**游戏业内一直流传着一句话“2D游戏其本质就是动画的集合”。**

1. 影响游戏加载速度因素：(不考虑玩家)

（1）、接入方式和接入带宽 ＞＜

用户选用的接入方式、选购的接入带宽，要与经常使用的业务应用相匹配。如果用户经常使用速率高的业务，但选购的接入带宽较低，也会影响感知速率。

（2）、访问网站的用户激增

同时访问某一网站的用户激增，访问该网站的用户所体验的网速就会变慢，网站也可能瘫痪。

（3）、高峰时段上网

高峰时段，上网用户增多，对整个网络资源的占用量加大，网络瓶颈环节增加，网速也会变慢。

（4）、网站的系统配置

网站服务器的并行处理能力不够强大，难以承载较多用户的访问，也会造成用户感受的速率较低。

（5）、网站的连接带宽

网站连接到互联网的带宽不够高，也会影响到用户的感受速率。

（6）、少数用户的应用占据大量带宽

互联网上的资源是在线用户共享的。有些用户使用特殊软件下载或上传信息，占用了绝大部分带宽，影响其他用户使用。

（7）、加载资源的大小

解决：

在开发之前、上线之前，都应该将项目所用的所有资源，包括声音、图片和文件，压缩一下。例如：图片——可以在线压缩【https://tinypng.com/】,文件——使用谷歌的closure compiler压缩工具压缩一下。

注意：将文件压缩、图片打包时，涉及浏览器的加载方式。chrome浏览器是异步加载方式，【不知道其他浏览器是不是异步的？】因此将cocos项目中的JS文件，全部压缩成一个文件、图片打包成一个文件，此时，单个的文件变得很大，加载时间很长。所以不建议将所有的文件压缩成一个文件，但是可以将单个文件单独压缩，当然最好是大文件。

（8）、加载服务器上没有的资源。

优化方式：

1、开发之前，将所有的图片、音频资源，打包；

2、开发中，使用Cocos压缩引擎。调试时，没有错误提示；

1. 游戏加载速度优化
2. 将首页和游戏页，分隔开来【首页使用H5做，游戏页使用Cocos做】；
3. 在首页加载完毕之后，将游戏中所用的资源【包括声音、图片和引擎】；

var res\_list = [

//js

"src/cocos2d-js-v3.1-lite.js",

"src/Resources.js",

//pic

//Loading页面

"res/loadBg.png",

"res/jiafen.png",

"res/loadProcessUp.png",

"res/loadProcessDown.png",

//bg

"res/bg.png",

"res/8\_sky.jpg",

//guide

"res/game.png",

//Processsbar

"res/loadProcessDown.png",

"res/loadProcessUp.png",

//house

"res/house.png",

//barrier

"res/zaw.png",

//cloud

"res/yun.png",

//car

"res/car.png",

//Map

"res/Map.png",

//logo

"res/jiafen.png",

//Road

"res/cj1\_lu.jpg",

"res/cj2\_lu.jpg",

"res/cj3\_lu.jpg",

"res/cj4\_lu.jpg",

"res/cj5\_lu.jpg",

"res/cj6\_lu.jpg",

"res/cj7\_lu.jpg",

"res/cj8\_lu.jpg",

//Labelbg

"res/k\_gj.png",

"res/k\_gls.png",

"res/k\_yl.png",

//button

"res/btn\_share.png",

"res/btn\_wycj.png",

"res/btn\_replay.png",

"res/btn\_goonplay.png",

"res/btn\_goon.png",

"res/btn\_zy.png",

//popup

"res/pass.png",

"res/jiayou.png",

//GameOver

"res/tk\_wzj.png",

"res/tk\_zj.png",

"res/stars.png",

"res/bg\_map.jpg"

];

1. 加载JS文件【引擎、资源列表】：

function LoadJS(filename){

var fileref = document.createElement('script');

fileref.setAttribute("type","text/javascript");

fileref.setAttribute("src",filename);

document.getElementsByTagName("head")[0].appendChild(fileref);

}

1. 加载图片资源：

function LoadPic(filename){

//Picture

var fileref = document.createElement('img');

fileref.setAttribute("type","image/jpeg");

fileref.setAttribute("src",filename);

}

1. 为了避免首页加载时间过长，可以在游戏页面加载完毕【window.onload()】之后，再开始加载资源：

window.onload = function(){

for(var i=0;i<res\_list.length;++i){

loadRes(res\_list[i]);

}

};

function loadRes(filename){

var filetype = filename.substring(filename.lastIndexOf("."),filename.length);

switch (filetype){

case ".js":

case ".plist":

LoadJS(filename);

break;

case ".jpg":

case ".jpeg":

case ".png":

LoadPic(filename);

break;

}

}

* 问题：虽然游戏中，也可能用到\*.plist和\*.js【程序】，为什么并没有使用这种方法实现预加载？

解答：

因为在JS中，程序可以使用引擎中的各种变量，解析时，控制台会报错。\*.plist文件在解析时，控制台也会发生报错。

1. 判断是否为微信浏览器？

if(cc.sys.browserType == cc.sys.BROWSER\_TYPE\_WECHAT)

1. 限制仅能在微信浏览器中打开:

* 原理:

方法一:

检测是否为微信浏览器，如果是，则打开。判定条件:

(cc.sys.isMobile && window.navigator.userAgent.indexOf("MicroMessenger") != -1),true:是微信浏览器。

方法二:

微信浏览器内部初始化成功以后，会调用**WeixinJSBridgeReady**接口，因此只要监听这个接口即可。

window.onload = function () {

//隐藏微信右上角菜单

document.addEventListener('WeixinJSBridgeReady',

function onBridgeReady() {

cc.game.onStart = function () {

//适配模式

var mode = cc.sys.isMobile && window.navigator.userAgent.indexOf("MicroMessenger") != -1 ? cc.ResolutionPolicy.EXACT\_FIT : cc.sys.isMobile ? cc.ResolutionPolicy.EXACT\_FIT : cc.ResolutionPolicy.SHOW\_ALL;

cc.view.setDesignResolutionSize(640, 960, mode);

cc.view.resizeWithBrowserSize(true);

//load resources

MyLoaderScene.preLoad(g\_preRes, g\_resources, function () {

cc.spriteFrameCache.addSpriteFrames(res.card\_plist, res.card\_png);

cc.director.runScene(new IndexScene());

}, this);

};

cc.game.run("gameCanvas");

});

};

1. 限制:在上述示例的基础上，限制不可在电脑端的微信浏览器上打开，仅可手机微信中打开

…

MyLoaderScene.preLoad(g\_preRes, g\_resources, function () {

if((cc.sys.OS\_ANDROID== cc.sys.os)||(cc.sys.OS\_IOS== cc.sys.os))

{

cc.director.runScene(new GameScene());

}else{

$("#Cocos2dGameContainer").css("display","none");

alert("请在手机微信浏览器中打开！");

}

}, this);

…

1. .html和.jsp文件的区别：

.jsp是动态网页，.html是静态网页，静态网页就是一旦发布网页，内容就不能修改，除非修改代码，而动态网页就是网页中的某些内容可以修改，比如说：新闻发布，我们可以把新闻内容放到数据库，通过修改数据库里的内容来发布最新新闻。

* 将.html改为.jsp的步骤：

1. <%@ page language="java" import="java.util.\*" pageEncoding="gbk"%> 选择性的加，不过一般都加上的；
2. 将后缀名改为“.jsp”。
3. chrome浏览器打开cocos 2d-js项目，出现：cocos2d: ERROR: Failed to link program: [object WebGLProgram]

原因：Uniforms with the same name but different type/precision

解决：修改CCGLProgram.js文件中：

var preStr = (type == this.\_glContext.VERTEX\_SHADER) ? “precision highp float;” : “precision mediump float;”;

将”mediump”改为”highp”

1. 在chrome浏览器中，查看打开一个网页所需要加载的各种资源的时间：

chrome浏览器🡪F12或者打开《开发者工具》🡪Network🡪打开对应的网页。

注意：第三步和第四步 不能颠倒。

1. Cocos2d-X的特点：

基于OpenGL，支持硬件加速，似的Cocos2D-X开发出来的项目FPS是非常高的。

1. cocos 2d-js中index.html文件解析

<head>

<!—标题可以显示中文-->

<meta charset="utf-8">

<!—书签标题-->

<title>COCOS 2014 秋季开发者大会报名</title>

<!—网页图标-->

<link rel="icon" type="image/GIF" href="res/favicon.ico"/>

<!—外部CSS-->

<link href="style.css" rel="stylesheet" type="text/css"/>

<meta name="apple-mobile-web-app-capable" content="yes"/>

<meta name="full-screen" content="yes"/>

<meta name="screen-orientation" content="portrait"/>

<meta name="viewport" content="target-densitydpi=high-dpi,width=device-width"/>

<meta name="x5-fullscreen" content="true"/>

<meta http-equiv="Access-Control-Allow-Origin" content="\*"/>

<meta name="360-fullscreen" content="true"/>

</head>

说明：使用外部样式表CSS,你就可以通过更改一个文件来改变整个站点的外观。

1. Viewport的作用：

手机浏览器是把页面放在一个虚拟的“窗口”（viewport）中，通常这个虚拟的“窗口”（viewport）比屏幕宽，这样就不用把每个网页挤到很小的窗口中（这样会破坏没有针对手机浏览器优化的网页的布局），用户可以通过平移和缩放来看网页的不同部分。

<meta name="viewport"

content="

height = [pixel\_value | device-height] ,

width = [pixel\_value | device-width ] ,

initial-scale = float\_value ,

minimum-scale = float\_value ,

maximum-scale = float\_value ,

user-scalable = [yes | no] ,

target-densitydpi = [dpi\_value | device-dpi | high-dpi | medium-dpi | low-dpi]"

/>

属性值：

* width

控制 viewport 的大小，可以指定的一个值，如果 600，或者特殊的值，如 device-width 为设备的宽度（单位为缩放为 100% 时的 CSS 的像素）。

* height

和 width 相对应，指定高度。

* target-densitydpi

一个屏幕像素密度是由屏幕分辨率决定的，通常定义为每英寸点的数量（dpi）。Android支持三种屏幕

像素密度：低像素密度，中像素密度，高像素密度。一个低像素密度的屏幕每英寸上的像素点更少，而一个高像素密度的屏幕每英寸上的像素点更多。Android Browser和WebView默认屏幕为中像素密度。

target-densitydpi 属性的取值范围:

* device-dpi –使用设备原本的 dpi 作为目标 dp。 不会发生默认缩放。
* high-dpi – 使用hdpi 作为目标 dpi。 中等像素密度和低像素密度设备相应缩小。
* medium-dpi – 使用mdpi作为目标 dpi。 高像素密度设备相应放大， 像素密度设备相应缩小。 这是默认的target density.
* low-dpi -使用mdpi作为目标 dpi。中等像素密度和高像素密度设备相应放大。
* dpi\_value – 指定一个具体的dpi 值作为target dpi. 这个值的范围必须在70–400之间。

为了防止Android Browser和WebView 根据不同屏幕的像素密度对你的页面进行缩放，你可以将viewport的target-densitydpi 设置为 device-dpi。当你这么做了，页面将不会缩放。相反，页面会根据当前屏幕的像素密度进行展示。在这种情形下，你还需要将viewport的width定义为与设备的width匹配，这样你的页面就可以和屏幕相适应。

* initial-scale

初始缩放。即页面初始缩放程度。这是一个浮点值，是页面大小的一个乘数。例如，如果你设置初始缩放为“1.0”，那么，web页面在展现的时候就会以target density分辨率的1:1来展现。如果你设置为“2.0”，那么这个页面就会放大为2倍。

* minimum-scale
* maximum-scale

最小与最大缩放。即允许的最大缩放程度。这也是一个浮点值，用以指出页面大小与屏幕大小相比的最大乘数。例如，如果你将这个值设置为“2.0”，那么这个页面与target size相比，最多能放大2倍。

注意：所有的缩放值都必须在0.01–10的范围之内。

* user-scalable

用户调整缩放。即用户是否能改变页面缩放程度。如果设置为yes则是允许用户对其进行改变，反之为no。默认值是yes。如果你将其设置为no，那么minimum-scale 和 maximum-scale都将被忽略，因为根本不可能缩放。

//不允许用户缩放

<meta name="viewport" content="user-scalable=no"/>

//允许用户移动

<meta name="viewport" content="target-densitydpi=high-dpi"/>

//设置屏幕密度为高频，中频，低频自动缩放

<meta name="viewport" content="width=device-width,target-densitydpi=high-dpi,initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0"/>

1. Cocos 2d-JS中，main.js文件解析：

cc.game.onStart = function () {//游戏开始

//设置浏览器meta来适配屏幕，引擎内部会根据屏幕大小来适配meta的viewpoint值，会达到更好的屏幕适配效果。

cc.view.adjustViewPort(true);

cc.view.enableAutoFullScreen(false);

//针对手机浏览器和PC浏览器的不同，启用不同的分辨率适配策略。

var mode = cc.sys.isMobile && window.navigator.userAgent.indexOf("MicroMessenger") != -1 ? cc.ResolutionPolicy.FIXED\_HEIGHT : cc.sys.isMobile ? cc.ResolutionPolicy.FIXED\_WIDTH : cc.ResolutionPolicy.SHOW\_ALL;

cc.view.setDesignResolutionSize(640, 831, mode);

//预加载图片声音等资源

cc.LoaderScene.preload(g\_resources, function () {

//执行场景

cc.director.runScene(new MainScene());

}, this);

};

cc.game.run();//运行游戏

1. Cocos 2D-js中，resources.js文件解析：

var res = {//资源列表

background\_png: "res/background.jpg",

map\_png:"res/map.png"

};

var g\_resources = [];

for (var i in res) {//将所有资源路径存储到数组中

g\_resources.push(res[i]);

}

1. JavaScript

使用Html只能制作出静态的网页，无法独立完成与客户端动态交互的网页设计。虽然也有其他的语言如CGI、ASP、java等能制作出交互的网页，但是因为其编程方法较为复杂，因此Netscape公司开发出了JavaScript语言。引入了Java语言的概念，是内嵌于HTML中的脚本语言。

JavaScript语言是网页中广泛使用的一种脚本语言，使用JavaScript可以使网页产生动态效果，JavaScript以其小巧简单而备用用户的欢迎。

由于JavaScript由Java集成而来的，因此它是一种面向对象的程序设计语言。它所包含的对象有两个组成部分，即变量和函数，也称为属性和方法。

JavaScript是一种解释型的、基于对象的脚本语言。尽管与C++这样的成熟的面向对象的语言相比，JavaScript的功能要弱一些，但对于它的预期用途而言，JavaScript的功能已经足够大了。JavaScript是一种宽松类型的语言。宽松类型意味着不必显示定义变量的数据类型。事实上JavaScript更进一步，无法在JavaScript中明确地定义数据类型。此外，在大多数情况下，JavaScript会根据需要自动进行转换。

JavaScript为网页设计人员提供了极大的灵活性，它能够将网页中的文本、图形、声音和动画等各种媒体形式捆绑在一起，形成了一个紧密结合的信息源。

* Java和JavaScript的异同点

Java和JavaScript语言虽然在语法上很相似，但它们仍然是两种不同的语言。JavaScript仅仅是一种嵌入到HTML文件中的描述性语言，它并不编译产生机器代码，只是有浏览器的解释器将其动态的处理成可执行的代码。而Java语言与JavaScript相比，则是一种比较复杂的编译型语言。

* JavaScript基本使用方法：

<script type="text/javascript">

document.write("Hello World!")

</script>

<script> 标签用于定义客户端脚本，比如 JavaScript。

script 元素既可以包含脚本语句，也可以通过 src 属性指向外部脚本文件。必需的 type 属性规定脚本的 MIME 类型。

必选的属性

属性 值 描述

type MIME-type 指示脚本的 MIME 类型。

可选的属性

属性 值 描述

async async 规定异步执行脚本（仅适用于外部脚本）。

charset charset 规定在外部脚本文件中使用的字符编码。

defer defer 规定是否对脚本执行进行延迟，直到页面加载为止。

language script 不赞成使用。规定脚本语言。请使用 type 属性代替它。

src URL 规定外部脚本文件的 URL。

xml:space preserve 规定是否保留代码中的空白。

* JavaScript具有以下语言特点：
  + JavaScript是一种脚本编写语言，采用小程序端的方式实现编程，开发过程非常简单；
  + JavaScript是一种基于对象的语言，它能运用已经创建的对象；
  + JavaScript具有简单性。首先它是一种基于Java基本语句和控制流之上的简单而紧凑的设计语言，其次它的变量类型是采用弱类型，并未使用严格的数据类型。
  + JavaScript是动态的，它可以直接对用户或客户的输入做出响应，无需经过Web服务器程序。
  + JavaScript是一种安全性语言，他不允许访问本地硬盘，并且不能将数据存入到服务器上，不允许对网络文档进行修改和删除，只能通过浏览器实现信息浏览或动态交互，从而有效地防止数据丢失。
  + JavaScript具有跨平台性。它依赖于浏览器本身，与操作环境无关。
* JavaScript的事件
* 浏览器的内部对象

使用浏览器的内部对象，可以实现与HTML文档进行交互。浏览器的内部对象主要包括以下几个：

浏览器对象(navigator)：提供有关浏览器的信息

文档对象(document):document包含了与文档匀速一起工作的对象；

窗口对象(window):window对象处于对象层次的最顶端，它提供了处理浏览器窗口的方法和属性

位置对象(location):location对象提供了与当前打开的URL一起工作的方法和属性，它是一个静态的对象。

历史对象(history):history对象提供了与历史清单有关的信息。

* + navigator对象

Navigator 对象包含有关浏览器的信息

Navigator 对象集合

集合 描述

plugins[] 返回对文档中所有嵌入式对象的引用。

该集合是一个 Plugin 对象的数组，其中的元素代表浏览器已经安装的插件。Plug-in 对象提供的是有关插件的信息，其中包括它所支持的 MIME 类型的列表。

Navigator 对象属性

属性 描述

appCodeName 返回浏览器的代码名。

appMinorVersion 返回浏览器的次级版本。

appName 返回浏览器的名称。

appVersion 返回浏览器的平台和版本信息。

browserLanguage 返回当前浏览器的语言。

cookieEnabled 返回指明浏览器中是否启用 cookie 的布尔值。

cpuClass 返回浏览器系统的 CPU 等级。

onLine 返回指明系统是否处于脱机模式的布尔值。

platform 返回运行浏览器的操作系统平台。

systemLanguage 返回 OS 使用的默认语言。

userAgent 返回由客户机发送服务器的 user-agent 头部的值。

userLanguage 返回 OS 的自然语言设置。

Navigator 对象方法

方法 描述

javaEnabled() 规定浏览器是否启用 Java。

taintEnabled() 规定浏览器是否启用数据污点 (data tainting)。

Navigator 对象包含的属性描述了正在使用的浏览器。可以使用这些属性进行平台专用的配置。

Navigator 对象的实例是唯一的，可以用 Window 对象的 navigator 属性来引用它。

* + Document对象

每个载入浏览器的 HTML 文档都会成为 Document 对象。

Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

提示：Document 对象是 Window 对象的一部分，可通过 window.document 属性对其进行访问。

Document 对象集合

集合 描述

all[] 提供对文档中所有 HTML 元素的访问。

anchors[] 返回对文档中所有 Anchor 对象的引用。

applets 返回对文档中所有 Applet 对象的引用。

forms[] 返回对文档中所有 Form 对象引用。

images[] 返回对文档中所有 Image 对象引用。

links[] 返回对文档中所有 Area 和 Link 对象引用。

Document 对象属性

属性 描述

body 提供对 <body> 元素的直接访问。

对于定义了框架集的文档，该属性引用最外层的 <frameset>。

cookie 设置或返回与当前文档有关的所有 cookie。

domain 返回当前文档的域名。

lastModified 返回文档被最后修改的日期和时间。

referrer 返回载入当前文档的文档的 URL。

title 返回当前文档的标题。

URL 返回当前文档的 URL。

Document 对象方法

方法 描述

close() 关闭用 document.open() 方法打开的输出流，并显示选定的数据。

getElementById() 返回对拥有指定 id 的第一个对象的引用。

getElementsByName() 返回带有指定名称的对象集合。

getElementsByTagName() 返回带有指定标签名的对象集合。

open() 打开一个流，以收集来自任何 document.write() 或 document.writeln() 方法的输出。

write() 向文档写 HTML 表达式 或 JavaScript 代码。

writeln() 等同于 write() 方法，不同的是在每个表达式之后写一个换行符。

* + window对象

Window 对象表示浏览器中打开的窗口。

如果文档包含框架（frame 或 iframe 标签），浏览器会为 HTML 文档创建一个 window 对象，并为每个框架创建一个额外的 window 对象。

Window 对象集合

集合 描述

frames[]

返回窗口中所有命名的框架。

该集合是 Window 对象的数组，每个 Window 对象在窗口中含有一个框架或 <iframe>。属性 frames.length 存放数组 frames[] 中含有的元素个数。注意，frames[] 数组中引用的框架可能还包括框架，它们自己也具有 frames[] 数组。

Window 对象属性

属性 描述

closed 返回窗口是否已被关闭。

defaultStatus 设置或返回窗口状态栏中的默认文本。

document 对 Document 对象的只读引用。请参阅 Document 对象。

history 对 History 对象的只读引用。请参数 History 对象。

innerheight 返回窗口的文档显示区的高度。

innerwidth 返回窗口的文档显示区的宽度。

length 设置或返回窗口中的框架数量。

location 用于窗口或框架的 Location 对象。请参阅 Location 对象。

name 设置或返回窗口的名称。

Navigator 对 Navigator 对象的只读引用。请参数 Navigator 对象。

opener 返回对创建此窗口的窗口的引用。

outerheight 返回窗口的外部高度。

outerwidth 返回窗口的外部宽度。

pageXOffset 设置或返回当前页面相对于窗口显示区左上角的 X 位置。

pageYOffset 设置或返回当前页面相对于窗口显示区左上角的 Y 位置。

parent 返回父窗口。

Screen 对 Screen 对象的只读引用。请参数 Screen 对象。

self 返回对当前窗口的引用。等价于 Window 属性。

status 设置窗口状态栏的文本。

top 返回最顶层的先辈窗口。

window window 属性等价于 self 属性，它包含了对窗口自身的引用。

screenLeft

screenTop

screenX

screenY

只读整数。声明了窗口的左上角在屏幕上的的 x 坐标和 y 坐标。IE、Safari 和 Opera 支持 screenLeft 和 screenTop，而 Firefox 和 Safari 支持 screenX 和 screenY。

Window 对象方法

方法 描述

alert() 显示带有一段消息和一个确认按钮的警告框。

blur() 把键盘焦点从顶层窗口移开。

clearInterval() 取消由 setInterval() 设置的 timeout。

clearTimeout() 取消由 setTimeout() 方法设置的 timeout。

close() 关闭浏览器窗口。

confirm() 显示带有一段消息以及确认按钮和取消按钮的对话框。

createPopup() 创建一个 pop-up 窗口。

focus() 把键盘焦点给予一个窗口。

moveBy() 可相对窗口的当前坐标把它移动指定的像素。

moveTo() 把窗口的左上角移动到一个指定的坐标。

open() 打开一个新的浏览器窗口或查找一个已命名的窗口。

print() 打印当前窗口的内容。

prompt() 显示可提示用户输入的对话框。

resizeBy() 按照指定的像素调整窗口的大小。

resizeTo() 把窗口的大小调整到指定的宽度和高度。

scrollBy() 按照指定的像素值来滚动内容。

scrollTo() 把内容滚动到指定的坐标。

setInterval() 按照指定的周期（以毫秒计）来调用函数或计算表达式。

setTimeout() 在指定的毫秒数后调用函数或计算表达式。

Window 对象表示一个浏览器窗口或一个框架。在客户端 JavaScript 中，Window 对象是全局对象，所有的表达式都在当前的环境中计算。也就是说，要引用当前窗口根本不需要特殊的语法，可以把那个窗口的属性作为全局变量来使用。例如:

可以只写 document，而不必写 window.document。同样，可以把当前窗口对象的方法当作函数来使用，如只写 alert()，而不必写 Window.alert()。

除了上面列出的属性和方法，Window 对象还实现了核心 JavaScript 所定义的所有全局属性和方法。

* + location位置对象

Location 对象包含有关当前 URL 的信息。

Location 对象是 Window 对象的一个部分，可通过 window.location 属性来访问。

Location 对象属性

属性 描述

hash 设置或返回从井号 (#) 开始的 URL（锚）。

host 设置或返回主机名和当前 URL 的端口号。

hostname 设置或返回当前 URL 的主机名。

href 设置或返回完整的 URL。

pathname 设置或返回当前 URL 的路径部分。

port 设置或返回当前 URL 的端口号。

protocol 设置或返回当前 URL 的协议。

search 设置或返回从问号 (?) 开始的 URL（查询部分）。

Location 对象方法

属性 描述

assign() 加载新的文档。

reload() 重新加载当前文档。

replace() 用新的文档替换当前文档。

Location 对象存储在 Window 对象的 Location 属性中，表示那个窗口中当前显示的文档的 Web 地址。它的 href 属性存放的是文档的完整 URL，其他属性则分别描述了 URL 的各个部分。这些属性与 Anchor 对象（或 Area 对象）的 URL 属性非常相似。当一个 Location 对象被转换成字符串，href 属性的值被返回。这意味着你可以使用表达式 location 来替代 location.href。

不过 Anchor 对象表示的是文档中的超链接，Location 对象表示的却是浏览器当前显示的文档的 URL（或位置）。但是 Location 对象所能做的远远不止这些，它还能控制浏览器显示的文档的位置。如果把一个含有 URL 的字符串赋予 Location 对象或它的 href 属性，浏览器就会把新的 URL 所指的文档装载进来，并显示出来。

实例：把用户带到一个新的地址

<html>

<head>

<script type="text/javascript">

function currLocation()

{

alert(window.location)

}

function newLocation()

{

window.location="/index.html"

}

</script>

</head>

<body>

<input type="button" onclick="currLocation()" value="显示当前的 URL">

<input type="button" onclick="newLocation()" value="改变 URL">

</body>

</html>

* + history历史对象

History 对象属性

属性 描述

length 返回浏览器历史列表中的 URL 数量。

History 对象方法

方法 描述

back() 加载 history 列表中的前一个 URL。

forward() 加载 history 列表中的下一个 URL。

go() 加载 history 列表中的某个具体页面。

History 对象最初设计来表示窗口的浏览历史。但出于隐私方面的原因，History 对象不再允许脚本访问已经访问过的实际 URL。唯一保持使用的功能只有 back()、forward() 和 go() 方法。

实例：

下面一行代码执行的操作与单击后退按钮执行的操作一样：

history.back()

下面一行代码执行的操作与单击两次后退按钮执行的操作一样：

history.go(-2)

1. JQeury库

jQuery 库可以通过一行简单的标记被添加到网页中。jQuery 是一个 JavaScript 函数库。

jQuery 库包含以下特性：

* HTML 元素选取
* HTML 元素操作
* CSS 操作
* HTML 事件函数
* JavaScript 特效和动画
* HTML DOM 遍历和修改
* AJAX
* Utilities

jQuery有两个版本可供使用：

* + - Production version - 用于实际的网站中，已被精简和压缩。
    - Development version - 用于测试和开发（未压缩，是可读的代码）

将jQuery添加到网页上，有两种方法：

* 本地JQuery

jQuery 库是一个 JavaScript 文件，您可以使用 HTML 的 <script> 标签引用它：

<head>

<script src="jquery.js"></script>

</head>

注意：此引用<script> 标签应该位于页面的 <head> 部分。

提示：您是否很疑惑为什么我们没有在 <script> 标签中使用 type="text/javascript" ？

在 HTML5 中，不必那样做了。JavaScript 是 HTML5 以及所有现代浏览器中的默认脚本语言！

* 通过 CDN（内容分发网络）引用它

谷歌和微软的服务器都存有 jQuery 。

Google CDN:

<head>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js">

</script>

</head>

* 以上两种方法的对比：

使用谷歌或微软的 jQuery，有一个很大的优势：

许多用户在访问其他站点时，已经从谷歌或微软加载过 jQuery。所有结果是，当他们访问您的站点时，会从缓存中加载 jQuery，这样可以减少加载时间。同时，大多数 CDN 都可以确保当用户向其请求文件时，会从离用户最近的服务器上返回响应，这样也可以提高加载速度。

1. Array数组

JS中，Array提供对创建任何数据类型的数组的支持。

* + join(separator)：将数组的元素组成一个字符串,元素由指定的分隔符分隔开来，默认以逗号分割:

var array=[1,2,3,4,5];

var str="";

str=array.join("-");

结果：1-2-3-4-5

* + slice(start,end)：返回从原数组中指定开始下标start到结束下标end之间的项组成的新数组：

var array=[1,2,3,4,5];

var arra= array.slice(1,3);

结果：2,3

注意：包含起始值，但是不包含结束值。

* + revease()：数组反转

var array=[1,2,3,4,5];

array= array.reverse();

结果：5,4,3,2,1

* + splice()：截取原数组中的某段区域中的一段,删除该段数值，并返回该值

var array=[1,2,3,4,5];

str= array.splice(2,1);

结果：str数组——[3]

* + concat()：连接成一个新的数组，并返回新数组

var array=[1,2,3,4,5];

array= array.concat(6,6);

结果：1,2,3,4,5,6,6

* + push()：在数组最后，添加元素，

var array=[1,2,3,4,5];

array.push(1);

结果：1,2,3,4,5,1

注意：返回的是数组的长度。

* + pop()：删除原数组最后一项，并返回删除元素的值

var array=[1,2,3,4];

array.pop();

结果：[1,2,3]

* + unshift():在原数组的开头，添加对应的元素，并返回数组【所有元素】

var array=[1,2,3,4,5];

array.unshift(-2,-1);

结果：[-2,-1,1,2,3,4,5]

* + shift()：删除原数组第一项，并返回删除元素的值

var array=[1,2,3,4,5];

str= array.shift();

结果：[2,3,4,5]

* + sort()：

说明：

如果调用该方法时没有使用参数，将按字母顺序对数组中的元素进行排序，说得更精确点，是按照字符编码的顺序进行排序。要实现这一点，首先应把数组的元素都转换成字符串（如有必要），以便进行比较。

如果想按照其他标准进行排序，就需要提供比较函数，该函数要比较两个值，然后返回一个用于说明这两个值的相对顺序的数字。比较函数应该具有两个参数 a 和 b，其返回值如下：

若 a 小于 b，在排序后的数组中 a 应该出现在 b 之前，则返回一个小于 0 的值。

若 a 等于 b，则返回 0。

若 a 大于 b，则返回一个大于 0 的值。

var array=[];

for(var i=0;i<15;++i){

array.push(i);

}

* 缺省时排序：

array.sort();

结果：0,1,10,11,12,13,14,2,3,4,5,6,7,8,9

* 升序：

array.sort(function(a,b){return a>b?1:-1});

结果：0,1,2,3,4,5,6,7,8,9,10,11,12,13,14

* 降序：

array.sort(function(a,b){return a<b?1:-1});

结果：14,13,12,11,10,9,8,7,6,5,4,3,2,1,0

* 混乱排列：

array.sort(function(){return Math.random()<0.5?0:1;});

结果：7,0,13,3,4,5,6,12,8,9,11,10,1,2,14

结论:

1.数组调用sort方法后，会影响本身(而非生成新数组)

2.sort()方法默认是按字符来排序的，所以在对数字型数组排序时，不可想当然的以为会按数字大小排序！

3.要改变默认的sort行为(即按字符排序)，可以自行指定排序规则函数。

特殊实例：

* + 数组的拷贝

var array=[1,2,3];

var b=array.slice(0,array.length);

* + 清空数组

var array=[1,2,3];

array.splice(0,array.length);

1. Cocos 2D-js中，project.json文件解析：

项目相关配置，由原来的cocos2d.js中转移到project.json中，该文件需要与index.html同级，一般建议放在根目录下。

由原来的cocos2d.js改为project.json的好处如下：

* 原来的cocos2d.js中参杂了一些逻辑代码，无法很好体现其就是作为项目的配置文件。
* 用json格式作为配置文件会更友好。也可以使得cocos-console、cocos-utils，甚至是用户自定义脚本工具能够使用同一个配置文件。

字段说明：

* debugMode:

相当于原来的COCOS2D\_DEBUG，其中：0表示关闭调试模式，1表示log，2表示warn，3表示error，5表示logForWebPage，6表示warnForWebPage，7表示errorForWebPage。

* renderMode

表示渲染模式。0为默认，由引擎自动选择绘制模式。1为强制使用Canvas绘制模式，2为强制使用WebGL绘制模式，但是实际上WebGL仍然可能会在一些移动浏览器上被忽略，而自动使用Canvas绘制模式。

* showFPS

表示是否开启帧率显示。默认为false。若取值为真则会在游戏窗口左下角显示绘制函数调用次数，渲染时间和帧率。

* frameRate

表示设置帧率。默认为60。

* engineDir

用于指定引擎目录。区别在于，在v2版本中，指向的是../cocos2d-html5/cocos2d，但在v3版本中，只要指向../cocos2d-html5就行了。默认为"frameworks/cocos2d-html5"。在debug模式下，如果使用完整引擎来开发你的游戏，你需要将Web引擎的相对地址设置在这个字段下，但是如果你使用单文件引擎，可以忽略这个字段。

* jsList

相当于原来的appFiles，用于罗列用于js列表。用户JS脚本列表，游戏中依赖的JS脚本都应该列入这个列表中。

* id

相当于原来的tag，用于指定canvas的id。Web引擎页面中canvas元素的id，仅服务于Web引擎。

* modules

用于指定加载模块。将你的游戏需要引入的模块添加到一个数组中，仅服务于Web引擎。

在v2版本中，只能通过loadExtension、box2d、chipmunk、pluginx等开关来指定是否使用某个模块。

在v3版本中则使用modules字段，灵活进行配置，这也能较好的支持今后引擎进行模块裁剪。配置如下：

"modules" : ["extensions", "pluginx", "box2d", "chipmunk"]

//示例:

{

"project\_type": "javascript",//工程语言

"debugMode" : 1,//调试模式

"showFPS" : false,//是否显示FPS

"frameRate" : 60,//刷新频率，默认为1/60

"id" : "gameCanvas",//游戏页面<canvas>的ID值，必须保持一致

"renderMode" : 1,//渲染模式

"engineDir":"frameworks/cocos2d-html5",//引擎的相对路径

"modules" : ["core", "actions", "menus","progress-timer"],

"jsList" : [//游戏中用到的JS文件，像微信接口在游戏中使用不到的地方，就不用添加了

"src/resource.js",

"src/register.js"

]

}

如果想知道引擎中具体提供了哪些模块，可以参见frameworks\cocos2d-html5\moduleConfig.json文件中的module字段。

1. moduleConfig.json解析：

该配置文件相当于v2版本中的jsloader.js。改造的目的是为了使得配置纯粹化，同时也能比较好的支持cocos-console、cocos-utils甚至是用户自定义脚本工具。

字段说明

* module

配置各个模块的js列表。key名即为模块名称。这些key名将会在project.json的modules字段中使用。倘若不清楚project.json里面究竟有哪些模块可以配置，就可以直接查看该文件。每个模块的配置对象是一个数组，数组项分两种，一种是模块名，一种是js路径。

例如：

"menus" : [

"core", "actions",

"cocos2d/menus/CCMenuItem.js",

"cocos2d/menus/CCMenu.js"

]

此配置的意思是，menus模块依赖于core和actions模块，并且自身包含cocos2d/menus/CCMenuItem.js和cocos2d/menus/CCMenu.js。

Web引擎支持模块化，引擎被划分为多个子模块，模块之间相互依赖并提供给游戏对应的功能，下面是Web引擎的模块列表：

Web版本模块化的原理和使用方法

| **模块名称** | **描述** |
| --- | --- |
| core | Web引擎核心模块，包含游戏导演，节点，场景，图层，精灵，TTF文本，事件管理器，计时器和贴图等核心功能。 |
| actions | 动作模块包含节点的基础动作以及缓动动作，重复动作，变速动作等修饰性动作 |
| audio | 声音系统 |
| menus | 包含菜单和菜单项节点，用于制作游戏菜单 |
| labels | 包含位图文本节点以及Atlas文本节点 |
| ccui | 支持布局的Cocos UI界面系统，包含众多类型的UI控件 |
| shape-nodes | 包含DrawNode类型节点，可以用来绘制直线，多边形，曲线等 |
| render-texture | 包含RenderTexture类型节点用于绘制附加内容到一张贴图上 |
| particle | 粒子系统模块，内建丰富粒子系统效果并支持ParticleDesigner导出的粒子系统 |
| progress-timer | ProgressTimer节点模块可以将一个普通节点转换为线性或环形进度条节点 |
| transitions | 场景转换模块包含众多内建场景转换特效 |
| text-input | 简单的文本输入框节点 |
| tilemap | 砖块地图模块支持TMX文件读取并创建砖块地图图层 |
| parallax | ParallaxNode可以为其子结点提供Parallax带景深移动效果 |
| editbox | 编辑框可以提供各种复杂的文字输入框效果 |
| clipping-nodes | 剪裁节点模块提供ClippingNode节点，可以将其子结点按蒙板节点的外形来进行剪裁 |
| motion-streak | MotionStreak节点可以为目标节点提供跟踪尾巴特效，支持颜色填充或贴图填充的尾巴 |
| node-grid | 用于支持高级特效的网格节点 |
| actions3d | 可以应用在节点之上的3D特效：翻页，晃动，波浪等等 |
| gui | 另一种简单的GUI系统，包含一些游泳的UI控件，但不支持智能布局 |
| Cocos Studio | Cocos Studio模块可以读取Cocos Studio编辑器导出项目 |
| ccbreader | Cocos Builder读取模块，可以读取Cocos Builder导出项目 |
| ccpool | 内建的对象缓冲池支持 |
| pluginx | 社交网络插件系统 |
| plugin-facebook | Cocos2d-JS内建的Facebook SDK支持 |
| socketio | ScoketIO库支持 |
| box2d | Box2d物理引擎支持 |
| chipmunk | Chipmunk物理引擎支持 |
| spine | Spine骨骼动画编辑器支持 |
| cocos2d | 快捷模块包，包含Cocos2d-JS的所有主要模块："core", "render-texture", "motion-streak", "node-grid", "clipping-nodes", "effects", "shape-nodes", "actions", "actions3d", "progress-timer", "transitions", "labels", "particle", "text-input", "menus", "tilemap", "parallax", "audio" |
| extensions | 快捷模块包，包含所有扩展模块："Cocos Studio", "ccbreader", "ccpool", "gui", "editbox", "spine" |
| external | 快捷模块包，包含外部库模块："box2d", "chipmunk", "socketio", "pluginx" |

