PhoneGap

笔记本： Android
创建时间： 2015/12/15 14:54
标签： HTML5

PhoneGap
2015年12月15日
14:54

1，PhoneGap API

PhoneGap的缺点：

1. 运行效率基于webkit的处理速度，比原生的慢
2. 不能完全调用原生的API特性
3. 不同平台的节目差异化不足，因为基于html界面看起来一样

1，PhoneGap基本工程

1，在assets中建立www目录，添加cordova.js，jquery.js，jquery-mobile-min.js以及自己编写的js文件，如main.js。添加jquery-mobile-min.css以及自己编写的css。

在界面布局html中做如下的引用：

```
  <meta charset="utf-8"/>

<link rel="stylesheet" href="css/jquery.mobile-1.4.5.min.css">

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

<script type="text/javascript" charset="utf-8" src="js/cordova.js">

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

<script src="js/jquery.mobile-1.4.5.min.js"></script>

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

要是页面加载时就执行某个javascript命令，就需要在body中过设置onload属性为某个函数

2，引用cordova.jar文件，保证activity可以正常的继承DroidGap类。在activity中加载html文件：

```
public class MainActivity extends DroidGap {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    super.loadUrl("file:///android_asset/www/index.html");
  }}
```

在html布局中，可以使用 设置一个完整的显示页面，要跳转到该页面设置一个超链接就可以产生和intent一样的作用：

```
<a href="#index" data-transition="slide">

<img src="images/button.jpg" width="80" height="40" border="0">

</a>
```

要返回之前的页面则直接跳转到#即可，不能使用#id，不然会跳到新的id页面：

```
<a href="#" data-rel="back">

<img src="images/button.jpg" width="80" height="40" border="0">
```


可以使用如下方式解决分辨率自适应的问题，自动的进行缩放：

```
<meta name="viewport"
content="target-densitydpi=device-dpi,width=device-width,initial-scale=1,
minimum-scale=0.1,maximum-scale=1" />
```

相关参数设置说明如下。

- target-densitydpi=device-dpi：以设备标准 dpi 作为标准，以下各要素均以此作为标准。
- width=device-width：使用设备的宽度。
- initial-scale=1：定义屏幕缩放比，1 为不缩放。
- minimum-scale=0.1：最小放大倍数，有些手机默认不支持缩小显示，直接设置 initial-scale 为 0.5 这样的缩小倍数是不支持的，因此要定义一下。

API可以调用移动设备的原生特性进行移动APP开发，当前phonegap可以使用除蓝牙外的原生功能特性。以下时phonegap支持的原生功能特性：

- 本地事件处理：通过 JavaScript 调用原生事件。
- 加速器传感器：调用设备的运动传感器。
- 摄像头：通过设备的摄像头拍照。
- 多媒体采集：通过设备的媒体采集应用程序来采集媒体文件。
- 指南针：获取设备指向的方向。
- 网络连接：获取网络连接状态（Wi-Fi 或移动网络）。
- 联系人：访问与操作手机通讯录。
- 设备：获取设备软、硬件信息。
- 文件：通过 JavaScript 对设备文件系统进行处理。
- 地理位置：调用设备传感器的 GPS 地理位置服务。
- 多媒体：播放音频与录制音频。
- 通知：调用设备的视觉、听觉和触觉通知。
- 存储：通过 JavaScript 对设备进行数据存储操作。

2，本地事件处理API

1，deviceready

在使用API之前，首先需要监听deviceready事件，以便确认phonegap加载完成。当phonegap加载完成并且开始和本地设备进行通信时就会触发该事件：

```
<script type="text/javascript" charset="utf-8" src="../js/cordova.js"></script>

<script type="text/javascript" charset="utf-8">

//监听deviceready事件

document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){

alert("加载完成");

}

</script>
```

2，pause和resume

当phonegap程序被放到后台的时候会触发pause事件，可以用于保存当前程序状态：

```
function onDeviceReady() {
        document.addEventListener("pause", onPause, false);
}
// 处理 pause 事件
function onPause() {
}
```

当phonegap程序恢复到前台运行时触发resume事件，可用于恢复程序状态：

```
function onDeviceReady() {
        document.addEventListener("resume", onResume, false);
```

3，online和offline

当phonegap连接到互联网时触发的事件：

```
document.addEventListener("online", yourCallbackFunction, false);
```

当phonegap网络中断时触发offline事件：

```
document.addEventListener("offline", yourCallbackFunction, false);
```

4，系统button触发

当用户点击android中的后退按钮时触发的事件为**backbutton**：

```
document.addEventListener("backbutton", yourCallbackFunction, false);
```

当用户点击android中的菜单按钮时触发的事件为**menubutton**：

```
document.addEventListener("menubutton", yourCallbackFunction, false);
```

当用户点击android中的搜索按钮时触发**searchbutton**事件；

当用户点击“通话”按钮时会触发**startcallbutton**事件；

当用户点击“挂断”按钮时会触发**endcallbutton**事件；

当用户点击“减小音量”按钮时会触发**volumedownbutton**事件；

当用户点击“增大音量”按钮时会触发**volumeupbutton**事件；

5，battery相关

当phonegap程序发现电池电量到临界值时触发的事件**batterycritical**：

```
window.addEventListener("batterycritical", yourCallbackFunction, false);
```

该事件触发后会传递一个info对象给监听程序，info的主要参数是：level——电量剩余百分比；isPlugged——是否接通电源

```
// 处理 batterycritical 事件
function onBatteryCritical(info) {
    alert("Battery Level Critical " + info.level + "%\nRecharge Soon!");
}
```

与batterycritical同样用法的还有**batterylow**事件——电量降低单一个较低水平时触发，info参数一致。

监听电池状态发生变化可以使用**batterystatus**事件，当电量发生至少1%的改变时就会触发，也会返回一个和上面一样的info对象，其用法一样。

3，传感器

1，指南针

1，获取指南针当前朝向

使用 compass.getCurrentHeading 可以获取指南针的当前朝向。相关代码如下所示：

```
navigator.compass.getCurrentHeading(compassSuccess, compassError, compassOptions);
```

指南针是一个检测设备方向或朝向的传感器，它使用度作为衡量单位，取值范围是 0°~359.99°。这里通过 compassSuccess 回调函数返回指南针朝向相关数据。

```
<script type="text/javascript" charset="UTF-8">
    document.addEventListener("deviceready",onDeviceReady(),flase);
    function onDeviceReady(){
      navigator.compass.getCurrentHeading(onSuccess,onError);
    }
    //成功则显示当前方位数
    function onSuccess(heading){
      alert("Heading:"+heading.magneticHeading);
    }
    //失败
    function onError(compassError){
      alert("compass error"+compassError.code);
    }
  </script>
```

2，周期性获取指向

使用 compass.watchHeading 在固定的时间间隔获取指南针朝向的角度。相关代码如下所示：

```
var watchID = navigator.compass.watchHeading(compassSuccess,compassError, [compassOptions]);
```

compass.watchHeading 每隔一个固定时间就获取一次设备的当前朝向。每次取得朝向后，headingSuccess 回调函数会被执行。通过 compassOptions 对象的 frequency 参数可以设定以毫秒为单位的时间间隔。

返回的 watch ID 是指南针监听周期的引用，可以通过 compass.clearWatch 调用该 watch ID 以停止对指南针的监听。

3，指南针度数改变时获取度数

当罗盘改变一定度数时，使用 compass.watchHeadingFilter 获取指南针的朝向度数。相关代码如下所示：

```
var watchID = navigator.compass.watchHeadingFilter(compassSuccess,
compassError, compassOptions);
```

compass.watchHeadingFilter 方法获取当设备朝向发生一个指定值的改变后的朝向。每次朝向的改变大于或者等于某个指定值时，headingSuccess 回调函数就会被调用。特定的度数值将通过 compassOptions 对象的 filter 参数指定。

返回的 watchID 引用指向指南针的监听间隔，compass.clearWatchFilter 方法能使用 watchID 停止对指南针改变特定度数的监听。每次只有一个 watchHeadingFilter 是有效的，如果 watchHeadingFilter 是有效的，调用 getCurrentHeading 或者 watchHeading 方法时会使用有效的过滤值来监听指南针的改变。在 iOS 平台上，这个方法比 iOS 制造商提供的 compass.watchFilter() 方法更加有效。

当前只适用于iphone

2，加速计传感器

加速计用于捕获设备运动过程中 x、y、z 坐标中的位置数据，然后使用这些数据推断设备的运动状态或方向。

1，获取坐标位置

```
<script type="text/javascript" charset="utf-8">

//监听deviceready事件
```

```
document.addEventListener("deviceready",onDeviceReady,false);
function onDeviceReady(){
//设备加载完成后，获取xyz坐标值
navigator.accelerometer.getCurrentAcceleration(onSuccess,onError);
}
//获取数据成功
function onSuccess(acceleration){
alert('Acceleration X:'+acceleration.x+'\n'+
'Acceleration Y:'+acceleration.y+'\n'+
'Acceleration Z:'+acceleration.z+'\n'+
'Timestamp'+acceleration.timestamp+'\n'
);
}
//获取数据失败
function onError(){
alert('onError');
}
</script>
```

2，周期性获取坐标位置

```
//每隔3秒读取一次位置
var options={frequency:3000};
navigator.accelerometer.watchAcceleration(onSuccess,onError,options);
```

Success和error的处理和上面一致

要取消周期性的读取使用下面程序：

```
Navigator.accelerometer.clearWatch(watchID);
```

4，多媒体

1，拍照

```
navigator.camera.getPicture( cameraSuccess, cameraError, [ cameraOptions ] );
```

camera.getPicture 函数用于打开设备的默认摄像头应用程序，以便用户进行拍照（设置 Camera.sourceType = Camera.PictureSourceType.CAMERA，默认值）。一旦拍照完毕，摄像头应用程序会关闭并返回到你的应用程序。

如果设置 Camera.sourceType = Camera.PictureSourceType.PHOTOLIBRARY 或者 Camera.PictureSourceType.SAVEDPHOTOALBUM，系统将显示照片选择对话框，用户可以从相册中选择照片。其返回值将会按照用户通过 cameraOptions 参数所设定的格式发送给 cameraSuccess 函数，包括：

拍照——调用系统拍照程序

```
navigator.camera.getPicture(onPhotoURISuccess,onFail,{qulity: 50});
//当成功获取照片时以URI的格式显示出来
  function onPhotoURISuccess (imageURI) {
```

```
    var largeImage=document.getElementById("largeImage");
    largeImage.style.display="block";
    largeImage.src=imageURI;
  }
```

读取照片库——调用系统照片库

```
navigator.camera.getPicture(onPhotoURISuccess,onFail,{qulity: 50,destinationType:
destinationType.FILE_URI,sourceType: source});
```

除了使用camera对象还可以使用capture，同样可以系统的拍照程序完成图片采集：

使用 capture.captureImage 方法，可以从摄像头应用程序中采集与保存大量的图片。相关代码如下所示：