每个模块具体所依赖的文件可以参考frameworks/cocos2d-html5/modulesConfig.json。

在project.json中配置modules字段可以为开发者的游戏选择依赖模块，好的模块配置可以帮助开发者，显著降低游戏脚本的大小和游戏页面加载速度。当Cocos Console在调试模式下测试游戏的时候，仅会引入所需要的模块的脚本。在发布模式下编译生成的时候，将仅打包模块定义中定义包含的模块。

1. Cocos 2D-JS中,.cocos-project.json文件解析：

该文件作用：好像是生成Release版本所必须要的配置文件。

内容：

{

"has\_native": true,

"project\_type": "js"//工程语言

}

1. Cocos2d-JS的屏幕适配方案

Cocos2d-JS中的屏幕适配模式在Web引擎中和原生引擎中的差异比较大，但是API还是统一一致的。

* 关于Web引擎屏幕适配

对于如何将网页的内容适配到不同尺寸的浏览器窗口，基于CSS3的Responsive Design是目前非常热门的解决方案。可惜它不适用于Canvas中的游戏内容，也不适用于解决原生平台，所以Cocos2d引擎为游戏开发者提供了屏幕适配策略（Resolution Policy）解决方案。

* 设置屏幕适配策略（Resolution Policy）

如果你还没有用过Resolution Policy，只需要在游戏载入过程完成之后（cc.game.onStart函数回调中），调用下面的代码：

cc.view.setDesignResolutionSize(320, 480, cc.RESOLUTION\_POLICY.SHOW\_ALL);

setDesignResolutionSize函数的前两个参数是你想要在你的代码中使用的游戏分辨率，第三个参数就是你选择的适配方案。引擎中内置了5种适配方案，每种都有自己独特的行为，详见下文。

如果你已经设置了设计分辨率，那么你可以直接设置你的Resolution Policy：

cc.view.setResolutionPolicy(cc.RESOLUTION\_POLICY.NO\_BORDER);

* + SHOW\_ALL (PROPORTION\_TO\_FRAME + SHOW\_ALL)

SHOW\_ALL模式会尽可能按原始宽高比放大游戏世界以适配外框(Frame)，同时使得游戏内容全部可见，所以浏览器宽高比不同于游戏宽高比时，窗口中会有一定的留白。留白部分的颜色或背景可以通过设置外框的CSS属性来修改。

* + NO\_BORDER (EQUAL\_TO\_FRAME + NO\_BORDER)

NO\_BORDER模式会尽可能按原始宽高比放大游戏世界以适配外框，并且保证不留空白。所以浏览器宽高比不同于游戏宽高比时，游戏世界会被部分切割。同时，在这种情况下，cc.visibleRect代表的就是Canvas在游戏世界中的视窗，大小比cc.winSize要小一些。

* + EXACT\_FIT (EQUAL\_TO\_FRAME + EXACT\_FIT)

EXACT\_FIT模式会忽略原始宽高比放大游戏世界以完全适应外框，所以浏览器宽高比不同于游戏宽高比时，游戏世界会被一定程度拉伸。

* + FIXED\_WIDTH (EQUAL\_TO\_FRAME + FIXED\_WIDTH)

FIXED\_WIDTH模式会横向放大游戏世界以适应外框的宽度，纵向按原始宽高比放大。结果有两种可能，类似与SHOW\_ALL模式的结果（如图），或者类于NO\_BORDER模式。它与前面两种模式的差别在于，在FIXED\_WIDTH模式下游戏世界坐标系等同于Canvas元素坐标系，并且Canvas元素必然占满整个外框。

注意：图中情况下与SHOW\_ALL模式的区别，此时Canvas大小是整个外框大小，所以可显示内容区域实际上比SHOW\_ALL模式更多。

* + FIXED\_HEIGHT (EQUAL\_TO\_FRAME + FIXED\_HEIGHT)

与前一个模式相反，FIXED\_HEIGHT模式会纵向放大游戏世界以适应外框的高度，横向按原始宽高比放大。结果同上。

在这个模式下，与NO\_BORDER模式的区别是此时游戏世界坐标系和大小等同于Canvas坐标系。

* Resolution Policy的意义

使用Resolution Policy的好处很明显，不论设备屏幕大小如何，也不论浏览器窗口的宽高比，你的游戏场景都会被自动放缩到屏幕大小。更重要的是，在游戏代码中，你将永远使用你所设计的游戏分辨率来布置游戏场景。比如说，如果你将设计分辨率设置为320 \* 480，那么在游戏代码中你的游戏窗口右上角坐标将永远是(320, 480)(在FIXED\_WIDTH模式高度可能会被缩放，同样在FIXED\_HEIGHT模式下宽度可能存在缩放的情况)。

* 监听浏览器窗口大小变化事件

新的适配方案允许在浏览器大小变化的时候自动重新尝试适配。比如说，当用户拖拽来改变浏览器大小，或者更有用的情况，当他们转动自己手机方向的时候。游戏中任意时刻都可以开启这种行为，只需要调用cc.view的resizeWithBrowserSize函数：

cc.view.resizeWithBrowserSize(true);

为了更灵活得应对变化，我们为cc.view提供了一个新的函数，你可以通过setResizeCallback函数注册一个回调函数来监听浏览器窗口大小变化事件：

cc.view.setResizeCallback(function() {

// 做任何你所需要的游戏内容层面的适配操作

// 比如说，你可以针对用户的移动设备方向来决定所要应用的适配模式

});

* Fullscreen API

Fullscreen API是浏览器允许Web页面在获得用户全屏幕的一个新的制定中的API。

Cocos2d-JS在移动端浏览器中会尝试自动进入全屏幕来给用户更好的游戏体验（需要指出并不是所有浏览器都支持这个API）。

另一方面，桌面端几乎所有现代浏览器都支持Fullscreen API，如果你希望使用这个API，Cocos2d-JS也简化了它的使用方式：

尝试进入全屏模式（需要用户交互）: cc.screen.requestFullScreen(targetElement, onFullScreenCallback);

检测是否处于全屏模式: cc.screen.fullScreen();

退出全屏模式: cc.screen.exitFullScreen();

* 重要概念
  + 游戏外框 Frame

游戏外框是你的游戏Canvas元素的初始父节点，一般情况下，它是html文档的body元素。但是如果你愿意，它可以是DOM结构中的任意容器节点。Canvas元素的初始大小并不重要，屏幕适配过程中它会被自动放缩来适应你设置的外框大小。 再次提醒，如果你希望游戏窗口适应整个浏览器窗口，那么只需要将Canvas元素直接放在body下。

* + 游戏容器 Container

在Cocos2d-JS的初始化进程中，引擎会自动将你的Canvas元素放置到一个DIV容器中，而这个容器会被加入到Canvas的原始父节点（游戏外框）中。这个游戏容器是实现屏幕适配方案的重要辅助元素，你可以通过cc.container来访问它。

* + 游戏世界 Content

游戏世界代表游戏内使用的世界坐标系。

* + 视窗 Viewport

视窗是游戏世界相对于游戏Canvas元素坐标系中的坐标及大小.

* + 容器适配策略 Container Strategy

容器适配策略负责对游戏容器和游戏Canvas元素进行放缩以适应游戏外框。

* + 内容适配策略 Content Strategy

内容适配策略负责将游戏世界放缩以适应游戏容器，同时也会计算并设置视窗。

* Web引擎与原生引擎的差异

在原生引擎中，由于应用总是占据整个游戏窗口或在移动端占用全屏幕空间，我们沿用了Cocos2d-x中的屏幕适配方案，cc.view中的API一致，但是不提供Web引擎中的高级适配功能，下面是区别列表：

原生引擎只提供默认的5中适配策略。

容器适配策略和内容适配策略在原生引擎中都不存在，不能够自由组合。

不能够通过继承实现用户容器适配策略和内容适配策略。

1. 变形

注意：在EXACT\_FIT模式下,圆形图片会发生变形。尤其在旋转时，给人的误区是：旋转时，锚点改变了。实质上是因为图片变形了，产生的视觉误差，锚点并没有改变。

解决方案：

将canvas元素的尺寸设置为浏览器的可见区域宽高。保持EXACT\_FIT模式。

cc.view.setDesignResolutionSize(document.documentElement.clientWidth,document.documentElement.clientHeight,cc.ResolutionPolicy.EXACT\_FIT);

问题：虽然此方法可以做到圆形不变形、圆形旋转不发生偏移，但同时背景在不同的手机上有的显示不全、有的有黑边，也就是不再全屏。因此可以手动在程序中，将游戏场景缩放，实现全屏显示：

缩放比例 = (屏幕可见的宽/背景图的宽 , 屏幕可见的高/背景图的高);

注意：

1、获取屏幕可见的尺寸时，在进入游戏之前和进入游戏之后，获取屏幕可见区域的尺寸不同。后者较小。在游戏中，获取游戏之前屏幕可见区域尺寸，有两种方法。

（1）、在游戏页的game.html中，定义两个变量，存储游戏之前屏幕可见区域尺寸，以便在游戏中使用。不过不推荐使用该方法。

（2）、因为使用进入游戏之前的可见区域尺寸，设置的适配方案。所以获取适配方案的尺寸，就是进入游戏之前的可见区域尺寸cc.view.getDesignResolutionSize()。

2、因为在iPhone4中的可见区域尺寸为960\*1274，相应的图片可能会变小，所以，可以采用两种方法：

（1）、将图形按照960\*1274作图，问题：图片资源变大，加载资源时间变长。

（2）、可以按照小图做，但是在程序中，适当放大。问题：图片清晰度失真。

//show\_all模式下，修改圆形

* 注意：在<head>标签下，

1. 检测设备横屏/竖屏

在Html5页面上的<header>标签，设置屏幕朝向：

<meta name="screen-orientation" content="portrait"/>

属性content取值：landscape——横向 portrait——纵向

* 方法一：利用Html5，检测屏幕朝向：

<body>

<script>

var supportsOrientationChange = "onorientationchange" in window,

orientationEvent = supportsOrientationChange ? "orientationchange" : "resize";

// 监听事件

window.addEventListener(orientationEvent, function() {

if(window.orientation != 0){

alert("请竖屏玩耍！");

}

}, false);

</script>

</body>

window.orientation的值域：

0 – 正常的纵向方向

-90 – 从纵向顺时针旋转后的水平方向

90 – 从纵向逆时针旋转后的水平方向

180 – 目前还不支持，但应该是纵向旋转

* 方法二：因为在移动设备上，一般不会发生屏幕的尺寸改变。只有在横屏/竖屏切换时，才会发生屏幕尺寸改变，所以可以检测Cocos屏幕尺寸是否变化来判断

cc.view.resizeWithBrowserSize(true);

cc.view.setResizeCallback(function(){//尺寸大小改变后的回调函数

alert("改变了！");

});

1. 页面的相关尺寸

说明：以下数据，都是在iPhone4测试所得。

* 网页的可见区域：

//方法一：

document.body.clientHeight

document.body.clientWidth

//方法二：

document.body.offsetWidth

document.body.offsetHeight

注意：

方法一和方法二中获取页面可见高度时，获取到的只是页面最小的高度。比如页面只有一个button，那么该js语句获得的就是24。

实例：【964,150】

* 屏幕分辨率

window.screen.width

window.screen.height

实例：【320,480】

* 屏幕可用工作的区域

window.screen.availWidth

window.screen.availHeight

实例：【320,460】

* 获取网页正文全文的宽和高

//方法一：

document.body.scrollWidth

document.body.scrollHeight

//方法二：

document.documentElement.clientWidth

document.documentElement.clientHeight

实例：【980,1274】

1. 创建场景时，onEnter方法的作用是：

场景初始化完成即将展示的消息回调。在onEnter方法中，必须调用this.\_super()来确保场景正确的初始化。

1. 定时器、菜单按钮、动作的回调函数可以有参数，也可以没有参数。带参数是，其值为this对象。
2. 如何获取屏幕页面的大小尺寸？

方法一：var winsize=cc.director.getWinSize;

winsize.width//页面的宽

winsize.height//页面的高

方法二：cc.winSize.width//页面的宽

cc.winSize.height//页面的高

1. 在页面上绘制的所有图片都会发生放大、失真效果：

原因：

1、适配模式选择全屏模式cc.ResolutionPolicy.[EXACT\_FIT](file:///E:\%E6%9F%A5%E8%AF%A2\Cocos2d-JS-v3.0-API\symbols\cc.ResolutionPolicy.html#EXACT_FIT)【main.js函数中的cc.view.setDesignResolutionSize参数三】;

2、屏幕的长宽比与图片的长宽比不一致。【例如，图片是1000\*800，屏幕是320\*480，解决：将cc.view.setDesignResolutionSize的参数一和二，改为480\*320】

1. 如何将一个精灵不显示？

解决：

//创建一个精灵，居中，并添加到Layer中

var sprite=cc.Sprite.create(res.bk\_png);

sprite.setPosition(cc.p(cc.winSize.width/2,cc.winSize.height/2));

this.addChild(sprite);

//设置该精灵的属性，使其不显示

sprite.visible=false;//false:不显示，true:显示

同理 ，Label等控件也是这样设置的。

注意:

如果精灵的路径是网址时，虽然可以正常显示，但是获取该精灵的尺寸时，值均为0。因此可以:

* 定义一个全局变量:

var g\_image= new Image();

* 为该标签加载资源:

g\_image.src= "http://wx.qlogo.cn/mmopen/KydxAIB52xnFDwghvDYRqGwPyAdialHGr5G5c7EhUFXM283xSBHMsHKsBo73qQibwOwwicPdWkkLicHoK2MQYabibCzkEeawBQzNE/0";

* 根据尺寸缩放到(100,100)的大小:

var portraits= new cc.Sprite(g\_image.src);

portraits.attr({

x:cc.winSize.width\* 0.5,

y:cc.winSize.height\* 0.5,

scaleX:100/g\_image.width,

scaleY:100/g\_image.height

});

1. 如何实现一个精灵的移动效果？

//MoveTo

var moveToAction=cc.moveTo(2,cc.p(x,y));//参数1：动作执行时间，参数2要移动要的坐标点

//MoveBy

var moveByAction=cc.moveBy(2,cc.p(x,y));

注意：MoveTo与MoveBy参数二cc.p(x,y)两者之间的不同：

MoveTo中的坐标参数是相对于左下角原点的；

MoveBy中的坐标参数是相对于当前位置的；

//对应的精灵对应的动作

sprite1.runAction(moveToAction);

sprite2.runAction(moveByAction);

1. 如何实现一个动作的反动作？

varmoveToBackAction=moveToAction.reverse();

//其中moveToBackAction就是moveToAction的相反动作。

1. 如何实现：将两个或者多个动作连接起来？

解决:使用序列动作类cc.Sequence或者是同步动作类cc.Spawn。

//按一定的顺序执行动作

var eq=cc.Sequence(action1,delay1,action2,delay2);

sprite.runAction(seq);

//其中delay1、delay2是指：执行完动作action1后，间隔多长时间，执行动作action2【var delay = cc.delayTime(0.25);】

1. 如何实现：两个动作之间添加回调函数

var seq= cc.sequence(scale1, cc.callFunc(callback),scale2);

其中，callback是回调函数。

1. 如何实现：动作的循环执行？

上一实例中，sprite.runAction(seq.repeatForever);

1. 如何做到一个动作的往返操作？

var seq=cc.Sequence(action1,action1.reverse());//一个动作的往返动作

1. 如何将图片1和图片2两个文件，组成一个动画播放？

var animation=new cc.Animation();

//遍历将图片添加到动画中[图片名：dance\_1.png、dance\_2.png…]

for(var i=0;i<5;++i){

var frameName = "res/dance\_" +i+ ".png";

animation.addSpriteFrameWithFile(frameName);

}

// And display 60 frames per second

animation.setDelayPerUnit(1 / 60);

animation.setRestoreOriginalFrame(true);

//动画动作

var action = cc.animate(animation);

sprite.runAction(action);//执行动画

1. 如何做到遍历01、02、03……10、11？

解决:((i<10)?(”0”+i):i)

1. 如何在监视器中，调用this.sprite变量？

将要操作的语句放到一个函数中，例如：move().通过event.getCurrentTarget().move()调用。

1. 单点监听：

cc.eventManager.addListener({

event: cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches: true,

onTouchBegan: this.onTouchBegan,

onTouchMoved: this.onTouchMoved,

onTouchEnded: this.onTouchEnded

}, this);

onTouchBegan:function(touch, event) { return true;}，

onTouchMoved:function(touch, event) {},

onTouchEnded:function(touch, event) {return true;}

//获取触摸点的位置

var posX = touch.getLocationX();

var posY = touch.getLocationY();

//在触发函数上，调用别的函数：

问题：参数一二的作用：

1. 获取页面的宽度和高度：

* Cocos游戏代码中

cc.winSize.width//宽度

cc.winSize.height//高度

cc.director.getWinsize().width;//宽度

cc.director.getWinsize().height;//高度

* 网页中

<script>

var w1=document.documentElement.clientWidth;

var h1=document.documentElement.clientHeight;

if(w1<h1){

alert("请横屏游戏!");

}

</script>

说明：横屏或者竖屏游戏，主要是根据页面的宽度和高度比，来实现横屏还是竖屏游戏。

1. 设置精灵的显示效果：

* 缩放

sprite.setScale(x,y);

//参数一：宽的缩放比例

//参数二：高的缩放比例

* 旋转

sprite.setRotation(90);//设置旋转的角度

1. 设置精灵的动作：

* 缩放

varscaleAction= cc.scaleTo(5,5);

this.playerSprite.runAction(scaleAction);//执行缩放动画

//参数一：缩放动画时间

//参数二：缩放比例

* 旋转

1. 播放声音：

cc.audioEngine.playEffect(res.Hit\_wav);//音效

cc.audioEngine.playMusic(res.Hit\_wav);//背景音乐

问题：两者之间的不同之处：

1. 如何实现进度条效果？

//准备：前后两张图片【后面的是进度条背景、前面的是进度条】

思路：

多种进度条的实现方法。但是答题思路都是——初始化进度条背景和进度条、在定时器中，更新进度。

方法一：利用图片横向的拉伸，实现进度条的增长。

注意：假如进度条的增长方向是从右向左，拉伸时，所以前面的进度条的锚点设置为sprite.attr({anchorX:1,anchorY:0})

注意：将进度条减少到0的判定条件设置成：this.precent<=0。

原因：浮点数的精度问题。

//processBar Bg

var processDown = cc.Sprite.create("res/loadProcessDown.png");

processDown.attr({

x:cc.winSize.width/2+60,

y:cc.winSize.height/2+220,

scale:0.5

});

this.addChild(processDown);

//processBar fore

var processUp = cc.Sprite.create("res/loadProcessUp.png");

processUp.attr({

anchorX:0,

anchorY:0,

x:3,

y:5

});

processDown.addChild(processUp,0,1);

this.removeObj.push(processUp);

方法二:利用点九图实现拉伸

cc.spriteFrameCache.addSpriteFrames(res.ProgressBar\_plist,res.ProgressBar\_png);

//processBar Bg

var processDown =cc.Sprite.create("res/loadProcessDown.png");

processDown.attr({

x:cc.winSize.width/2+60,

y:cc.winSize.height/2+190,

scale:0.5

});

this.addChild(processDown);

//processBar fore

var processUp = cc.Scale9Sprite.create("loadProcessUp.png");

processUp.attr({

anchorX:0,

anchorY:0,

x:3,

y:5

});

processDown.addChild(processUp,0,2);

this.removeObj.push(processUp);

注意：点九图在使用时，图片资源必须是帧图片，且加载时，没有"#"。

方法三:设置显示的矩形大小，来实现进度条效果

1. 定义一个全局变量：processUpWidth:0,//进度条缩放后的宽度
2. 创建进度条的前后背景精灵：

//注意：此方法中，仅实现了左边减少，没做其他的三种。

var processDown = cc.Sprite.create(res.processDown\_png);

processDown.attr({

anchorX:0.5,

anchorY:0.5,

x:cc.winSize.width/2,

y:cc.winSize.height/2,

scale:0.5

});

this.addChild(processDown,0);

this.processUp = cc.Sprite.create(res.processUp\_png);

//左边增加、减少——进度条的锚点设置为：(0,0.5)位置X设置：进度条背景的X位置-(进度条的宽度\*缩放比例)的一半

this.processUp.attr({

anchorX:0,

anchorY:0.5,

x:(cc.winSize.width-this.processUp.\_getWidth()\*0.5)/2,

y:cc.winSize.height/2,

scale:0.5

});

this.addChild(this.processUp,1);

//注意;此时已经缩放完毕，所以this.processUp.\_getWidth()就是获取的缩放后的宽

this.processUpWidth=this.processUp.\_getWidth();

//开始时，从最长减到0

this.width=this.processUp.\_getWidth();

1. 在定时器中，添加：

if(this.width<=0){

this.width = this.processUpWidth;

}else{

this.width-=1;

}

this.processUp.setTextureRect(cc.rect(0,0,this.width,this.processUp.\_getHeight()));

方法四: cc.ProgressTimer——水平进度条

initProcessBar\_Control\_Horizontal:function(){

//label

var label = cc.LabelTTF.create("ProgressTimer—Horizontal","微软雅黑",15);

label.attr({

x:cc.winSize.width/2-60,

y:cc.winSize.height/2+130,

color:cc.color(0,0,0)

});

this.addChild(label);

//processBar Bg

var processDown = cc.Sprite.create("res/loadProcessDown.png");

processDown.attr({

x:cc.winSize.width/2+60,

y:cc.winSize.height/2+110,

midPoint:cc.p(0,0),

scale:0.5

});

this.addChild(processDown);

var processBar = new cc.ProgressTimer(new cc.Sprite("res/loadProcessUp.png"));

processBar.attr({

midPoint:cc.p(0,0),

x:3,

y:5,

type:cc.ProgressTimer.TYPE\_BAR,

barChangeRate:cc.p(1,0)

});

processDown.addChild(processBar,0,4);

this.removeObj.push(processBar);

}

//更新

this.removeObj[i].setPercentage(this.s);

方法五: cc.ProgressTimer——圆形

initProcessBar\_Control\_Radial:function(){

//label

var label = cc.LabelTTF.create("ProgressTimer—Radial","微软雅黑",15);

label.attr({

x:cc.winSize.width/2-60,

y:cc.winSize.height/2+70,

color:cc.color(0,0,0)

});

this.addChild(label);

var processDown = cc.Sprite.create("res/radial\_bg.png");

processDown.attr({

x:cc.winSize.width/2+60,

y:cc.winSize.height/2+70,

midPoint:cc.p(0,0),

scale:0.5

});

this.addChild(processDown);

var processBar = cc.ProgressTimer.create(new cc.Sprite("res/radial\_fore.png"));

processBar.attr({

x:0,

y:0,

midPoint:cc.p(0.5,0.5),

type:cc.ProgressTimer.TYPE\_RADIAL,

barChangeRate:cc.p(1,0)

});

//另一种写法:

/\*

self.ProcessBarUp.setType(cc.ProgressTimer.TYPE\_RADIAL);

self.ProcessBarUp.setBarChangeRate(cc.p(1, 0));

self.ProcessBarUp.setPercentage(0);

self.ProcessBarUp.setMidpoint(cc.p(0.5, 0.5));

\*/

processDown.addChild(processBar,0,5);

this.removeObj.push(processBar);

}

//更新

this.removeObj[i].setPercentage(this.s);

方法六：控件—cc.ControlPotentiometer

initProcessBar\_Control\_Radial2:function(){

//label

var label = cc.LabelTTF.create("ControlPotentiometer","微软雅黑",15);

label.attr({

x:cc.winSize.width/2-60,

y:cc.winSize.height/2,

color:cc.color(0,0,0)

});

this.addChild(label);

var processBar = cc.ControlPotentiometer.create(

"res/radial\_bg.png",

"res/radial\_fore.png",

"res/radial\_center.png");

processBar.attr({

x:cc.winSize.width/2+60,

y:cc.winSize.height/2,

scale:0.5

});

//processBar.addTargetWithActionForControlEvents(this, this.valueChanged, cc.CONTROL\_);

this.addChild(processBar,0,6);

this.removeObj.push(processBar);

//this.valueChanged(processBar,cc.CONTROL\_EVENT\_VALUECHANGED);

}

//更新

this.removeObj[i].setValue(this.s\*0.01);

方法七：裁剪和蒙版

说明：此方法也是我认为最完美的方法。原因：解决了——进度条两端是圆弧时，缩放的不自然现象。

分析：

1. 在进度条背景的区域，添加对应的蒙版。
2. 将进度条设置在蒙版区域之外；
3. 根据蒙版的特性，区域外的内容不显示，仅显示区域内的内容。

initPerfect:function(){

//label

var label = cc.LabelTTF.create("Perfect","微软雅黑",15);

label.attr({

x:cc.winSize.width/2-80,

y:cc.winSize.height/2-40,

color:cc.color(0,0,0)

});

this.addChild(label);

//processBar Bg

var processDown = cc.Sprite.create("res/loadProcessDown.png");

processDown.attr({

x:cc.winSize.width/2+60,

y:cc.winSize.height/2-40,

scale:0.5

});

this.addChild(processDown);

//processBar fore

var processUp = cc.Sprite.create("res/loadProcessUp.png");

processUp.attr({

anchorX:0.5,

anchorY:0.5,

x:-327\*0.5,

y:10

});

//蒙版

var stencil= new cc.DrawNode();

stencil.drawRect(cc.p(0,0),cc.p(327,28),cc.color(255,0,0),1,cc.color(0,0,0));

//裁剪

var clipper= new cc.ClippingNode();

clipper.attr({

x:3,

y:8

});

clipper.setStencil(stencil);

clipper.addChild(processUp,0,1);

processDown.addChild(clipper,0,7);

this.removeObj.push(clipper);

},

//更新

this.removeObj[i].getChildByTag(1).setPositionX(327\*(this.s/100-0.5));

方法八、设置圆环的旋转角度

资源：

代码：

//进度percent

//更新

circle.setRotaion(percent\*360);

效果：在进度条的后面，紧跟着加载进程



//Html 5中实现

1. 添加各种标签

<div id="time1" style="position:absolute; width:90%; left:5%; display: none">

<img id= "progress\_back" src="../images/time-1.png" style="width: 100%; height:10%;">

</div>

<div id="process" style="position:absolute; width:90%;left:20.5%; display: none">

<img id= "progress\_fore" src="../images/time-2.png" style="width: 100%; height:3%;">

<!--TODO:2-->

<div id="progress-img" class="progress-img" style="position:absolute;">

0%

</div>

</div>

1. 更新

//获取对应的div

var num= Math.floor((g\_answer\_index)/13\*100);

var progress= num+"%";

$("#progress\_fore").css("width",num\*0.9\*0.9+"%");

$("#progress-img").css("marginLeft",num\*0.9\*0.9+"%");

var label= $("#progress-img");

label.html(progress);

注意：上述是从右向左减少，改为从上向下减少，需要的步骤是：

1、this.processUp的X改为cc.winSize.width/2;

2、this.processUp的Y改为(cc.winSize.height-this.processUp.\_getWidth()\*0.5)/2;

3、将processDown、this.processUp逆时针旋转90度

processDown.setRotation(-90);

this.processUp.setRotation(-90);

1. 在html页面上添加注释

标签<!-添加注释内容->

1. 查看该游戏是横版还是竖版：

* 查看index.html的<canvas>标签

<!-横版->

<canvas id="gameCanvas" width="480" height="320"></canvas>

<!-竖版->

<canvas id="gameCanvas" width="320" height="480"></canvas>

* 查看main.js文件中适配模式

cc.view.setDesignResolutionSize(480, 320, cc.ResolutionPolicy.SHOW\_ALL);//横版

cc.view.setDesignResolutionSize(320, 480, cc.ResolutionPolicy.SHOW\_ALL);//竖版

1. 在Resources.js文件中，简单快速添加资源？

var res = {

helloBG\_png : "res/hellobg.jpg",

Start\_N\_png : "res/start\_n.png",

Start\_S\_png : "res/start\_s.png"}

varg\_resources = [];//定义数组

for (var i in res) {

g\_resources.push(res[i]);

}

//注意不要使用以下方法，在红色环境【最后两项是plist时】下，下面方法会报错。

var res = {

helloBG\_png : "res/helloBG.png",

start\_n\_png : "res/start\_n.png",

start\_s\_png : "res/start\_s.png",

PlayBG\_png : "res/PlayBG.png",

running\_plist : "res/running.plist",

running\_png : "res/running.png"

};

varg\_resources = [];//定义数组

for (var i=0;i<sign\_res.length;i++){

g\_resources.push(sign\_res[i]);

}

1. 自定义加载页面

* 自定义加载场景
  + 方法一：

//以下代码直接拷贝即可，放到一个\*.js文件中。真正有用的仅仅只是红色字体

varMyLoaderScene = cc.Scene.extend({

\_interval : null,

\_length : 0,

\_count : 0,

\_label : null,

\_img:null,

\_className:"MyLoaderScene",//类名

init : function(){

var self = this;

// 纯背景色层

varbgLayer = self.\_bgLayer = cc.LayerColor.create(cc.color(32, 32, 32, 255));

bgLayer.setPosition(cc.visibleRect.bottomLeft);

self.addChild(bgLayer, 0);

//显示当前加载速度

var label = self.\_label = cc.LabelTTF.create("玩命加载中... 0%", "Arial", 24);

label.setPosition(cc.pAdd(cc.visibleRect.center, cc.p(0, 0)));

label.setColor(cc.color(180, 180, 180));

bgLayer.addChild(this.\_label, 10);

//加载时显示的图标、图片

varimg = self.\_img = new cc.Sprite(res.loading\_png);

img.setPosition(cc.pAdd(cc.visibleRect.center, cc.p(0, 85)));

img.setColor(cc.color(180, 180, 180));

bgLayer.addChild(this.\_img, 10);

return true;

},

\_initStage: function (img, centerPos) {

var self = this;

var texture2d = self.\_texture2d = new cc.Texture2D();

texture2d.initWithElement(img);

texture2d.handleLoadedTexture();

var logo = self.\_logo = cc.Sprite.create(texture2d);

logo.setScale(cc.contentScaleFactor());

logo.x = centerPos.x;

logo.y = centerPos.y;

self.\_bgLayer.addChild(logo, 10);

},

onEnter: function () {

var self = this;

cc.Node.prototype.onEnter.call(self);

self.schedule(self.\_startLoading, 0.01);

},

onExit: function () {

cc.Node.prototype.onExit.call(this);

vartmpStr = "玩命加载中... 0%";

this.\_label.setString(tmpStr);

},

/\*\*

\* init with resources

\* @param {Array} resources

\* @param {Function|String} cb

\*/

initWithResources: function (resources, cb) {

if(typeof resources == "string") resources = [resources];

this.resources = resources || [];

this.cb = cb;

},

\_startLoading: function () {

var self = this;

self.unschedule(self.\_startLoading);

var res = self.resources;

self.\_length = res.length;

self.\_count = 0;

cc.loader.load(res, function(result, count){ self.\_count = count; }, function(){

if(self.cb)

self.cb();

});

self.schedule(self.\_updatePercent);

},

//进度更新函数

\_updatePercent: function () {

var self = this;

var count = self.\_count;

var length = self.\_length;

var percent = (count / length \* 100) | 0;

percent = Math.min(percent, 100);

self.\_label.setString("玩命加载中... " + percent + "%");

if(count >= length) self.unschedule(self.\_updatePercent);

}

});

MyLoaderScene.preload = function(resources, cb){

var \_myLoaderScene = null;

if(!\_myLoaderScene) {

\_myLoaderScene = new MyLoaderScene();

\_myLoaderScene.init();

}

\_myLoaderScene.initWithResources(resources, cb);

cc.director.runScene(\_myLoaderScene);

return \_myLoaderScene;

};

* + 方法二：

var MyLoaderScene = cc.Scene.extend({

\_interval : null,

\_length : 0,

\_count : 0,

\_className:"MyLoaderScene",

init : function(){

var self = this;

// bg

var bgLayer = self.\_bgLayer = cc.LayerColor.create(cc.color(229, 17, 17));

bgLayer.setPosition(cc.visibleRect.bottomLeft);

self.addChild(bgLayer, 0);

var ProcessBarDown = self.ProcessBarDown = new cc.Sprite(res.loadProcessDown\_png);

ProcessBarDown.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

bgLayer.addChild(this.ProcessBarDown, 10);

var ProcessBarUp = self.ProcessBarUp = new cc.Sprite(res.loadProcessUp\_png);

ProcessBarUp.attr({

x:cc.winSize.width/2-327\*0.5,

y:cc.winSize.height/2,

scaleX:0,

anchorX:0,

anchorY:0.5

});

bgLayer.addChild(this.ProcessBarUp, 10);

return true;

},

onEnter: function () {

var self = this;

cc.Node.prototype.onEnter.call(self);

self.schedule(self.\_startLoading, 0.5);

},

onExit: function () {

cc.Node.prototype.onExit.call(this);

this.ProcessBarUp.scale = 0.02;

},

initWithResources: function (resources, cb) {

if(typeof resources == "string") resources = [resources];

this.resources = resources || [];

this.cb = cb;

},

\_startLoading: function () {

var self = this;

self.unschedule(self.\_startLoading);

var res = self.resources;

self.\_length = res.length;

self.\_count = 0;

cc.loader.load(res, function(){

self.\_count++;

var count = self.\_count;

var length = self.\_length;

var percent = (count / length \* 100) | 0;

percent = Math.min(percent, 100);

self.ProcessBarUp.scaleX = percent/100;

}, function(){

if(self.cb)

self.cb();

});

}

});

MyLoaderScene.preload = function(resources, cb){

var \_myLoaderScene = null;

if(!\_myLoaderScene) {

\_myLoaderScene = new MyLoaderScene();

\_myLoaderScene.init();

}

\_myLoaderScene.initWithResources(resources, cb);

cc.director.runScene(\_myLoaderScene);

return \_myLoaderScene;

};

说明：

方法一和方法二的实现原理都是一样的——利用cc.loader.load()函数加载资源，并在其回调函数中，设置加载资源的进程。并将获取的进程，显示出来。

不同之处在于：

方法一：使用的定时器来显示加载进程。优点：因为是隔一段时间，才会更新进度，所以，显得加载速度快。缺点：使用self.schedule(this.OnUpate);因为没有指定定时器的时间，所以有时会显得不正常。

方法二：每次加载完一个资源，就会更新一下进程。优点：能够显示出完整的加载进程。缺点：正因为每加载完一个资源之后，都会更新一下，所以在用户看来加载速度慢，所以，可以将两者综合一下:

1. 在加载资源cc.loader.load()的回调函数中,记录加载进程；
2. 在开始加载资源文件时，开启更新进程定时器：self.schedule(self.onUpdate,0.3);
   * 方法三：方法一和方法二的结合

var MyLoaderScene = cc.Scene.extend({

\_length : 0,

\_count : 0,

\_className:"MyLoaderScene",

init : function(){

var self = this;

var ProcessBarDown = self.ProcessBarDown = new cc.Sprite(res.loadProcessDown\_png);

ProcessBarDown.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

self.addChild(self.ProcessBarDown, 10);

var ProcessBarUp = self.ProcessBarUp = new cc.Sprite(res.loadProcessUp\_png);

ProcessBarUp.attr({

x:0,

y:0.5\*38,//其中38加载进度条背景的高度。

scaleX:0,

anchorX:0,

anchorY:0.5

});

self.ProcessBarDown.addChild(self.ProcessBarUp, 10);

return true;

},

onEnter: function () {

var self = this;

cc.Node.prototype.onEnter.call(self);

self.\_startLoading();

},

onExit: function () {

cc.Node.prototype.onExit.call(this);

},

/\*\*

\* init with resources

\* @param {Array} resources

\* @param {Function|String} cb

\*/

initWithResources: function (resources, cb) {

if(typeof resources == "string") resources = [resources];

this.resources = resources || [];

this.cb = cb;

},

\_startLoading: function () {

var self = this;

//start the timer,update the process

self.schedule(self.\_updateData,0.001);

var res = self.resources;

self.\_length = res.length;

self.\_count = 0;

cc.loader.load(res, function(){

self.\_count++;

}, function(){

if(self.cb){//Loaded

self.cb();

}

});

},

\_updateData:function(delta){

var self= this;

var count = self.\_count;

var length = self.\_length;

var percent = (count / length \* 100) | 0;

percent = Math.min(percent, 100);

self.ProcessBarUp.scaleX = percent/100;

}

});

MyLoaderScene.preload = function(resources, cb){

var \_myLoaderScene = null;

if(!\_myLoaderScene) {

\_myLoaderScene = new MyLoaderScene();

\_myLoaderScene.init();

}

\_myLoaderScene.initWithResources(resources, cb);

cc.director.runScene(\_myLoaderScene);

return \_myLoaderScene;

};

说明：

1. 该方法在项目的资源很少时，甚至看不到自定义的加载场景【cc.LoaderScene可以看见图标和加载0%】。
2. 在使用该方法时，必须手动指定进度条背景的高度。

* 在\*.json文件中，加载上面创建的\*.js文件；
* 在index.html或者是main.js文件中，使用该类添加资源：

MyLoaderScene.preload(g\_resources, function () {

cc.director.runScene(new MenuScene());

}, this);

//其中，MyLoaderScene要与上面的类名对应。

* 想法：关于加载时间慢的问题的一些想法

在JS项目中，大部分加载的时间花费在以下：cocos引擎、JQuery架包、图片资源。其中cocos引擎和JQuery架包，可以通过压缩的方法变小，加速加载时间。图片资源可以这样做，加载资源不是一次性加载到页面中的。第一次加载的仅仅只是游戏的第一个场景中，用到的图片资源。进入游戏场景之后或者在使用之前的一个场景中，再加载对应的资源。

无论是原来的方法，还是现在的想法，都必须要将Loading页面中用到的资源首先加载。

//原来的

var res = {

//BackGroundLayer

bg\_jpg:"res/bg.jpg",

zhuan\_pan\_png:"res/zhuan\_pan.png",

award\_png:"res/award.png",

popup\_png:"res/popup.png",

fail\_png:"res/fail.png",

btn\_continue\_png:"res/btn\_continue.png",

loadProcessUp\_png:"res/loadProcessUp.png",

loadProcessDown\_png:"res/loadProcessDown.png",

btn\_return\_png:"res/btn\_return.png"

};

var g\_resources = [];

for(var i in res){

g\_resources.push(res[i]);

}

//Main.js中

MyLoaderScene.preload(g\_resources, function () {

cc.director.runScene(new GameScene());

}, this);

//想法

var res1 = {//加载页面和游戏中第一个场景中，要用到的资源

loadProcessUp\_png:"res/loadProcessUp.png",

loadProcessDown\_png:"res/loadProcessDown.png",

bg\_jpg:"res/bg.jpg",

zhuan\_pan\_png:"res/zhuan\_pan.png",

award\_png:"res/award.png"