```
navigator.device.capture.captureImage( CaptureCB captureSuccess,
CaptureErrorCB captureError, [CaptureImageOptions options]);
```

该方法通过设备的摄像头应用程序开始一个异步操作，从而采集图像。该操作允许设备用户在一个会话中同时采集多个图像。

当用户退出摄像头应用程序，或程序到达 CaptureImageOptions 的 limit 参数所设置的最大图片数时将会停止采集操作。如果没有设置 limit 参数的值，则使用其默认值 1，也就是说，当用户采集到一个图像后采集操作就会终止。

当采集操作结束时，系统会调用 CaptureCB 函数，并传递一个包含每个采集到的图像文件的 MediaFile 对象数组给该函数。如果用户在完成一个图像采集之前终止采集操作，系统会调用 CaptureErrorCB 函数，并传递一个包含 CaptureError.CAPTURE_NO_MEDIA_FILES 错误代码的 CaptureError 对象给该函数。

下面看一下其中涉及的可选参数。

- limit: 采集图像的数量，该值必须设定为大于或等于 1（默认值为 1）。
- mode：选定的图片采集模式，该值必须与 capture.supportedImageModes 的值匹配。

```
//当采集完时调用
  function captureSuccess(mediaFiles){
    var i,len;
    for(i=0,len=mediaFiles.length;i<len;i++){
      uploadFile(mediaFiles);
    }
  }
  //当采集发生异常时调用
  function captureError(error){
    var msg='发生错误：'+error.code;
    navigator.notification.alert(msg,null,"Uh oh!");
  }
  //单击采集图片按钮触发
  function captureImage(){
    //调用摄像头程序，并且最多允许采集两张图片
    navigator.device.capture.captureImage(captureSuccess,captureError,{limit:2});
  }
  //保存图片
```

```
function uploadFile(mediaFiles){
  //需要补充javascript保存文件的代码
  alert("我会保存图片的");
}
```

2，capture录像

要想使用设备录制视频，可以使用 capture.captureVideo 方法。相关代码如下所示：

```
navigator.device.capture.captureVideo(CaptureCB captureSuccess,
CaptureErrorCB captureError, [CaptureVideoOptions options]);
```

该方法通过设备的视频录制应用程序开始一个异步操作，从而进行视频采集。该操作允许设备用户在一个会话中同时采集多个视频。

当用户退出视频录制应用程序，或程序到达 CaptureVideoOptions 的 limit 参数所设置的最大录制数时将会停止采集操作。如果没有设置 limit 参数的值，则使用其默认值 1，也就是说，当用户录制了一个视频剪辑后采集操作就会终止。

当采集操作结束时，系统会调用 CaptureCB 函数，并传递一个包含每个采集到的视频剪辑文件的 MediaFile 对象数组给该函数。如果用户在完成一个视频剪辑采集之前终止了采集操作，系统会调用 CaptureErrorCB 函数，并传递一个包含 CaptureError.CAPTURE_NO_MEDIA_FILES 错误代码的 CaptureError 对象给该函数。

其中所涉及的可选参数如下：

- limit：在单个采集操作期间能够采集的视频剪辑数量的最大值，该值的设定必须大于或等于 1（默认值为 1）。
- duration：一个视频剪辑的最长时间，单位为秒。
- mode：选定的视频采集模式，该值必须与 capture.supportedVideoModes 的值匹配。

调用系统录像程序：

```
<script type="text/javascript" charset="utf-8">
  //当操作操作完成时调用
  function captureSuccess(mediaFiles){
    var i,len;
    for(i=0,len=mediaFiles.length;i<len;i++){
      uploadFile(mediaFiles);
    }
  }
  //当采集发生异常时调用
  function captureError(error){
    var msg='发生错误：'+error.code;
    navigator.notification.alert(msg,null,"Uh oh!");
  }
  //单击采集视频按钮触发
  function captureVideo(){
   navigator.device.capture.captureVideo(captureSuccess,captureError,{limit:2});
  }
  //保存视频文件
  function uploadFile(mediaFiles){
```

```
    //需要补充javascript保存文件的代码
  }
</script>
```

3，capture录音

使用 capture.captureAudio 方法，可以从录音应用程序中采集多个音频剪辑文件。相关代码如下所示：

```
navigator.device.capture.captureAudio( CaptureCB captureSuccess,
CaptureErrorCB captureError,  [CaptureAudioOptions options]);
```

该方法通过设备的录音应用程序开始一个异步操作，从而采集录音。该操作允许设备用户在一个会话中同时采集多个录音。

当用户退出录音应用程序，或程序到达 CaptureAudioOptions 的 limit 参数所设置的最大录音数时将会停止采集操作。如果没有设置 limit 参数的值，则使用其默认值 1，也就是说，当用户采集到一个音频剪辑后采集操作就会终止。

当采集操作结束后，系统会调用 CaptureCB 函数，并传递一个包含每个采集到的音频文件的 MediaFile 对象数组给该函数。如果用户在完成一个音频采集之前终止采集操作，系统会调用 CaptureErrorCB 函数，并传递一个包含 CaptureError.CAPTURE_NO_MEDIA_FILES 错误代码的 CaptureError 对象给该函数。

capture.captureAudio 方法中涉及的可选参数如下。

- limit: 表示在单个采集操作期间能够采集的音频剪辑数量的最大值，该值必须设定为大于或等于 1（默认值为 1）。
- duration：一个音频剪辑的最长时间，单位为秒。
- mode：选定的音频采集模式，该值必须与 capture.supportedAudioModes 的值匹配。