};

var res2= {//进入游戏场景后，要加载的资源

popup\_png:"res/popup.png",

fail\_png:"res/fail.png",

btn\_continue\_png:"res/btn\_continue.png",

btn\_return\_png:"res/btn\_return.png"

}

var g\_resources1 = [];

var g\_resources2 = [];

for(var i in res1){

g\_resources1.push(res1[i]);

}

for(var j in res2){

g\_resourcesj.push(res1[j]);

}

//Main.js中

MyLoaderScene.preload(g\_resources1, function () {

cc.director.runScene(new GameScene());

cc.loader.load(g\_resources2);

}, this);

* 想法：不想要Loading界面

可以不使用Loading界面，直接使用cc.loader.load()函数，示例:

cc.loader.load(g\_resources, function () {

cc.director.runScene(new GameScene());

}, this);

1. cc.loader

原来的cc.Loader被改造为一个单例cc.loader，采用了插件机制设计，让loader做更纯粹的事。

各种资源类型的loader可以在外部注册进来，而不是直接将所有的代码杂揉在cc.Loader中，更好的方便管理以及用户自定义loader的创建。

cc.loader中包含了一些网络资源获取的基本api，例如加载js、加载image等。

注：以下描述中，全路径的意思为 "资源的根路径" + "资源路径"，例如设置图片资源根路径(cc.loader.resPath)为"res"，图片资源路径为"a.png"，那么全路径为"res/a.png"，非全路径为"a.png"。

resPath——资源的根路径（音频资源除外）。

audioPath——音频资源的根路径。

getXMLHttpRequest——获取XMLHttpRequest对象。

loadJs——加载js文件

参数：

url 资源路径（注意，必须是全路径）

cb 回调函数

返回值： 无

说明：其中，url可以是js文件路径列表，也可以是单个的文件路径。

load

加载资源的入口api。相当于原来的cc.Loader.preload。

参数：

res 资源列表，可以是一个数组或者一个string或者一个包含`src`属性的对象。

option 选项（可选），有以下属性：

cb 回调函数

cbTarget 回调函数的执行者

trigger 触发器（函数）

triggerTarget 触发器的执行者

cb 回调函数（可选）

release——释放资源缓存数据。

参数：

url 资源路径（非全路径）

用法：cc.loader.release("a.png");

releaseAlll——释放所有的资源缓存数据

用法：cc.loader.releaseAll();

1. 使用多张图片实现动画效果

//使用三张图片

//动画帧缓存

var animation=new cc.Animation();

for(var i=0;i<2;i++){

varframeName="res/gou"+i+".png";

animation.addSpriteFrameWithFile(frameName);

}

//设置FPS

animation.setDelayPerUnit(1/10);

animation.setRestoreOriginalFrame(true);

//动画动作

varanimateAction=cc.animate(animation);

dog.runAction(animateAction);

//循环播放

dog.runAction(animateAction.repeatForever());

1. 将多个图片资源打包成一个图片资源和\*.plist文件

* 安装TexturePackerGUI；【注意：一定要选择正式版，否则打包成的图片会出现乱码现象。】;
* 点击Add Sprites菜单，添加要打包的图片资源。
* 设置生成文件的路径；
* 点击publish即可。

1. 使用\*.plist和一张图片，绘制精灵：

cc.spriteFrameCache.addSpriteFrames(res.running\_plist);

var sprite = new cc.Sprite("#runner0.png");

sprite.attr({x:80, y:85});

this.addChild(sprite);

1. 使用\*.plist和一张图片，实现动画效果:

精灵表因为以下几点被广泛应用：

1、减少文件读写时间。读取一张图片比读取一堆小文件肯定要快。

2、减少内存消耗。OpenGL ES 1.1仅仅能够使用2的几次方大小的图片（也就是宽度或者高度是2,4,864,128,256,512,1024,...）。也就是说，OpenGL ES1.1将会分配给每个图片2的几次方大小的内存空间，即使你这张图片达不到这样的宽度和高度也会分配大于此图片的2的n次方大小的空间。那么运用这种图片集的方式将会减少内存碎片。

3、减少OpenGL ES绘制调用并且加速渲染。

Cocos2d-x v2.0升级到了OpenGL ES2.0.OpenGL ES2.0不会再分配2的几次方的内存块了，但是减少读取时间和绘制调用的优势依然存在。

//利用\*.plist和\*.png创建一个动画

//利用SpriteBatchNode将所有的帧包含到精灵节点中，并将其添加到层中

var spriteNode=new cc.SpriteBatchNode(res.running\_png);

this.addChild(spriteNode);

//运用CCSpriteFrameCache实例来确保帧名字对应帧边界，即加载\*.plist

cc.spriteFrameCache.addSpriteFrames(res.running\_plist);

//此时，精灵帧和精灵表已经加载成功，并且将精灵帧添加到场景中。

//创建精灵帧，并将该精灵添加到精灵节点spriteNode中

//注意此时的sprite已经加载到spriteNode中，而spriteNode已经加载到层上了。所以没必要再将sprite加载层上。

var sprite =new cc.Sprite("#runner0.png");

sprite.attr({

x:85,

y:80

});

spriteNode.addChild(sprite);

//create array 创建数组，存放所有的动画帧

var animationFrames=[];

for(var i=0;i<8;++i){

var str="runner"+i+".png";

var frame=cc.spriteFrameCache.getSpriteFrame(str);

animationFrames.push(frame);

}

//create animation

var animation=new cc.Animation(animationFrames,0.1);

//create animate action

var animateAction=cc.animate(animation).repeatForever();

//sprite run animation action

sprite.runAction(animateAction);

1. 实现移动效果：

//精灵移动

//create sprite

var sprite=new cc.Sprite(res.dog\_png);

sprite.attr({

x:85,

y:80

});

this.addChild(sprite);

//create move action创建移动动作

var moveAction =new cc.MoveBy(2,cc.p(0,100));

//sprite run sprite 精灵执行动作

sprite.runAction(moveAction);

1. 实现变速移动效果：

//精灵变速移动效果

//create sprite

var sprite=new cc.Sprite(res.dog\_png);

sprite.attr({

x:85,

y:80

});

this.addChild(sprite);

//create move action创建移动动作

var moveAction =new cc.MoveBy(2,cc.p(0,100));

//将上述动作的拷贝动作按cc.easeSineOut()速度执行。

var move=moveAction.clone().easing(cc.easeSineOut());

//sprite run sprite 精灵执行动作

sprite.runAction(move);

1. 微信接口

* 创建微信接口wechat.js文件

window.appMessage = {

"imgUrl" : "http://www.91yixun.cn/jianai/images/fgy.png",

"timeLineLink" :"http://www.91yixun.cn/jianai/share.html",

"tTitle" : "法国浪漫双人游大抽奖",

"tContent" : "浪漫法国双人游大抽奖，你也快去抽吧！"

};

window.shareMessage = {

"imgUrl" : "http://www.91yixun.cn/jianai/images/fgy.png",

"timeLineLink" :"http://www.91yixun.cn/jianai/share.html",

"tTitle" : "法国浪漫双人游大抽奖",

"tContent" : "浪漫法国双人游大抽奖，你也快去抽吧！"

};

document.addEventListener('WeixinJSBridgeReady',

function onBridgeReady() {

//发送给朋友

WeixinJSBridge.on('menu:share:appmessage', function(argv) {

WeixinJSBridge.invoke('sendAppMessage', {

"img\_url" : window.appMessage.imgUrl,

"link" : window.appMessage.timeLineLink,

//分享内容【上面】

"desc" : window.appMessage.tContent,

//分享标题【下面】

"title" : window.appMessage.tTitle

}, function(res) {

onConfirm(res);

});

});

//分享到朋友圈

WeixinJSBridge.on('menu:share:timeline', function(argv) {

WeixinJSBridge.invoke('shareTimeline', {

"img\_url" : window.shareMessage.imgUrl,

"img\_width" : "640",

"img\_height" : "640",

"link" : window.shareMessage.timeLineLink,

"desc" : window.shareMessage.tContent,

"title" : window.shareMessage.tTitle

}, function(res) {

onConfirm(res);

});

});

}, false);

//回调函数的处理事件

function onConfirm(res){

switch (res.err\_msg){

//发送给朋友🡪成功

case 'send\_app\_msg:ok':

case 'send\_app\_msg:confirm':

{

alert("send\_app\_msg成功！");

break;

}

//分享到朋友圈🡪成功

case 'share\_timeline:ok':

case 'share\_timeline:confirm':

{

alert("share\_timeline成功！");

break;

}

};

}

注意以下两种情况：

1. 当《发送给朋友》的链接timeLineLink值为""时，《发送给朋友》按钮的功能被屏蔽，功能正常。
2. 上面的红色文本，就是《发送给朋友》《分享到朋友圈》的单击事件。将对应的文本注释掉，就可以屏蔽对应按钮的单击事件。

* project.json配置文件中，将该文件添加到项目中：【可有可无，如果cocos代码中，使用了微信分享中的变量，才需要在\*.json中添加。】

//JS列表

"jsList" : [

"src/wechatAPI.js"

]

注意：可以将wechatAPI文件添加到src目录下，也可以放在根目录下。最好放在wechatAPI.js目录下，因为在将项目打包之后，没有了src文件夹，因此将不会找到wechatAPI.js文件，会报错。同理，jquery-1.8.2.min.js文件也是。

* 在首页index.html文件中，引用该js文件：

<script src="src/wechatAPI.js"></script>

* 《分享到朋友圈》的按钮的实现函数：

$.post("updateWeChatServlet.action");//服务器后台跳转

window.appMessage.tTitle="这个游戏也太虐了吧！我只颠了"+g\_score+"个球，你也来吧！还有机会赢乐视蓝牙音箱！";//更新分享标题

1. 动态屏蔽或显示微信的右上角分享菜单按钮

<script>

function onBridgeReady(){

//函数接口可能已经过期了

WeixinJSBridge.call('hideOptionMenu');

}

if (typeof WeixinJSBridge == "undefined"){

if( document.addEventListener ){

document.addEventListener('WeixinJSBridgeReady', onBridgeReady, false);

}else if (document.attachEvent){

document.attachEvent('WeixinJSBridgeReady', onBridgeReady);

document.attachEvent('onWeixinJSBridgeReady', onBridgeReady);

}

}else{

onBridgeReady();

}

//按钮的单击事件

function onClick(){

WeixinJSBridge.call('showOptionMenu');

}

</script>

//添加按钮

<img id="btn\_start" style="position:absolute;" src="images/btn\_play.png" onclick = "onClick()"/>

1. Cocos游戏中，实现页面的跳转：

location.href=<http://v.t.sina.com.cn/>;

1. 页面之间跳转：

点击开始按钮图片，跳转到同一级目录下的index.html文件

<a href="playPcServlet.action"><img style="width:100%;height:100%;" src="images/start.jpg"></a>

playPcServlet.action = index.html

1. 分享到微博的跳转页面

var title="赢了建宏老师签名球不闲着，这个#周二现货日#在@乐视商城 玩颠球，球技杠杠的！微博分享成绩还有机会赢新品神兽级蓝牙耳机！我得加把劲啦~颠啊颠，颠到18号！#我爱超级电视#，快来跟我PK啊小伙伴们！";//分享的内容

$.post("updateWebServlet.action");

location.href="http://v.t.sina.com.cn/share/share.php?url=http://219.234.5.128/letv/games/football/Horizontal/pcServlet.action&title="+encodeURIComponent(title)+"&appkey=2038864787";//跳转链接

//红色为分享的链接。

1. 页面上，添加弹窗信息：alert("开始游戏！");
2. 开关按钮的实现

* 单击之后，修改菜单按钮图片

this.click.setNormalImage(new cc.Sprite(res.a\_bg));

* 文字开关菜单

var item = new cc.MenuItemToggle(

new cc.MenuItemFont("Off"),

new cc.MenuItemFont("On"),

this.onMenuCallback, this//回调函数

);

// 默认被选中菜单项

item.setSelectedIndex(2);

var menu = new cc.Menu(item);

this.addChild(menu);

* 图片开关菜单

this.click=new cc.MenuItemToggle(

new cc.MenuItemImage(res.a\_btnOff),

new cc.MenuItemImage(res.a\_btnOn),

this.onClick,this);//回调函数

this.click.attr({

anchorX:0.5,

anchorY:0.5,

x:cc.winSize.width-30,

y:cc.winSize.height/2-200,

scale:cc.winSize.width/640

});

this.click.setSelectedIndex(0);

var menu=new cc.Menu(this.click);

menu.x = 0;

menu.y = 0;

this.addChild(menu,10);

* 设置菜单不可用：this.click.setEnabled(false);

1. 攻击值文本1秒之后，自动消失

* 添加一个全局变量：fladetime:100
* 在每帧更新的函数update中，更新该变量：

//更新文本显示

if(this.fladetime==0){

this.timeLabel.string="";//时间到时，清空文本

}else if(this.fladetime>0){

this.fladetime--;

}

* 在判断是有效攻击时，重新设置全局变量的值：fladetime=100;

1. 查看指定服务器上的文件

* 打开winscp.exe
* 输入指定的Host name\user name\port\password;
* 找到指定的文件目录即可
* 使用域名或者IP地址访问时，下一级目录是从wepapps的子目录开始的。

1. 停止、启用指定的服务器

* 打开SSHclient.exe，配置好指定服务器的信息。

//配置服务器环境

1、打开sshClient.exe-->Profiles-->Add Profiles-->输入要显示的名字(随便)-->Add to Profiles;

2、Profiles-->选择刚才输入的名字-->Host填写IP User填写用户名 Port改为服务器的进程号;

3、选择刚才输入的名字-->单击-->输入密码；

* 停止应用：

ps -ef|grep java

kill -9 进程号

其中进程号是在输入上一行代码之后，显示的。举例：

//输入ps -ef|grep java之后，显示进程号3164

【root 3164 3150 0 09:13 pts/0 00:00:00 grep java】

* 启动应用：

cd /usr/java/tomcat/bin

sh startup.sh

1. 怎么样在在页面上添加游戏背景图？避免黑色或者白色

在index.html文件中，<bord>标签中，添加background属性，

方法一：

<body background="res/bg.jpg"></body>

方法二：

<body style="background-image:url(res/bg.jpg);background-size:100% 100%;background-repeat:no-repeat;">

两种方法的不同是：

方法一背景拉伸，方法二背景平铺。最好使用第二种方法。

1. 《分享到朋友圈》图片的实现功能：【点击《分享到朋友圈》按钮🡪显示图片，在图片上单击之后，图片隐藏】

//guide

var guide= new cc.Sprite(res.guide\_png);

guide.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.addChild(guide,20,10);

if(cc.sys.isMobile){

var clientSize = cc.view.getDesignResolutionSize();

var bgSize = guide.getContentSize();

/\*

console.log(clientSize.width+" "+clientSize.height);

console.log(bgSize.width+" "+bgSize.height);

\*/

guide.scale=cc.p(clientSize.width/bgSize.width,clientSize.height/bgSize.height);

}

var self=this;

//listener

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:function(touch,event){

var target= event.getCurrentTarget();

var size= target.getContentSize();

var rect= cc.rect(0,0,size.width,size.height);

var locationPos= target.convertToNodeSpace(touch.getLocation());

return cc.rectContainsPoint(rect,locationPos);

},

onTouchEnded:function(touch,event){

var target= event.getCurrentTarget();

//移除

target.removeFromParent();

self.scheduleUpdate();

}

});

cc.eventManager.addListener(listener,guide);

在onTouchEnded函数中移除分享引导图片，

1. 给某一层添加监听机制

cc.eventManager.addListener({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,//单触点模式

swallowTouches:true,

onTouchBegan:this.onTouchBegan

},this)

//游戏结束时，移除所有的监听机制

cc.eventManager.removeAllListeners();

1. 定义一个point对象，并初始化：

方法一：var point= new cc.Point(200,200);

方法二：var point=cc.p(300,300);

1. 定义一个RECT对象，并初始化：

方法一：var rect= new cc.Rect(0,0,200,200);

方法二：var rect=cc.rect(0,0,300,300);

参数说明：

参数一和参数二，分别是区域的左上角的x和y坐标;

参数三和参数四，分别是区域的宽和高。

1. 创建一个精灵时，只显示一张图片的指定区域：

new cc.Sprite(res.dog\_png,cc.rect(0,0,30,30));

1. 获取触摸点的位置：touch.getLocation()

var target = event.getCurrentTarget();

var locationInNode = target.convertToNodeSpace(touch.getLocation());

var s = target.getContentSize();

var rect = cc.rect(0, 0, s.width, s.height);

其中，convertToNodeSpace的作用：

//获得touch.getLocation()在target坐标系中的坐标

//convertToNodeSpace要传入的坐标为世界坐标

1. 检测点是否在某个rect中

var point=touch.getLocation();

var rect =cc.rect(0,0,100,100);

if(cc.rectContainsPoint(rect,point)){

alert("明白");

}

1. 锚点

锚点就是所有扭转，移动，缩放的参考点。cocos2-x中默认的锚点是中间点。锚点用比例来默示局限为0-1，（0，0）点代表左下点，（1，1）代表右上点。设置的函数为setAnchorPoint（ccp（0.5， 0.5））;

1. 将网上的《水果特工队》，在自己的机器上运行：

* 解压fruitAttck项目；
* 将游戏的引擎frameWorks拷贝到fruitAttck项目文件夹下；
* 在project.json文件中，配置引擎的位置：

"engineDir":"frameworks/cocos2d-html5"

* 在index.html文件中，更改CCBoot.js文件的引用路径：

<script src="frameworks/cocos2d-html5/CCBoot.js"></script>

1. 在html中引用JS：

<script src="main.js"></script>

1. lite版本中，将main.js文件删除，一直到index.html文件中：

在<body></body>标签之间添加：

<script type="text/javascript">

window.onload = function(){

cc.game.onStart = function(){

//load resources

cc.LoaderScene.preload(["HelloWorld.png"], function () {

var MyScene = cc.Scene.extend({

onEnter:function () {

this.\_super();

var size = cc.director.getWinSize();

var sprite = cc.Sprite.create("HelloWorld.png");

sprite.setPosition(size.width / 2, size.height / 2);

sprite.setScale(0.8);

this.addChild(sprite, 0);

}

});

cc.director.runScene(new MyScene());

}, this);

};

cc.game.run("gameCanvas");

};

</script>

1. html中标题名改为中文，需要添加：

在<head></head>标签之间，添加：<meta charset="utf-8">

1. 根据不同的屏幕大小，游戏中加载不同的图片

//设置一个默认的屏幕大小，判断是否为高清屏幕，用来加载不同的图片

var designSize = cc.size(480,800);

//获取浏览器窗口大小

var screenSize = cc.view.getFrameSize();

//设置资源加载路径信息

if(!cc.sys.isNative&&screenSize.height<800){

//如果不是原生设备，并且浏览器窗口高度小于设定值，则使用标准大小资源图片

designSize = cc.size(320,480);

cc.loader.resPath = "res/Normal";

}else{

cc.loader.resPath = "res/HD";

}

//根据指定的大小，设置游戏视图，并且满屏显示

cc.view.setDesignResolutionSize(designSize.width,designSize.height,cc.ResolutionPolicy.SHOW\_ALL);

例如：在资源文件中，设置资源路径为：btn\_png:"res/btn.png",那么为了实现上述效果，必须在项目的根目录下，添加res/HD/res/btn.png或者res/Normal/res/btn.png。

1. 在屏幕上显示或隐藏FPS

Cocos左下角的三行数值意义：

1.最上面一行是指的当前场景的渲染批次。（简单理解为需要渲染多少个贴图出来）

2.中间一行是渲染每一帧需要的时间。

3.最下行就是大家熟悉的FPS。

FPS是图像领域中的一个术语。

FPS（Frames Per Second）：每秒传输帧数。Frames Per Second更确切的解释是“每秒中填充图像的帧数（帧/秒）”。FPS是测量用于保存、显示动态视频的信息数量。通俗来讲就是指动画或视频的画面数。例如在电影视频及数字视频上，每一帧都是静止的图象；快速连续地显示帧便形成了运动的假象。每秒钟帧数 (FPS) 愈多，所显示的动作就会愈流畅。通常，要避免动作不流畅的最低FPS是30。某些计算机视频格式，例如 AVI，每秒只能提供15帧。

这里的“FPS”也可以理解为我们常说的“刷新率（单位为Hz）”，例如我们常在CS游戏里说的“FPS值”。我们在装机选购显卡和显示器的时候，都会注意到“刷新率”。一般我们设置缺省刷新率都在75Hz（即75帧/秒）以上（XP锁定在60Hz）。例如：75Hz的刷新率刷也就是指屏幕一秒内只扫描75次，即75帧/秒。而当刷新率太低时我们肉眼都能感觉到屏幕的闪烁，不连贯，对图像显示效果和视觉感观产生不好的影响。

fps表示帧数

30FPS是一般录像的常用帧数，60是一般游戏的常用帧数。

30FPS在快速动作的时候会感觉不流畅。但是60FPS对显卡的要求要高一些.如果电脑硬件不达标的话，看60FPS反而会卡...

方法一：在project.json中，添加设置showFPS参数的值。

false:隐藏 true：显示

方法二：或者使用API函数：cc.director.setDisplayStats(true);

1. 获取精灵的宽和高

spriteBg.getContentSize().height

spriteBg.getContentSize().width

1. JS中，数组的使用

* 在类中定义一个数组对象：removeObj:[]
* 向数组中，添加一个元素

var sprite=new cc.Sprite(res.bg\_png);

this.removeObj.push(sprite);

* 删除数组中的元素：

if(this.removeObj.length != 0){//再来一次，将所有列表清空

this.removeObj.splice(0,this.removeObj.length);

}

//参数一：数组索引，从该下标开始，删除数组元素

//参数二：要删除元素的个数

* 获取数组的长度：this.removeObj.length
* 遍历数组元素：

for(var i=0;i<this.removeObj.length;++i){

this.removeObj[i]

}

1. HTML文档中，实现滑动效果：

注意：在手机上的滑动效果，其实就是在电脑上的滚动条效果，只不过在手机上将滚动条隐藏，实现滑动效果。最主要的就是overflow:auto这个属性，实现滑动效果。

<div id="paihang" style="overflow:auto; position:absolute;">

<ul>

<li class="li\_first">

<ul>

<li class="li1">

1

</li>

<li class="li2">

139\*\*\*\*5689

</li>

<li class="li3">

102

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

2

</li>

<li class="li2">

185\*\*\*\*5623

</li>

<li class="li3">

95

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

3

</li>

<li class="li2">

139\*\*\*\*5689

</li>

<li class="li3">

85

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

4

</li>

<li class="li2">

185\*\*\*\*5623

</li>

<li class="li3">

76

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

5

</li>

<li class="li2">

139\*\*\*\*5689

</li>

<li class="li3">

66

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

6

</li>

<li class="li2">

185\*\*\*\*5623

</li>

<li class="li3">

54

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

7

</li>

<li class="li2">

139\*\*\*\*5689

</li>

<li class="li3">

43

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

8

</li>

<li class="li2">

185\*\*\*\*5623

</li>

<li class="li3">

32

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

9

</li>

<li class="li2">

139\*\*\*\*5689

</li>

<li class="li3">

21

</li>

</ul>

</li>

<li class="li\_first">

<ul>

<li class="li1">

10

</li>

<li class="li2">

185\*\*\*\*5623

</li>

<li class="li3">

12

</li>

</ul>

</li>

</ul>

</div>

1. 设置节点的位置：

label.setPosition(cc.pAdd(cc.visibleRect.center, cc.p(0, 0)));

以cc.visibleRect.center为原点，偏移量为cc.p(0, 0)，设置节点的位置。

其中，cc.visibleRect.center为可视区域

1. 动态改变适配模式

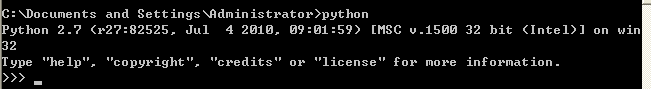
cc.view.setDesignResolutionSize(320,480,cc.ResolutionPolicy.SHOW\_ALL);

1. cocos开发环境配置
   1. 安装python，并将其安装目录添加到系统环境path变量中：

例如:python的安装目录是：C:\Program Files\Python2.7

我的电脑-->右击-->属性-->高级-->环境变量-->对话框中的靠下面《系统变量》-->找到path一行，编辑该值，添加“;C:\Program Files\Python2.7”

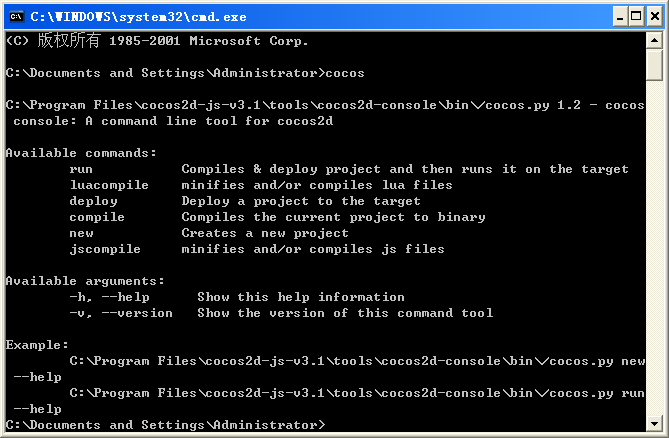
* 1. 测试是否设置成功：win+R-->cmd-->python,若显示,成功安装：



* 1. 将Cocos引擎解压，例如解压路径是：C:\Program Files\cocos2d-js-v3.1
  2. 在第一步的path的变量中，添加:

”;C:\Program Files\cocos2d-js-v3.1\tools\cocos2d-console\bin”

* 1. 测试是否设置成功：win+R-->cmd-->cocos -h,若显示，成功安装：



* 1. ant的配置。将ant解压，例如路径是：C:\apache-ant-1.9.2。在电脑的系统环境变量中，添加一个变量：ANT\_HOME，其值等于C:\apache-ant-1.9.2
  2. 添加新的环境变量：ANT\_HOME，其值为：C: \apache-ant-1.9.2
  3. 添加新的环境变量：ANT\_ROOT，其值为：C:\ apache-ant-1.9.2\bin
  4. 添加path变量的值：%ANT\_HOME%/bin;
  5. 至此，ANT环境已经配置好了，但是需要配置jdk的环境变量。下载JDK并安装，安装目录为：C:\Program Files\Java\jdk1.7.0\_67
  6. 添加新的变量：JAVA\_HOME，其值为：C:\Program Files\Java\jdk1.7.0\_67

1. 生成release版本

* 将引擎拷贝到项目中；
* 修改project.json文件中引擎的位置："engineDir":"frameworks/cocos2d-html5"

//切换路径

cd /d D:\MyProject

//生成一个最简新项目Hello,路径:D:\MyProject\Hello

cocos new Hello -l js --no-native

//命令行切换到项目文件夹D:\MyProject\Hello下，编译、运行该项目

cd /d D:\MyProject\Hello

cocos compile -p web

cocos run -p web

//生成release版本

cocos compile -p web -m release

//切换到release版本下,并运行

cd /d D:\MyProject\Hello\publish\html5

cocos run -p web

1. js混淆、压缩

* 压缩
* 下载googleclosurecompiler压缩工具，路径：D:\MyProject\closure-compiler；
* 修改项目的build.xml文件【如果没有，则新建一个build.xml文件，将下面的代码全部拷贝。】：

//原始build.xml文件

<?xml version="1.0"?>

<project name="Javascript compress project" basedir="." default="compile">

<!-- The classpath should be modified to the real closure compiler jar file path -->

<taskdef name="jscomp" classname="com.google.javascript.jscomp.ant.CompileTask" classpath="F:/myData/closure-compiler-master/compiler.jar"/>

<target name="compile">

<jscomp compilationLevel="simple" warning="quiet"

debug="false" output="./game.min.js">

<!-- Uncomment the line below to enable sourcemap generation -->

<!--sourceMapOutputFile="./cocos2d-js-sourcemap" sourceMapFormat="V3"> -->

<sources dir="./">

<!-- You may need to modify the file name to the actual downloaded file name -->

<file name="src/cocos2d-js-v3.1-lite.js"/>

<file name="src/MyLoaderScene.js"/>

<file name="src/resource.js"/>

<file name="src/app.js"/>

<file name="src/game.js"/>

<file name="src/globals.js"/>

<file name="src/AnimationLayer.js"/>

<file name="src/BackgroundLayer.js"/>

<file name="src/GameOverLayer.js"/>

<file name="src/StatusLayer.js"/>

<!-- Put your own js files here, dependency order is also important -->

</sources>

</jscomp>

</target>

</project>

//参数说明：

* <target name="compile">定义了任务目标名称，与前面的default的值相同;
* <jscomp compilationLevel="simple" 用于定义编译模式,有三种模式可以指定:
  + whitespace(去除空白行和回车,并把文件合并成一个文件);
  + simple(进行简单编译,在whitespace基础上加入了清除无用代码,改变局部变量名等.)
  + advanced(对js文件进行运行时优化,会改变函数名称,相当于压缩和混淆了.)

注意：cocos使用advanced模式时，会报错，好像不支持混淆操作。

* output="./game.min.js"用于指定打包后的文件放到哪里,取什么文件名
* <sources dir="${basedir}">用于定义js文件目录,可以指定当前目录或其他目录。
* <file name="src/cocos2d-js-v3.1-lite.js"/>用于指定要打包的文件,可以在文件中加入子目录名称等。剩下的事就是将需要打包的文件都加入到file列表中了。
* 修改classpathcompiler.jar文件的路径信息变量：classpath【在google closurecompiler文件夹中】
* 在build.xml文件中，修改要压缩的文件cocos2d-js-v3.1-lite.js路径信息：

<file name="cocos2d-js-v3.1-lite.js"/>

* 切换到对应的项目路径下，打开命令窗口输入ant
* 反压缩
* 利用chrome浏览器，打开对应的页面，F12🡪进入开发者模式；
* 打开对应的压缩js文件；
* 点击下面的{}图标，就可以查看还原之后的代码。

1. 精灵根据触摸点移动而移动：

//方法一：跟随动作

//移动精灵

this.sprite=new cc.Sprite(res.dog\_png);

this.sprite.attr({

x:85,

y:80

});

this.addChild(this.sprite);

//给Layer创建监听器

cc.eventManager.addListener({

event: cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches: true,

onTouchBegan: this.onTouchBegan,

onTouchMoved: this.onTouchMoved

}, this);

this.moveAction =new cc.MoveBy(0.01,cc.p(0,2));

onTouchBegan:function(touch,event){

//alert("触摸开始");

//event.getCurrentTarget().setBeganX(touch.getLocationX());

return true;

},

onTouchMoved:function(touch,event){

//var point = touch.getDelta();

event.getCurrentTarget().move(touch.getLocationX(),touch.getLocationY());

},

move:function(x,y){

this.sprite.setPosition(cc.p(x,y));

//this.sprite.runAction(this.moveAction);

},

CCPoint getDelta() const; //获取前后两次位置的偏移量，基于OpenGL坐标

//方法二：根据在船舵上的上下滑动位移，来控制船的位移。

* 定义一个变量：this.preY=0;
* 在onTouchBegan函数中，设置this.preY的变量值：

onTouchBegan:function(touch,event){

//alert("触摸开始");

event.getCurrentTarget().getParent().setBeganY(touch.getLocationY());

return true;

}，

setBeganY:function(y){

//alert("成功！");

this.preY=y;

},

* 在onTouchMoved函数中，移动精灵：

onTouchMoved: function (touch, event) {

var target = event.getCurrentTarget();

var locationInNode = target.convertToNodeSpace(touch.getLocation());

var s = target.getContentSize();

var rect = cc.rect(0, 0, s.width, s.height);

if(cc.rectContainsPoint(rect, locationInNode)){

event.getCurrentTarget().getParent().move(touch.getLocationY());

}

return false;

}

move:function(y){

if(y>this.preY){

cc.log("向上移动！");

this.sprite.setPositionY(this.sprite.getPositionY()+(y-this.preY));

}else{

cc.log("向下移动！");

this.sprite.setPositionY(this.sprite.getPositionY()-(this.preY-y));

}

this.preY=y;

//this.sprite.setPositionY(y);

},

注意：以上监听机制是添加给指定图片的，所用会使用getParent()

1. 模拟抛物线运动：

//模拟抛物线运动

this.football = new cc.Sprite(res.football\_png);

this.football.attr({

anchorX:0.5,

anchorY:0.5,

x:cc.winSize.width/2,

y:cc.winSize.height/2-100,

scale:0.5

});

this.addChild(this.football);

//this.scheduleUpdate();

this.schedule(this.onSchedule,0.05);//开启定时器 自我模拟

this.time=0;

onSchedule:function(){

this.time+=1;

var vt=v0-(this.time\*g);

var s=v0\*this.time-(this.time\*this.time\*0.5\*g);

cc.log("位移："+s);

cc.log("速度："+vt);

this.football.setPositionY(cc.winSize.height/2-100+s);

if(s<2&&s>0){

//回到水平线

this.time=0;

}

if(vt<2&&vt>0){

//已经达到最高点

this.unschedule(this.onSchedule);

}

},

1. 自定义字体

// font definition【自定义字体名fontDefBlueStroke】

var fontDefBlueStroke = new cc.FontDefinition();

fontDefBlueStroke.fontName = "Arial";//系统

fontDefBlueStroke.fontSize = 20;//字体大小

fontDefBlueStroke.textAlign = cc.TEXT\_ALIGNMENT\_CENTER;//居中显示

fontDefBlueStroke.verticalAlign = cc.VERTICAL\_TEXT\_ALIGNMENT\_TOP;

fontDefBlueStroke.fillStyle = redColor;//主体颜色

// stroke描边

fontDefBlueStroke.strokeEnabled = true;//显示描边

fontDefBlueStroke.strokeStyle = yellowColor;//描边颜色

//shadow 阴影

fontDefBlueStroke.shadowEnabled = true;

//阴影的位置，偏移量以字体的左下角为原点

fontDefBlueStroke.shadowOffsetX = 12;//左

fontDefBlueStroke.shadowOffsetY = -12;//下

//create font

//显示得分系统

this.score=cc.LabelTTF.create("Score:0",fontDefBlueStroke);

this.score.setPosition(cc.p(140,140));

this.addChild(this.score);

1. 精灵的时隐时现效果

var action = new cc.Sequence(cc.fadeOut(0.1),cc.fadeIn(0.1));

//精灵执行该动作即可，例如：

this.score.runAction(action.repeatForever());

1. 层的切换效果：

在层的初始化函数中，添加：

//层由最中间位置开始逐渐放大

this.attr({

anchorX:0.5,

anchorY:0.5,

x:0,

y:0,

scale:0

});

var scale = cc.scaleTo(1,1);

this.runAction(scale);

//同理，层由上到下切换

1. 改变页面的背景色：

在<body style="padding:0; margin: 0;background: #000;"></body>中添加：

bgcolor="black"

1. 从plist和png图片中分离小图片

* 安装python,并配置python的Path系统变量；
* 自己新建一个unpack\_plist.py脚本，内容如下：

#!python

import os,sys

from xml.etree import ElementTree

from PIL import Image

def tree\_to\_dict(tree):

d = {}

for index, item in enumerate(tree):

if item.tag == 'key':

if tree[index+1].tag == 'string':

d[item.text] = tree[index + 1].text

elif tree[index + 1].tag == 'true':

d[item.text] = True

elif tree[index + 1].tag == 'false':

d[item.text] = False

elif tree[index+1].tag == 'dict':

d[item.text] = tree\_to\_dict(tree[index+1])

return d

def gen\_png\_from\_plist(plist\_filename, png\_filename):

file\_path = plist\_filename.replace('.plist', '')

big\_image = Image.open(png\_filename)

root = ElementTree.fromstring(open(plist\_filename, 'r').read())

plist\_dict = tree\_to\_dict(root[0])

to\_list = lambda x: x.replace('{','').replace('}','').split(',')

for k,v in plist\_dict['frames'].items():

rectlist = to\_list(v['frame'])

width = int( rectlist[3] if v['rotated'] else rectlist[2] )

height = int( rectlist[2] if v['rotated'] else rectlist[3] )

box=(

int(rectlist[0]),

int(rectlist[1]),

int(rectlist[0]) + width,

int(rectlist[1]) + height,

)

sizelist = [ int(x) for x in to\_list(v['sourceSize'])]

rect\_on\_big = big\_image.crop(box)

if v['rotated']:

rect\_on\_big = rect\_on\_big.rotate(90)

result\_image = Image.new('RGBA', sizelist, (0,0,0,0))

if v['rotated']:

result\_box=(

( sizelist[0] - height )/2,

( sizelist[1] - width )/2,

( sizelist[0] + height )/2,

( sizelist[1] + width )/2

)

else:

result\_box=(

( sizelist[0] - width )/2,

( sizelist[1] - height )/2,

( sizelist[0] + width )/2,

( sizelist[1] + height )/2

)

result\_image.paste(rect\_on\_big, result\_box, mask=0)

if not os.path.isdir(file\_path):

os.mkdir(file\_path)

outfile = (file\_path+'/' + k).replace('gift\_', '')

print outfile, "generated"

result\_image.save(outfile)

if \_\_name\_\_ == '\_\_main\_\_':

filename = sys.argv[1]

plist\_filename = filename + '.plist'

png\_filename = filename + '.png'

if (os.path.exists(plist\_filename) and os.path.exists(png\_filename)):

gen\_png\_from\_plist( plist\_filename, png\_filename )

else:

print "确保\*.plist和\*.png在当前文件夹下！"

* 将三个文件拷贝到一个文件夹下，例如: c:\picture

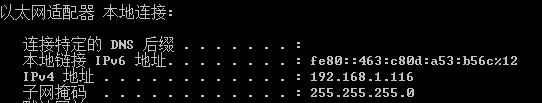
【unpack\_plist.py、\*.plist、\*.png三个文件】

* 打开Dos命令窗口：Win+R-->cmd;
* 切换到指定的文件路径下：cd /d c:\picture;
* 执行分解命令：python unpack\_plist.py explosion

其中 explosion是\*.plist的主文件名。本例中，\*.plist文件为explosion.plist

1. 获取本机的IP地址：

方法一：win+R🡪cmd🡪ipconfig🡪显示的IPv4地址就是你的本机IP



方法二：打开网络和共享中心🡪本地连接🡪详细属性🡪IPv4

1. 获取本机在公网上的IP地址：

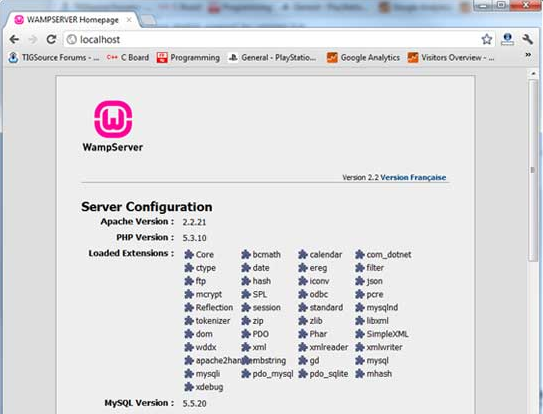
在百度中输入IP，回车，在打开的网页的上部，就是显示。



1. 查看能否及连接到指定的IP地址：ping 192.168.1.116
2. 安装Web服务器

如果你想充分利用这些cocos 示例，你需要一个已安装的web 服务器。这可以让你在不同的机器上都能访问你自己的东西。WAMP服务器是一个Apache，MySql，PHP和其他的一些程序的整合安装包，虽然它有点庞大，但它确实非常简单。

1. 下载WAMP安装包，并安装。安装路径为：“C:\wamp\”
2. 在安装过程中，会选择一个默认的浏览器。
3. 安装完成之后，在系统托盘处会有wamp的图标，点击🡪LocalHost🡪在浏览器中会显示下图，则说明安装成功：



1. 将JS项目中所有的文件，都拷贝到c:\wawp\www\目录下，在你的浏览器中输入：localhost/index.html，就可以看到你的index.html文件了。

//制定的IP地址

#Listen 12.34.56.78:80

Listen 192.168.1.116:80

//DNS

ServerName 192.168.1.116

1. 动作执行完毕的回调函数：

var seq = new cc.Sequence(animateAction,cc.callFunc(this.Ok,this));

//回调函数Ok()

Ok:function(){

this.player.visible =false;

}

说明：

创建动作序列对象【new cc.Sequence()和new cc.Spawn()】的时候，都可以加回调函数。但是和本身的意义一样，cc.Spawn同步执行，cc.Sequence异步执行。

1. 爆炸效果的实现：【难点在于：执行完动画之后，精灵消失】

//创建爆炸效果精灵

create:function(){

//点击开始

var player = new cc.Sprite();

player.attr({

anchorX:0.5,

anchorY:0.5,

x:cc.winSize.width/2,

y:cc.winSize.height/2-100

});

this.addChild(player);

//加载explosion.plist资源

cc.spriteFrameCache.addSpriteFrames(res.explosion\_plist);

//create array 创建数组，存放所有的动画帧

var animationFrames=[];

for(var i=1;i<35;++i){

var str="explosion\_"+((i<10)?("0" + i):i)+".png";

var frame=cc.spriteFrameCache.getSpriteFrame(str);

animationFrames.push(frame);

}

//create animation

var animation=new cc.Animation(animationFrames,0.02);

//create animate action

var animateAction=cc.animate(animation);

//var seq = new cc.Sequence(animateAction,cc.callFunc(this.Ok,player,true));

var seq = new cc.Sequence(animateAction,cc.callFunc(this.Ok,this));

player.runAction(seq);

},

//点击一下 爆炸效果 然后消失

Ok:function(nodeExecutingAction, data){

//方法一：删除数据

nodeExecutingAction.removeFromParent(data);

//方法二：将该精灵不显示

}

1. 精灵执行完对应的动作后，删除自身

//player1【旋转90-->隐藏fadeIn】

var player1 = new cc.Sprite("res/grossini.png");

player1.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.addChild(player1);

var rotateTo = cc.rotateTo(1,90);

var fadeOut = cc.fadeOut(1);

var seq = cc.sequence(rotateTo,fadeOut,cc.callFunc(this.removeSelf,this));

player1.runAction(seq);

removeSelf:function(pSender){

//方法一：删除子节点

this.removeChild(pSender);

//方法二：

this.parent.removeChild(this,true);

}

注意：上述删除自身的两种方法，二者只能取其一。

1. 文字

Cocos 2D-X引擎有三个用于处理文字的类：CCLabelAtlas、CCLabelBNFont和CCLabelTTF

* TTF：类型标签（CCLableTTF）。TTF类型字体文件的扩展名是.ttf,类型代码是tfil。优点：
  + - 任何一种TTF字体都包含了某种语言当中所有的字幕以及符号，开发者可以随意调整字体的大小、颜色以及样式;
    - TTF在很多系统都得到了普及;

缺点：

* + - 创建和更新的过程将会比较缓慢，这是由于字体包含的内容较多，并且初始化时需要创建纹理图片；
* 使用系统默认字体

this.score=cc.LabelTTF.create("Score:0","Helvetica",20);

this.score.setPosition(cc.p(40,40));

this.score.setColor(cc.color(255,255,0));

this.addChild(this.score);

* 加载自定义字体【几乎没作用】

var labelTTF = cc.LabelTTF.create("我们","res/迷你简双体.ttf",20);

labelTTF.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2,

color:cc.color(255,0,0),

strokeStyle: cc.color(0,0,0),//描边颜色

lineWidth: 2//描边宽度

});

this.addChild(labelTTF);

* 创建自定义字体

var fontDefine = new cc.FontDefinition();

//系统字体名

fontDefine.fontName = "Impact";

//字体主体颜色

fontDefine.fillStyle = cc.color(255,255,0);

//字体尺寸

fontDefine.fontSize = 30;

//水平方向上的对齐格式

fontDefine.textAlign = cc.TEXT\_ALIGNMENT\_CENTER;//居中

//竖直方向上的对齐格式

fontDefine.verticalAlign = cc.VERTICAL\_TEXT\_ALIGNMENT\_TOP;//

//stroke

//是否显示描边【默认不显示】

fontDefine.strokeEnabled = true;

//描边颜色

fontDefine.strokeStyle = cc.color(255,0,0);

//shadow阴影

fontDefine.shadowEnabled = true;

//偏移量【以字体的左下角为准】

fontDefine.shadowOffsetX = 12;

fontDefine.shadowOffsetY = -12;

//以新字体fontDefine，创建标签

var labelTTF = cc.LabelTTF.create("我们",fontDefine);

labelTTF.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.addChild(labelTTF);

除了使用平台系统中的提供的字体之外，开发者还可以直接引入TTF字体。只要是符合标准的TTF文件，就可以在引擎中使用。开发者只需向使用其他资源文件一样，将TTF文件放置在资源路径下。然后就是调用创建函数，通过字体名称就能获得引入的字体类型。

建议开发者一旦创建了TTF标签对象就不要再改变文字。因为每一次重建文字标签的内容，都需要建立一张纹理图片，这将会非常耗时，同时还有可能导致内存空间的浪费。

* BMFont标签类（CCLabelBMFont）

BMFont标签类是引擎当中最快速最自由的字体类。不过也是使用起来最麻烦的字体。

优点：

1. 具备了很快的创建以及更新速度。
2. 自由度非常高i，每一个字幕或者符号都是单独的精灵。
3. 自制的字体方式，开发者可以自定义的字母以及符号，甚至可以包含阴影、外框以及花纹。

使用BMFont编辑器之后，开发者将会得到两个文件：一个是字体纹理图集文件，另一个是以fnt为后缀的配置文件。

* 将以上两个文件，添加到资源路径下
* 创建

var labelBMF = new cc.LabelBMFont("qinghu", "res/bitmapFontTest3.fnt");

labelBMF.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.addChild(labelBMF,60);

BMFont类对象中每一个字幕或者符号都是一个独立的精灵对象，所以开发者可以轻易地就让一个字母跳动、旋转、变形、变色以及改变透明度。

获取字符串中的某个字符，可使用label.getChildByTag(索引)。【索引从0-(长度-1)】

* 实例：第一个和第六个字符 做不同的旋转动作

var label1= new cc.LabelBMFont("different Character",res.bitmapFontTest\_fnt);

label1.attr({

x:300,

y:cc.winSize.height/2-100

});

var label2= new cc.LabelBMFont("",res.bitmapFontTest\_fnt);

label2.attr({

x:300,

y:200

});

this.addChild(label1,1,1);

this.addChild(label2,1,2);

//旋转

var rotate1= cc.rotateBy(0.01,-1,0).repeatForever();

var rotate2= cc.rotateBy(0.01,1,0).repeatForever();

//first char

label1.getChildByTag(0).runAction(rotate1);

label1.getChildByTag(5).runAction(rotate2);

* Atlas标签类（CCLabelAtlas）

与之前的CCLabelTTF类相比，它具有更大的灵活性，同时，还占用了更少的资源。Atlas标签类的创建速度是远远超过TTF标签类的。这两种个字体的技术原理十分累世。只不过TTF标签会创建一张绘制着文字的图片，而Atlas则不会创建任何的纹理图片。Aalas标签类只会使用源纹理图片进行绘制。

* 一张png图片资源，放到资源路径下
* 创建

//create Atlas

var labelAtlas = new cc.LabelAtlas("01234", "res/fps\_images.plist", 16, 24, '.');

labelAtlas.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.addChild(labelAtlas);

1. 菜单按钮

* CCMenuItem

建议不要直接使用该类，因为它并不包含具体显示的功能。

* MenuItemLabel

此类的对象会将一个基本的文字标签转变成为一个菜单按钮。当玩家选中时，还会随文字放大的效果。

var labelAtlas = new cc.LabelAtlas("0123456789", s\_fpsImages, 16, 24, '.');

var item3 = new cc.MenuItemLabel(labelAtlas, this.onMenuCallbackDisabled, this );

item3.setDisabledColor( cc.color(32,32,64) );

item3.color = cc.color(200,200,255);

* MenuItemAtlasFont

类MenuItemAtlasFont是从MenuItemLabel类继承而来的，并没有太多的变化，只是支持使用LabelAtlas来创建按钮对象，同样具备了在玩家选中按钮时的文字放大效果。

* MenuItemFont

类MenuItemAtlasFont是从MenuItemLabel类继承而来的。

var item4 = new cc.MenuItemFont("I toggle enable items", function(sender) {

this.\_disabledItem.enabled = !this.\_disabledItem.enabled;

}, this);

item4.fontSize = 20;

item4.fontName = "Arial";

* MenuItemSprite

一个由精灵对象组成的菜单按钮。

//正常状态

var spriteNormal = new cc.Sprite(s\_menuItem, cc.rect(0,23\*2,115,23));

//选中状态

var spriteSelected = new cc.Sprite(s\_menuItem, cc.rect(0,23,115,23));

//无效状态

var spriteDisabled = new cc.Sprite(s\_menuItem, cc.rect(0,0,115,23));

var item1 = new cc.MenuItemSprite(spriteNormal, spriteSelected, spriteDisabled, this.onMenuCallback, this);

//this.onMenuCallback回调函数

* MenuItemImage

类MenuItemImage从类MenuItemSprite继承而来的，将原本按钮中的精灵对象换成了三张纹理图片。最好使用尺寸相同的图片，这样在玩家按下按钮时，产生的动画效果不会发生偏移。

var sendScoreSF = new cc.SpriteFrame(s\_sendScore, cc.rect(0, 0, 145, 26));

cc.spriteFrameCache.addSpriteFrame(sendScoreSF, "send\_score\_sf");

var item2 = new cc.MenuItemImage("#send\_score\_sf", s\_pressSendScore, this.onMenuCallback2, this);

* MenuItemToggle

类MenuItemToggle在内部拥有一个MenuItem数组，用来负责展示不同的按钮状态。因为使用了一个菜单按钮的数组，所以此类的对象可以实现状态切换。

var item4= new cc.MenuItemToggle(

new cc.MenuItemFont("on"),

new cc.MenuItemFont("off"),

this.onItem4,this

);

注意：开关菜单点击到最后的时候，会从头开始。因此，只需要在开关菜单的回调函数中，根据getSelectedIndex()索引值的不同和switch语句，就可以做不同的功能。

示例：

* 创建一个菜单

addMenu:function(){

//menu

var itemimage= new cc.MenuItemImage(

"#but-start.png",

null,

this.onStart,this

);

itemimage.attr({

x:this.layerGradient.getContentSize().width/2,

y:this.layerGradient.getContentSize().height/2

});

var menu= new cc.Menu(itemimage);

menu.setPosition(cc.p(0,0));

this.layerGradient.addChild(menu,1,10);

}

* 操作菜单中的itemimage对象

var menu = this.layerGradient.getChildByTag(10);

//获取itemimage对象【0——new cc.Menu()中的第一个子类】

var menuItem= menu.children[0];

//颜色还原

menuItem.setColor(cc.color(255,255,255));

//设置是否可用

menuItem.setEnabled(true);

1. Tiled Map

两种常见的用户交互渠道：一种为触摸操作，另一种为加速度计操作。

通常游戏中背景大致分为以下三种：

（1）、平面滚动类型：可将其细分为横版滚轴、纵版滚轴和自由滚轴三个类型。

1. 创建一个精灵类，并在游戏的指定层显示出来：

* 自定义一个类，继承自cc.Sprite

var Cat = cc.Sprite.extend({

ctor:function(){

this.\_super(res.catSprite\_png);//类对象的图片

//类对象的坐标属性

this.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

}

});

* 在指定的层中，创建类对象变量，添加精灵：

self.catSprite = new Cat();

self.addChild(self.catSprite);

问题：怎么样实现：不同的精灵，使用不同的图片？

1. Cocos2D-X音频播放

由于浏览器对音频支持的情况各有不同，现在的AudioEngine有三套机制管理器来播放音频：

1. WebAudio来播放音频，现在Chorme与firefox都支持标准来播放；
2. 普通Audio元素，一些手机浏览器和IE浏览器；
3. 只能播放单音频，意思就是当前只有一个音频在播放，其他在播放的音频都得停下来等。

之所以会出现第三种机制是因为：通过测试，一些手机浏览器(如小米2S的默认浏览器)在播放一个音频时，再播放另一个音频，就会让另一个音频，再也不发音了，或者多播放几次就不发声了。

为此，我们还创建了一个白名单，支持可同时播放多个音频的浏览器，我们都加入CCBoot.js里的multipleAudioWhiteList里。虽然加入这个机制，我们也不能保证音频能在所有浏览器都能正常播放。这与浏览器对音频的支持有关。

1. 粒子系统

当我们希望在游戏中模拟这些大规模运动的物体时，通常有如下两个方法：

1. 使用帧动画来模拟：设计帧动画并把它渲染为图片序列来模拟特效，不但生成的动画体积庞大，也无法调整其运动参数，因此有失灵活性；
2. 粒子效果：我们把每一个对象看做成一个粒子，赋予它们一定的属性（例如外观、位置、速度、加速度和生存时间等），使它们按照一定的规律产生、运动并最终消失。

在粒子效果中，通常存在一个对所有粒子进行统一调度的引擎，称作粒子系统(particle system),它负责粒子的产生，随时间改变粒子的状态，以及最后回收不再需要的粒子。

Cocos2d-X为我们提供的粒子系统由CCParticleSystem类实现。CCParticleSystem实现了对粒子的控制与调度，对粒子的操作包括如下几种：

* 产生粒子：这部分也称作粒子发射器(emitter);
* 更新粒子状态：引擎会随时间更新粒子的位置、速度以及其他状态;
* 回收无效粒子：当粒子的生存周期结束后，就会被系统回收;

创建一个全新的粒子系统，通常较为繁琐，大多数情况下，我们更乐意把粒子系统的参数保存在文件中，而Cocos2d-X就是使用Plist文件来保存这些参数的。

* 使用系统本身的粒子系统：

//粒子效果

//粒子效果

var particle = new cc.ParticleFireworks();

particle.initWithFile("res/FireWorks.plist");

particle.setTexture(cc.textureCache.addImage("res/stars.png"));

this.addChild(particle);

* 自己模拟的粒子效果：【方法一：更新(x,y)坐标】
  + 定义两个变量，记录时间和位移：

this.time = 0;

this.s = 0;

* + 定义一个数组，记录所有的粒子对象。初始化时并将所有数组清空

testObj:[],

this.testObj.splice(0,this.testObj.length);

* + 填充数组

for(var i=0;i<8;i++){

var sprite = cc.Sprite.create("res/stars.png");

sprite.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.testObj.push(sprite);

this.addChild(this.testObj[i],1,0);

}

* + 开启定时器，更新粒子的坐标

this.schedule(this.Update,0.01);

Update:function(){

var x = 0;

var y = 0;

var rad = 0;

this.time++;

this.s = this.time\*2;

if(this.s>100){

this.unschedule(this.Update);

return;

}

for(var i=0;i<8;i++){

rad = 2\*3.14\*(i\*45)/360;

x = Math.cos(rad);

y = Math.sin(rad);

var color = cc.color(Math.ceil(Math.random()\*255),Math.ceil(Math.random()\*255),0);

var point = this.testObj[i].getPosition();

this.testObj[i].setPosition(cc.p(x+point.x,y+point.y));

this.testObj[i].setColor(color);

}

}

* 自己模拟的粒子效果：【方法二：半径更新】

for(var i=0;i<8;i++){

var sprite = cc.Sprite.create("res/stars.png");

sprite.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2,

color:cc.color(Math.ceil(Math.random()\*255),Math.ceil

(Math.random()\*255),0)

});

this.addChild(sprite,2);

var x = 0;

var y = 0;

var rad = 0;

rad = 2\*3.14\*(i\*45)/360;

x = 50\*Math.cos(rad);

y = 50\*Math.sin(rad);

var point = sprite.getPosition();

var action1 = cc.moveTo(0.5,cc.p(point.x+x,point.y+y));

//变大后，隐藏

var fadeOut = cc.fadeOut(0.5);

var seq1 = new cc.Sequence(action1,fadeOut);

sprite.runAction(seq1);

}

//TODO：爆炸后隐藏。在渐隐动作执行后的回调函数中，将精灵子节点删除。

* TODO：自己定义一个粒子类对象。

1. 动画

为了方便的记录文理的显示信息，Cocos2D-X提供了框帧类CCSpriteFrame。一个框帧包含两个属性，纹理与区域。纹理指的是将要被显示的纹理，而区域指的是此纹理将要被显示的部分。一个帧框可以完整地描述精灵显示的内容，因此在动画中，我们使用框帧来表示每一帧的内容。

然而，我们也经常需要非匀速的动画效果，此时每一帧显示的时间就需要特殊设置了。为了描述一帧，除了框帧，显然我们还需要记录帧的持续时间。动画帧类CCAnimationFrame同样包含两个属性，其一是对一个框帧的引用，其二是帧的延时。

我们使用CCAnimation描述一个动画，而精灵显示动画的动作则是一个CCAnimate对象。动画动作CCAnimate是精灵显示动画的动作。动画与动画动作的关系就CD光盘与CD播放机的关系一样——前面记录了动画的内容，而后者是播放动画的工具。

动画功能是在游戏制作中必须的一种表现技术，其原理就是利用人眼的视觉残留而产生动态的效果。

* 精灵帧

新版的Cocos2D-X引擎专门设计了类CCAnimationFrame为动画帧对象。在旧版本的引擎当中，开发者更习惯直接使用类CCSpriteFrame(精灵帧)。类CCAnimationFrame的作用就是作为动画帧。而精灵帧被看做是一个附属于精灵对象的属性，则更为合理一些。

其实在类CCAnimationFrame中都持有了一个类CCSpriteFrame的对象，而且是双射的对应关系。

所谓的精灵帧缓冲类CCSpriteFrameCache,其实就是一个存放精灵帧对象的缓冲帧。在引擎运行中，它将会作为一个单例对象。

* 精灵帧缓冲

所谓的精灵帧缓冲类CCSpriteFrameCache,其实就是一个存放精灵帧对象的缓冲池。在引擎运行中，它将会作为一个单例对象。

说明：所谓单例对象，就是在引擎运行时，内存中只存在此类的唯一一个对象。

精灵帧缓冲，能够被算作是一种处理动画的高级技术，是因为它的目的在于提升动画的性能。而不是为了实现动画功能。这句话的含义就是动画系统中可以没有精灵帧缓冲，没有它只会表现得不好罢了。

精灵帧缓冲的优势在于提升动画帧的加载速度，可以让精灵更快地执行动画效果。在引擎启动之处，就会在内存当中创建一个区域，专门用来存放精灵帧。开发者可以从缓冲区中更快速的获取精灵帧。如果想要更高效地利用缓冲区，开发者最好在缓冲区中放置将那些经常用到的精灵帧。在引擎当中为开发者提供了一个动画类CCAnimation。它就是用在精灵之上的动画对象。它包含一系列的动画帧以及动画帧之间的播放间隔。所以动画类的对象是将动画帧元素组织起来的、准备播放的集合对象。它决定了动画帧的播放顺序以及时间间隔。

虽然我们已经获得了动画类CCAnimation,但是它并不是能够执行的动作。他只是作为一个精灵帧的有序集合。同时，它也存有一些与动画相关的播放属性。不过，Cocos2D-X引擎也提供了一个专门用来执行动画的持续动作类，那就是类CCAnimate。

另外一点，动画动作也是存在反序动作动作的。他的执行效果看上去就像是录像的倒带播放。

SpriteX是一款开源、免费的2D动画编辑器。这可是在国内游戏开发圈中非常知名的一款游戏编辑器。

* 动画编辑器

Cocos2D-X引擎当中并没有配备动画编辑器。但它明确了一种动画文件的格式。开发者则可以通过这个动画文件的格式来编写动画编辑器。这个动画文件是一个XML数据格式的文件。

* + 方法一：【多幅图片资源组成的动画】

//create sprite

var player = cc.Sprite.create(res.dog\_png);

player.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2,

scale:1

});

this.addChild(player,0);

//create Animation

var animation = new cc.Animation();

//fill animation with spriteFrame

var array = [res.dog1\_png,res.dog2\_png];

for(var i=0;i<2;++i){

var spriteFrame = array[i];

animation.addSpriteFrameWithFile(spriteFrame);

}

//settings

animation.setDelayPerUnit(1/6);

//go back to the first

animation.setRestoreOriginalFrame(true);

//create animate

var animate = new cc.Animate(animation);

player.runAction(animate);

* + 方法二：利用\*.plist和\*.png文件创建动画

//add SpriteFrameCache

cc.spriteFrameCache.addSpriteFrames(res.running\_plist);

//create spirte

var player = cc.Sprite.create("#runner0.png");

player.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.addChild(player,1);

//create Animation array

var array = [];

for(var i=0;i<8;++i){

var path = "runner"+i+".png";

var spriteFrame = cc.spriteFrameCache.getSpriteFrame(path);

array.push(spriteFrame);

}

//create Animation

var animation = new cc.Animation(array,0.1);

animation.setRestoreOriginalFrame(true);

//create animate

var animate = new cc.Animate(animation);

//sprite run animate

player.runAction(animate);

* + 方法三:

var animation= new cc.Animation();

for(var i=0; i< arrPath.length; ++i){

var frame= cc.spriteFrameCache.getSpriteFrame(arrPath[i]);

animation.addSpriteFrame(frame);

}

animation.setDelayPerUnit(delay);

animation.setRestoreOriginalFrame(bRestoreOriginal);

var animate= cc.animate(animation);

target.runAction(animate.repeatForever());

* + 方法四:直接利用动画文件播放
    - 资源：grossini.plist、grossini.png和animations-2.plist
    - 在动画文件\*.plist的最后，添加：

<key>properties</key>

<dict>

<key>spritesheets</key>

<array>

<!—动画列表-->

<string>grossini.plist</string>

</array>

<key>format</key>

<integer>2</integer>

</dict>

* + - 创建精灵，执行动画

//create spirte

var player = cc.Sprite.create("res/runner0.png");

player.attr({

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

this.addChild(player,1);

var animCache = cc.animationCache;

animCache.addAnimations("res/animations-2.plist");

//\*.xml文件中，dance\_1子节点动画

var animation2 = animCache.getAnimation("dance\_1");

var action2 = cc.animate(animation2);

player.runAction(action2);

1. 场景特效

replaceScene方法：把当前场景切换为另一个场景。然而replaceScene方法的场景切换并没有过多的修饰，仅仅是停止当前场景，再播放下一个场景。

场景是一类特殊的场景，它包含了另一个场景，在运行场景特效时，被包含的原场景会以添加了特效的方式显示出来。因此，特效场景的使用方法与复合动作也类似。首先创建一个场景，称作原场景，然后把原场景当做参数来创建一个特效场景，使用时只需要把特效场景传入CCDirector的replaceScene方法即可。相关代码如下：

CCDirector::sharedDirector()->replaceScene(CCTransitionFlipX::transitionWithDuration(2，pScene));

场景特效切换时机，有两种：

1. 任何时候，直接利用cc.director.replaceeScene()方法进行切换；
2. 在场景的init函数中，执行场景切换。

下表简单罗列了部分具有代表性的场景特效。全部的场景特效的代码都位于引擎目录中CCTransition.h文件中。

常用的场景特效

CCTransitionJumpZoom 跳跃缩放，当前场景会先缩小，然后跳跃切换

CCTransitionFade 淡出淡入效果，可以设置覆盖颜色

CCTransitionFlipx(CCTransitionFlipY) 沿着x轴向左翻(沿着y轴向上翻)

CCTransitionShrinkGrow 交错切换场景

CCTransitionMoveInL 从左进入覆盖原场景

CCTransitionSlideInL 从左进入推进原场景

【要想看全部的场景切换特效，可以查看引擎下的CTranstion.js文件。在<http://cn.cocos2d-x.org/doc/jsdoc/index.html>网页上，可以查看对应的类的作用（中文）】

知识点简介：

* 框帧（CCSpriteFrame）:包含纹理与纹理中的一个矩形区域，表示纹理的一部分。一个精灵显示的内容就可以用框帧表示，同时框帧还是帧动画的基本元素。
* 帧动画（CCAnimationFrame）:由框帧与单位延时组成，可以表示变速动画中的一帧。通常，匀速动画的单位延时为1。
* 动画（CCAnimation）:由动画帧组成,表示一个动画的内容；
* 动画动作（CCAnimate）动画的播放器，使用动画对象创建，只能作用于精灵。为了播放一个动画，通常首先创建动画帧或框帧，然后用它们创建动画，最后利用动画创建动画动作，并指派一个精灵来执行此动作。
* 场景特效（CCTransitionScene）:一类特殊的场景，可以把另一个场景包装起来，实现诸如特殊翻页、波纹等华丽的场景切换特效。

1. 加速度计

除了触摸，移动设备上一个很重要的输入来源是设备的方向，因此大多数设备都配备了加速度计，用于测量设备静止或匀速运动时所受到的重力方向。

说明：在IOS与Android手持设备中都提供了重力加速度计感应器。

只有那些主流的、用户习惯的游戏操作方式，才会被引擎采纳。其中在游戏中使用比较频繁的就算是加速度计的用户操作了。

在Cocos2D-X引擎当中，使用了类CCAccelerometer类存储加速度计的信息。类CCAcceleromete的作用于用户操作的分发器；类似。区别在于用户操作的分发器可以拥有很多的委托对象。二加速度计只存在一个委托对象。这是因为一台手持设备只有一个硬件，所以接口进行了简化。类CCAccelerometerDelegate就是加速度计的委托对象。开发者同样需要采用注册的方式来使用它。

不过任何一种操作方式，都需要用户适应一段时间。前几次电子设备的更新都推动了游戏产品的发展，比如鼠标、触摸、体感等等。一旦用户接受了新的操作体验，必将会成为游戏产品又一次热潮。

说明：体感或称躯体感觉，是现今被家用游戏机发明并得以发展的一种用户操作方式。

目前，加速度计在传统的PC平台上尚未提供支持。重力感应来自于移动设备的加速度计，通常支持X、Y和Z三个方向的加速度感应，所以又称为三向加速度计。在实际应用中，可以根据3个方向的力度大小来计算手机倾斜的角度或方向。iOS和Android等设备三轴加速度计的坐标系是右手坐标系，即：设备竖直向上，正面朝向用户，水平向右为X轴正方向，竖直向上为Y轴正方向，Z轴正方向是从设备指向用户方向。

方法一：利用Cocos开发

和触摸事件一样，重力感应的处理先被引擎抽象为一个触摸代理协议，然后由CCAcceleration是一个结构体，包含了加速度计获得的3个方向的加速度。

3.0中，新的事件机制下，我们需要通过创建一个加速计监听器EventListenerAcceleration，其静态create方法中有个Acceleration的参数需要注意。

typedef struct{

double x;

double y;

double z;

double timestamp

}CCAcceleration;

该类中每个方向的加速度大小都为一个重力加速度大小。

* 水平
* 正面朝上：(0,0,-1)【角度关系——对应值\*90——(0,0,-90)】
* 背面朝上：(0,0,1) 【角度关系——对应值\*90——(0,0,90)】
* 竖直
* 正面朝向用户：(0, -1，0)【角度关系——对应值\*90——(0, -90，0)】
* 背面朝向用户：(0, 1，0)【角度关系——对应值\*90——(0, 90，0)】
* 倾斜【初始状态——水平、正面朝上】
* 向左翻转90度：(-1, 0，0)【角度关系——对应值\*90——(-90, 0，0)】
* 向右翻转90度：(1, 0，0)【角度关系——对应值\*90——(90, 0，0)】

x、y和z的取值范围是（-1,1），其值乘以9.8就是加速度,其值乘以90就是对应方向上的倾斜角度。

注意：如果浏览器支持——随着手机的朝向不同，浏览器的视图可自动发生旋转。那么当X轴方向上的旋转角度达到45度以上，此时就应该会发生了旋转。【

注意：

1. 不同的手机，旋转时所需要的临界角度不同——苹果4：45度 华为H30-U10：25度；
2. 不同手机的x、y和z的取值范围也不同，例如: 华为H30-U10水平放置时，z轴的取值在3.6-3.7之间。不知道是不是个例。】

* 假设：浏览器支持——随着手机的朝向不同，浏览器的视图可自动发生旋转。注意:此时浏览器的坐标系，不会发生变化，仍然是右手坐标系。不过X和Y轴发生了交换。【X轴:窄屏方向🡪宽屏方向 Y轴:宽屏方向🡪窄屏方向】
* 引擎:

其中lite版本、自选版本【core、audio、actions、menus】的引擎，均不支持重力感应。

在自选版本中，勾选process-timer项，就会支持重力感应。

* 在使用加速计事件监听之前，首先应该判断系统是否支持：

if( 'accelerometer' in cc.sys.capabilities )

说明:

此种判定条件，在电脑浏览器上，判定条件值为true。只不过没有响应不了而已。

* 启用此设备：

Device::setAccelerometerEnabled(true);

或者是

//开启重力感应

cc.inputManager.setAccelerometerEnabled(true);

//设置监听机制时间间隔

cc.inputManager.setAccelerometerInterval(1/30);

其中，setAccelerometerEnabled()函数在WebStrom中会被标记错误，但是不影响程序运行。

* 添加监听机制:

cc.eventManager.addListener({

event: cc.EventListener.ACCELERATION,

callback: function(accelEvent, event){

if((accelEvent.x!=0)||(accelEvent.z!=0)){

sprite.setPosition(cc.p(cc.winSize.width/2+(accelEvent.x\*30),cc.winSize.height/2+(accelEvent.y\*30)));

}

}

}, this);

* 记录摇动次数:

if(accelEvent.x> 0.01){

if(!g\_bADD){

g\_bADD= true;

times++;

}

}else{

g\_bADD= false;

}

* 示例代码：

//添加重力感应

addAccelerometer:function(){

//开启重力感应

cc.inputManager.setAccelerometerEnabled(true);

//设置监听机制的时间间隔

cc.inputManager.setAccelerometerInterval(1/10);

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.ACCELERATION,

callback:self.showNum

},self);

cc.eventManager.addListener(listener, self);

},

//加速计的回调函数

showNum:function(accelEvent, event){

var self= event.getCurrentTarget();

if(accelEvent.x> 0.01){

if(!g\_bADD){

g\_bADD= true;

cc.audioEngine.playEffect(res.effect\_mp3);

}

}else{

g\_bADD= false;

}

}

* 方法二：利用html测试加速度计【晃动DeviceMotionEvent事件】

if (window.DeviceMotionEvent) {

window.addEventListener("devicemotion", motionHandler, false);

}

* devicemotion简介:

html5提供了几个新的DOM事件来获得设备物理方向及运动的信息，包括：陀螺仪、罗盘及加速计。

第一个DOM事件是deviceorientation，其提供设备的物理方向信息，表示为一系列本地坐标系的旋角。

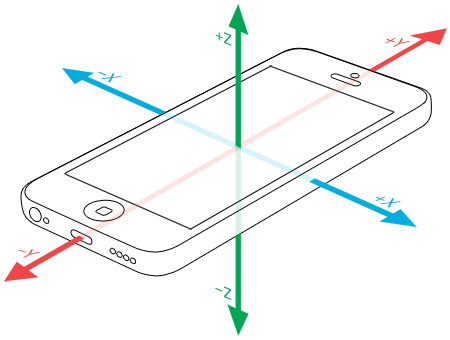
第二个DOM事件是devicemotion，其提供设备的加速信息，表示为定义在设备上的坐标系中的卡尔迪坐标。其还提供了设备在坐标系中的自转速率。

第三个DOM事件是compassneedscalibration，其用于通知Web站点使用罗盘信息校准上述事件。

* 原理:

开发者从各个内置传感器那里获取未经修改的传感数据，并观测或响应设备各种运动和角度变化。这些传感器包括陀螺仪、加速器和磁力仪(罗盘)。

加速器和陀螺仪的数据都是描述沿着iOS设备三个方向轴上的位置，对于一个竖屏摆放的iPhone来说，X方向从设备的左边(负)到右边(正)，Y方向则是由设备的底部(-)到顶部(+)，而Z方向为垂直于屏幕由设备的背面(-)到正面(+)。



DeviceMotionEvent 会在设备发生有意义的摆动(或运动)时产生.事件对象封装有产生的间距值,旋转率,和设备加速度.

加速度的计算方式是重力和用户产生的两个加速度矢量之和.设备是通过陀螺仪和加速计来区别这两者的.