```
//当采集完时调用
  function captureSuccess(mediaFiles){
    var i,len;
    for(i=0,len=mediaFiles.length;i<len;i++){
      uploadFile(mediaFiles);
    }
  }
  //当采集发生异常时调用
  function captureError(error){
    var msg='发生错误：'+error.code;
    navigator.notification.alert(msg,null,"Uh oh!");
  }
  //单击录音按钮触发
  function captureAudio(){
    //调用录音程序，并且最多允许采集两个录音
    navigator.device.capture.captureAudio(captureSuccess,captureError,{limit:2});
  }
  //保存录音
  function uploadFile(mediaFiles){
```

//需要补充javascript保存文件的代码

alert("我会保存录音的");

}

4，Media对象

```
var media = new Media(src, mediaSuccess, [mediaError]);
```

备注：Media的当前实现并没有遵守W3C媒体捕获的相关规范，目前只是为了提供方便。未来的实现将遵守最新的W3C规范并可能不再支持当前的APIs。

参数：

- src：一个包含音频内容的URI。（DOMString类型）
- mediaSuccess：（可选项）当一个Media对象完成当前的播放、录制或停止操作时触发的回调函数。（函数类型）
- mediaError：（可选项）当出现错误时调用的回调函数。（函数类型）
- mediaStatus：（可选项）当状态发生变化的时候调用的回调函数。（函数类型）

方法：

- media.getCurrentPosition：返回一个音频文件的当前位置。
- media.getDuration：返回一个音频文件的总时长。
- media.play：开始或恢复播放音频文件。
- media.pause：暂停播放音频文件。
- media.release：释放底层操作系统的音频资源。
- media.seekTo：在音频文件中移动到相应的位置。
- media.startRecord：开始录制音频文件。
- media.stopRecord：停止录制音频文件。
- media.stop：停止播放音频文件。

播放本地音乐的示例：

```
<script type="text/javascript" charset="UTF-8">
    var myMedia=null;
    var mediaTimer=null;
    var path=null;
    function playAudio(){
        //获取要播放的文件
        window.requestFileSystem(LocalFileSystem.PERSISTENT, 0, gotFS, Error);

        if(myMedia==null){//创建media对象
            myMedia=new Media(path,onSuccess,onError);
        }
        //播放音乐
        myMedia.play();
        //每一秒更新一次界面上显示的播放位置
        mediaTimer=setInterval(function(){
            //获取播放位置
            myMedia.getCurrentPosition(function(pos){
                if(pos>-1) setAudioPos((pos)+" sec");
            },function(error){
                setAudioPos("Error:"+error);
            });
        },1000);
    }
    function stopAudio(){
        if(myMedia)  myMedia.stop();
        clearInterval(mediaTimer);
        mediaTimer=null;
    }
    function pauseAudio(){
        if(myMedia)  myMedia.pause();
```

```
    }
    //其他函数
    function gotFS(fileSystem) {
        fileSystem.root.getFile("/sdcard/Music/Honor.mp3", null, gotFileEntry, Error);
    }
    function Error(evt) {
        alert(evt.target.error.code);
    }
    function gotFileEntry(fileEntry) {
        path=fileEntry.fullPath;
        console.log("文件路径看过来："+fileEntry.fullPath);
    }
    function onSuccess(){
        console.log("加载文件成功");
    }
    function onError(e){
        console.log("出现错误："+e);
    }
    function setAudioPos(pos){
        document.getElementById("audio_pos").innerHTML=pos;
    }
</script>
```

使用media录制声音的示例：

```
//使用media录制声音
function recordAudio(){
    var src="file:///sdcard/Music/record.mp3";
    var media_rec=new Media(src,onrecSuccess,onrecerror);
    //开始录制
    media  rec.startRecord();
    //10s后停止录制
    var rectime=0;
    var recinterval=setInterval(function(){
        rectime=rectime+1;
        setAudioRecPos("recTime:"+rectime);
        if(rectime>=10){
            clearInterval(recinterval);
            media_rec.stopRecord();
        }
    },1000);
}
```

5，网络

1，获取网络连接类型

Connection对象提供了对设备的蜂窝和wifi网络连接的访问，可以通过navigator.network接口获得该对象。Connection中可以获取到的状态信息如下：

```
function checkConnection(){

        var networkState=navigator.network.connection.type;

        var states={};
```

```
        states[Connection.UNKNOWN]='Unknown connection';
        states[Connection.ETHERNET]='Ethernet connection';
        states[Connection.WIFI]='Wifi connection';
        states[Connection.CELL_2G]='2G connection';
        states[Connection.CELL_3G]='3G connection';
        states[Connection.CELL_4G]='4G connection';
        states[Connection.NONE]='no network connection';
        alert("Connection type:"+states[networkState]);
    }
```

6，手机通讯录

Contacts对象提供了对通讯录数据库的访问

1，创建联系人

navigator.contacts.create();可以创建一个contact对象。调用contact.save()可以保存一个联系人。Contact对象包含的主要属性和属性类型如下：

- **id**: A globally unique identifier. *(DOMString)*
- **displayName**: The name of this Contact, suitable for display to end-users. *(DOMString)*
- **name**: An object containing all components of a persons name. *(ContactName)*
- **nickname**: A casual name by which to address the contact. *(DOMString)*
- **phoneNumbers**: An array of all the contact's phone numbers. *(ContactField[])*
- **emails**: An array of all the contact's email addresses. *(ContactField[])*
- **addresses**: An array of all the contact's addresses. *(ContactAddress[])*
- **ims**: An array of all the contact's IM addresses. *(ContactField[])*
- **organizations**: An array of all the contact's organizations. *(ContactOrganization[])*
- **birthday**: The birthday of the contact. *(Date)*
- **note**: A note about the contact. *(DOMString)*
- **photos**: An array of the contact's photos. *(ContactField[])*
- **categories**: An array of all the user-defined categories associated with the contact. *(ContactField[])*
- **urls**: An array of web pages associated with the contact. *(ContactField[])*