通过DeviceMotion对设备运动状态的判断，则可以帮助我们在网页上就实现“摇一摇”的交互效果。

* 事件监听:

if (window.DeviceMotionEvent) {

window.addEventListener('devicemotion', deviceMotionHandler, false);

} else {

alert("Not supported on your device！");

}

* 捕捉重力加速度

function deviceMotionHandler(eventData) {

var acceleration =eventData.accelerationIncludingGravity;

x = acceleration.x;

y = acceleration.y;

z = acceleration.z;

}

1. 精灵反转
   * 左右反转：setFlippedX(true);
   * 上下反转：setFlippedY(true);
2. 动态修改精灵的层次感

在跑酷横版中，玩家移动到障碍物的上方时，应该是障碍物遮蔽玩家；玩家移动到障碍物的下方时，应该是玩家遮蔽障碍物。

主要用到函数——getLocalZOrder()和setLocalZOrder()函数，来修改Z值。

实现上面需求：获取精灵和障碍物的位置，根据位置关系，动态修改精灵的Z值。

if(this.removeArray[i].getPositionY()<this.playerSprite.getPositionY()){

this.removeArray[i].setLocalZOrder(this.playerSprite.getLocalZOrder()+1);

}

注意:

通过上述两个方法，来修改对象的Z值时，必须保证该对象是cc.Layer的直接子节点,否则将会没有效果。

1. 淡入淡出效果

方法说明：

* cc.fadeTo(duration, opacity)/ cc.FadeTo(duration, opacity)
* cc.fadeBy(duration, opacity)/ cc.FadeBy(duration, opacity)

渐变一个对象的透明度（遵循cc.RGBAProtocol协议）。从当前透明度到一个自定义值。参数说明：

* + duration——动作执行时间
  + opacity——透明度。取值范围【0-255】。0——全透明
* cc.FadeIn(duration)

淡入一个对象（遵循cc.RGBAProtocol协议）。透明度从0变化到255。

"reverse"动作是FadeOut。

* cc.FadeOut(duration)

淡出一个对象（遵循cc.RGBAProtocol协议）。透明度从255变化到0。

"reverse"动作是FadeIn。

* cc.fadeTo(duration, opacity)

渐变一个对象（遵循cc.RGBAProtocol协议）透明度。透明度从当前值变化到指定的透明度opactity值。

注意：cc.fadeTo动作函数没有与之相对的cc.fadeBy。

var fadeOut = cc.fadeOut(1);//淡出

var fadeIn = cc.fadeIn(1);//淡入

var delay = cc.delayTime(5);//延时

var seq = new cc.Sequence(fadeOut,delay,fadeIn);

player.runAction(seq.repeatForever());

1. 用户交互

新型游戏：一款用摄像头来操作飞碟移动的游戏。玩家需要在游戏开始的时候，保持头部不动，然后就可以在游戏中用头来控制飞船行进了。但是别人可不知道呢，还以为晚间是在用意念控制游戏。这类游戏确实能够给玩家带来耳目一新的感觉。

鼠标和触摸屏的区别：

* + 鼠标可是实现悬停效果，触摸屏不可以；
  + 鼠标只有一个焦点，而触摸屏大多数支持多个触摸点；
  + 鼠标通常包含两个功能键从而区分左右键，而触摸屏只能实现一种点按操作

总概：

* 事件监听器(cc.EventListener) 封装用户的事件处理逻辑
* 事件管理器(cc.eventManager) 管理用户注册的事件监听器，根据触发的事件类型分发给相应的事件监听器
* 事件对象(cc.Event) 包含事件相关信息的对象

事件监听器包含以下几种类型：

* 触摸事件监听器 (cc.EventListenerTouch)

//单触点监听

addListener:function(){

if('touches' in cc.sys.capabilities){

this.addTouch();

}/\*else if('mouse' in cc.sys.capabilities){

this.addMouse();

}\*/else{

alert("该设备不支持！");

}

},

addTouch:function(){//触摸

cc.eventManager.addListener({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

onTouchBegan:this.onTouchBegan,

onTouchMoved:this.onTouchMoved

},this);

},

onTouchBegan:function(touch,event){//开始触摸

var target= event.getCurrentTarget();

var point= touch.getLocation();

var size= cc.rect(0,0,target.width,target.height);

if(cc.rectContainsPoint(size,point)){

target.label.setString(point.x);

return true;

}

return false;

},

onTouchMoved:function(touch,event){//移动中

var target= event.getCurrentTarget();

var point= touch.getLocation();

var size= target.getContentSize();

var rect= cc.rect(0,0,size.width,size.height);

if(cc.rectContainsPoint(rect,point)){

target.label.setString(point.x);

}

}

//多点触点监听

addListener:function(){

if('touches' in cc.sys.capabilities){

this.addTouch();

}else{

alert("该设备不支持！");

}

},

addTouch:function(){//触摸

cc.eventManager.addListener({

event:cc.EventListener.TOUCH\_ALL\_AT\_ONCE,

onTouchesBegan:this.onTouchesBegan,

onTouchesMoved:this.onTouchesMoved

},this);

},

onTouchesBegan:function(touches,event){//开始触摸

var target= event.getCurrentTarget();

for(var i=0;i<touches.length;++i){

var point= touches[i].getLocation();

var size= cc.rect(0,0,target.width,target.height);

if(cc.rectContainsPoint(size,point)){

return true;

}

}

return false;

},

onTouchesMoved:function(touches,event){//开始触摸

var target= event.getCurrentTarget();

for(var i=0;i<touches.length;++i){

var point= touches[i].getLocation();

var size= cc.rect(0,0,target.width,target.height);

if(cc.rectContainsPoint(size,point)){

target.label.setString(touches.length);

}

}

}

* 键盘事件监听器 (cc.EventListenerKeyboard)

addListener:function(){

if('keyboard' in cc.sys.capabilities){

this.addTouch();

}else{

alert("该设备不支持键盘！");

}

},

addTouch:function(){//触摸

cc.eventManager.addListener({

event:cc.EventListener.KEYBOARD,

onKeyPressed:this.onKeyPressed,

onKeyReleased:this.onKeyReleased

},this);

},

onKeyPressed:function(key,event){

var target= event.getCurrentTarget();

target.label.setString(String.fromCharCode(key));

},

onKeyReleased:function(key,event){

var target= event.getCurrentTarget();

target.label.setString("0");

}

* 加速计事件监听器 (cc.EventListenerAcceleration)

addListener:function(){

if( 'accelerometer' in cc.sys.capabilities ) {

cc.inputManager.setAccelerometerInterval(1/30);

cc.inputManager.setAccelerometerEnabled(true);

cc.eventManager.addListener({

event: cc.EventListener.ACCELERATION,

callback: function(accelEvent, event){

var target = event.getCurrentTarget();

var w = cc.winSize.width;

var h = cc.winSize.height;

var x = w \* accelEvent.x + w/2;

var y = h \* accelEvent.y + h/2;

x = x\*0.2 + target.prevX\*0.8;

y = y\*0.2 + target.prevY\*0.8;

target.prevX = x;

target.prevY = y;

target.ball.x = x;

target.ball.y = y ;

}

}, this);

this.prevX = 0;

this.prevY = 0;

} else {

alert("ACCELEROMETER not supported");

}

},

onExit:function(){

this.\_super();

if( 'accelerometer' in cc.sys.capabilities )

cc.inputManager.setAccelerometerEnabled(false);

}

* 鼠标事件监听器 (cc.EventListenerMouse)

addListener:function(){

if( 'mouse' in cc.sys.capabilities ) {

cc.eventManager.addListener({

event:cc.EventListener.MOUSE,

onMouseMove:this.onMouseMove

},this);

} else {

alert("不支持鼠标");

}

},

onMouseMove:function(event){

var target= event.getCurrentTarget();

target.ball.setPosition(event.getLocation());

}

* 自定义事件监听器 (cc.EventListenerCustom)

在监听器cc.EventListener中实现各种事件的处理逻辑，然后将监听器加入到事件管理器cc.eventManager中, 当事件触发时，事件管理器会根据事件类型分发给相应的事件监听器。

一般情况下，我们可以通过TouchEnabel属性来开启或关闭接收触摸输入。

CCLayer实现了以下4个方法，当引擎接收到触摸事件时，这些方法就会被调用：

* virtual void ccTouchesBegan(CCSet\* pTouches,CCEvent\* pEvent);
* virtual void ccTouchesMoved(CCSet\* pTouches,CCEvent\* pEvent);
* virtual void ccTouchesEnded(CCSet\* pTouches,CCEvent\* pEvent);
* virtual void ccTouchesCancelled(CCSet\* pTouches,CCEvent\* pEvent);

以上4个方法都声明为虚函数，这意味着我们通过重载的方式来处理用户输入事件。实际上，这4个函数来自于CCStandardTouchDelegate接口，是触摸事件的回调函数，分别对应触摸的开始、移动、结束和取消操作。传入参数pTouches是一个CCTouch对象（表示一个触摸点）的集合，其中包含当前触摸事件中的所有触摸点。

触摸事件分为4类，其中结束事件和取消事件容易使人困惑。通常，当玩家的触摸动作完成时（例如抬手和手指离开屏幕等），会引起触摸结束事件。而触摸取消的情况较少，当触摸过程中，程序被调入后台时才会出现。

事件处理函数：void ccTouchesMoved(CCSet\* pTouches,CCEvent\* pEvent);

其中pTouches->count()【触摸点的数量】

由于在不同平台下触摸点的坐标系与OpenGL呈现区域的参数可能不尽相同，所以触摸点的位置通常与系统相关。不过，Cocos2d-x已经为我们处理好了这个问题，只需要调用CCTouch：：locationInView方法获取游戏画面中的点位置，然后再调用CCDirector::convertToGL方法把屏幕坐标转换为游戏坐标，就可以获取触摸点在游戏中的位置了。

var point = touch.locationInView();

point = cc.director.converToGL(point);

利用层里实现触摸十分简便，然后只要玩家出没了屏幕，所有响应触摸事件的层都会被触发。当层的数量很多时，维护多个层的触摸事件就成了一件复杂的事情。因此，在实际开发中，我们通常单独建立一个触摸层。用触摸层来接收用户输入事件，并根据需要通知游戏中的其他部件来响应触摸事件。

* 设置监听机制是否可用：cc.eventManager.setEnabled(参数);

参数取值：false—不可用 true—可用

* 移除所有的单点监听机制：

cc.eventManager.removeListeners(cc.EventListener.TOUCH\_ONE\_BY\_ONE);

* 动态移除或添加对应的监听机制：

//移除

cc.eventManager.removeListener(this.listener);

//添加

cc.eventManager.addListener(this.listener,this.block1);

* 暂停或回复某个对象的监听机制：

//暂停：

cc.eventManager.pauseTarget(this,true);

//恢复

cc.eventManager.resumeTarget(this,true);

注意:

在自定义一个类的时候，在类中添加监听机制时，有时会发生失效情况。【我的解决办法是:设定类对象的显示位置之后，再添加监听机制。】

1. 两种Cocos2d-x触摸事件

当我们开启CCLayer的TouchEnable属性后，层中的4个回调函数就会在接收到事件时被触发，我们把这类事件称作标准触摸（standard touch）事件，它的特点是任何层都会平等地接收全部触摸事件。

除此之外，Cocos2d-x还提供了另一种被称作带目标的触摸事件（targeted touch）机制。在带目标的触摸机制中，接受者并不平等，焦躁处理事件的接收者有权停止事件的分发，使它不再继续传递其他接受者。换句话说，带目标的触摸事件并不一定会被广播给所有的接受者。通常，游戏的菜单按钮、摇杆按钮等元素常使用目标触摸事件，以保证触摸事件不对其他层产生不良影响。

Cocos2D-X引擎实现跨平台的秘密。针对各个设备平台使用不同的CCEGLView文件来处理相同的用户操作信息。然而将信息传递到引擎当中。由于引擎提供了统一的对外接口，在开发者使用时，就可以忽略各个平台存在的差异。读者可以进入到引擎源代码中的platform\目录，就会看到针对不同平台而编写的CCEGLView文件。在Android平台就能看到用Java语言编写的代码。它们不仅解决了跨平台绘制界面的问题，同时也对用户操作提供了统一的函数。【IOS中，引擎使用了EAGLView来处理画面显示与用户交互。】

在引擎当中，用户操作将会经过接受、分发、处理三个步骤。

* 标准触摸事件

不难发现，类CCLayer只提供了两种用户交互的方式，因为只存在触摸操作与加速度计操作的响应函数。

实际上，并不是只有层才去支持接收触摸事件，任何一个游戏元素都可以接受时间，只不过层中提供了现成的支持。

在引擎中，CCTouchDispatcher负责触摸事件的分发处理，此处的addStandardDelegate方法会把当前对象注册到分发器中。被注册的对象必须实现CCStanardTouchDelegate接口，当引擎从系统接收到触摸事件时，就会调用接口中对应的方法，触发触摸事件。

相比之下，关闭触摸则简单得多：只需要调用CCTouchDispatcher的removeDelegate方法即可。

因此可以总结，为了是一个对象接受标准触摸事件，主要有以下4个步骤：

1. 需要此对象实现CCStandardTouchDelegate接口；
2. 使用addStanardDelegate方法把自己注册给触摸事件分发器；
3. 重载事件回调函数，处理触摸事件；
4. 当不需要接受触摸事件时，使用removeDelegate方法来注销触摸事件的接受。

回调函数

* virtual void ccTouchesBegan(CCSet\* pTouches,CCEvent\* pEvent);
* virtual void ccTouchesMoved(CCSet\* pTouches,CCEvent\* pEvent);
* virtual void ccTouchesEnded(CCSet\* pTouches,CCEvent\* pEvent);
* virtual void ccTouchesCancelled(CCSet\* pTouches,CCEvent\* pEvent);
* 带目标的触摸事件

标准触摸事件中存在两个较为不便的地方，具体如下所示：

* + 只要事件分发器接受到用户的触摸事件，就会分发给所有的订阅者，因此常常会出现按下按钮时，触摸事件穿透按钮分发给后面的层这种尴尬的情况；
  + 当系统存在多个触摸点时，标准触摸事件会吧所有触摸点都传递给回调韩侯蔌，然而在许多情况下每个触摸点之间是独立的，屏幕上是否存在其他触摸点我们并不关心，因此我们不必为了处理多个触摸事件手动遍历一边触摸点；

为了，Cocos2d-X为我们提供了一个简化的解决方案：带目标的触摸事件。与标准触摸事件类似，我们也需要首先使接受时间的对象实现一个借口CCTargetTouchDelegate，然后把对象注册到触摸分发器中，最后当不再需要接受触摸事件时注销自身。

和CCStandardTouchDelegate接口类似，为了接受触摸事件，我们童颜需要实现下面所示的回调函数：

* virtual bool ccTouchesBegan(CCTouch\* pTouch,CCEvent\* pEvent);
* virtual void ccTouchesMoved(CCTouch\* pTouch,CCEvent\* pEvent);
* virtual void ccTouchesEnded(CCTouch\* pTouch,CCEvent\* pEvent);
* virtual void ccTouchesCancelled(CCTouch\* pTouch,CCEvent\* pEvent);

这个接口和CCStandarTouchDelegate存在两处不同，这两处不同也对应着带目标的触摸事件的两个特点：

* + 时间参数不再是集合，而是一次只传入一个触摸点；
  + ccTouchBegan方法返回一个布尔值，表示声明是否要捕捉这个触摸点，只有再次方法中捕捉到的触摸点才会继续引发其他3个事件，否则次触摸点的其他时间都会被忽略。

通过CCTargetedTouchDelegate的方式接受触摸事件，就无法直接利用CCLayer提供的属性了，因此我们必须主动把自身注册到引擎的触摸分发器：

CCTouchDispatcher::sharedDispatcher()->addTargetDelegate(this,0,true)；

在addTargetedDeletage方法中，前两个参数分别对应触摸接受对象和优先级，其中优先级是一个整型参数，值越低，则优先级越高，也就越早获得触摸事件。通常，为了获得较高的优先级，可以将其指定为负数。第三个参数较为有趣，表明了是否“吞噬”一个触摸，如果设置为true，一个触摸一旦被捕捉，那么所有优先级更低的接受对象都无法接受触摸。CCMenu就是一个会“吞噬”且优先级为1-128的触摸接收器，由于它的优先级很高，所以菜单按钮总能获得触摸响应。

综上所述，带目标的触摸事件的使用步骤如下所示：

* + 实现CCTargetedTouchDelegate接口；
  + 使用addTargetedDelegate方法注册到触摸事件分发器；
  + 重载事件回调函数。注意，我们必须在触摸开始事件中针对需要接受的事件返回true以捕捉事件；
  + 当不再需要接受触摸事件时，使用removeDelegate方法来注销触摸事件的接受。与标准触摸事件相比，不同之处主要在于开始触摸事件需要返回一个代表是否捕捉事件的值。
* 接受操作

为了在引擎中，将操作数据作为对象操作和传递的便利，引擎开发者就设计了一个专门存储用户操作的类CCTouch。

每个CCTouch类的对象都包含了一个用户操作信息。其属性当中存储了id标识、x坐标和y坐标。这就是类CCTouch的作用。它能够将采集自不同平台的触摸信息，保存为统一的格式。

* 触摸分发器原理

然而无论使用哪一种机制，最关键的一步都是把接受事件的对象注册到触摸分发器中，这是因为触摸分发器会利用系统的API获取触摸事件，然后把事件分发给游戏中接收触摸事件的对象。

另一个方面。时间分发器从系统接受到了触摸事件之后还需要逐一分发。分发事件的相关代码主要集中在touches方法之中。

引擎专门提供了一个负责分发用户操作信息的类。类CCTouchDispatcher的作用就是向响应对象分发用户操作信息，即类CCTouch对象。

所谓的分发，就是类CCTouchDispatcher不仅仅只发送一次，它会向所有需要响应的对象。发送用户操作信息。类CCTouchDelegate就是分发器发送用户操作信息的目标对象。我们姑且将此类型的对象称为用户操作委托对象吧！类CCTouchDelegate具有三个子类：CCLayer、CCStandardTouchdelagate和CCTargetedTouchDelegate。后两个，一个为标准的委托对象，另一个为目标委托对象。

作为目标委托对象，其响应函数所传递的参数都为单一的用户操作信息。而标准委托对象，其响应函数所传递的参数则是一个用户信息的集合。说得明白一个，目标委托对象是用来响应单点触摸的，比如一个精灵或者一个按钮。玩家是很难在一个按钮上进行多点触碰操作的。而与之对应的标准委托对象则是用在触碰面积较大的层对象之上。

分发过程遵循以下的规则：

* 对于触摸集合中的每个触摸点，按照游戏询问每一个注册到分发器的对象。对于同一优先级的对象，访问顺序并不确定。
* 当接收到开始事件时，如果接受者返回true，则称对象捕捉了此事件。只有被捕捉，后续事件（如移动、结束等）才会继续分发给目标对象。
* 如果设置了吞噬属性，则捕捉到的点会被吞噬。被吞噬的点将立即溢出触摸集合，不再分发给后续目标（包括注册了标准触摸事件的目标）。
* 将没有被吞噬的触摸点集按优先级的顺序分发给每一个注册了注册了标准触摸时间的目标对象，同一优先级之间对象的访问顺序并不确定。
* 为了避免时间分发中时间处理对象呗改变，Cocos2d-x仔细维护了两个临时表，因此开发者无论何时都可注册或注销触摸事件。
* 触摸中的陷阱

1. 接受触摸的对象并不一定正显示在屏幕上。触摸分发器和引擎中的绘图是相互独立的，所以并不关心触摸代理是否处于屏幕上。因此，在实际使用中，应该不需要的时候从分发器中溢出触摸代理，尤其是自定义的触摸对象。而CCLayer也仅仅会在切换场景时从分发器中移除，所以同场景内手动切换CCLayer的时候，也需要注意禁用触摸来从分发器移除自己。
2. 另一个陷阱出自CCTargetTouchDelegate。尽管灭此值传入一个触摸点，也只有在开始阶段被声明过的触摸点后续才会传入，但是这并不因为这只会接受一个触摸点：只要被声明过触摸点都会传入，而且可能是乱序的。因此，一个良好的习惯是，如果使用CCTargetedTouchDelagate,那么只声明一个触摸，针对一个触摸做处理即可。

* 手势识别
* 多点触摸

说明：多点触碰是可以包含单点触碰的，因此标准委托对象可以替代目标委托对象的功能。

* 文字输入

在没有实体键盘的触摸手机上，文字输入通过虚拟键盘来实现。不幸的是，CCLayer并没有为我们提供一个简化的封装实现；而幸运的是，在Cocos2d-x中已经存在一个现成的文本框控件CCTextFiledTTF了。尽管这个控件的接口稍微有些复杂，但使用起来并不麻烦。它的外观为屏幕上一块可输入的区域，如同各种平台下的文本框一样，点击后可弹出键盘。

CCTextFieldTTF提供了3个常用的操作接口，具体如下所示：

const char\* getString(void);//获取当前字符串

bool attachWithIME();//激活输入法

bool detachWithIME();//取消激活输入法

这3个方法基本可以满足开发需要了。如果我们在开始场景中添加一个让用户输入用户名的文本框，只需要在StartScene类的初始化代码总添加一个CCTextFieldTTF控件即可。

var textField = new cc.TextFieldTTF("<click here for input>",

"Impact",

20);

textField.setPosition(cc.p(200,200));

textField.attachWithIME();//弹出虚拟键盘

this.addChild(textField);

问题：弹出的虚拟键盘会将屏幕下半部分盖住。

在很多情景下，如果文本框敲好在屏幕的下半部分，会直接被弹出的键盘覆盖，用户无法正常输入文字。我们应该在输入法弹出和收起的时候调整布局到一个合适的位置。

CCTextFieldTTF预留了一个代理CCTextFieldDelegate协议来通知相关事件。这个代理协议实际上是输入法代理协议的简化封装，其中包含了文本输入框的5个常用事件:

//即将激活输入法，如果不想激活，返回true

virtual bool onTextFieldAttachWithIME(CCTextFieldTTF \* mysender){

//启动键盘

return false;

//不启动键盘

//return ture;

}

//即将取消输入法，如果不想取消，应该返回true

virtual bool onTextFieldDetachWithIME(CCTextFieldTTF \* mysender){

//关闭键盘

return false;

//不关闭键盘

//return ture;

}

//即将插入一段文字，如果不想插入，应该返回true

virtual bool onTextFieldInsertText(CCTextFieldTTF \* mysender,const char \* text,int textnum){

//输入字符

return false;

//不输入字符

//return ture;

}

//即将删除一段文字，如果不想删除，应该返回true

virtual bool onTextFieldDeleteBackward(CCTextFieldTTF \* mysender,const char \* text,int textnum){

//删除字符

return false;

//不删除字符

//return ture;

}

//如果不希望绘制这个输入法，返回true

virtual bool onDraw(CCTextFieldTTF\* sender);

实现了CCTextFieldDelegate协议的类可以获取输入框的以上5个事件。我们就可以利用其中的前两个接口来重新布局屏幕。

最后，不要忘记设置文本输入框和协议响应方：

text->setDelegate(this);

注意：使用CCTextFieldTTF类时，必须在project.json文件中，添加：

"modules" : ["extensions"]，

注意：事件的监听机制和菜单的单击相比，菜单的优先级更高。【好像为-1】

1. CCNode与坐标系

Cocos2d-X把渲染树上的每一个游戏元素抽象为一个节点，即CCNode。一切游戏元素都继承自CCNode，因此它们都具有CCNode所提供的特性。

CCNode定义了一个可会知对象的通用特性，包括位置、缩放、是否可见、旋转角度等。

* 节点的最基本的功能包括：
* 包含其他的CCNode对象；
* 接受各种事件与回调函数，如定时器事件；
* 运行动作；
* Cocos2d-x中的坐标系

在Cocos2d-x中，存在两种坐标系:

* 绘图坐标系

它是最常见的坐标系，与OpenGL采用的坐标系相同，以左下角为原点，向右为x轴正方向，向上为y轴正方向，左手坐标系。在Cocos2d-x中，一切绘图相关的操作都使用绘图坐标系，如游戏元素中的Position和AnchorPoint等属性。

* 纹理坐标系

纹理坐标系以左上角为原点，向右为X轴正方向，向下为Y轴正方向。在Cocos2d-x中，只有从纹理中截取部分矩形时才使用这个坐标系。如CCSprite的TextureRect属性。

* 绘图属性

利用这些属性，我们可以对精灵呈现的方式进行精确的控制。

* + CCSize ContentSize

获取或设置次节点的内容大小。任何一个节点都需要确定它的内容大小，一百年进行图形变换。对于精灵来说，ContentSize是它的纹理显示部分的大小；对于层或场景等全屏的大型节点来说，ContentSize则是屏幕大小。

* + CCPoint AnchorPoint与CCPoint Position

AnchorPoint用于设置一个锚点，一边精确地控制节点的位置和变换。AnchorPoint的两个参量x和y的取值通常都是0到1之间的实数，表示锚点相对于节点长宽的位置。例如，把节点左下角作为锚点，值为（0,0）；把节点的中心作为锚点，值为（0.5,0.5）；把节点右下角作为锚点，值为（1,0）。精灵的AnchorPoint默认值为（0.5,0.5）,其他节点的默认值为（0,0）。

Position用于设置节点的位置。由于Position指的是锚点在父节点中的坐标值，节点显示的位置通常与锚点有关。

* + float Rotation

获取或设置节点的旋转角度。节点以自己的锚点为中心顺时针旋转一定量，单位是角度。旋转角度可以是任意实数。

* + float scale(float ScaleX与float ScaleY)

Scale用于获取或设置及诶按的缩放比例。节点以锚点为中心缩放该比例。Scale的值代表整体缩放比例。而ScaleX与ScaleY分别代表X方向与Y方向的缩放比例。默认情况下这三个属性的值都是1，表示节点不被缩放。如果设置Scale属性，则ScaleX和ScaleY都会随之变为相同的值。

* + bool Visible

获取或设置节点的可见性。在节点不被显示的时候，也不会被调用绘图方法（visit与draw）。

* + float SkewX与float SkewY

获取或者是斜切角度。节点以锚点为中心，平行x轴或Y轴方向作为一定角度的变形。SkewX为平行X轴顺时针的变形，，SkewY为平行y轴逆时针的变形，单位为角度。SkewX与SkewY的默认值为0，表示节点没有斜切变形。

* + int Tag

获取或设置节点的标号。

* + void\* UserData

获取或设置与节点相关的额外信息。UerData为void\*类型，我们可以利用这个属性来保存任何数据。

* + CCArray\* Children

获取保存了该节点所有子引用的数组。

* + CCNode\* Parent

获取或设置该节点的父节点

* + CCCamera\* Camera

获取或设置该节点的摄像机状态。摄像机（CCCamera）定义了绘制该节点时的视点，通查那个用于实现特效。

* + CCGridBase\* Grid

获取或设置该节点的网格特效状态。网格应用于绘图平面上，可以四号线水纹等3D特效

* + CCGLProgram\* ShaderProgram

获取或设置该节点的Shader程序。CCGLProgram是OpenGl的glProgram的封装

* + CCActionManager\* ActionManager

获取或设置该节点所使用的动作。当为节点设置了新的动作管理器时，正在实行的动作都会被丢弃。

* + CCSchedule\* Scheduler

获取或设置该节点所使用的计时器管理器。当为节点设置了新的计时器管理器时，正在运行的计时器都会被丢弃。

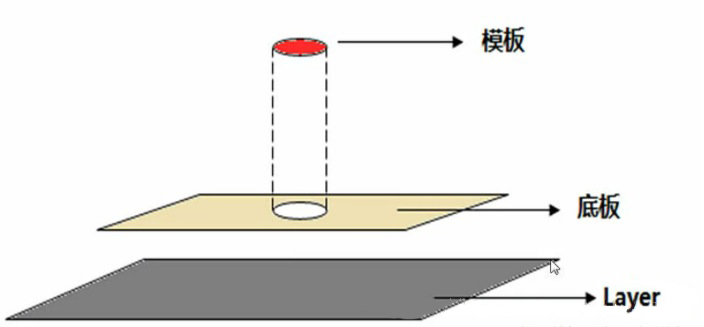
1. ClippingNode的使用

ClippingNode(裁剪节点)可以用来对节点进行裁剪，可以根据一个模板切割图片的节点，生成任何形状的节点显示。

ClippingNode是Node的子类，可以像普通节点一样放入Layer，Scene，Node中。

* ClippingNode 原理：

ClippingNode是利用模板遮罩来完成对Node区域裁剪的技术。如何理解ClippingNode的遮罩？看下图的例子吧。



所谓模板，就是一个形状，透过该形状可看到底板上的图层，如果底板上没有任何内容，则直接看到Layer上的内容，而底板上的东西又不会妨碍Layer上的东西，即模板在底板之外的空间对于Layer来说是透明的。【模板，等同于PhotoShop中的蒙版。就是可以挡住部分像素，露出下面图层的工具。】

ClippingNode（裁剪节点）可以用来对节点进行裁剪。ClippingNode是Node的子类，可以像普通节点一样放入Layer，Scene，Node中。

主要是根据一个模板（Stencil）切割图片的节点，生成任何形状的节点显示。

ClippingNode是利用模板遮罩来完成对Node区域裁剪的技术。注意使用该技术时，要将配置文件project.json中的renderMode字段，修改为1。

* ClippingNode使用步骤：
  + - * + 创建ClippingNode

var cliper= new cc.ClippingNode();

* + - * + 创建裁剪模板stencil

//模板可以是精灵

var stencil= new cc.Sprite(res.game\_title\_png);

//也可以是绘图

var stencil = new cc.DrawNode();

var triangle = [cc.p(-100, -100),cc.p(100, -100), cc.p(0, 100)];

var green = cc.color(0, 255, 0);

stencil.drawPoly(triangle, green, 3, green);

* + - * + 创建底板，并其添加到ClippingNode

底板和裁剪模板一样，可以是精灵，也可以是绘图文本等等。

var content=new cc.Sprite(res.game\_title\_png);

clipper.addChild(content,2,2);

* + - * + 将ClippingNode放入某个节点下，例如Layer

this.addChild(clipper,6,6);

* 另外两个重要方法：
  + - * + setInverted

可以使用void setInverted(bool inverted);方法，设置是显示被裁剪的部分，还是显示裁剪。true 显示剩余部分。false显示被剪掉部分。 如下：

clipper->setInverted(true);//设置底板可见，显示剩余部分

* + - * + setAlphaThreshold

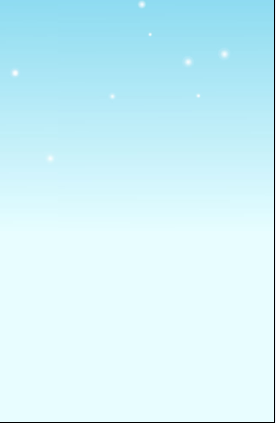
可以使用void setAlphaThreshold(GLfloat alphaThreshold);，设置alpha阈值， 只有模板（stencil）的alpha像素大于alpha阈值（alphaThreshold）时内容才会被绘制。 alpha阈值（threshold）范围应是0到1之间的浮点数。 alpha阈值（threshold）默认为1。 如下：

clipper->setAlphaThreshold(0);//设置绘制底板的Alpha值为0

* 所有资源：









所有资源从左到右、从上到下的规则，编号为1-6。其中4是背景，就不再每个实例中指明了。

* 实例一：标题闪亮特效
  + - * + 资源:1、2
        + 效果:



* + - * + 解析:

裁剪模板：1

底板：1、2

* + - * + 源码:

initLabel:function(){//标题闪亮特效

var stencil= new cc.Sprite(res.game\_title\_png);

//创建裁剪区域

var cliper= new cc.ClippingNode();

//设置模板

cliper.setStencil(stencil);

cliper.setAlphaThreshold(0);

cliper.setContentSize(stencil.getContentSize().width,stencil.getContentSize().height);

cliper.attr({

x: cc.winSize.width/2-80,

y: cc.winSize.height/8,

scale:0.5

});

//将ClippingNode添加到层节点中

this.addChild(cliper,1,1);

var content= new cc.Sprite(res.spark\_png);

cliper.addChild(content,2,2);

//背景

var label= new cc.Sprite(res.game\_title\_png);

cliper.addChild(label,1,1);

//虽然是同样的图片，但是前后顺序

//动作

var cliperSize= cliper.getContentSize();

var moveAction = cc.MoveTo.create(0.6, cc.p(cliperSize.width, 0));

var moveBackAction = cc.MoveTo.create(0.6, cc.p(-cliperSize.width, 0));

var seq = cc.Sequence.create(moveAction, moveBackAction);

var repeatAction = cc.RepeatForever.create(seq);

content.runAction(repeatAction);

},

* 实例二：三角形外的内容不显示
  + - * + 资源:1
        + 效果:



* + - * + 解析:

底板:1

裁剪模板:cc.DrawNode()

* + - * + 源码:

initTransparentTriangle:function(){//精灵旋转，三角形

var content=new cc.Sprite(res.game\_title\_png);

var stencil = new cc.DrawNode();

var triangle = [cc.p(-100, -100),cc.p(100, -100), cc.p(0, 100)];

var green = cc.color(0, 255, 0);

stencil.drawPoly(triangle, green, 3, green);

//this.addChild(stencil);

var clipper= new cc.ClippingNode();

clipper.setStencil(stencil);

clipper.attr({

x:cc.winSize.width/2-80,

y:cc.winSize.height/3+20,

scale:0.5

});

this.addChild(clipper,2,2);

clipper.addChild(content,2,2);

//rotate

var rotate= cc.rotateBy(1,90).repeatForever();

content.runAction(rotate);

},

* 实例三：裁剪模板外图案显示，内的不显示
  + - * + 说明:

原来想直接在实例二的基础上，设置setInverted(true),但是不知什么原因，使用DrawNode时没有裁剪效果，因此将裁剪模板改成了精灵。和实例二相比，仅仅只是添加设置了ClippingNode节点对象的inverted属性为true。

* + - * + 资源:1、5
        + 效果:



* + - * + 解析:

裁剪模板：5

底板：1

inverted:true

* + - * + 源码:

initInvertedTransparentTriangle:function(){//精灵旋转，三角形

var content=new cc.Sprite(res.game\_title\_png);

var stencil= new cc.Sprite(res.hole\_effect\_png);

stencil.scale = 2;

var clipper= new cc.ClippingNode();

clipper.setStencil(stencil);

clipper.setInverted(true);

clipper.attr({

x:cc.winSize.width/2-80,

y:cc.winSize.height\*0.65,

scale:0.5

});

this.addChild(clipper,6,6);

clipper.addChild(content,2,2);

//rotate

var rotate= cc.rotateBy(1,90).repeatForever();

stencil.runAction(rotate);

}

* 实例四：
  + - * + 资源:1
        + 效果:



* + - * + 解析:

底板:cc.DrawNode()

裁剪模板:1

* + - * + 源码:

initGreenTriangle:function(){//精灵旋转，三角形

//sprite

var stencil=new cc.Sprite(res.game\_title\_png);

var content= new cc.DrawNode();

var triangle = [cc.p(-100, -100),cc.p(100, -100), cc.p(0, 100)];

var green = cc.color(0, 255, 0);

content.drawPoly(triangle, green, 3, green);

var clipper= new cc.ClippingNode();

clipper.setStencil(stencil);

clipper.alphaThreshold = 0.05;

clipper.attr({

x:cc.winSize.width/2+80,

y:cc.winSize.height/3+20,

scale:0.5

});

this.addChild(clipper,3,3);

clipper.addChild(content,2,2);

//rotate

var rotate= cc.rotateBy(1,90).repeatForever();

content.runAction(rotate);

},

* 实例五：
  + - * + 说明：与上一实例，仅仅修改了ClippingNode节点的inverted属性。
        + 资源:1
        + 效果:



* + - * + 解析:

底板:cc.DrawNode()

裁剪模板:1

inverted:true

* + - * + 源码:

initInvertedGreenTriangle:function(){//精灵旋转，三角形

var stencil=new cc.Sprite(res.game\_title\_png);

var content= new cc.DrawNode();

var triangle = [cc.p(-100, -100),cc.p(100, -100), cc.p(0, 100)];

var green = cc.color(0, 255, 0);

content.drawPoly(triangle, green, 3, green);

var clipper= new cc.ClippingNode();

clipper.setStencil(stencil);

//仅仅比上一个多了这句话

clipper.setInverted(true);

clipper.alphaThreshold = 0.05;

clipper.attr({

x:cc.winSize.width/2+80,

y:cc.winSize.height\*0.63,

scale:0.5

});

this.addChild(clipper,3,3);

clipper.addChild(content,2,2);

//rotate

var rotate= cc.rotateBy(1,90).repeatForever();

content.runAction(rotate);

}

* 实例六：在一个矩形框中，拖动精灵，矩形外部分不显示
  + - * + 资源:3
        + 效果:



* + - * + 解析:

底板：3

裁剪模板:cc.DrawNode()

全局变量：

this.isMove//记录是否移动图片

\_lastPoint:cc.p(0,0),//图片最近移动的位置

监听机制：开启onTouchBegan、onTouchMoved和onTouchEnded

* + - * + 源码:

initMovePicture:function(){

//clipper

var clipper= new cc.ClippingNode();

clipper.attr({

anchorX:0.5,

anchorY:0.5,

x:cc.winSize.width/2,

y:cc.winSize.height\*0.9,

width:200,

height:200,

scale:0.5

});

this.addChild(clipper,5,5);

//content

var content= new cc.Sprite(res.meinv\_png);

content.attr({

x:clipper.width/2,

y:clipper.height/2

});

clipper.addChild(content,1,1);

//stencil

var stencil= new cc.DrawNode();

var rectangle= [

cc.p(0, 0),

cc.p(clipper.width, 0),

cc.p(clipper.width, clipper.height),

cc.p(0, clipper.height)

];

stencil.drawPoly(rectangle);

clipper.setStencil(stencil);

//旋转

var rotate= cc.rotateBy(1,90).repeatForever();

clipper.runAction(rotate);

},

onTouchBegan:function(touch, event){

var target = event.getCurrentTarget();

//移动图片

var clipper= target.getChildByTag(5);

var Rect = cc.rect(0, 0, clipper.width, clipper.height);

point= clipper.convertToNodeSpace(touch.getLocation());

if (cc.rectContainsPoint(Rect,point)){

target.\_isMove=true;

target.\_lastPoint=point;

return true;

}

return false;

},

onTouchMoved:function(touch,event){

var target = event.getCurrentTarget();

if(!target.\_isMove){

return;

}

var clipper= target.getChildByTag(5);

var content= clipper.getChildByTag(1);

//要移动的相对距离

var point = clipper.convertToNodeSpace(touch.getLocation());

var diff = cc.pSub(point, target.\_lastPoint);

content.setPosition(cc.pAdd(content.getPosition(), diff));

target.\_lastPoint=point;

},

onTouchEnded:function(touch,event){

var target = event.getCurrentTarget();

target.\_isMove=false;

return false;

}

* 实例七：官方的《打洞》效果
  + - * + 资源:1、5、6
        + 效果:



* + - * + 解析:

底板：1

裁剪模板:cc.ClippingNode()以底图宽和高绘制的矩形。

全局变量：

\_Clipper:null,//裁剪节点

\_holesStencil:null,//所有的模板

\_holes:null,//所有的弹痕

监听机制：开启onTouchBegan

注意：要加两个裁剪模板。

* + - * + 源码:

initHole:function(){

//content

var content = new cc.Sprite(res.game\_title\_png);

content.attr({

anchorX:0,

anchorY:0

});

//stencil——矩形

var stencil = new cc.DrawNode();

var rectangle = [

cc.p(0, 0),

cc.p(content.width\*content.scale, 0),

cc.p(content.width\*content.scale, content.height\*content.scale),

cc.p(0, content.height\*content.scale)

];

stencil.drawPoly(rectangle, cc.color(255, 0, 0, 255), 0, cc.color(255, 255, 255, 0));

//放射矩阵

var transform = cc.affineTransformMakeIdentity();

transform = cc.affineTransformScale(transform, content.scale, content.scale);

var ocsize = cc.sizeApplyAffineTransform(cc.size(content.width, content.height), transform);

//创建一个包含所有弹孔的节点

this.\_holesStencil = new cc.Node();

//创建一个包含所有弹痕的节点

this.\_holes=new cc.Node();

//裁剪节点

var holesClipper = new cc.ClippingNode();

holesClipper.addChild(content);

holesClipper.setInverted(true);

holesClipper.setAlphaThreshold(0.05);

holesClipper.setStencil(this.\_holesStencil);

holesClipper.addChild(this.\_holes);

//clipper

this.\_Clipper = new cc.ClippingNode();

this.\_Clipper.setStencil(stencil);

this.\_Clipper.attr({

width:ocsize.width,

height:ocsize.height,

anchorX:0.5,

anchorY:0.5,

x:this.width \* 0.5+80,

y:this.height/8,

scale:0.5

});

this.addChild(this.\_Clipper);

this.\_Clipper.addChild(holesClipper);

//Action

this.\_Clipper.runAction(cc.rotateBy(1, 45).repeatForever());

},

pokeHoleAtPoint:function (point) {

//弹痕

var hole = new cc.Sprite(res.hole\_effect\_png);

hole.attr({

x: point.x,

y: point.y

});

this.\_holes.addChild(hole);

//弹孔

var holeStencil = new cc.Sprite(res.hole\_stencil\_png);

holeStencil.attr({

x: point.x,

y: point.y

});

this.\_holesStencil.addChild(holeStencil);

//action

this.\_Clipper.runAction(cc.sequence(cc.scaleBy(0.05, 0.95), cc.scaleTo(0.125, 0.5)));

},

onTouchBegan:function(touch, event){

var target = event.getCurrentTarget();

//《打洞》

var rect = cc.rect(0, 0, target.\_Clipper.width, target.\_Clipper.height);

var point = target.\_Clipper.convertToNodeSpace(touch.getLocation());

if (cc.rectContainsPoint(rect,point)){

target.pokeHoleAtPoint(point);

return false;

}

},

* 实例八：文本滚动
  + - * + 资源:无
        + 效果:



* + - * + 解析:

裁剪模板:cc.DrawNode()

底板：cc.LabelTTF

* + - * + 源码:

initMoveLabel:function(){

var content= new cc.LabelTTF("","微软雅黑",20);

content.attr({

color:cc.color(255,0,0),

textAlign:cc.TEXT\_ALIGNMENT\_CENTER

});

var str="";

str+="womenhoadlfjlsadjfklskdhg\r\n"

+"kldfhlkg;jsad'gk;lfds\r\n"

+"hslkdf;gjlksdfhgklds\r\n"

+"flgjsdf;lkgjlkdsf\r\n";

content.setString(str);

//stencil

var stencil= new cc.DrawNode();

var rectangle=[

cc.p(0,0),

cc.p(0,20),

cc.p(200,20),

cc.p(200,0)

];

stencil.drawPoly(rectangle,cc.color(255,255,0),3,cc.color(255,255,0));

var clipper=new cc.ClippingNode();

clipper.setStencil(stencil);

clipper.setPosition(cc.p(cc.winSize.width/2+10,cc.winSize.height\*0.8));

clipper.addChild(content,1,1);

this.addChild(clipper,7,7);

//上移

var moveUP=cc.moveBy(0.01,cc.p(0,1)).repeatForever();

content.runAction(moveUP);

},

* 实例九：
  + - * + 资源:3
        + 效果:



* + - * + 解析:

底板：cc.LabelTTF

裁剪模板：3

* + - * + 源码:

initCircleLabel:function(){

var content= new cc.LabelTTF("","微软雅黑",20);

content.attr({

color:cc.color(255,0,0),

textAlign:cc.TEXT\_ALIGNMENT\_LEFT

});

var str="";

str+="womenhoadlfjlsadjfklskdhg\r\n";

content.setString(str);

var stencil= new cc.Sprite(res.meinv\_png);

var clipper=new cc.ClippingNode();

clipper.setStencil(stencil);

clipper.attr({

x:cc.winSize.width/2+10,

y:cc.winSize.height\*0.8,

width:200,

height:60

});

stencil.attr({

x:clipper.width/2,

y:clipper.height/2-20,

scale:0.2

});

clipper.addChild(content,1,1);

this.addChild(clipper,7,7);

//上移

var moveLeft=cc.moveBy(0.01,cc.p(-2,0)).repeatForever();

stencil.runAction(moveLeft);

},

* 扩展【人脸化妆、文字上贴图、进度条】

1. 系统属性
   * 定义了一个系统属性变量

cc.sys = {};

var sys = cc.sys;

* + 设备可支持的性能【cc.sys.capabilities】，取值范围：

cc.sys.capabilities =[

"canvas",//画布

"opengl", //OpenGL

"touches", //触摸

"mouse", //鼠标

"keyboard",//键盘

"accelerometer" //加速计

]

实例：

1. 遍历设备可支持的性能：

var capabilities= cc.sys.capabilities;

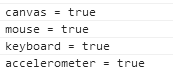
for(var i in capabilities){

//console中输出

console.log(i + " = "+ capabilities[i]);

}

结果：



1. 检测指定的性能，是否在该设备中支持：

if("mouse" in cc.sys.capabilities){

console.log("成功！")

}

* + 系统语言【cc.sys.language】

sys.LANGUAGE\_ENGLISH = "en";

sys.LANGUAGE\_CHINESE = "zh";

sys.LANGUAGE\_FRENCH = "fr";

sys.LANGUAGE\_ITALIAN = "it";

sys.LANGUAGE\_GERMAN = "de";

sys.LANGUAGE\_SPANISH = "es";

sys.LANGUAGE\_DUTCH = "du";

sys.LANGUAGE\_RUSSIAN = "ru";

sys.LANGUAGE\_KOREAN = "ko";

sys.LANGUAGE\_JAPANESE = "ja";

sys.LANGUAGE\_HUNGARIAN = "hu";

sys.LANGUAGE\_PORTUGUESE = "pt";

sys.LANGUAGE\_ARABIC = "ar";

sys.LANGUAGE\_NORWEGIAN = "no";

sys.LANGUAGE\_POLISH = "pl";

实例：

var language= "";

switch(cc.sys.language){

case cc.sys.LANGUAGE\_CHINESE:

language= "简体中文";

break;

case cc.sys.LANGUAGE\_ENGLISH:

language= "英文";

break;

}

alert(language);

* + 操作系统【cc.sys.os】

sys.OS\_WINDOWS = "Windows";

sys.OS\_IOS = "iOS";

sys.OS\_OSX = "OS X";

sys.OS\_UNIX = "UNIX";

sys.OS\_LINUX = "Linux";

sys.OS\_ANDROID = "Android";

sys.OS\_UNKNOWN = "Unknown";

【判定接口cc.sys.os== cc.sys.OS\_ANDROID】

* + 平台【cc.sys.platform】

sys.WINDOWS = 0;

sys.LINUX = 1;

sys.MACOS = 2;

sys.ANDROID = 3;

sys.IPHONE = 4;

sys.IPAD = 5;

sys.BLACKBERRY = 6;

sys.NACL = 7;

sys.EMSCRIPTEN = 8;

sys.TIZEN = 9;

sys.WINRT = 10;

sys.WP8 = 11;

* + 浏览器类型【cc.sys.isMobile】

sys.MOBILE\_BROWSER = 100;

sys.DESKTOP\_BROWSER = 101;

* + 浏览器种类【cc.sys.browserType】

sys.BROWSER\_TYPE\_WECHAT = "wechat";

sys.BROWSER\_TYPE\_ANDROID = "androidbrowser";

sys.BROWSER\_TYPE\_IE = "ie";

sys.BROWSER\_TYPE\_QQ = "qqbrowser";

sys.BROWSER\_TYPE\_MOBILE\_QQ = "mqqbrowser";

sys.BROWSER\_TYPE\_UC = "ucbrowser";

sys.BROWSER\_TYPE\_360 = "360browser";

sys.BROWSER\_TYPE\_BAIDU\_APP = "baiduboxapp";

sys.BROWSER\_TYPE\_BAIDU = "baidubrowser";

sys.BROWSER\_TYPE\_MAXTHON = "maxthon";

sys.BROWSER\_TYPE\_OPERA = "opera";

sys.BROWSER\_TYPE\_OUPENG = "oupeng";

sys.BROWSER\_TYPE\_MIUI = "miuibrowser";

sys.BROWSER\_TYPE\_FIREFOX = "firefox";

sys.BROWSER\_TYPE\_SAFARI = "safari";

sys.BROWSER\_TYPE\_CHROME = "chrome";

sys.BROWSER\_TYPE\_UNKNOWN = "unknown";

1. JS 字符/ASCII码转换
   * 字符🡪ASCII码：charCodeAt();

var str= "A";

var code= str.charCodeAt();//字符-->ASCII值

* + ASCII码🡪字符：fromCharCode();

String.fromCharCode(code);//ASCII值-->字符

1. 显示或者隐藏FPS

true：显示 false：隐藏

* + 方法一：

cc.director.setDisplayStats(true);

* + 方法二：

在project.json文件中，修改showFPS的值：

"showFPS" : false,

1. Cocos2d-JS中，获取系统的时间

//利用JS函数

var date = new Date();

alert(date.getHours());

1. "无法找到“XXX.exe”的调试信息，或者调试信息不匹配。未使用调试信息生成二进制文件"

环境：

Debug模式，运行时完全正常，但是一调试就出现对话框，显示出错信息：

“无法找到“XXX.exe”的调试信息，或者调试信息不匹配。未使用调试信息生成二进制文件。”

解决：

首先打开菜单 项目->项目属性页🡪配置属性->链接器->调试->生成调试信息 改为 是

1. XP系统下，下载资源，避免输入登录信息、扣积分
   1. 右击IE图标🡪属性🡪浏览历史记录🡪设置🡪查看文件🡪删除文件夹下的所有文件；
   2. 打开IE，并输入该资源路径信息，打开网页；
   3. 在打开的文件夹下，查找你要的资源文件即可。
2. 定时器

Cocos2d中，一共有三种定时器：schedule，scheduleUpdate，scheduleOnce。

注意：

定时器的回调函数的执行时间，不是固定的秒数，而是与屏幕的刷新频率frameRate、FPS有关。当回调函数的执行时间小于屏幕的刷新频率，以屏幕的刷新频率为准。否则以回调函数的执行时间为准。

说明：

前两种定时器的回调函数的参数delta,是指两帧之间的刷新时间间隔。

* schedule(callback\_fn, interval, repeat, delay)

功能：隔几秒就执行某个自定义的函数。

参数说明：

* callback\_fn:回调函数
* interval：间隔，单位/s
* repeat:重复次数
* delay：延时
  + - 开启定时器

两种方式

* + - * 参数二是执行时间间隔

this.schedule(this.onAbout,0.02);

* + - * 当schedule没有添加参数二时，相当于scheduleUpdate，每帧都会执行。

this.schedule(this.onAbout);

* + - 回调函数

onAbout:function(delta){

console.log(delta);

}

* scheduleUpdate

功能：前节点后，程序会每帧都会自动执行一次默认的update函数。【注意：回调函数一个是update】

* + - 开启定时器

this.scheduleUpdate();

* + - 回调函数

update:function(delta){

console.log(delta);

}

* scheduleOnce

功能：在几秒之后执行，并且只执行一次。

* + - 开启定时器

scheduleOnce(this.onAbout, 1.0f); //在1.0f之后执行，并且只执行一次。

* + - 回调函数

onAbout:function(delta){

console.log(delta);

}

* 停止定时器
  + 停止自定义的定时器

this.unschedule(this.OnAbout);

* + 停止默认定时器

this.unscheduleUpdate();

* + 停止所有定时器

this.unscheduleAllSelectors();

注意：在开启定时器和注销定时器时，都不需要在回调函数的后面，加上小括号。

1. Cocos2d-JS中，类和对象的使用

* 自定义一个类,在CPlayer.js定义

var CPlayer = cc.Sprite.extend({

\_texture:null,

ctor:function(){

this.\_super();

},

SetTexture:function(texture){

this.\_texture = texture;

this.initWithTexture(texture);

}

});

* 在project.json中，添加JS列表：

"src/Classes/Player.js"

* 在对应的层中，创建精灵初始化函数：

initPlayer:function(pos){

//create player

var player = new CPlayer();

//texture

var texture = cc.textureCache.addImage("res/cat.png");

player.SetTexture(texture);

//attribute

player.attr({

x:pos.x,

y:pos.y,

scale:1

});

this.addChild(player,2);

return player;

},

* 创建对象

for(var i =0;i<5;i++){

var pos = cc.p(cc.winSize.width/2+20\*(-1\*i),cc.winSize.height/2+20\*(-1\*i));

var player = this.initPlayer(pos);

this.removeObj.push(player);

}

1. OpenGL绘图

OpenGL ES是OpenGL三维图形API的子集，专门针对移动设备设计，其1.0版本是针对固定管线硬件的，而2.0版本已经扩展到支持可编程管线硬件。Cocos2d-x 2.0正是将底层绘图从OpenGL ES 1.0升级到了OpenGL ES 2.0。

分段数：即绘制曲线一般都是通过绘制“样条曲线”来实现，而分段数即样条段数。

绘图的方式有两种：

* 使用OpenGL的绘图原语DrawPrimitives。

使用DrawPrimitives绘图原语绘制的图形，可以是实心的，也可以是空心的。

* 使用DrawNode。

使用DrawNode绘制的图形都是实心的。【其中该方法，也可以绘制空心的，只要将对应的填充颜色设置为null即可。】

* 方法一：使用DrawNode绘图
* 创建DrawNode对象,并将其添加到对应的父节点中。

var draw = new cc.DrawNode();

this.addChild(draw,10);

* 清理以前使用drawNode画的所有图形【原理：将缓存数组的长度改为0】:

draw.clear();

* 单个的点

drawDot(pos, radius, color)

功能:画点，半径:radius 颜色:color

参数说明：缺省值：radius—0 color—cc.color(255,255,255,255)

* + 示例：

draw.drawDot(cc.p(cc.winSize.width/2,cc.winSize.height/2));

* 多个相同的点

drawDots(points, radius, color)

功能: 画几个相同的点。

参数说明：points是几个点的组成的数组，其他参数同上。

* + 示例：

var points = [cc.p(60, 60), cc.p(70, 70), cc.p(60, 70), cc.p(70, 60)];

draw.drawDots(points,5);

* 线段

drawSegment(from, to, lineWidth, color)

功能:画线段，线宽:lineWidth 颜色:color

参数说明：缺省值：lineWidth—1 color—cc.color(255,255,255,255)

注意：当线粗到一定程序之后，就会发现：线段的一端是椭圆形的【想法:可不可以利用这个来做一个进度条】。

* + 示例：

draw.drawSegment(cc.p(0,0),cc.p(cc.winSize.width-200,cc.winSize.height-200),2);

* 填充矩形

drawRect(origin, destination, fillColor, lineWidth, lineColor)

参数说明：

\* @param origin—左下角点

\* @param destination—右上角点

\* @param fillColor—填充颜色 null:不填充

\* @param lineWidth—线宽

\* @param lineColor—矩形边框颜色

* + 示例：

draw.drawRect(cc.p(100,0),cc.p(200,100),cc.color(255,255,0),2,cc.color(255,0,0));

* 非填充矩形

draw.drawRect(cc.p(0,0),cc.p(100,100),null,2,cc.color(255,0,0));

* 多边形—三角形

绘制三个顶点的多边形，既是三角形。

drawPoly(verts, fillColor, lineWidth, color)

参数说明：

\* @param verts—多边形顶点数组

\* @param fillColor—填充颜色 null:不填充

\* @param lineWidth—线宽

\* @param color—矩形边框颜色

* + 示例：

//filled

var trianglePoints1 = [ cc.p(0,150), cc.p(300,150), cc.p(150,210) ];

draw.drawPoly(trianglePoints1, cc.color(255,255,0), 2, cc.color(0,255,0,255));

//not filled

var trianglePoints = [ cc.p(0,270), cc.p(300,270), cc.p(150,210) ];

draw.drawPoly(trianglePoints, null, 2, cc.color(0,0,0,255));

* 多边形—星星

绘制4个顶点的多边形

* + 示例：

// star poly (triggers bugs)

var o=80;

var w=20;

var h=50;

var star = [

cc.p(o+w,o-h), cc.p(o+w\*2, o), // lower spike

cc.p(o + w\*2 + h, o+w ), cc.p(o + w\*2, o+w\*2), // right spike

cc.p(o +w, o+w\*2+h), cc.p(o,o+w\*2), // top spike

cc.p(o -h, o+w), cc.p(o,o) // left spike

];

draw.drawPoly(star, cc.color(255,0,0,255), 2, cc.color(0,0,255,255) );

// star poly (doesn't trigger bug... order is important un tesselation is supported.

o=180;

w=20;

h=50;

star = [

cc.p(o,o), cc.p(o+w,o-h), cc.p(o+w\*2, o), // lower spike

cc.p(o + w\*2 + h, o+w ), cc.p(o + w\*2, o+w\*2), // right spike

cc.p(o +w, o+w\*2+h), cc.p(o,o+w\*2), // top spike

cc.p(o -h, o+w) // left spike

];

draw.drawPoly(star, cc.color(255,0,0,255), 2, cc.color(0,0,255,255) );

* 空心圆

注意:计算机实际上,并不能直接绘制圆形，而是后台通过绘制多边形drawPoly的方法，近似模拟圆形。

了解了计算机绘制圆形的原理后，也就可以理解：为什么第四个参数太小时，圆就变成多边形了。

drawCircle(center, radius, angle, segments, drawLineToCenter, lineWidth, color)

参数说明：

\* @param center—圆点

\* @param radius—半径

\* @param angle—每一条线段的旋转角度 正值向上偏移

\* @param segments—圆分几段 近似圆形 值为36

\* @param drawLineToCenter—是否从第一条线段到圆形,画一条线段。angle参数影响较大。

\* @param lineWidth—线宽

\* @param color—颜色

* + 示例：

draw.drawCircle(cc.p(cc.winSize.width / 2, cc.winSize.height / 2), 100, 0, 36, false, 6, cc.color(0, 255, 0, 255));

* 实心圆

使用drawCircle函数绘制的圆形是空心的，而cc.DrawNode类中，又没有封装好的实心圆的绘制函数。但是drawCircle()函数的本质是：利用drawPoly函数和分段数来近似模拟圆形，因此开发者可以自己计算分段数、填充颜色和drawPoly函数来绘制实心圆。

* + 示例：

//顶点数

var segments = 36;

//圆心半径

var radius = 50;

//圆心

var center = cc.p(cc.winSize.width/2,cc.winSize.height/2);

//顶点数组

var vertices = [];

//偏转弧度

var coefStep = cc.PI2/ segments;

for(var i= 0;i< segments;++i){

var rad = i\* coefStep;

var x = radius \* Math.cos(rad)+ center.x;

var y = radius \* Math.sin(rad)+ center.y;

vertices.push(cc.p(x,y));

}

draw.drawPoly(vertices,cc.color(255,0,0),2,cc.color(0,0,255));

* 二次贝塞尔曲线

drawQuadBezier(origin, control, destination, segments, lineWidth, color)

功能：二次贝塞尔曲线：起点终点之间的一条抛物线，利用一个控制点来控制抛物线的形状。

参数说明:

\* @param origin—曲线的起始点

\* @param control—曲线的控制点

\* @param destination—曲线的终点

* + 示例：

draw.drawQuadBezier(cc.p(0,cc.winSize.height),cc.p(cc.winSize.width/2,0),cc.p(cc.winSize.width,0),50,3);

* 圆弧
  + 示例：

draw.drawQuadBezier(cc.p(50,0),cc.p(50,50),cc.p(0,50),50,5,cc.color(255,255,0,255));

* 三次贝塞尔曲线

drawCubicBezier(origin, control1, control2, destination, segments, lineWidth, color)

功能：三次次贝塞尔曲线：起点终点之间，利用两个控制点来控制曲线的形状。

参数说明:

\* @param origin—曲线的起始点

\* @param control1—曲线的控制点1

\* @param control2—曲线的控制点2

\* @param destination—曲线的终点

* + 示例：

draw.drawCubicBezier(cc.p(cc.winSize.width / 2, cc.winSize.height / 2), cc.p(cc.winSize.width / 2 + 30, cc.winSize.height / 2 + 50),

cc.p(cc.winSize.width / 2 + 60, cc.winSize.height / 2 - 50), cc.p(cc.winSize.width, cc.winSize.height / 2), 100, 2, cc.color(255, 0, 255, 255));

* 基本的样条曲线

drawCardinalSpline(config, tension, segments, lineWidth, color)

功能:绘制一个基本的样条曲线

参数说明：

\* @param config—点数组【5个点—四个顶点，最后一个是回到起点】

\* @param tension—张力

* + 示例：

var centerPos = cc.p(cc.winSize.width/2,cc.winSize.height/2);

var vertices4 = [

cc.p(centerPos.x - 130, centerPos.y - 130),

cc.p(centerPos.x - 130, centerPos.y + 130),

cc.p(centerPos.x + 130, centerPos.y + 130),

cc.p(centerPos.x + 130, centerPos.y - 130),

cc.p(centerPos.x - 130, centerPos.y - 130)

];

draw.drawCardinalSpline(vertices4, 0.5, 100, 2, cc.color(255, 255, 255, 255));

* CatmullRom样条曲线

drawCatmullRom(points, segments, lineWidth, color)

功能:绘制一个CatmullRom样条曲线

本质：就是使用drawCardinalSpline函数绘制

参数说明：

\* @param points—点数组【5个点—四个顶点，最后一个是回到起点】

* + 示例：

vertices4 = [

cc.p(centerPos.x - 100, centerPos.y - 100),

cc.p(centerPos.x - 100, centerPos.y + 100),

cc.p(centerPos.x + 100, centerPos.y + 100),

cc.p(centerPos.x + 100, centerPos.y - 100),

cc.p(centerPos.x - 100, centerPos.y - 100)

];

draw.drawCatmullRom(vertices4, 100, 2, cc.color(255, 255, 0, 255));

* + 方法二：使用绘图原语DrawPrimitives绘图

要使用绘图原语DrawPrimitives来绘制几何图形时，必须要重载CCNode的函数draw()。但是在JS版本中，重载draw函数后，并没有响应，所以并不在此多介绍。【C++版本中，不知道可不可以使用。】

注意：绘制曲线一般都是通过绘制“样条曲线”来实现，而分段数即样条段数。圆形也是如此。计算机不能直接绘制圆形，就是通过分段数绘制的。

* 涂鸦
* 为对应的类，添加几个全局变量：

draw:null,

prePos:cc.p(0,0),

curPos:cc.p(0,0),

* 添加监听机制：

this.addListener();

addListener:function(){

var self = this;

cc.eventManager.addListener({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches:true,

onTouchBegan:self.onTouchBegan,

onTouchMoved:self.onTouchMoved

},this);

},

* 绘图：

onTouchBegan:function(touch,event){

var target = event.getCurrentTarget();

target.prePos = target.curPos = touch.getLocation();

return true;

},

onTouchMoved:function(touch,event){

var target = event.getCurrentTarget();

target.curPos = touch.getLocation();

var step = Math.sqrt((target.curPos.x-target.prePos.x)\*(target.curPos.x-target.prePos.x) + (target.curPos.x-target.prePos.x)\*(target.curPos.x-target.prePos.x));

//当前后两次坐标间隔大于5时，才会绘制新的线段。

if(step>5){

target.draw.drawSegment(target.prePos,target.curPos,2,cc.color(255,255,0));

target.prePos = target.curPos;

}

}

1. <canvas>标签

这个 HTML 元素是为了客户端矢量图形而设计的。它自己没有行为，但却把一个绘图 API 展现给客户端 JavaScript 以使脚本能够把想绘制的东西都绘制到一块画布上。

可以通过HTML5的<canvas>元素来使用JavaScript绘制形状和推行、形状和Web页面上创建动画，而这些并不需要通过使用其他语言(例如SVG)或插件(如Flash)来实现。

<canvas>元素支持脚本，这意味着可以创建一个根据用户输入而变化的canvas。所有的绘图都可以在<canvas>元素中完成，但使用<canvas>元素来为页面嵌入所有图像或添加页面的header都是不适合的，应当采用<img>和<header>标签来代替<canvas>元素。但可以用canvas显示动态图形或创建在线游戏。将<canvas>元素添加至HTML中HTML文档的方法很简单:

<canvas></canvas>

此代码在浏览器中创建了一个空白的canvas画布，但它不包含宽度、高度或内容，也不会在屏幕上有任何显示。大部分情况下需要为canvas注明宽度及高度，以及一个用来在脚本进行引用的ID。

<canvas id="gameCanvas" width="320" height="480"></canvas>

上述代码会在HTML上生成一个320\*480像素的空白空间。

注意：

在Cocos2d-JS的初始化进程中，引擎会自动将你的Canvas元素放置到一个DIV容器Cocos2dGameContainer中【在Chrome浏览器中，开发者工具🡪Elements中就可以看见。】，而这个容器会被加入到Canvas的原始父节点（游戏外框）中。这个游戏容器是实现屏幕适配方案的重要辅助元素，你可以通过cc.container来访问它。

注意：让空白canvas可见。

在第一次使用<canvas>元素时，可以为它加上一个边框，这是能看见它的位置的最简单的方法。在样式表中<style>加入一行——canvas{border:solid thin black;},这样为页面上的所有canvas加入一个细边框。编辑完成后，可以将样式删除。

<canvas> 标记由 Apple 在 Safari 1.3 Web 浏览器中引入。对 HTML 的这一根本扩展的原因在于，HTML 在 Safari 中的绘图能力也为 Mac OS X 桌面的 Dashboard 组件所使用，并且 Apple 希望有一种方式在 Dashboard 中支持脚本化的图形。

Firefox 1.5 和 Opera 9 都跟随了 Safari 的引领。这两个浏览器都支持 <canvas> 标记。

我们甚至可以在 IE 中使用 <canvas> 标记，并在 IE 的 VML 支持的基础上用开源的 JavaScript 代码（由 Google 发起）来构建兼容性的画布。

要从同一图形的一个 <canvas> 标记中移除元素，往往需要擦掉绘图重新绘制它。

* + 属性

属性 值 描述

height pixels 设置 canvas 的高度。

width pixels 设置 canvas 的宽度。

* + <canvas> 标记和 SVG 以及 VML 之间的差异

<canvas> 标记和 SVG 以及 VML 之间的一个重要的不同是，<canvas> 有一个基于 JavaScript 的绘图 API，而 SVG 和 VML 使用一个 XML 文档来描述绘图。

这两种方式在功能上是等同的，任何一种都可以用另一种来模拟。从表面上看，它们很不相同，可是，每一种都有强项和弱点。例如，SVG 绘图很容易编辑，只要从其描述中移除元素就行。

* + 如何使用 <canvas> 标记绘图

大多数 Canvas 绘图 API 都没有定义在 <canvas> 元素本身上，而是定义在通过画布的 getContext() 方法获得的一个“绘图环境”对象上。

Canvas API 也使用了路径的表示法。但是，路径由一系列的方法调用来定义，而不是描述为字母和数字的字符串，比如调用 beginPath() 和 arc() 方法。

一旦定义了路径，其他的方法，如 fill()，都是对此路径操作。绘图环境的各种属性，比如 fillStyle，说明了这些操作如何使用。、

* + 定义和用法

<canvas> 标签定义图形，比如图表和其他图像。

<canvas> 标签只是图形容器，您必须使用脚本来绘制图形。

* + 实例

要让canvas显示出内容，需要用JavaScript为其编写脚本。

* + - * 添加一个画布:

<canvas id="gameCanvas" width="320" height="480"></canvas>

* + - * 设置<canvas>元素的背景色：

//方法一：<header>标签中，添加：

<style>

#gameCanvas {

background: #000000;

}

</style>

//方法二：创建<canvas>元素时，指定：

<canvas id="gameCanvas" width="320" height="480" style="background: #FFFF00"></canvas>

* + - * 如何通过 canvas 元素来显示一个红色的矩形：

<script type="text/javascript">

//1、在文档中，找到ID为“gameCanvas”元素

var canvas=document.getElementById('gameCanvas');

//2、将环境设为二维

var ctx=canvas.getContext('2d');

//3、定义填充颜色为蓝色

ctx.fillStyle='#FF0000';

//4、绘出矩形

ctx.fillRect(0,0,80,100);

</script>

警告：必须设置content环境。

在<canvas>元素上惊醒绘制时，必须将字符串“2d”传入getCentext()方法中。否则，该<canvas>元素将什么都不显示。

* + - * 动态指定<canvas>元素的尺寸

//方法一：JQeury【此方法好像有问题】

var canvas =$("#gameCanvas");

canvas.width = 100;//画布的宽度改为100

//方法二：HTML5

var canvas = document.getElementById("gameCanvas");;

canvas.width = 100;//画布的宽度改为100

注意：document.getElementById("gameCanvas").width=100;//错误

应该写成：document.getElementById("gameCanvas").style.width=100;

* + - * 动态显示或隐藏<canvas>元素【display的值：block——显示 none——隐藏】

//方法一：JQeury

var canvas= $("#gameCanvas");

canvas.css("display","none");

//方法二：HTML5

var canvas= document.getElementById("gameCanvas");

canvas.setAttribute("display","none");

注意：

Canvas API 非常紧凑的一个原因上它没有对绘制文本提供任何支持。要把文本加入到一个 <canvas> 图形，必须要么自己绘制它再用位图图像合并它，或者在 <canvas> 上方使用 CSS 定位来覆盖 HTML 文本。

JS中，定义<canvas>标签

<canvas id="gameCanvas" width="320" height="480"></canvas>

在HTML5中，动态执行Cocos绘图:cc.game.run("gameCanvas");

在HTML5中，隐藏Cocos画布:

cc.director.pause();

cc.director.popScene();

//方法一：使用Jquery方法获取对应的<canvas>标签，并隐藏

var div = $("#gameCanvas ");//ID

div.css("display","none");//隐藏

//方法二：使用Html5方法获取对应的<canvas>标签，并隐藏

var div = document.getElementById("gameCanvas");;//ID

div.setAttribute("display:none");//隐藏

1. window.onload函数的作用：

当你的整个页面加载完毕的时候，会调用这个onload，然后你里面可以写一些函数进行一些操作。 但是它里面只能写一个函数。而且整个页面有且仅有一个该函数。

当加载网页文档是，会产生该事件。onLoad事件的作用是在首次载入一个页面文件时，检测cookie的值，并用一个变量为其赋值，使其可以被源代码使用。

发生在文档全部下载完毕的时候，全部下载完毕意味着不但HTML文件，而且包含的图片，插件，控件，小程序等全部内容都下载完毕。本事件是window的事件，但是在Html中指定时间处理程序的时候，我们是把它写在<body>标记中的。

1. HTML5的Web存储

在HTML5之前，人们只能通过将信息添加至DOM或使用Cookie来存储信息。但这两种方式都存在缺陷:

* + - * 用HTML保存数据，即DOM,倒是十分简单。但需要借助JavaScript,且数据只能保存于打开的浏览器或应用程序中。一旦用户终止session(会话)，数据将会丢失。另外，存储在HTML中的数据是任何人都可以看见的。
      * Cookie是一段限制在4KB左右的数据，这个大小存储的信息量较少。另外，Cookie会随着每个HTTP请求发送，从而降低应用程序的速度。Cookie没有加密，因此这样存储的数据有可能被嗅探器(Sniffer)获取，除非为整个应用程序使用SSL(Secure Sockets Layers，安全套接层)，但SSL也会降低页面速度。

HTML5本地存储可以提供给用户更多存储空间，并且只会在应用程序请求时，才与服务器进行传输。

Web存储设计的不仅仅是本地存储。还可以通过两种方式来创建本地数据库并在其中存储信息:Web SQL和Indexed Database(又称为Indexed DB)。

* Web存储

Web存储原本是属于HTML5规范的一个独立API，而如今已成为HTML5范围所支持的独立文档。一些浏览器制造商有时称它为本地存储(Local Storage)或DOM存储(DOM Storage)。

注意：一些插件也可以创建持久存储。

Web存储让开发者可以将Web应用程序中的数据作为键/值【key/value】对保存在本地计算机中。数据在用户离开程序、关闭浏览器或关闭电脑后依然保留。当用户重新开始使用该Web应用程序时，该应用程序可以继续使用Web存储中的数据。

数据存储能许多Web应用程序提供帮助。

* + 使用Web存储的一个简单的应用就是保存用户的访问位置，便于他们下次接着访问。
  + 使用Web存储的另一种方式是在用户离线时，将通常保存在服务器的数据保存到本地中。
* Web存储于Cookie的不同之处

许多人错误地认为Web存储和Cookie相同，只是在尺寸上更大。尽管Web存储的确提供比Cookie更多的空间，但它们还有一些的差别。

* Web存储只在客户端提出请求时才将数据发送给页面，而Cookie则是随每个HTTP请求发送。
* Web存储只能通过客户端(JavaScript)访问，而Cookie还能通过PHP这类服务器脚本访问。
* Cookie对应浏览器及域名，但不会区分同一浏览器下的不同窗口；而Web存储(特别是会话存储)会区分窗口、浏览器和域名。

Web存储并不仅仅是更高版本的Cookie。Cookie作用于某个用途后，必须一直使用它。例如，Cookie使用两个标志位——HttpOnly和secure——来提高安全性。HttpOnly通知浏览器此Cookie不能用于客户端脚本，而secure要求Cookie在安全(SSL)连接下使用。

如果要将会话数据保存到Cookie中，你至少得把Cookie设置为HttppOnly和secure——否则黑客可以通过会话信息得到你的用户信息。

最好将Web存储当成存储应用程序配置、非敏感用户信息、游戏存档和内容存档等信息的保存场所。Web存储的支持并不如Cookie广泛，后者自1995年便获得了浏览器支持。Web存储虽然被所有新款浏览器支持，但也仅限以下特定版本。【Andriod 2.0以上，Chrome 4以上，Firefox 3.5以上， IE8.0以上，ios 2.0以上，Opera 10.5以上,Safari 4.0以上】

* 会话存储和本地存储

Web存储规范中支持的两种存储类型分别为本地存储和会话存储。

本地存储指的是存储在本地磁盘中的键/值对，每个网站都拥有独立的存储区域，存储在该区域的数据即使在用户离开网站或关闭浏览器后也可以保存。数据仅在其指定的域名下可用，并仅在域名发出请求时发送数据。

本地存储最适合用来保存占用很多空间的非敏感数据。它针对需要跨会话保存的数据，这样用户回到网站时可以使用这些数据。

注意：Web存储将所有数据保存为字符串。

在Web存储中数据被保存为字符串。这意味着如果使用的是结构性数据(如数据库信息)或数字(整数或浮点)，需要将它们转化为字符串，以便保存在Web存储中。可以在取回数后使用parseInt()和parseFloat()方法将它还原成整数或浮点。

会话存储将页面会话以键/值对的形式作为一个项目保存到本地磁盘。Session存储仅在保存数据时相同的窗口或域名请求时，才会传递所保存的数据。在窗口关闭后(非特定情况下)，保存的数据将被删除。若同一网站打开新的标签页或窗口，将会创建一个新的会话存储实例。

会话储存被用于解决Cookie存在的问题。如果访问的网站把数据保存在Cookie中，当在第二个窗口打开该网站时，新窗口也可以访问之前保存的Cookie数据，甚至污染它。

会话存储非常适合用来保存临时数据，并防止浏览器意外刷新。跟本地存储一样，它不适合用来保存敏感数据。

注意：会话存储可在重载后保留。

如果浏览器支持重载后恢复会话，那么会话存储也可以保留下来。在这些浏览器中，存储在会话中的数据也可以一并恢复。

不同浏览器无法共享localStorage或sessionStorage中的信息。相同浏览器的不同页面间可以共享相同的localStorage（页面属于相同域名和端口），但是不同页面或标签页间无法共享sessionStorage的信息。这里需要注意的是，页面及标签页仅指顶级窗口，如果一个标签页包含多个iframe标签且他们属于同源页面，那么他们之间是可以共享sessionStorage的。

* 使用Web存储

与所有的HTML5新特性一样，在使用Web存储前需要先确认浏览器是否支持。

if(window.localStorage){

alert('This browser supports localStorage');

}else{

alert('This browser does NOT support localStorage');

}

使用Web存储时，要做的第一件事是定义存储对象。这将告诉用户代理使用的是本地存储还是会话存储。这两个对象分别如下：

1. window.localStorage
2. window.sessionStorage

* Html5中

Web存储接口的5个方法，如下所示：

* + - getItem(key)——获取key所定义的条目(item)的值；
    - setItem(key,value)——创建一个key所定义的条目，并将值设定为value;
    - removeItem(key)——从存储中移除有key所定义的条目
    - clear()——从存储中完全清除当前对象的所有条目
    - key(n)——返回存储区域中第n个存储条目的键名。

存储对象还有一个length属性，调用该属性将返回对象所有键/值对的数目。

警告：Web存储会被hack

如果网站没有采取措施防御跨站脚本(XSS),那么Web存储和Cookie一样都会遭到hack。本地存储所有条目的名称都被保存在JavaScript的一个清晰文本中，因此黑客可以很容易地通过XSS获取它。要保护用户，应该在所有页面头部设置<meta charset>元素——即在页面中出现任何文本之前，不要用Web存储保存会话ID——如果有这个需要，宁可选择会话存储。另外，也不要在本地存储中保存敏感数据。

跨站脚本攻击是利用字符集漏洞。但设置为GBK、GB2312这些字符集照样存在跨域攻击漏洞。因此对于中文网站而言，安全的方式是使用UTF-8。

如果调用setItem()指定了一个已经存在的键，那么其原有值将被新值覆盖。如果调用getItem()指定了一个不存在的值，将会返回null。

Web存储API也提供了一个事件，用于跟踪存储区域变化。该事件会创建一个StorageEvent对象，并拥有以下属性:

* key——指定的键（的内容）被添加、移除或修改。
* oldValue——条目的上一个值(包括空值)。
* newValue——条目的新值（包括空值，即如果条目被移除）
* url——调用函数并触发变动的页面。

关于存储事件还需要知道的最后一件事是：该事件不可取消。它仅仅是一个告诉你浏览器中正在做什么事情。

* Cocos 2D-JS中

在Cocos 2D-JS中，没有会话存储，只有本地存储。而且该对象好像只有三个方法：

* setItem(key,value)
* getItem(key)
* removeItem(key);
  + 定义对象：var ls = cc.sys.localStorage;
  + 添加键/值对:

var key = 'key\_' + Math.random();

ls.setItem(key, "Hello world");

* + 获取键/值对：

var r = ls.getItem(key);

* + 移除键值：

ls.removeItem(key);

* Web SQL
* Indexed DB

1. Bake

* 设计意图

在游戏开发的过程中，经常会遇到作为UI或者不怎么修改的背景的层(Layer)， 这些层内容并不怎么变动。 而在游戏的渲染过程中，这些层往往又会消耗大量的渲染时间，特别是比较复杂的UI界面，比如：在Canvas渲染模式中，一个Button会调用9次绘图(drawImage)。在复杂一些的UI场景中，会出现UI的绘图次数远大于实际游戏的绘图次数的情况，这对于性能资源非常稀缺的手机浏览器来说，会带来灭顶之灾。

对于上述情况，我们给cc.Layer加入了bake的接口，调用了该接口的层，会将自身以及其子节点都备份（烘焙/bake)到一张画布(Canvas)上，只要自身或子节点不作修改，下次绘制时，将直接把画布上的内容绘制上去。这样，原来需要调用N次绘图的层，就只需要调用一次绘图了。 当该层不需要bake时，可以调用unbake来取消该功能。

* 使用场景

1. 复杂UI层。 UI层含有大量的面板(Panel)，按钮(Button)等，这些控件的绘制会有大量的绘图调用，而这些控件并不经常修改。
2. 游戏中作为静态的背景或障碍物的层。

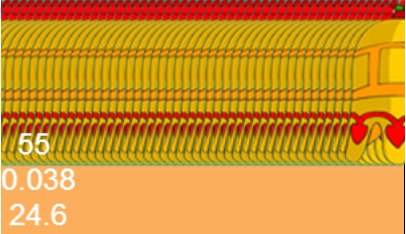
* 使用方法

使用bake功能非常简单:

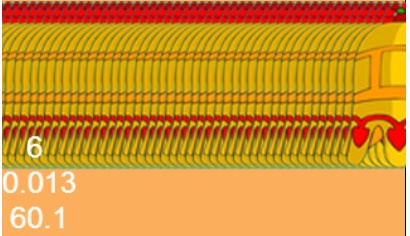
1. 将需要bake的节点元素加入到一个cc.Layer或其子类(cc.LayerColor,cc.LayerGradient)对象中；
2. 然后调用该对象的bake函数就可以了。

* 实例：
  + 效果：

//unbake



//bake



注意:两帧的刷新频率。

* + 源码：

initBakeLayer:function(){

//bake layer

var bakeLayer= new cc.Layer();

this.addChild(bakeLayer,1,1);

var playerArray=[];

for(var i=0;i<50;++i){

var width= cc.winSize.width/50;

var player= new cc.Sprite(res.player\_png);

player.attr({

x:width\*(i+0.5),

y:cc.winSize.height/4

});

playerArray.push(player);

bakeLayer.addChild(player,1);

}

bakeLayer.bake();

}

* 注意事项

1. 注意bake功能的对象只能是Layer。
2. 对于子节点经常会变的层， 启用bake功能，会给游戏性能带来额外的开销，建议对于不常修改子节点的层才开启该功能。
3. 该功能仅在Canvas渲染模式下有效, 在JSB与WebGL渲染模式下调用bake函数，不会产生效果。
4. 层

Layer是处理玩家事件响应的Node子类。与场景不同，层通常包含的是直接在屏幕上呈现的内容，并且可以接受用户的输入事件，包括触摸，加速度计和键盘输入等。我们需要在层中加入精灵，文本标签或者其他游戏元素，并设置游戏元素的属性，比如位置，方向和大小；设置游戏元素的动作等。通常，层中的对象功能类似，耦合较紧，与层中游戏内容相关的逻辑代码也编写在层中，在组织好层后，只需要把层按照顺序添加到场景中就可以显示出来了。要向场景添加层，我们可以使用addChild方法。

addChild( Node child )

addChild( Node child, int zOrder )

addChild( Node \*child, int zOrder, int tag )