```
var myContact=navigator.contacts.create();
        //三种名称
        myContact.displayName="renxiuhu";
        myContact.nickname="xiuxiu";
        var name=new ContactName();
        name.givenName="Tiger";
        name.familyName="Ren";
        myContact.gender="male";//性别
        //添加号码
        var phoneNumbers = [];
        phoneNumbers[0] = new ContactField('work', '212-555-1234', false);
        phoneNumbers[1] = new ContactField('mobile', '917-555-5432', true); // preferred number
        phoneNumbers[2] = new ContactField('home', '203-555-7890', false);
        myContact.phoneNumbers=phoneNumbers;
        myContact.save(onError,onSuccess);//保存联系人
```

2，查询联系人

Navigator.contacts.find方法可以用于查找指定的联系人。Find有四个参数，分别是：查询条件，成功时调用方法，失败时调用方法，过滤选项。

```
var fields=["displayName","老婆"];//查询条件

var options=new ContactFindOptions();//过滤选项

options.filter="老";

navigator.contacts.find(fields,onSuccess1,onError,options);
```

3，联系人复制和删除

- clone：返回一个新的 Contact 对象，它是调用对象的深度副本，其 id 属性被设为 null。
- remove：从通讯录数据库中删除联系人。当删除不成功时，触发以 ContactError 对象为参数的错误处理回调函数。
- save：将一个新联系人存储到通讯录数据库中，如果通讯录数据库中已经包含与其 id 相同的记录，则更新已有记录。

删除：一般先查询要删除的联系人，然后调用remove

```
function onSuccess1(contacts){

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

   alert("DisplayName:"+contacts[i].displayName);

  }

   contacts[0].remove(onSuccess,onRemoveError);//删除联系人

}
```

Clone直接调用即可

7，设备信息

Device对象用于描述设备的硬件和软件信息。可以通过phonegap获取的信息有：设备型号，设备的phonegap版本，操作系统名称，操作系统版本号和UUID。

```
var element=document.getElementById("deviceinfo");

element.innerHTML="Device name:"+device.name+"<br/>"+

      "Device Phonegap:"+device.phonegap+"<br/>"+

       "Device Platform:"+device.platform+"<br/>"+

       "Device Version:"+device.version+"<br/>"+

       "Device UUID:"+device.uuid+"<br/>";
```

8，文件处理基本对象

1，File对象

File 包含了单独的文件属性，可以通过调用 FileEntry 对象的 file 方法获得一个 File 对象实例。具体内容如下。

- name：文件的名称（DOMString 类型）。
- fullPath：文件的完整路径，包含文件名称（DOMString 类型）。
- type：文件的 mime 类型（DOMString 类型）。
- lastModifiedDate：文件最后被修改的时间（日期类型）。
- size：以字节为单位的文件大小（长整型）。

2，FileReader对象：

FileReader 是一个允许用户读取文件的对象，它的属性如下。

- readyState: 当前读取器所处的状态，取值为 EMPTY、LOADING 和 DONE 三者之一。
- result：已读取文件的内容（DOMString 类型）。
- error：包含错误信息的对象（FileError 类型）。
- onloadstart：读取启动时调用的回调函数（函数类型）。
- onprogress：读取过程中调用的回调函数，用于提示读取进度（progress.loaded 和 progress.total）。
- onload：读取操作安全完成后调用的回调函数（函数类型）。
- onabort：读取被中止后调用的回调函数（函数类型）。
- onerror：读取失败后调用的回调函数（函数类型）。
- onloadend：请求完成后调用的回调函数（无论请求是成功还是失败）（函数类型）。

使用方法介绍如下。

- readAsDataURL：读取文件，结果以 Base64 编码的 data URL 形式返回（data URL 的格式由 IETF 在 RFC2397 中定义）。
- readAsText：读取文件，结果以文本字符串返回。
- abort：中止读取文件。

1，获取本地文件系统，获取到的文件系统将作为gotFS函数的参数

```
window.requestFileSystem(LocalFileSystem.PERSISTENT, 0, gotFS, Error);
```

2，通过本地文件系统可以访问文件，获取FileEntry对象，会作为函数参数

```
function gotFS(fileSystem) {
   fileSystem.root.getFile("/sdcard/readme.txt", null, gotFileEntry, Error);
 }
```

3，获取到fileenrty后使用file方法读取文件内容，参数分别时读取函数和错误函数

```
function gotFileEntry(fileEntry) {
        fileEntry.file(readDataUrl, Error);
}
```

4，获取文件内容的方法，有两种方式，URL和Text

```
    function readDataUrl(file) {
        var reader = new FileReader();
        reader.onloadend = function(evt) {
          console.log("Read as data URL");
          alert("as URL的输出："+evt.target.result);
        };
        reader.readAsDataURL(file);
     }


    function readAsText(file) {
        var reader = new FileReader();
        reader.onloadend = function(evt) {
          console.log("Read as text");
          alert("as Text的输出："+evt.target.result);
        };
```

```
        reader.readAsText(file);
    }
```

3，FileWrite对象

FileWriter 是允许用户写文件的对象，它的相关属性如下。

- readyState：当前写入器所处的状态，取值为 INIT、WRITING 和 DONE 三者之一。
- fileName：要写入的文件名称（DOMString 类型）。
- length：要写入文件的当前长度（长整型）。
- position：文件指针的当前位（长整型）。
- error：包含错误信息的对象（FileError 类型）。
- onwritestart：写入操作启动时调用的回调函数（函数类型）。
- onprogress：写入过程中调用的回调函数，用于提示写入进度（progress.loaded 和 progress.total）。
- onwrite：当写入成功完成后调用的回调函数（函数类型）。
- onabort：写入被中止后调用的回调函数（函数类型）。
- onerror：写入失败后调用的回调函数（函数类型）。
- onwriteend：请求完成后调用的回调函数（无论请求是成功还是失败）（函数类型）。

下面是关于它的一些主要方法。

- seek：移动文件指针到指定的字节位置。
- truncate：按照指定长度截断文件。
- write：向文件中写入数据。
- abort：中止写入文件。

1，获取文件系统：