其中，Child参数就是节点。对于场景而言，通常我们添加的节点就是层。先添加的层会被置于后添加的层之下。如果需要为它们指定先后次序，可以使用不同的zOrder值。tag是元素的标识号码，如果为子节点设置了tag值，就可以在它的父节点中利用tag值就可以找到它了。层可以包含任何Node作为子节点，包括Sprites(精灵), Labels(标签)，甚至其他的Layer对象。

注意：

Layer的锚点默认为(0,0)，即左下角。并且忽略锚点的设置，即使你setAnchorPoint了锚点，Layer的锚点也不会改变，依然是（0,0）。

复杂的应用会要求你定义特定的“CCLayer”子类，cocos2d提供了几个预设的子类，例如“CCMenu”（一个简单的菜单层）、“CCColorLayer”（一个绘制纯色的颜色层）以及“CCLayerMultiplex”（一个允许你复用子类的层，一次激活一个子类时会关闭其他子类）。

cc.Layer的派生类有：

* + - cc.Menu//菜单
    - cc.LayerColor//纯色
    - cc.LayerMultiplex
    - cc.LayerGradient//渐变颜色层

创建：cc.LayerGradient(start, end, v)

CCLayerGradient 是cc.LayerColor的子类，可以绘制具有梯度的背景。具备所有cc.LayerColor具有的功能，加入了如下的功能

方向

最终颜色

差值模式

将会在给定的开始颜色和最终颜色之间进行插值计算。向量（起点开始，终点结束），如果没有指定向量，那么使用默认向量（0,-1），代表从顶部渐变到底部。以左下角为原点(0,0),向颜色梯度的方向向量v画的方向，就是颜色梯度渐变的方向。

如果compressedInterpolation模式被关闭，将不会看到任何基于非基础向量的起始和结束颜色，将会保持两个端点之间的平滑梯度。如果compressedInterpolation模式开启（默认情况下），将会看到梯度的起始和结束颜色。

//开启与关闭：layerGradient.setCompressedInterpolation(true/false);

//判断是否已经开启compressedInterpolation模式 layerGradient.isCompressedInterpolation()

* + 创建一个纯色背景层

this.colorLayer= new cc.LayerColor();

this.colorLayer.attr({

color:cc.color(255,255,0),

width:100,

height:100,

x:this.width/2,

y:this.height/2

});

注意：

1、Layer的锚点默认为(0,0)，即左下角。屏幕居中显示，必须忽略锚点的设置：this.colorLayer.ignoreAnchor=false;

2、利用层的width和height属性，可以动态修改层的尺寸。

* + 颜色混合

注意：

使用颜色混合时,，必须使用WebGL渲染模式，因此最好将project.json中的“renderMode”的值为0。

this.layer.setBlendFunc(cc.ONE\_MINUS\_DST\_COLOR,cc.ZERO);

* + 创建一个颜色渐变层：

this.layerGradient= new cc.LayerGradient(cc.color(255,0,0),cc.color(0,255,0));

this.layerGradient.attr({

width:300,

height:300,

x:this.width/2,

y:this.height/2,

ignoreAnchor:false

});

this.addChild(this.layerGradient);

可以利用layerGradient.setVector(diff)函数和向量diff，动态改变渐变层的颜色插值。

疑问：

* + - 为什么创建cocos类的对象时，有的可以给基类对象传递参数，有的却不可以？又不像C++中的类一样有构造函数，怎么可以做到传递参数呢？

例如：

var layer = new cc.LayerColor(cc.color(255, 255, 0, 80), 100, 300);

原因：

使用new创建对象时，等价于调用了基类的init()函数，因此只要基类的init函数有参数，就可以在创建的对象的时候传参数，初始化。

1. 音频播放
   * + Cocos 2D-JS的cc.audioEngine机制

cc.audioEngine是一个单例对象，它提供了简单的音频接口。背景音——Music 音效——Effect

* + 背景音

//播放 参数二是：true——循环播放

cc.audioEngine.playMusic(res.background\_mp3,true);

//暂停

cc.audioEngine.pauseMusic();

//恢复

cc.audioEngine.resumeMusic();

//停止

cc.audioEngine.stopMusic();

//是否播放

if(cc.audioEngine.isMusicPlaying()){

alert("播放中！");

}else{

alert("没有播放！");

}

//增加音量【音量最大是1.0，最小是0.0】

cc.audioEngine.setMusicVolume(cc.audioEngine.getMusicVolume()+0.1);

//减少音量

cc.audioEngine.setMusicVolume(cc.audioEngine.getMusicVolume()-0.1);

* + 音效

//对应的音效

var g\_effectID=null;

//播放

g\_effectID=cc.audioEngine.playEffect(res.effect2\_mp3);

//循环播放

g\_effectID=cc.audioEngine.playEffect(res.effect2\_mp3,true);

//暂停

cc.audioEngine.pauseEffect(g\_effectID);

//停止

cc.audioEngine.stopEffect(g\_effectID);

//恢复

cc.audioEngine.resumeEffect(g\_effectID);

//从内部缓冲区释放预加载的音效

cc.audioEngine.unloadEffect(g\_effectID);

//暂停所有的音效

cc.audioEngine.pauseAllEffects();

//恢复所有的音效

cc.audioEngine.resumeAllEffects();

//停止所有音效

cc.audioEngine.stopAllEffects();

* + - Cocos 2D-X的
    - <audio>标签

注释：Internet Explorer 8 以及更早的版本不支持 <audio> 标签。<audio> 标签是 HTML 5 的新标签。

* + 定义和用法

<audio> 标签定义声音，比如音乐或其他音频流。

提示：可以在开始标签和结束标签之间放置文本内容，这样老的浏览器就可以显示出不支持该标签的信息。

* + 属性

属性 值 描述

autoplay autoplay 如果出现该属性，则音频在就绪后马上播放。

controls controls 如果出现该属性，则向用户显示控件，比如播放按钮。

loop loop 如果出现该属性，则每当音频结束时重新开始播放。

muted muted 规定视频输出应该被静音。

preload preload

如果出现该属性，则音频在页面加载时进行加载，并预备播放。

如果使用 "autoplay"，则忽略该属性。

src url 要播放的音频的 URL。

* + 方法

使用标签的play和pause函数，就可以动态播放、暂停音频。

* + - <bgsound>

网页上，播放背景音乐的标签。【问题：不知道<audio>和<bgsound>标签，一个播放背景音，一个播放音效。是否会发生冲突？】

1. 播放音效:【自动播放】

* 方法一:

<audio id="ring" preload="load" loop="loop" autoplay="autoplay" src="ring.mp3" type="audio/mpeg">

</audio>

* 方法二:

<audio id="ring" autoplay="autoplay" src="ring.mp3" type="audio/mpeg">

</audio>

* 方法三:

<audio id="ring" autoplay="autoplay" src="ring.mp3" type="audio/mpeg">

</audio>

<script>

$(function(){//手动自动播放，避免在一些手机中不能自动播放

var myAudio= $("#ring");

myAudio.play();

});

</script>

* 方法四:

<audio id="ring" autoplay="autoplay" src="ring.mp3" type="audio/mpeg">

</audio>

<script>

var myAudio= $("#ring");

myAudio.ready(function(){//手动自动播放，避免在一些手机中不能自动播放

myAudio.play();

});

</script>

1. 播放视频:
2. <video>标签:

<video id="media" autoplay loop="loop">

<source src="movie.mp4" type="video/mp4" />

</video>

* 缺陷:

循环播放:在chrome浏览器中，可以自动循环、自动播放,但是在微信浏览器中必须手动播放，而且不能自动循环;

1. <embed>标签

<embed src="movie.mp4" autostart="true" loop="true" >

</embed>

* + 在微信浏览器上，不能自动播放：

原理:

检测到<video>标签播放完毕之后的触摸事件中，添加自动播放功能，不就实现了，自动循环播放的功能。

function Ended(){

var video= document.getElementById("media");

video.play();

}

<video id="media" autoplay onended="Ended()">

<source src="movie.mp4" type="video/mp4" />

</video>

1. 颜色混合

注意：使用颜色混合时,，必须使用WebGL渲染模式和“openGL”绘图，因此最好将project.json中的“renderMode”的值为0。同时首先判断设备是否支持：

"opengl" in cc.sys.capabilities

Sprite不仅继承了Node，还继承了纹理协议接口TextureProtocol。

TextureProtocol纹理协议接口主要是负责纹理图片的管理。

在游戏开发中，如果我们需要实现闪光的灯，照明弹效果等等，我么你可以采用混合模式来实现。

如果学习过OpenGL(ES)，就知道里面使用glBlendFunc函数实现的。在cocos2d-x里肯定也有，对于精灵，可以使用mySprite->setBlendFunc()来现。

颜色混合：简单来说就是，将RGBA中的A,经行操作处理。具体一点，就是把某一像素位置原来的颜色和将要画上去的颜色，通过某种方式混在一起，从而实现特殊的效果。

OpenGL 会把源颜色和目标颜色各自取出，并乘以一个系数（源颜色乘以的系数称为“源因子”，目标颜色乘以的系数称为“目标因子”），然后相加，这样就得到了新的颜色。（也可以不是相加，新版本的OpenGL可以设置运算方式，包括加、减、取两者中较大的、取两者中较小的、逻辑运算等）。

glBlendFunc有两个参数，前者表示源因子，后者表示目标因子。这两个参数可以是多种值，下面介绍比较常用的几种。

GL\_ZERO： 表示使用0.0作为因子，实际上相当于不使用这种颜色参与混合运算。

GL\_ONE： 表示使用1.0作为因子，实际上相当于完全的使用了这种颜色参与混合运算。

GL\_SRC\_ALPHA：表示使用源颜色的alpha值来作为因子。

GL\_DST\_ALPHA：表示使用目标颜色的alpha值来作为因子。

GL\_ONE\_MINUS\_SRC\_ALPHA：表示用1.0减去源颜色的alpha值来作为因子。

GL\_ONE\_MINUS\_DST\_ALPHA：表示用1.0减去目标颜色的alpha值来作为因子。

CCSprite有一个ccBlendFunc类型的blendFunc\_结构体成员，可以用来设置描绘时的颜色混合方案, cc.TextureProtocol采用的协议。ccBlendFunc包含了一个src和一个dst，分别表示目标和源的运算因子。

如果我们对一个Sprite使用setBlendFunc方法，如：

Sprite.setBlendFunc(cc.ONE,cc.ZERO);

会以这个Sprite作为源，Sprite所在位置的其它像素作为目标，进行混合运算:

源的RGBA变量：Rs，Gs，Bs，As；

目标的RGBA： Rd，Gd，Bd，Ad；

源的各个运算因子： N\_Rs，N\_Gs，N\_Bs，N\_As；

目标的各个运算因子： N\_Rd，N\_Gd，N\_Bd，N\_Ad；

混合后的RGBA为：(Rs\*N\_Rs+ Rd\* N\_Rd，Gs\*N\_Gs+ Gd\* N\_Gd，

Bs\*N\_Bs+ Bd\* N\_Bd，As\*N\_As+ Ad\* N\_Ad)

其中的运算因子包括：

cc.ONE：1.0

cc.ZERO：0.0

cc.SRC\_ALPHA：源的Alpha值作为因子

cc.DST\_ALPHA：目标Alpha作为因子

cc.ONE\_MINUS\_SRC\_ALPHA：1.0减去源的Alpha值作为因子

cc.ONE\_MINUS\_DST\_ALPHA：1.0减去目标的Alpha值作为因子

我们可以在CCSprite的init方法中看到，默认的状态下使用cc.ONE, cc.ONE\_MINUS\_SRC\_ALPHA这两个参数，这也是我们看到的情况，Sprite会覆盖下面的描绘，但如果有透明的地方，则会显示下面的色值。

使用ALPHA相关的因子，会根据透明度来计算，比如透明度高的颜色占较大比重等。

常数 相关因子 融合因子结果

cc.ZERO 源因子或目的因子 (0,0,0,0)

cc.ONE 源因子或目的因子 (1,1,1,1)

cc.DST\_COLOR 源因子 (Rd,Gd,Bd,Ad)

cc.SRC\_COLOR 目的因子 (Rs,Gs,Bs,As)

cc.ONE\_MINUS\_DST\_COLOR 源因子 (1,1,1,1)-(Rd,Gd,Bd,Ad)

cc.ONE\_MINUS\_SRC\_COLOR 目的因子 (1,1,1,1)-(Rs,Gs,Bs,As)

cc.SRC\_ALPHA 源因子或目的因子 (As,As,As,As)

cc.ONE\_MINUS\_SRC\_ALPHA 源因子或目的因子 (1,1,1,1)-(As,As,As,As)

cc.DST\_ALPHA 源因子或目的因子 (Ad,Ad,Ad,Ad)

cc.ONE\_MINUS\_DST\_ALPHA 源因子或目的因子 (1,1,1,1)-(Ad,Ad,Ad,Ad)

cc.SRC\_ALPHA\_SATURATE 源因子 (f,f,f,1); f=min(As,1-Ad)

* + cc.SRC\_ALPHA,cc.ONE\_MINUS\_SRC\_ALPHA

作用：

如果偏上的精灵半透明，那么最终呈现的画面将会隐隐的露出偏下的精灵图片，

如果偏上的精灵完全不透明，那么偏上精灵所在的部分会完全将底下的精灵图片遮盖住。

* + cc.DST\_COLOR, cc.ONE\_MINUS\_SRC\_ALPHA

效果:



作用：正片叠底

* + cc.SRC\_ALPHA, cc.ONE\_MINUS\_SRC\_ALPHA

效果：



* + cc.ZERO， cc.ONE\_MINUS\_SRC\_ALPHA

效果：



作用：背景图片在黄色光圈区域的图片尤为阴暗

* + cc.ZERO， cc.DST\_ALPHA

效果：



作用：仅仅显示出了原始的背景图片

* + cc.ZERO， cc.ONE\_MINUS\_DST\_ALPHA

效果：



作用：加黑

* + cc.ONE， cc.ZERO

效果：



作用：仅显示黄色光圈图片

* + cc.ONE，cc.ONE

效果：



作用：纯粹的添加光圈图片到背景图片上面

* + cc.ONE，cc.DST\_COLOR

效果：



作用：高亮

* + cc.ONE， cc.ONE\_MINUS\_DST\_COLOR

效果：



作用：朦胧

* + cc.ONE， cc.SRC\_ALPHA\_SATURATE

效果：



作用：纯粹的添加光圈图片到背景图片上面

* + cc.ONE， cc.SRC\_ALPHA

效果：



作用：黄色光圈区域显示背景图片，其他区域显示黑色(黄色光圈很刺眼)

* + cc.SRC\_COLOR，cc.DST\_COLOR

效果：



* + cc.ZERO，cc.SRC\_ALPHA

效果：



作用：仅显示光晕下的精灵。其他区域颜色都为黑色，包括背景。

源码：hole.setBlendFunc(cc.ZERO, cc.SRC\_ALPHA);

* + 高亮效果

var flare = new cc.Sprite("res/1\_zaw2.png");

this.addChild(flare, 1, 1);

flare.setBlendFunc(cc.SRC\_ALPHA, cc.ONE);

flare.attr({

x: cc.winSize.width/2,

y: cc.winSize.height/2-200,

scale: 0.5

});

var opacityAnim = cc.fadeTo(1, 255);//变亮

var opacDim = cc.fadeTo(1, 0);//变暗

flare.runAction(cc.sequence(opacityAnim, opacDim).repeatForever());

1. \*.plist文件

在cocos2d-x中可以用.plist格式的文件来保存数据，它是XML文件格式的一种，在cocos2d-x解析.plist方面相关的资料比较少，但本身也很简单，要解析.plist文件可以参考cocos2d-x类库中的CCSpriteFrameCache类和CCParticleSystem类，它主要是使用CCDictionary类来对.plist文件进行操作。

这里需要注意的是，当在读取'pos'的时候，它的值一个{x, y}的字符串，这是.plist文件中的数组存储规则，我们可以通过cocos2d-x提供函数api将这样的字符串转化为CCpoint对象。

CCPoint point = CCPointFromString(farScene->valueForKey("pos")->getCString());

上面这句话就是做了这样的一个转化的过程，同样的cocos2d-x还支持CCSize、CCRect的字符串的转化。他们转化的方法以及在.plist中对应的字符串格式如下：

CCPoint: CCPointFromString();{x, y}

CCSize:  CCSizeFromString();{w, h}

CCRect: CCSizeFromString();{x, y, w, h}

这样我们2D游戏所初始化所需要的数据都基本上够用了，可以尝试将游戏的初始数据放在.plist中，然后修改调整数值就可以直接修改plist文件，而无需重新编译程序了，从而实现游戏数据和游戏逻辑的分离。

在绘制精灵时最好使用CCSpriteBatchNode和CCSpriteFrameCache，前面只说过可以通过图片合并后使用.plist文件在程序内部进行引用，这回就具体说说这些文件是如何生成并使用的。这样的纹理图片打包工具应该是不只一个的，我常用的就是Texture packerGUI，好东西总会有盗版，用之……大笑

要说为什么要打包使用纹理图片的话，那是因为可以节省载入次数，这样加载效率就高了，然后还有一点就是OPenGL ES1.0需要纹理图片以2的n次方的尺寸载入，非常麻烦，所以使用这样的工具是非常有必要的。

这个工具可以根据不同的游戏引擎使用最适合的打包方式比如：苹果专用的图片格式PVR。简单的操作就一句话：点击add Sprites按钮添加一堆精灵图片，设置dataFile导出到的目录、图片格式及imageFormat质量，最后publish即可

使用\*.plist加载动画

cc.spriteFrameCache.addSpriteFrames(res.runner\_plist);

this.spriteSheet = new cc.SpriteBatchNode(res.runner\_png);

this.addChild(this.spriteSheet);

if(GameStatus=="playing"){

//游戏状态。判断球是上升，还是下降

varleftFootBallPosY= this.leftFootballSprite.getPositionY()-this.preLeftFootBallY;

if(leftFootBallPosY>0){

leftFootBallStatus="up";

}else{

leftFootBallStatus="down";

}

if(leftFootBallStatus=="down"){

if(leftFootBallStatus<0.00001){

GameStatus="end";

}

}

}

1. spawn和sequence同时执行，执行的同时，再顺序执行某一项动作。

var action = cc.spawn(

cc.jumpBy(2, cc.p(300, 0), 50, 4),

cc.rotateBy(2, 720));

varseq=cc.sequence(action,action.reverse());

1. 给图片添加监听事件：

var introduce = new cc.Sprite(res.a\_introduce);

introduce.attr({

anchorX:0.5,

anchorY:0.5,

x:cc.winSize.width/2,

y:cc.winSize.height/2,

scale:cc.winSize.width/640

});

this.addChild(introduce,10);

var self = this;

var listener = cc.EventListener.create({

event: cc.EventListener.TOUCH\_ONE\_BY\_ONE,

swallowTouches: true,

onTouchBegan: function (touch, event) {

var target = event.getCurrentTarget();

varlocationInNode = target.convertToNodeSpace(touch.getLocation());

var s = target.getContentSize();

varrect = cc.rect(0, 0, s.width, s.height);

if(cc.rectContainsPoint(rect, locationInNode)) {

self.removeChild(introduce,true);

cc.director.resume();

return false;

}

return false;

}

});

cc.eventManager.addListener(listener,introduce);

//动态的移除或者添加监听机制

//移除

cc.eventManager.removeListener(listener);

//添加

cc.eventManager.addListener(listener,introduce);

1. 怎么样在一个层中 访问另一个层的变量？

//获取ID为TagOfLayer.Status层的对象

varstatusLayer = this.getChildByTag(TagOfLayer.Status);

statusLayer.updateTimer(g\_time);//访问该层的函数

1. 实现精灵动态切换：

* Cocos2d-X
* 直接通过图片更换

使用setTexture(CCTexture2D\*)函数，可以重新设置精灵类的纹理图片。

CCTexture2D\* texture = CCTextureCache::sharedTextureCache()->addImage("bg2.png");

bg->setTexture(texture);

* 通过缓存帧替换

使用setDisplayFrame(CCSpriteFrame\*)函数，利用精灵帧更换精灵图片。

//加载plist文件到缓存

CCSpriteFrameCache::sharedSpriteFrameCache()->addSpriteFramesWithFile("bg\_0.plist");

//使用精灵帧，创建精灵

CCSprite\* bg = CCSprite::createWithSpriteFrameName("bg1.png");

//更换精灵图片

CCSpriteFrame\* frame = CCSpriteFrameCache::sharedSpriteFrameCache()->spriteFrameByName("bg2.png");

bg->setDisplayFrame(frame);

* Cocos2d-JS
* 直接通过图片切换

var texture = cc.textureCache.addImage(res.Arrow\_png);

this.leftFootballSprite.setTexture(texture);

* 通过缓存帧替换

在cocos2d-js中，setDisplayFrame已经被弃用，使用setSpriteFrame代替。

cc.spriteFrameCache.addSpriteFrames(res.running\_plist);

var texture = cc.spriteFrameCache.getSpriteFrame("runner1.png");

player.setDisplayFrame(texture);

//其中runner1.png是res.running\_plist资源中的一个图片。

1. 点击某一项菜单，切换图片

this.click.setNormalImage(cc.Sprite.create(res.a\_btnOn));

1. 在图片上添加监听事件之后，如何调用图片所在的层的函数?

var target1 = event.getCurrentTarget();

target1.getParent().move();

1. js 字符串转化成数字

三种方法: 转换函数、强制类型转换、利用js变量弱类型转换。

* + - 转换函数：

js提供了parseInt()和parseFloat()两个转换函数。前者把值转换成整数，后者把值转换成浮点数。只有对String类型调用这些方法，这两个函数才能正确运行；对其他类型返回的都是NaN(Not a Number)。

* + - 强制类型转换

使用强制类型转换（type casting）处理转换值的类型。使用强制类型转换可以访问特定的值，即使它是另一种类型的。

ECMAScript中可用的3种强制类型转换如下：

Boolean(value)——把给定的值转换成Boolean型；

Number(value)——把给定的值转换成数字（可以是整数或浮点数）；

String(value)——把给定的值转换成字符串。

用这三个函数之一转换值，将创建一个新值，存放由原始值直接转换成的值。这会造成意想不到的后果。

当要转换的值是至少有一个字符的字符串、非0数字或对象（下一节将讨论这一点）时，Boolean()函数将返回true。如果该值是空字符串、数字0、undefined或null，它将返回false。

* + - 利用js变量弱类型转换

方法一：调用数值类型对象的toString()方法，转换；

方法二：直接将数值类型赋给字符串变量

1. 强制显示数字中的固定位数的小数点

使用数值类型的toFixed()函数。

1. 固定小数点后数字的位数之后，删除用不到的0【例如：1.000🡪1】

(g\_bestMile\*0.007).toFixed(3)\*1000/1000

固定几位，乘以再除以10的位数次方。

1. 子节点中父节点中移除：

layerColor.removeFromParent();

1. 效果：触摸某对象时，透明；释放时，还原

注意：一幅图像的原始颜色值为：cc.color(255,255,255,255);

在onTouchBegan函数中，设置对象的透明度：target.setOpacity(128);//半透

在onTouchEnded函数中，设置对象的透明度：target.setOpacity(255);//还原

同理，设置颜色的时候，也可以使用cc.color(255,255,255)还原。

1. 效果：拖动图像移动

在onTouchMoved函数中，利用touch.getDelta()函数，在屏幕坐标系中，返回从前一个触摸点到当前点的delta距离。获取了两者之间的距离，加上原来的位置，就可以实现拖动效果了。

onTouchMoved:function(touch,event){

var target= event.getCurrentTarget();

var point= touch.getDelta();

var desPoint=cc.pAdd(target.getPosition(),point);

target.setPosition(desPoint);

}

分析：为什么不直接利用触摸点的位置，来设定图像的位置？

可能的原因之一：因为锚点的位置和当前触摸点之间有偏移。在第一次移动时，明显有跳跃的问题；

可能的原因之二：当前项目的要求就是以当前触摸点为拖动点，不能发生偏移。

1. 效果：抽奖效果展示

背景:后台抽奖、前台跑马灯效果显示抽奖结果

原理:前台：停在从后台传到前台的抽中索引对应的奖品项。

分析：

因为不知道什么时候会从后台将数据传过来，但是跑马灯效果一定要一直执行。

跑马灯效果停止的条件是：当前的奖品索引==抽中的奖品索引;

因此,在后台没有传回来数据之前，可以将抽中的奖品索引值，设置为-1等等无效的数据。因为是无效的数据，所以跑马灯效果，将会一直执行。而后台传过来的数据一定是有效的，因此，只要后台传过来数据，跑马灯效果就会停止。

1. 效果：抽奖转盘，先加速，后减速，听到后台指定的位置处

* 获取需要旋转的角度

//转动的圈数

var turnNum= Math.ceil(Math.random()\*3+2);

//奖品索引值

var index= getAwardIndex();

//转到指定的位置处，需要旋转的角度【圈数\*360+一圈之内应该停止的角度数】

g\_totalRotation= turnNum\*360+ index\*Math.floor(360/g\_species)+Math.floor(Math.random()\*360/g\_species);

* 先加速，后减速

将所需要旋转的角度数，分成两半，一半加速，一半减速。两者之间的加速度是一样的，仅方向不同。

g\_accelerationSpeed=g\_initialSpeed\*g\_initialSpeed/(2\*g\_totalRotation/2);

//用到的物理公式

vt^2 – v0^2 = 2\*a\*s;

* 注意：当加速到最高速度时，加速度方向改变，同时，运动时间重置：

var rotationSpeed=this.time\*g\_accelerationSpeed;

if(rotationSpeed>=g\_initialSpeed){

g\_accelerationSpeed\*=-1;

this.time=0;

}

if(g\_accelerationSpeed<0){

rotationSpeed+=g\_initialSpeed;

}

//用到的物理公式

vt = v0-at;

* 注意：转动角度变化=当前位置+速度\*渲染该帧花费的时间

this.tabel.setRotation(this.tabel.getRotation()+delta\*rotationSpeed);

示例：

var Layer4= GameLayer.extend({

rotationSpeed:0,//the rotation speed

time:0,//the time of the movement

init:function(){

this.\_super(cc.color(43,89,0),cc.color(0,162,155),"Rotary Table","slowly stop");

this.initRotaryTable();

},

initRotaryTable:function(){

var content= new cc.Sprite(res.stencil\_png);

content.attr({

x:0,

y:130

});

this.stencil= new cc.Sprite(res.rotarytable\_1\_png);

//ClippingNode

var clipper= new cc.ClippingNode();

clipper.attr({

x:this.\_layerWidth/2,

y:this.\_layerHeight/2,

width:content.getContentSize().width,

height:content.getContentSize().height

});

clipper.setAlphaThreshold(0);

this.layerGradient.addChild(clipper,2,10);

clipper.setStencil(this.stencil);

clipper.addChild(content,1,1);

this.tabel= new cc.Sprite(res.rotarytable\_1\_png);

clipper.addChild(this.tabel,0,1);

//award button

var item= new cc.MenuItemImage(

res.award\_png,

null,

this.OnAward,this

);

var menu= new cc.Menu(item);

menu.setPosition(cc.p(this.\_layerWidth/2,this.\_layerHeight/2));

this.layerGradient.addChild(menu,2,11);

//arrow

var arrow= new cc.Sprite(res.arrow\_png);

arrow.attr({

x:this.\_layerWidth/2,

y:this.\_layerHeight/2,

scaleY:0.7,

opacity:0//Set the opacity to zero——not visible

});

this.layerGradient.addChild(arrow,2,12);

},

OnAward:function(pSender){

//get the index of the award

this.getAwardIndex();

//set menu not enabled

pSender.getParent().setEnabled(false);

//restore

this.tabel.setRotation(0);

//arrow

var arrow=this.layerGradient.getChildByTag(12);

//menu fade out,arrow fade in

var fadeOut= cc.fadeOut(0.5);

var fadeIn= cc.fadeIn(0.5);

var self=this;

var seq2=cc.sequence(fadeIn,cc.callFunc(function(pSender){

//start schedule

self.scheduleUpdate();

}));

var seq1= cc.sequence(fadeOut,cc.callFunc(function(pSender){

arrow.runAction(seq2);

}));

pSender.getParent().runAction(seq1);

},

update:function(delta){

this.time+=delta;

var rotationSpeed=this.time\*g\_accelerationSpeed;

//因为加速度是有方向的，所以速度只可能比g\_initialSpeed大一次

if(rotationSpeed>=g\_initialSpeed){

g\_accelerationSpeed\*=-1;

this.time=0;

}

//是否为减速了？

if(g\_accelerationSpeed<0){

rotationSpeed+=g\_initialSpeed;

}

if(rotationSpeed<=0){

rotationSpeed=0;

this.unscheduleUpdate();

//result

this.result();

this.time=0;

return;

}

//rotate

this.tabel.setRotation(this.tabel.getRotation()+delta\*rotationSpeed);

},

getAwardIndex:function(){

//the number of the turns

var turnNum= Math.ceil(Math.random()\*3+2);

//the index of the award

var index= getAwardIndex();

g\_totalRotation= turnNum\*360+ index\*Math.floor(360/g\_species)+Math.floor(Math.random()\*360/g\_species);

g\_accelerationSpeed=g\_initialSpeed\*g\_initialSpeed/(2\*g\_totalRotation/2);

},

result:function(){

//Set the arrow is not visible

var arrow= this.layerGradient.getChildByTag(12);

//Set the menu is visible

var menu= this.layerGradient.getChildByTag(11);

var fadeOut= cc.fadeOut(0.5);

var fadeIn= cc.fadeIn(0.5);

var seq1=cc.sequence(fadeOut,cc.callFunc(function(pSender){

menu.runAction(seq2);

}));

var seq2= cc.sequence(fadeIn,cc.callFunc(function(pSender){

menu.setEnabled(true);

}));

arrow.runAction(seq1);

alert("恭喜您抽中了"+g\_awardIndex+"——"+g\_awardName[g\_awardIndex]);

}

});

1. 效果：在指定的时间内，计数

说明：

普通的计数方式是：

this.schedule(this.OnSchedule,1);

OnSchedule:function(){

i++;

}

原意:1秒记一次数。但是，因为设备不同，因此性能也千差万别，可能有的设备在1秒之后加一次，但是有的可能1.5秒才加一次，所以同样是20秒，速度快的可能是20，但是速度慢的可能只是10、15。因此将会违背程序员意愿。

解决：

//定义一个全局变量

this.time=1;

this.scheduleUpdate();

onUpdate:function(delta){

this.time-=delta;

if(this.time<=0){

i++;

this.time+=1;

}

};

其中，delta是每两帧之间的渲染时间。

注意：

1. 判断是否为1秒时，应该判断<=0。原因是：delta是浮点数，可能会出现误差；
2. 将时间再还原时，this.time+=1。原因是：this.time中的负数，也是属于20秒之内的。
3. 也可以改为：this.time+=delta;但是一定要注意：时间到的时候，this.time有一部分是属于总时长的。
4. 效果：在指定的时间内，掉落规定数量物品
5. 首先计算应该多长时间，创建一个物品：

创建两个物品之间的时间间隔= 总游戏时间 / 要下落物品的数量

1. 使用上一实例中的计时器方法定时，只不过将定时器的时间1改为创建两个物品之间的时间间隔。
2. 效果：游戏时间到了。将悬浮在空中的物品，下落之后，游戏再结束，现实弹窗。
3. 在游戏时间的检测事件中，添加一个判定条件：

if(g\_gameTime<=0){

//是否还有悬浮的物品,等价于判断下落数组的长度是否为空。

if(g\_goodsArray.length==0){

//游戏真正结束 切换状态

g\_gameStatus=GameOver;

}

}

1. 添加一个标记，是否可以发生碰撞：g\_isCollision。并在检测碰撞区域的地方，加上判定条件。
2. 效果：当玩家的成绩大于某一限制时，增加障碍物。而且必须在游戏的规定时间之前，全部显示出来

分析：

1. 添加标记：g\_isAdd=false;【标记是否已经增加过障碍物】
2. 在更新成绩的地方，添加一个判定条件：

if((!g\_isAdd)&&(g\_score>=50)){

g\_isAdd=true;

//创建新的障碍物，并添加到下落数组中。

}

注意：

* + 出现以下情况,可以将以上两步，可以合为一步，不用使用全局变量g\_score。

1、玩家的成绩只能1分1分加，不能出现扣分、+2等等情况。

原因是：在以上情况下，只会出现一次50的情景。

1. 创建新物体的时间，改为:

（总游戏时间 – 已经进行的游戏时间） / (剩余要下落的物体数量【包括新添加的数量】)

1. 效果：卷轴

卷轴效果，可以分为两种：1、从中间向两边慢慢打开；2、从一端向另一端打开。不过相似点是：必须将整个卷轴，分为三部分：中间、两边卷筒。

* 资源：



* 自定义一个卷轴类,包含三个精灵。

var Rouleau = cc.Node.extend({

ctor:function(way){

this.\_super();

this.init();

},

init:function(){

//center

var center= new cc.Sprite(res.rouleau2\_png);

this.addChild(center,0,1);

//size

var size = center.getContentSize();

//top

var top= new cc.Sprite(res.rouleau1\_png);

top.setPosition(cc.p(size.width/2,size.height));

//bottom

var bottom= new cc.Sprite(res.rouleau1\_png);

bottom.setPosition(cc.p(size.width/2,0));

center.addChild(top,0,2);

center.addChild(bottom,0,3);

}

});

* + 上一步骤中，生成的卷轴是展开的，为了实现：从中间向两边，展开的效果，需要修改一下两边卷筒的位置、以及中间精灵可视区域大小：

//定义两个全局变量

//1、向某一端可以扩展的高度

centerheight:0,//the half of the extensible height

//2、保存以前图片的可视区域

preRect:cc.rect(0,0,0,0),

center\_way:function(){

var center = this.getChildByTag(1);

var top= center.getChildByTag(2);

//get the height of the rouleau

var height= top.getContentSize().height;

var size= center.getContentSize();

this.centerheight=(size.height/2-height/2)\*this.scale;

this.preRect=cc.rect(0,this.centerheight,size.width,height);

//console.log("原来的:" + this.preRect.x +" "+this.preRect.y + " " +this.preRect.width +" "+this.preRect.height);

center.setTextureRect(this.preRect);

top.setPosition(cc.p(size.width/2,height));

this.schedule(this.centerExtensible,0.01);

}

* + 开启定时器

centerExtensible:function(){

var center = this.getChildByTag(1);

this.centerheight-=g\_preHalfHeight;

if(this.centerheight<=0){

this.centerheight=0;

this.unschedule(this.centerExtensible);

return;

}

this.preRect.y-=g\_preHalfHeight;

this.preRect.height+=2\*g\_preHalfHeight;

//console.log("修改后的:" + this.preRect.x +" "+this.preRect.y + " " +this.preRect.width +" "+this.preRect.height);

center.setTextureRect(this.preRect);

var top= center.getChildByTag(2);

top.setPositionY(top.getPositionY()+2\*g\_preHalfHeight);

}

* + 同理，从一端向另一端打开

top\_way:function(){

var center = this.getChildByTag(1);

center.setPosition(cc.p(0-center.getContentSize().width/2,center.getPositionY()+center.getContentSize().height/2-10));

center.setAnchorPoint(cc.p(0,1));

var top= center.getChildByTag(2);

//get the height of the rouleau

var height= top.getContentSize().height;

var size= center.getContentSize();

this.preRect= cc.rect(0,0,size.width,height);

center.setTextureRect(this.preRect);

top.setPosition(cc.p(size.width/2,height));

this.schedule(this.topExtensible,0.01);

},

topExtensible:function(){

var center = this.getChildByTag(1);

//console.log("原来的:" + this.preRect.x +" "+this.preRect.y + " " +this.preRect.width +" "+this.preRect.height);

this.preRect.height+=2\*g\_preHalfHeight;

if(this.preRect.height>=this.centerPicheight){

this.unschedule(this.topExtensible);

return;

}

center.setTextureRect(this.preRect);

var top = center.getChildByTag(2);

top.setPositionY(top.getPositionY()+2\*g\_preHalfHeight);

}

1. 效果：长按3秒以上

在onTouchBegan函数中，开启定时器，累计时间，并判断是否已经达到3秒以上,是—停止定时器，并且可以移除监听机制，执行相应的操作，累计时间清零；

在onTouchEnded函数中，关闭定时器，累计时间清零。

1. 效果：避免用户不输入或者输入的信息全部由空格组成

获取字符串，使用trim()函数,将两边的空格删除【无论字符串两端有多少个空格，都可以删除】，然后判断字符串的长度是否大于1，当大于1时，为有效输入，否则，用户没有输入或者输入的信息全部由空格组成。

//剔除空格

if(str.trim().length){

//str中是用户的有效输入。

}

1. 效果：分享成功以后，调用场景类的子节点中的一个函数：

//在Utility.js中定义

function shareSuccess(){

if(g\_GameStatus!=g\_GameStatusTag.Playing) return;

var gameLayer= cc.director.getRunningScene().getChildByTag(g\_LayerTag.GameLayerTag);

gameLayer.automaticAddTag();

}

1. 效果：刮刮卡
   * + 方法一：Cocos实现

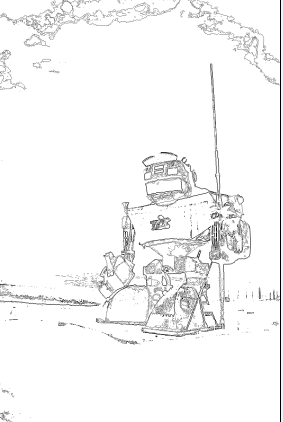
原理：

使用裁剪节点cc.ClippingNode()和蒙版的原理【蒙版中的内容可以看见，蒙版外的可见】，当移动的时候，动态添加蒙版。

content:遮蔽层

stencil:圆点

资源：







* 全局变量

content:null,

stencil:null,

* 创建裁剪节点

initClipping:function(){

//content

this.content= new cc.Sprite(res.robot\_redux\_png);

//stencil

this.stencil= new cc.Sprite();

//clippingNode

var clipper= new cc.ClippingNode();

clipper.setInverted(true);

clipper.attr({

anchorX:0,

anchorY:0,

x:cc.winSize.width/2,

y:cc.winSize.height/2

});

clipper.setStencil(this.stencil);

clipper.addChild(this.content);

this.addChild(clipper);

},

* 监听机制

addListener:function(){

var self= this;

var listener= cc.EventListener.create({

event:cc.EventListener.TOUCH\_ONE\_BY\_ONE,

onTouchBegan:self.onTouchBegan,

onTouchMoved:self.onTouchMoved

});

cc.eventManager.addListener(listener,self);

},

onTouchBegan:function(touch,event){

var target= event.getCurrentTarget().content;

var size= target.getContentSize();

var rect= cc.rect(0,0,size.width,size.height);

var point=target.convertToNodeSpace(touch.getLocation());

//检测

return cc.rectContainsRect(rect,point);

},

onTouchMoved:function(touch,event){

var target= event.getCurrentTarget().content;

var size= target.getContentSize();

var rect= cc.rect(0,0,size.width,size.height);

var location=target.convertToNodeSpace(touch.getLocation());

if(!cc.rectContainsRect(rect,location)){

return;

}

var stencil= new cc.Sprite(res.point\_png);

var point= cc.pSub(touch.getLocation(),cc.visibleRect.center);

stencil.setPosition(point);

event.getCurrentTarget().stencil.addChild(stencil);

}

* + - 方法二：JQeury实现

1. 效果:Html5中，实现进度条

//设置进度条的宽度为0%，指定进度条的高度

<img id= "progress\_fore" src="../images/time-2.png" style="width: 0%; height:10%;">

//在按钮的点击事件中或者定时器中，更新其宽度即可。

var progress= Math.floor((g\_answer\_index-1)/13\*100)+"%";

$("#progress\_fore").css("width",progress);

1. 效果：在html中，显示变量的值

//在Html中，定义标签

<p id="#num">123</p>

//定义

var g\_num=13522490962;

//使用jQuery方法，赋值

$("#num").html(g\_num);

//如果某标签没有id，只有class,如何获取该标签【id使用“#”,class使用“.”】

<p class="num">123</p>

$(“.num”).html(g\_num);

1. 效果：在html，修改某元素的背景

addClass() 方法向被选元素添加一个或多个类。该方法不会移除已存在的 class 属性，仅仅添加一个或多个 class 属性。

提示：如需添加多个类，请使用空格分隔类名。

示例：

1、在css文件中，添加一个类——green

.green {

color: rgb(0, 191, 0);

}

2、在html文件中，引用css

<link rel="stylesheet" type="text/css" href="../css/reset.css"/>

3、获取对应的元素并设置

例如：在html中定义的<p id="#num">123</p>

获取该对象$("#num").addClass("green");

4、移除该元素

$("#num").removeClass("green");

1. 效果:在html5中，添加定时器:

setInterval(onTest, 1000);

参数一:定时器执行函数

参数二:时长，单位为毫秒。

1. 效果:在html5中,模拟按钮的单击事件

function onTest(){

$('#me').trigger('click');

}

$(document).ready(function(){

setInterval(onTest, 1000);

});

同理:将click改为touchstart 即为 开始触摸 事件。

1. 效果:
2. 效果：动态修改<img>标签的图片资源路径

//图片

$("#pic1").attr({src:"../images/"+g\_answer\_index+".png"});

1. 效果：在Html 5中，图片资源预加载

在JS文件中，

* 方法一：

function LoadPic(filename){

var fileref = document.createElement('img');

fileref.setAttribute("type","image/jpeg");

fileref.setAttribute("src",filename);

}

* 方法二:

function LoadPic(filename){

var image = new Image();

console.log(filename);

image.src=filename;

}

1. 效果：在Html 5中，JS文件资源预加载

* 方法一：

document.write("<script src='src/cocos2d-js-v3.1-lite.js'></script>");

* 方法二：

document.write("<scri"+"pt type='text/javascript' src='src/cocos2d-js-v3.1-lite.js'></scr"+"ipt>");

* 方法三：

function LoadJS(filename){

var fileref = document.createElement('script');

fileref.setAttribute("type","text/javascript");

fileref.setAttribute("src",filename);

document.getElementsByTagName("head")[0].appendChild(fileref);

}

* 说明：

方法一和方法二在页面加载完毕【window.onload】之后时使用，将会报错。而方法三在任何时候，都可以正常使用。

1. 效果：获取用户点击时，<li>标签的对应索引值

<ul id="answer">

<li id="qt1" class="white" style="display: none">图中是你</li>

<li id="qt2" class="white" style="display: none">图中是我</li>

<li id="qt3" class="white" style="display: none">图中是他</li>

<li id="qt4" class="white" style="display: none">图中是你</li>

<li id="qt5" class="white" style="display: none">图中是我</li>

<li id="qt6" class="white" style="display: none">图中是他</li>

</ul>

1. 方法一：

* 给<ul>标签，添加监听事件

$(document).ready(function(){

$("#answer li").click(function(){

//获取被点击的对象

var \_this = $(this);

//获取被选中的文本

var answer= \_this.html();

//循环判断，获取的文本是否在选项中

var selectedID=0;

for(var i=0;i<g\_question[g\_answer\_index-1][1];++i){

if(answer.indexOf(g\_question[g\_answer\_index-1][3+i])>-1){

selectedID= i+1;

break;

}

}

其中selectID就是获取对应的ID。

}

}

1. 方法二：笨法子，给每一个<li>添加一个onclick事件，回调函数也不同。
2. 功能:在网页链接中,获取或者添加数据:【单个数值】
3. 添加数据:

在当前网页链接的后面,添加数据，以“?”作为分隔符。

window.location.href= window.location.href+"?"+score;

1. 获取数据

var score= window.location.href.substring(window.location.href.indexOf('?')+1,window.location.href.length);

1. 功能:在网页链接中,获取或者添加数据:【Json对象】

JSON（JavaScript Object Notation ）是一种轻量级的数据交换格式，采用完全独立于语言的文本格式，JSON是JavaScript原生数据格式。

在JSON中，有两种数据结构：对象和数组。

1.一个对象以“{”开始，“}”结束。每个“名称”后跟一个“：”，“名称/值”对之间运用“，”分隔，名称用引号括起来，如果是字符串则必须用括号，数值类型则不需要。

在数据传输流程中，json是文本，即以字符串形式传递，而JS操作的是json对象，所以，JSON对象和JSON字符串之间可以相互转换。

JSON字符串：

var str1 = '{ "name": "cxh", "sex": "man" }';

JSON对象：

var str1 = { "name": "cxh", "sex": "man" };

* JSON字符串转换为JSON对象：

var Obj = eval('('+ str +')');

var Obj = JSON.parse(str);

var Obj = str.parseJSON();

转换之后调用:Obj.name

* JSON对象转化为JSON字符串。

var str1 = Obj.toJSONString（）;

var str2 = JSON.stringgify（obj）;

* 获取json串:

当前页面链接: "http://219.234.5.130/game.html?openid=13522490962"

* + - 方法一:正则分析法

function getQueryString(name) {

var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)", "i");

var r = window.location.search.substr(1).match(reg);

if (r != null) return unescape(r[2]); return null;

}

举例:

var g\_openID= getQueryString(openid);

* + - 方法二:

<Script language="javascript">

function GetRequest() {

var url = location.search; //获取url中"?"符后的字串

var theRequest = new Object();

if (url.indexOf("?") != -1) {

var str = url.substr(1);

strs = str.split("&");

for(var i = 0; i < strs.length; i ++) {

theRequest[strs[i].split("=")[0]]=unescape(strs[i].split("=")[1]);

}

}

return theRequest;

}

</Script>

调用方法：

<Script language="javascript">

var Request = new Object();

Request = GetRequest();

var 参数1,参数2,参数3,参数N;

参数1 = Request['参数1'];

参数2 = Request['参数2'];

参数3 = Request['参数3'];

参数N = Request['参数N'];

</Script>

* 拼接json串:

var obj= {

score:g\_score,

openid:g\_openid,

period:g\_censorship+1,

data:g\_postData

};

var str= "data="+ JSON.stringify(obj);

调用方法：要链接的地址+ "?"+str即可。

1. Html5中，定时器和已经延时执行

//定时器

setInterval(code,delay)

//延时执行

setTimeout(code,delay)

参数说明：

code是回调函数。

举例：

setTimeout('setTitle()',500);

其中setTitle()是回调函数。

1. 效果：Html中，<input>的使用

* 定义：

<input id="question" type="text">

* 获取用户输入:

//1、JQuery

var question= $("#question");

var str= question.attr("value");

//2、Html

var str= document.getElementById("question").value

* 清空<input>中的内容：

var question= $("#question");

question.attr("value","");

1. 效果:Html中，获取id

var question= $("#question");

\_this.attr("id");

1. 判断字符串中是否是中文

var isChina=true;

var str;

* 方法一：

for(var i=0;(i<str.length)&&(isChina);++i){

if(str[i].charCodeAt()<=255){

isChina=false;

}

}

缺点：中文输入法下，标点符号也是中文字符。

* 方法二：

var id= escape(str).indexOf("%u");

if(id<0){

isChina= false;

}

缺点：中文输入法下，标点符号也是中文字符;数值也检测不出来。

* 方法三：

for(var i=0;(i<str.length)&&(isChina);++i){

if((str[i].charCodeAt()<=0x4e00)||(str[i].charCodeAt()>=0x9fff)){

isChina=false;

}

}

目前为止，没有测试出来缺点，因此可以算是比较好的方法。

1. JS中，字符串和数值之间的转换：

【字符串🡪数值】方法主要有三种：转换函数、强制类型转换、利用js变量弱类型转换。

* 转换函数

parseInt()和parseFloat()两个转换函数，前者把值转换成整数，后者把值转换成浮点数。

* 强制类型转换

Boolean(value)——把给定的值转换成Boolean型；

Number(value)——把给定的值转换成数字（可以是整数或浮点数）；

String(value)——把给定的值转换成字符串。

用这三个函数之一转换值，将创建一个新值，存放由原始值直接转换成的值。这会造成意想不到的后果。

* 利用js变量弱类型转换

var str="012.345";

var x = str-0;

x = x\*1;

上例利用了js的弱类型的特点，只进行了算术运算，实现了字符串到数字的类型转换，不过这个方法还是不推荐的。

【数字🡪字符串】方法主要只有一种：

var i = 10;

var s = i.toString();

1. 碰撞检测

碰撞检测技术运用了很多数学和几何知识。

* 平面几何在碰撞检测中的应用

将游戏画面转化为只有点和线组成的线框世界。在只有点和线组成的世界中，碰撞检测问题变得简单明了。一个坦克成了一个矩形，一发子弹变成了一个点。

因此只要判断是否在矩形之内。而此时的游戏中的碰撞检测就是判断点是否与线组成的形状发生碰撞，或者是两个形状之间是否发生碰撞。比如在程序中，我们需要检测一个矩形与另一个矩形是否发生碰撞。最直接的方法，就是查看第一个矩形的四个顶点是否在第二个矩形之中。这样无论是二维游戏还是三维游戏都可以通过判断点和线的关系来解决碰撞检测的问题。

注意：不要以为如此简单的就解决了游戏碰撞检测的问题。这只是最简单的方式哦！

* + 点与矩形的碰撞检测

原理：【与Cocos中cc.rectContainsPoint函数的原理相同】

要判定点在矩形内，就要判断

1. 点的X坐标在矩形的X的最大值和最小值之间;
2. 并且同时满足：点的Y坐标也在矩形的Y最大值和最小值之间;

isRectContainsPoint:function(rect,point){

return (point.x>=cc.rectGetMinX(rect)&&point.x<=cc.rectGetMaxX(rect)&&

point.y>=cc.rectGetMinY(rect)&&point.y<=cc.rectGetMaxY(rect));

}

* + 矩形与矩形的碰撞检测

因为在程序中，定下来一个矩形的依据是：左上角、宽度和高度。确切的是左上角和右下角【有左上角、宽度和高度定下来的右下角点】，因此，可以利用（左下角，右下角）来判定两个矩形之间的关系。

* 相交

var maxax = ra.x + ra.width,

maxay = ra.y + ra.height,

maxbx = rb.x + rb.width,

maxby = rb.y + rb.height;

return !(maxax < rb.x || maxbx < ra.x || maxay < rb.y || maxby < ra.y);

* 包容

原理：两个矩形rect1,rect2。如果rect2的左下角和右下角点，同时在rect1矩形之间，那么rect1包容rect2。

//判断是否为空

if (!rect1 || !rect2)

return false;

return !((rect1.x >= rect2.x) || (rect1.y >= rect2.y) ||

( rect1.x + rect1.width <= rect2.x + rect2.width) ||

( rect1.y + rect1.height <= rect2.y + rect2.height));

* + 圆与圆的碰撞检测

原理:

圆的碰撞检测比较简单，只要判断两圆心距离是否小于半径相加（r1+r2）即可。

isCollision:function(target){

var football= this.layerGradient.getChildByTag(1);

//获取两个圆的圆心

var pos1= football.getPosition();

var pos2= target.getPosition();

//计算两个圆心之间的长度

var length= cc.pDistance(pos1,pos2);

//获取两个圆的半径[假设半径==图片的宽度的一半]

var rad1=football.getContentSize().width/2;

var rad2=target.getContentSize().width/2;

console.log(length +" "+(length<(rad1+rad2)));

return (length<(rad1+rad2));

}

* + 圆与矩形的碰撞检测

有两种方法，两种方法之间的比较：

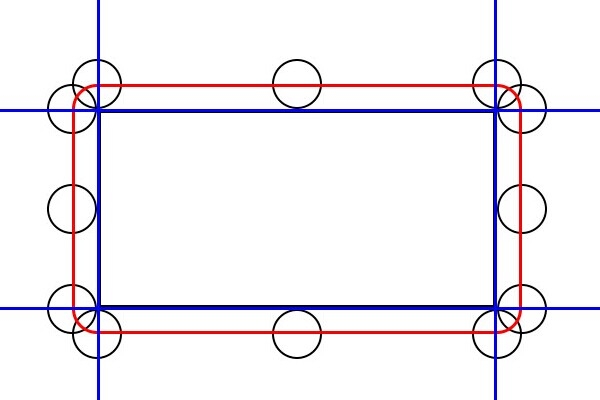
1. 精确度：方法一是精确地；方法二是不精确地；
2. 圆与矩形发生碰撞的方向。例如：圆碰到了矩形的那条边？方法二可以直观的回答。方法一则不行。可以用于碰撞反弹。

* 方法一：

矩形和圆的判断比较复杂，请看以下分析。

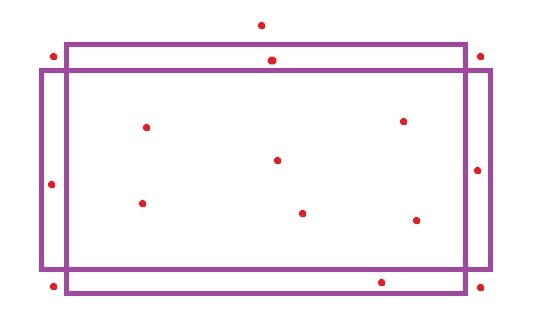
    （1）首先，我们让圆在矩形外沿着矩形的边滚一圈，然后将圆心移动的轨迹连线，就可以得到一个圆角矩形。

    （2）如下图红色区域为圆角矩形，显然我们只要判断圆心是否在圆角矩形区域内部即可。



    （3）如果除去圆角矩形四个角上的4个四分之一圆的部分，仅仅让你判断圆心是否落在剩下的区域内，你应该能很快想出解决办法吧？

       只要判断圆心是否在两个矩形的任意其中之一的内部即可。



    （4）然后再判断圆心是否在四个角上的四分之一圆的区域部分即可。

        显然，只要判断圆心与矩形的四个顶点的距离是否小于圆的半径即可。

  （5）综合上诉：（3）（4）的判断，即可得出圆是否矩形相交。

isCollision:function(target){

var football= this.layerGradient.getChildByTag(1);

//圆心的位置

var point= football.getPosition();

//假设：圆的半径==图宽度的一半

var radius= football.getContentSize().width/2;

var size= target.getContentSize();

//矩形区域

var rect=cc.rect(target.getPositionX()-size.width/2,target.getPositionY()-size.height/2,size.width,size.height);

var draw= new cc.DrawNode();

this.layerGradient.addChild(draw,3);

draw.drawRect(cc.p(cc.rectGetMinX(rect),cc.rectGetMinY(rect)),cc.p(cc.rectGetMaxX(rect),cc.rectGetMaxY(rect)), cc.color(255,255,0),2,cc.color(0,0,0));

//矩形的四个顶点

var MaxX= cc.rectGetMaxX(rect);

var MinX= cc.rectGetMinX(rect);

var MaxY= cc.rectGetMaxY(rect);

var MinY= cc.rectGetMinY(rect);

//圆心与矩形四个顶点之间的距离

var distance1= cc.pDistance(cc.p(MaxX,MaxY),point);

var distance2= cc.pDistance(cc.p(MaxX,MinY),point);

var distance3= cc.pDistance(cc.p(MinX,MaxY),point);

var distance4= cc.pDistance(cc.p(MinX,MinY),point);

//是否相交

//判断距离是否小于半径【圆弧】

if((distance1<radius)||(distance2<radius)||(distance3<radius)||(distance4<radius)){

return true;

}

//横向矩形内

if( (point.x > (MinX-radius)) && (point.x < (MaxX+radius)) && (point.y>MinY) && (point.y<MaxY)){

return true;

}

//纵向矩形内

if( (point.x > MinX) && (point.x < MaxX) && (point.y > (MinY-radius)) && (point.y < (MaxY+radius)) ){

return true;

}

}

* 方法二：

简单的讲，不关心圆弧的。只是将圆弧改成近似的看成矩形。

isCollision:function(target){

var football= this.layerGradient.getChildByTag(1);

//圆心的位置

var point= football.getPosition();

//假设：圆的半径==图宽度的一半

var radius= football.getContentSize().width/2;

var size= target.getContentSize();

//矩形区域

var rect=cc.rect(target.getPositionX()-size.width/2,target.getPositionY()-size.height/2,size.width,size.height);

//矩形的四个顶点

var MaxX= cc.rectGetMaxX(rect);

var MinX= cc.rectGetMinX(rect);

var MaxY= cc.rectGetMaxY(rect);

var MinY= cc.rectGetMinY(rect);

var MidX= cc.rectGetMidX(rect);

var MidY= cc.rectGetMidY(rect);

//横向矩形内

//触碰左边

if((point.y>MinY)&&(point.y<MaxY)){

if((point.x > (MinX-radius)) && (point.x < MidX)){//右边

return true;

}else if((point.x > MidX) && (point.x < (MaxX+radius))){//左边

return true;

}

}

//纵向矩形内

//触碰下边

if((point.x>MinX)&&(point.x<MaxX)){

if((point.y > (MinY-radius)) && (point.y < MidY)){//上边

return true;

}else if((point.y > MidY) && (point.y < (MaxY+radius))){//下边

return true;

}

}

}

* + Cocos矩形中，常用的矩形函数：

cc.rectContainsPoint(rect,point) 判断某个矩形中是否包含某个点，若包含，返回true

c.pointEqualToPoint(point1,point2) 判断两个点是否相等，相等返回true

cc.rectEqualToRect(rect1,rect2)判断两个矩形是否相等

cc.\_rectEqualToZero(rect)判断该矩形是否为0矩阵

cc.rectOverlapsRect(rectA,rectB)判断两个矩形是否部分重叠

cc.rectIntersection(rectA,rectB)返回两个矩形重叠部分

cc.rectUnion(rectA,rectB)返回包含这两个矩形的最小矩形

* 物体的包围盒

利用高中所学平面几何的数学知识，就能实现的碰撞检测。只能适用于游戏中的物体可以被近似为矩形。或者虽然不是矩形，但是碰撞精度要求不高的情况下，每个物体记录一个能够将自己框住的最小矩形的左上角坐标和矩形宽高。

碰撞检测中一个最基本的概念就是包围盒(bounding box)。所谓包围盒，就是指用一种规则的几何形状代表物体的边缘。这样可以简化物体的边缘，从而减少碰撞检测的计算。

原理：

由于游戏中的物体外形千差万别、形态各异，要想进行精确的碰撞检测的运算，这将是一个非常耗费性能的事。尤其是在性能受限的设备上，这是一件非常困难的事。因此开发者想到了包围盒。它能够将原本复杂的物体形状，通过简单的几何图形包围起来。

说明：虽然包围盒能够简化碰撞检测的预算，但是同时也会损失检测的精度。

点与矩形的碰撞检测方法，虽然很简单，但它的实用范围很小，因为多数游戏中的元素，其外形边缘都会比较复杂。如果只是单一使用矩形来进行碰撞检测，还是很难达到游戏产品的要求的。因此开发者选择合适的几何形状作为物体的包围盒。为了能够提供更加精确的方法，开发者需要选择最合适的包围盒。

注意：除非游戏物体是一块豆腐或者板砖，否则无论采用怎样的包围盒，总会存在一些空白区域，影响碰撞检测的准确性。

虽然球形检测在某些情况下提高了精度，但却损失了计算的速度。点与圆碰撞检测，需要用到平方与开方的数学方法。这对于一些不支持浮点运算的手持设备来说，将是非常大的开销。游戏开发者一直以来都在精确度与运算性能之间进行选择。

* 像素碰撞

实要精确的多，只有存在像素接触的时候才会判断，但是，像素判断要不停的判断每个像素有没有接触，非常消耗CPU资源。所以我们一般不选用。

* AABB碰撞检测技术

1. 获取矩形的X或Y的最大值、最小值

//最小值==左上角的顶点

rect.x

rect.y

//最大值==右下角的点

rect.x+rect.width

rect.y+rect.height

1. Cocos中，关于点的操作

* 计算关于原点对称的点：

cc.pNeg = function (point) {

return cc.p(-point.x, -point.y);

};

* 两个向量的加法：

cc.pAdd = function (v1, v2) {

return cc.p(v1.x + v2.x, v1.y + v2.y);

};

* 两个向量的减法：

cc.pSub = function (v1, v2) {

return cc.p(v1.x - v2.x, v1.y - v2.y);

};

* 数乘向量

cc.pMult = function (point, floatVar) {

return cc.p(point.x \* floatVar, point.y \* floatVar);

};

* 两向量的中心点

cc.pMidpoint = function (v1, v2) {

return cc.pMult(cc.pAdd(v1, v2), 0.5);

};

* 两个向量的数量积

a•a=|a|的平方。

cc.pDot = function (v1, v2) {

return v1.x \* v2.x + v1.y \* v2.y;

};

* 向量叉积

cc.pCross = function (v1, v2) {

return v1.x \* v2.y - v1.y \* v2.x;

};

* 关于Y轴对称

cc.pPerp = function (point) {

return cc.p(-point.y, point.x);

};

* 关于X轴对称

cc.pRPerp = function (point) {

return cc.p(point.y, -point.x);

};

* 两个向量之间的角度

cc.pAngle = function (a, b) {

var angle = Math.acos(cc.pDot(cc.pNormalize(a), cc.pNormalize(b)));

if (Math.abs(angle) < cc.POINT\_EPSILON) return 0.0;

return angle;

};

* 角度与弧度之间的转换

//角度🡪弧度

cc.degreesToRadians = function (angle) {

return angle \* cc.RAD;

};

其中cc.RAD= cc.PI / 180;

//弧度🡪角度

cc.radiansToDegrees = function (angle) {

return angle \* cc.DEG;

};

其中，cc.DEG = 180 / cc.PI;

1. cocos-JS版本，将未压缩版本改为压缩版本的方法步骤：

* 使用压缩工具，将项目所有的代码压缩成game.min.js;
* 将游戏首页中的引擎引用

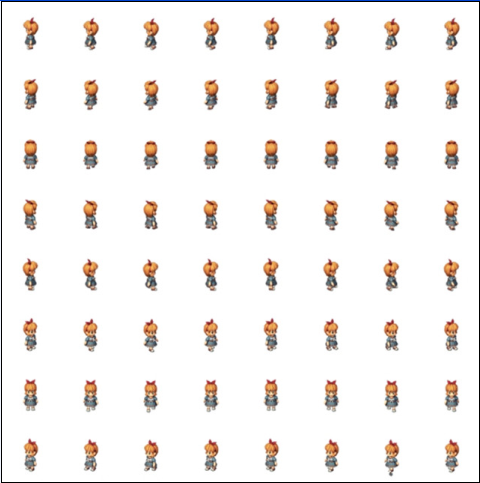
<script type="text/javascript" src="src/cocos2d-js-v3.1-lite.js" charset="UTF-8"></script>

//改为

<script type="text/javascript" src="game.min.js" charset="UTF-8"></script>

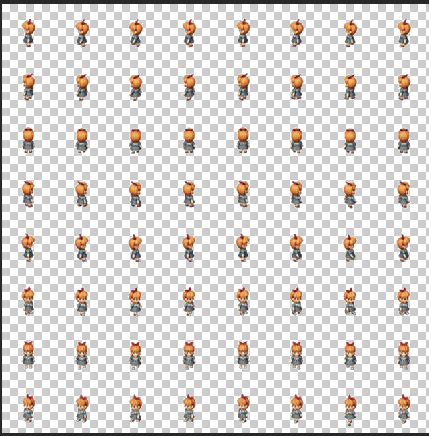
* 清空project.json中"jsList"字段：【注意：此文件一定要修改，否则即便修改了上两步，游戏仍然使用的是未压缩版】

1. 将下图改为透明背景



打开PS🡪Ctrl+J新建一个副本，避免直接在原图上修改🡪打开左侧工具栏的的第四个图标，长按 选择魔棒工具🡪选中整张图，按下退格键，保存为\*.png文件即可。

结果：



问题：有时候，将部分精灵同时也删除掉了

解决：将容差值改小一点，例如5等等。

1. Tiled Map瓦片地图

TMX地图格式是Tiled编辑器采用的一种灵活的基于瓦片的地图描述格式。用它来描述的地图有以下特点：任意数量的图块，任意数量的层（layer），任意数量的图块集（tile sets），还有附加于这些元素（elements）之上的用户自定义属性。除了地图的多分层控制之外，它也提供了可以自由放置的对象组（Objects Group）。

* \*.tmx文件解析【4\*4的地图】

<?xml version="1.0" encoding="UTF-8"?>

<map version="1.0" orientation="orthogonal" width="4" height="4" tilewidth="32" tileheight="32">

<tileset firstgid="1" name="block" tilewidth="32" tileheight="32" spacing="2" margin="2">

<image source="block.png" width="640" height="400"/>

</tileset>

<layer name="块层 1" width="4" height="4">

<data encoding="base64" compression="zlib">

eJyTZGBgkAJiaQYI0AZiHSDWZUAAOwZUgMwHADLYAU8=

</data>

</layer>

</map>

参数说明：

其实\*.tmx文件，就是一个XML文件。<map></map>标签元素之内的东西，就是Tiled Map编辑器制作的瓦片地图。

* version：在<map>标签中，指示TMX格式版本号，一般为1.0。
* orientation：地图朝向，目前支持“正交——orthogonal”和“45度等距——isometric”两种方式。可以在创建地图的时候指定：文件🡪新文件🡪在地图方向下拉列表中，选择即可。
* width：地图的宽度（tile个数。isometric则是斜边的tile个数）
* height：地图的高度宽度（tile个数）。可以在创建地图的时候指定：文件🡪新文件🡪地图大小🡪宽度和高度
* tilewidth：单个块tile的宽度
* tileheight：单个块tile的高度。可以在创建地图的时候指定：文件🡪新文件🡪块大小🡪宽度和高度。

tilewidth与tileheight两个属性决定了地图网格的大小，个别tiles允许有不同的大小，所有的Tile将靠挂在左下角，超出的部分将在顶部和右侧延伸。

<tileset>

* firstgid：

此图块集的第一个图块在全局图块集中的位置。

* source：

图像来源：如果此图块集来自于一个外部图块定义文件，此属性的值为该图块定义文件路径。TSX文件同样拥有相同的属性结构。在TSX文件中没有firstgid与source属性，如果这个图块集定义文件中的图块在地图中被使用，这两个属性将在TMX文件之中被设置。

* name：

图块集的名称。可以在创建地图的时候指定：视图🡪图块🡪右下角🡪重命名。

* tilewidth:

图块的宽度

* tileheight：

图块的高度。设置：地图🡪新图块🡪块宽度和块高度

* spacing：

图块的间距，在原图上采样图块时，图块与图块之间的间隔。设置：地图🡪新图块🡪间距

* margin：

图块的边距，在原图上采样图块时，图像左侧与上方采样的剔除边界大小。设置：地图🡪新图块🡪边距

<image>

Tiled Map自动填写该标签。

* format:

嵌入图片时所使用的格式，TiledQT版本当前不支持此属性。

* id:

在Tiled Java的某些版本中被使用，TiledQT版本当前不支持此属性。

* source:

图块集对应图像文件的路径。（Tiled支持大多数常见图像格式：bmp,gif,jpeg,jpg,pbm,pgm,png,ppm,tif,tiff,xbm,xpm）

* trans:

定义一种用于表示半透的颜色，将原图像中所有相同颜色值的alpha值改为0。

* width：

图像的宽度

* height：

图像的高度

<layer>

* name:

图层的名称。设置：图层🡪图层重命名

* x:

该层的X坐标（单位为图块）。默认为0，该值在QT版本的Tiled编辑器中不允许修改。

* y:

该层的Y坐标（单位为图块）。默认为0，该值在QT版本的Tiled编辑器中不允许修改。

* width:

该层的宽度（单位为图块）。历史版本要求存在，该值在QT版本的Tiled编辑器中与地图的宽度相同。

* height:

该层的高度（单位为图块）。历史版本要求存在，该值在QT版本的Tiled编辑器中与地图的高度相同。

* opacity:

不透明度，范围0~1,0为全透，1为不透。

* visible:

是否可见，1可见，0不可见，默认值为1。

<data>

* encoding:

图层数据的编码方式，当前提供两种“base64”和“csv”。

Base64是一种通用的方法，其原理很简单，就是把三个Byte的数据用4个Byte表示。在这四个Byte中，实际用到的都只有前面6bit，这样就不存在只能传输7bit的字符的问题了。Base64的缩写一般是“B”。

　　CSV逗号分隔值文件（COMMA Separated value）,是一种用来存储数据的纯文本文件格式。

* compression:

图层数据的压缩方式，QT版本的Tiled编辑器支持gzip或者zlib方式的数据压缩。设置：编辑🡪参数🡪另存为数据层下拉菜单中，选择【

下面的一个Base64 (zlib压缩)🡪压缩;

上面的一个Base64 (zlib压缩)🡪不压缩;】。

<tile>

* gid:

图块的全局ID。不用因为图块存在于图块集中而感到困惑，这个元素定义了可用于图层的一个基本图块。这是最有效率的层数据存储方式，通常应该避免。

<objectgroup>

对象组事实上就是一个图层，因此在TiledQt版本中我们也称之为对象层。可以包含下列元素：属性，对象。

* name:

对象组的名称。

* color:

当前对象组的颜色。

* x:

对象组的X坐标。默认为0，TiledQT版本此属性无法修改。

* y:

对象组的Y坐标。默认为0，TiledQT版本此属性无法修改。

* width:

对象组的宽度，当前无意义。

* height:

对象组的高度，当前无意义

* opacity:

对象组的图透明度0~1，默认为1不透。

* visible:

对象组的可见性 1显示 0隐藏，默认为1。

<object>

* name: 对象名称。字符串表示
* type: 对象类型。字符串表示
* x: 对象的X坐标，单位像素
* y: 对象的Y坐标，单位像素
* width: 对象的宽度，单位像素
* height: 对象的高度，单位像素
* gid: 图块的索引（可选）
* visible：可见性，1显示0隐藏，默认为1

<polygon>

多边形对象由一系列空格符分隔的坐标点组成。这些坐标点的原点是父对象的位置。默认情况下，第一个点的坐标为0,0 指示对象的存放位置。

* points: 一系列X,Y坐标点（单位像素）。

<polyline>

多段线段

* points：一系列X,Y坐标点（单位像素）。

<properties>

可以包含任意数量的自定义属性。可以用来作为地图，图块，层，对象组，对象的子元素。

* name: 属性名
* value: 属性值
* Tiled Map地图

cc.TMXTiledMap 知道如何解析并渲染一个TMX地图。将cc.TMXTiledMap加入支持TMX 它支持等距斜视(isometric),六边形(hexagonal),直角鸟瞰(orthogonal)tiles。也支持对象组,多对象以及多属性。

特性如下:

- 每个tile都被当作一个cc.Sprite - 这些精灵在需要时被创建而且仅当调用"layer.tileAt(position)"时才会被创建

- 由于每个tile是一个cc.Sprite，所以它们可以旋转/移动/缩放/着色/“透明化”

- Tiles可以在运行时添加或删除

- Tiles的z-order亦可在运行时修改

- 每个tile的锚点是(0,0)

- TMXTileMap的锚点是(0,0)

- TMX layers可以当作子节点添加

- TXM layers默认会设置一个别名

- Tileset图片可以在使用cc.TextureCache加载

- 每个tile都有一个唯一的tag

- 每个tile都有一个唯一的z值.

常用方法：

getLayer(layerName)

通过layerName获取对应的TMXLayer，其中layerName是Tiled Map编辑器中右边中的"Layer 0"，返回值为cc.TMXLayer对象：



cc.TMXLaye——用来表示TMX格式的layer，它是cc.SpriteBatchNode的一个子类。

cc.TMXLayer用来表示TMX格式的layer。

它是cc.SpriteBatchNode的一个子类. 默认情况下，通过使用一个cc.TextureAtlas来渲染瓦片.

如果运行时修改一个tile，那么tile将变成一个cc.Sprite对象，反之，则不会有Sprite对象被创建。

使用cc.Sprite对象作为tiles有如下好处：

- tiles(即cc.Sprite)可以通过完善的API进行旋转/缩放/移动

其中，该层的宽和高，就是对应的瓦块地图二维数组的行数和列数。

常用方法：

1、{cc.Size} getLayerSize()

获取layer的尺寸。

2、{cc.Sprite} getTileAt(pos, y)

返回指定坐标处的瓦片精灵tile(cc.Sprite)

返回的cc.Sprite已经是cc.TMXLayer的Child,不要再添加这个cc.Sprite可以像别的cc.Sprite一样, 旋转,缩放,透明, 变色, 等等

也可以删除通过调用

- layer.removeChild(sprite, cleanup);

- or layer.removeTileAt(ccp(x,y));

3、removeChild(sprite, cleanup)

删除Child

参数：cleanup,默认为true。

4、removeTileAt(pos, y)

通过给定的坐标删除一个tile

* 资源一：\*.tsx、\*tmx和\*.png
  + - 首先加载以上三种资源,即便是不显性使用，也要使用预加载功能，加载到内存中；
    - 创建对象：

var size= this.layerGradient.getContentSize();

var map = new cc.TMXTiledMap(res.orthogonal\_test1\_tmx);

map.attr({

anchorX:0.5,

anchorY:0.5,

x:size.width/2,

y:size.height/2

});

this.layerGradient.addChild(map,1,10);

* 资源二：\*.tmx和\*.png
* 首先加载以上资源；
* 创建对象：

var size= this.layerGradient.getContentSize();

var map = new cc.TMXTiledMap(res.orthogonal\_test2\_tmx);

map.attr({

anchorX:0.5,

anchorY:0.5,

x:size.width/2,

y:size.height/2

});

this.layerGradient.addChild(map,1,10);

* 示例：放大(1,3),移除（1，0）

initTiledMap:function(){

var size= this.layerGradient.getContentSize();

var map = new cc.TMXTiledMap(res.man\_tmx);

map.attr({

anchorX:0.5,

anchorY:0.5,

x:size.width/2,

y:size.height/2

});

this.layerGradient.addChild(map,1,10);

this.runAction\_Self(map);

},

runAction\_Self:function(map){

var layer = map.getLayer("Layer 0");

var size = layer.getLayerSize();

//action

var scaleY = cc.scaleTo(1,1,2);

//sprite——scaleY

var sprite1= layer.getTileAt(cc.p(1,3));

sprite1.runAction(scaleY);

//sprite——remove

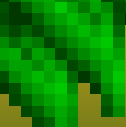
var sprite2= layer.getTileAt(cc.p(1,0));

layer.removeChild(sprite2,true);

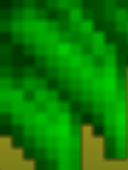
}

* 示例：对每一个纹理节点设置《反锯齿》效果

默认——锯齿



反锯齿



源码：

if ("opengl" in cc.sys.capabilities)

layer.texture.setAntiAliasTexParameters();

* 示例：移除整行，更新图块，重绘整行

initTiledMap:function(){

var size = this.layerGradient.getContentSize();

var centerPos = cc.p(size.width/2,size.height/2);

var map = new cc.TMXTiledMap(res.man\_tmx);

map.attr({

anchorX:0.5,

anchorY:0.5,

x:centerPos.x,

y:centerPos.y

});

this.layerGradient.addChild(map,1,10);

//对每一个纹理节点，设置《反锯齿》效果

var layer = map.getLayer("Layer 0");

if ("opengl" in cc.sys.capabilities)

layer.texture.setAntiAliasTexParameters();

this.removeAllRow();

this.gid=0;

this.schedule(this.updateAllRow,1);

this.schedule(this.repaintAllCol,1);

},

removeAllRow:function(){

//map

var map = this.layerGradient.getChildByTag(10);

var layer = map.getLayer("Layer 0");

for(var i=0;i<layer.getLayerSize().width;++i){

layer.removeTileAt(cc.p(i,1));

}

},

updateAllRow:function(){

//map

var map = this.layerGradient.getChildByTag(10);

var layer = map.getLayer("Layer 0");

for(var i=0;i<layer.getLayerSize().height;++i){

layer.setTileGID(this.gid,cc.p(i,2));

}

this.gid++;

},

repaintAllCol:function(){

//map

var map = this.layerGradient.getChildByTag(10);

var layer = map.getLayer("Layer 0");

for(var i=0;i<layer.getLayerSize().height;++i){

var gid = layer.getTileGIDAt(cc.p(i,layer.getLayerSize().width-1));

layer.setTileGID(gid+1,cc.p(i,layer.getLayerSize().width-1));

}

}

* 说明：其中\*.tmx文件中，orientation="hexagonal"。hexagonal指的是：六边形地图。虽然在Tile Map编辑器中，不能编辑六边形地图，但是可以使用cc.TMXTiledMap显示出来。

//随时暂停或恢复动作pauseTarget函数没有作用

雾化效果

//类的作用

cc.LayerGradient();

cc.LayerMultiplex();

cc.LayerDefineForWebGL();

//追踪导弹原理

movePlayer:function(pos){

var player= this.getChildByTag(2);

player.stopAllActions();

var o = pos.x - player.x;

var a = pos.y - player.y;

var at = Math.atan(o / a) \* 57.29577951; // radians to degrees

//skew动作 函数作用 参数

//人体时钟

//hone hone clock

//vector

//坐标系

// cc.EventMouse.BUTTON\_LEFT

//JQeure

按钮变化的三种状态、效果：

//正常【有阴影效果】

//选中【取消阴影、向下偏移、改变颜色、放大】

//不可用【颜色变灰】

//动作的暂停与恢复微信消息

//原理是是什么？var point= draw2.convertToNodeSpace(draw1.getPosition());