```
window.requestFileSystem(LocalFileSystem.PERSISTENT, 0, gotFS, Error);
```

2，获取FileEntry对象，提供参数为新建

```
function gotFS2(fileSystem) {
    fileSystem.root.getFile("/sdcard/readme.txt", {create: true, exclusive: false}, gotFileEntry2, Error2);
}
```

3，fileEntry通过createWriter制定写入文件的函数

```
function gotFileEntry2(fileEntry) {
    fileEntry.createWriter(gotFileWriter, Error2);
 }
```

4，写入文件的函数，使用write，seek，truncate等方法

```
function gotFileWriter(writer) {
        writer.onwriteend = function(evt) {
          alert(" 'some sample text'");
          writer.truncate(11);
          writer.onwriteend = function(evt) {
            alert("'some sample'");
            writer.seek(4);
            writer.write(" different text");
            writer.onwriteend = function(evt){
              alert(" 'some different text'");
            }
```

```
        };
      };
      writer.write("some sample text");
    }
```

4，FileSystem对象

FileSystem 对象表示一个文件系统，相关属性介绍如下。

- name：文件系统的名称（DOMString 类型）。
- root：文件系统的根目录（DirectoryEntry 类型）。

FileSystem 对象代表当前文件系统的信息。文件系统的名称在公开的文件系统列表中是唯一的。它的 root 属性包含一个代表当前文件系统根目录的 DirectoryEntry 对象。

5，FileEntry对象

FileEntry 对象代表文件系统中的一个文件，W3C 目录和系统规范对其进行了定义。它的相关属性如下。

- isFile：返回值总是 true（布尔类型）。
- isDirectory：返回值总是 false（布尔类型）。
- name：FileEntry 的名称，不包含前置路径（DOMString 类型）。
- fullPath：从根目录到当前 FileEntry 的完整绝对路径（DOMString 类型）。
- filesystem：FileEntry 驻留的文件系统名称（FileSystem 类型）。

FileEntry 对象包括以下几个方法。

- getMetadata：获得文件的元数据。
- moveTo：移动一个文件到文件系统中不同的位置。
- copyTo：复制一个文件到文件系统中不同的位置。
- toURI：返回一个可以定位文件的 URI。
- remove：删除一个文件。
- getParent：查找父级目录。
- createWriter：创建一个可以写入文件的 FileWriter 对象。
- file：创建一个包含文件属性的 File 对象。

各个方法具体介绍：

1. getMetadata

getMetadata 中的参数如下。

- successCallback：获取元数据成功后调用的回调函数，参数为一个 Metadata 对象（函数类型）。
- errorCallback：试图检索元数据发生错误后调用的回调函数，参数为一个 FileError 对象（函数类型）。

2. moveTo

在尝试进行以下操作时会发生错误：

- 同级移动（将一个文件移动到它的父目录中）时没有提供和当前名称不同的名称。
- 移动文件到一个目录所占用的路径中。

此外，尝试移动一个文件到另一个已经存在的空文件上时，系统会尝试删除并替换已存在的文件。

下面来看看它所涉及的参数。

- parent：将文件对象移动到的父级目录（DirectoryEntry 类型）。
- newName：文件的新名称。如果没有指定，那么默认为当前名称（DOMString 类型）。
- successCallback：移动成功后调用的回调函数，参数为移动后新文件的 FileEntry 对象（函数类型）。
- errorCallback：试图移动文件发生错误时调用的回调函数，参数为一个 FileError 对象（函数类型）。

3. copyTo

对于 copyTo，在尝试进行同级复制（将一个文件复制到它的父目录中）时，如果没有提供和当前名称不同的名称，那么将会发生错误。

下面介绍一下它所涉及的参数。

- parent：要将文件对象复制到的父级目录（DirectoryEntry 类型）。
- newName：文件的新名称。如果没有指定，那么默认为当前名称（DOMString 类型）。
- successCallback：复制成功后调用的回调函数，参数为复制后新文件的 FileEntry 对象（函数类型）。
- errorCallback：试图复制文件发生错误时调用的回调函数，其参数为一个 FileError 对象（函数类型）。

5. remove

remove 的作用是删除一个文件。它所涉及的相关参数如下。

- successCallback：文件删除成功后调用的回调函数，无参数（函数类型）。
- errorCallback：试图删除文件发生错误时调用的回调函数，其参数为一个 FileError 对象（函数类型）。

6，DirectoryEntry对象

directoryEntry代表文件系统中的一个目录，其属性和FileEntry一样。

另外，DirectoryEntry 对象有以下方法可以被调用。

- getMetadata：获得目录的元数据。
- moveTo：复制一个目录到文件系统中不同的位置。
- copyTo：复制一个目录到文件系统中不同的位置。
- toURI：返回一个可以定位目录的 URI。
- remove：删除一个目录，被删除的目录必须是空的。
- getParent：查找父级目录。
- createReader：创建一个可以从目录中读取条目的新 DirectoryReader 对象。
- getDirectory：创建或查找一个目录。
- getFile：创建或查找一个文件。
- removeRecursively：删除一个目录以及它的所有内容。

该对象的各个方法和FileEntry一样。其他方法如下：

8. getDirectory

getDirectory 的作用是创建新的目录或查询一个存在的目录，在尝试创建一个直属父级目录但尚不存在的目录时会发生错误。

getDirectory 中涉及以下相关参数。

- path：查找或创建的目录路径，可以是一个绝对路径或者是对应当前 DirectoryEntry 的相对路径（DOMString 类型）。
- options：用于指定如果查找的目录不存在时是否创建该目录的选项（Flags 类型）。
- successCallback：获取成功后调用的回调函数，参数为查找到或创建的 DirectoryEntry 对象（函数类型）。
- errorCallback：创建或查找目录发生错误时调用的回调函数，其参数为一个 FileError 对象（函数类型）。

9. getFile

getFile 的作用是创建新的文件或查询一个存在的文件，在尝试创建一个直属父级目录但尚不存在的文件时会发生错误。

getFile 中涉及以下相关参数。

- path：查找或创建的文件路径，可以是一个绝对路径或者是对应当前 DirectoryEntry 的相对路径（DOMString 类型）。
- options：用于指定如果查找的文件不存在时是否创建该文件的选项（Flags 类型）。
- successCallback：获取成功后调用的回调函数，参数为查找到或创建的 FileEntry 对象（函数类型）。
- errorCallback：创建或查找文件发生错误时调用的回调函数，其参数为一个 FileError 对象（函数类型）。

7，DirectoryReader对象

该对象包含目录中的所有的文件和子目录的列表对象，可以通过readEntries方法读取目录中的所有条目：

```
function success(entries) {
  var i;
  for (i=0; i<entries.length; i++) {
    console.log(entries[i].name);
  }
}
function fail(error) {
  alert("Failed to list directory contents: " + error.code);
}
var directoryReader = dirEntry.createReader();

directoryReader.readEntries(success,fail);
```

8，FileTransfer对象

FileTransfer 是一个允许用户向服务器上传文件的对象。 下面是它所涉及的方法。

- upload：上传一个文件到服务器。
- download: 从服务器下载一个文件。

FileTransfer 对象提供了一种将文件上传到远程服务器的方法，可以通过 HTTP 和 HTTPS 进行请求。可以传递一个由 FileUploadOptions 对象设定的可选参数给 upload 方法。上传成功后，系统会调用成功回调函数并传递一个 FileUploadResult 对象。如果出现错误，那么系统会调用错误回调函数并传递一个 FileTransferError 对象。

1. upload

upload 涉及以下参数。

- filePath：设备上文件的完整路径。
- server：接收服务器文件的 URL。
- successCallback：成功后调用的回调函数（函数类型）。
- errorCallback：失败后调用的回调函数（函数类型）。
- options：文件名和 MIME 类型的可选参数。

```
var win = function (r) {
  console.log("Code = " + r.responseCode);
  console.log("Response = " + r.response);
  console.log("Sent = " + r.bytesSent);
}
var fail = function (error) {
  alert("An error has occurred: Code = " + error.code);
  console.log("upload error source " + error.source);
  console.log("upload error target " + error.target);
}
var options = new FileUploadOptions();
options.fileKey = "file";
```

```
options.fileName = fileURI.substr(fileURI.lastIndexOf('/') + 1);
options.mimeType = "text/plain";
var params = {};
params.value1 = "test";
params.value2 = "param";
options.params = params;

var ft = new FileTransfer();
ft.upload(fileURI, encodeURI("http://some.server.com/upload.php"), win, fail, options);
```

2. download

涉及以下参数。

- source：接收服务器文件的 URL。
- target：设备上文件的完整路径。
- successCallback：成功后调用的回调函数（函数类型）。
- errorCallback：失败后调用的回调函数（函数类型）。

```
var fileTransfer = new FileTransfer();
var uri = encodeURI("http://some.server.com/download.php");
fileTransfer.download(
  uri,
  filePath,
  function(entry) {
    console.log("download complete: " + entry.fullPath);
  },
  function(error) {
    console.log("download error source " + error.source);
    console.log("download error target " + error.target);
    console.log("upload error code" + error.code);
  },
  false,
  {
    headers: {
      "Authorization": "Basic dGVzdHVzZXJuYW1lOnRlc3RwYXNzd29yZA=="
    }
  }
);
```

9，GPS定位服务

1，获取当前地理位置

Geolocation对象可以使用设备传感器的GPS位置服务，getCurrentPosition可以获取当前位置：

```
navigator.geolocation.getCurrentPosition(geolocationSuccess,
[geolocationError], [geolocationOptions]);
```

地理位置数据的 Position 对象是通过读取 geolocationSuccess 回调函数的返回值获取的。如果发生读取错误，那么将触发 geolocationError 函数并传递一个 PositionError 对象给该函数。

```
navigator.geolocation.getCurrentPosition(onSuccess, onError);

// onSuccess Geolocation

unction onSuccess(position) {

    var element = document.getElementById('geolocation');

   element.innerHTML = '维度: '+ position.coords.latitude+ '<br />' +

            '经度: '+ position.coords.longitude + '<br />' +

            '高度: '+ position.coords.altitude+ '<br />' +

            '精确度: '+ position.coords.accuracy+ '<br />' +

        'Altitude Accuracy: '+ position.coords.altitudeAccuracy+ '<br />' +

            '运动方向: '+ position.coords.heading + '<br />' +

            '地面速度: '+ position.coords.speed+ '<br />' +

            '时间戳: '+ position.timestamp + '<br />';

    }
```

2，周期性获取位置变化

使用 geolocation.watchPosition 方法，可以周期性地获得地理位置信息。相关代码如下所示：

```
var watchID = navigator.geolocation.watchPosition(geolocationSuccess,
[geolocationError], [geolocationOptions]);
```

每次取得位置数据后，数据将传递给 geolocationSuccess 函数。通过 geolocationOptions 对象的 frequency 参数可以设定以毫秒为单位的时间间隔。返回的 watchID 是地理位置监视周期的引用，可以通过 geolocation.clearWatch 调用该 watchID 以停止监视设备的位置变化。

```
function onDeviceReady() {

    var options = { enableHighAccuracy: true };

   watchID = navigator.geolocation.watchPosition(onSuccess, onError, options);

}

    // onSuccess Geolocation

function onSuccess(position) {

    var element = document.getElementById('geolocation');

    element.innerHTML = 'Latitude: '  + position.coords.latitude     + '<br />' + 'Longitude: ' +
position.coords.longitude     + '<br />' + '<hr />'     + element.innerHTML;

}

    // clear the watch that was started earlier

 function clearWatch() {

       if (watchID != null) {

          navigator.geolocation.clearWatch(watchID);

          watchID = null;
```

```
    }
}
```

10，notification消息提示

1，警告框

使用 notification.alert 方法可以显示一个自定义的警告框。相关代码如下所示：

```
navigator.notification.alert(message, alertCallback, [title], [buttonName]);
```

下面介绍其所涉及的参数。

- ❑ message：对话框信息（字符串类型）。
- ❑ alertCallback：当对话框消失后调用的函数（函数类型）。
- ❑ title：对框框标题（可选，默认值：“Alert”）（字符串类型）。
- ❑ buttonName：按钮名称（可选，默认值：“OK”）（字符串类型）。

在大多数平台上，PhoneGap 都可以对原生对话框使用这一特性。然而，在少数平台上只能使用浏览器的 alert 方法，不能使用 PhoneGap 自定义对话框。例如，在 BlackBerry 或者 webOS，不能设置对话框的回调函数、标题与按钮名称。

```
navigator.notification.alert("你赢了",alertDismissed,"游戏结束","确定");
```

2，确认框

使用 notification.confirm 方法可以显示一个自定义的确认框。相关代码如下所示：

```
navigator.notification.confirm(message, confirmCallback, [title], [buttonLabels]);
```

下面介绍其所涉及的参数。

- ❑ message：对话框信息（字符串类型）。
- ❑ confirmCallback：当对话框消失后调用的函数，并传递点击按钮的索引值（1、2 或 3）给该函数（数字类型）。
- ❑ title：对话框标题（可选，默认值：“Confirm”）（字符串类型）。
- ❑ buttonLabels：逗号分隔的按钮标签字符串（可选，默认值：“OK，Cancel”）（字符串类型）。

notification.confirm 显示一个原生的对话框，该对话框比浏览器的 confirm 函数具有更好的自定义性。

```
navigator.notification.confirm("你赢了",alertDismissed,"游戏结束","OK,Exit");
```

3，蜂鸣声音提示

使用 notification.beep 方法可以让设备播放蜂鸣声音。相关代码如下所示：

```
navigator.notification.beep(times);
```

参数 times 表示播放蜂鸣声音的次数（数字类型）。

Androids 平台上会播放在“设置 / 音效及显示”面板中指定的默认“通知铃声”。iPhone 平台将忽略蜂鸣次数参数。

4，震动

使用 notification.vibrate 方法，可以让设备振动。相关代码如下所示：

```
navigator.notification.vibrate(milliseconds);
```

参数 milliseconds 表示振动时长，单位为毫秒（数字类型）。在 iPhone 平台将忽略振动时长参数，振动时长为预先设定值。

11，客户端数据存储

1，openDatabase

openDatabase 用于创建一个新 DataBase 数据库。相关代码如下所示：

```
var dbShell = window.openDatabase(name, version, display_name, size);
```

window.openDatabase 将返回一个新的 Database 对象。该方法会创建一个新的 SQL Lite 数据库，并返回该 Database 对象，即可使用 DataBase 对象进行数据库相关的操作。

相关参数介绍如下。

- name：数据库的名称。
- version：数据库的版本号。
- display_name：数据库的显示名。
- size：以字节为单位的数据库大小。

上面方法会返回一个Database对象，包含了允许用户操作数据库的方法。

- transaction：运行一个数据库事务。
- changeVersion：该方法允许脚本自动验证版本号，并更新版本号以完成架构更新。调用 window.openDatabase() 将返回一个 Database 对象。

1，进行事务操作：

SQLTransaction 包含允许用户对 Database 对象执行 SQL 语句的方法，executeSql 表示执行一条 SQL 语句。

当调用 Database 对象的 transaction 方法时，其回调函数将被调用并接收一个 SQLTransaction 对象。用户可以通过多次调用 executeSql 来建立一个数据库事务处理。

```
function onDeviceReady() {
    var db = window.openDatabase("Database", "1.0", "Cordova Demo", 200000);
    db.transaction(populateDB, errorCB, successCB);
}
function populateDB(tx) {
    tx.executeSql('DROP TABLE IF EXISTS DEMO');
    tx.executeSql('CREATE TABLE IF NOT EXISTS DEMO (id unique, data)');
    tx.executeSql('INSERT INTO DEMO (id, data) VALUES (1, "First row")');
    tx.executeSql('INSERT INTO DEMO (id, data) VALUES (2, "Second row")');
}
```

2，修改数据库版本号

```
var db = window.openDatabase("Database", "1.0", "PhoneGap Demo", 200000);
db.changeVersion("1.0", "1.1");
```

2，SQLResultSet

当 SQLTransaction 对象的 executeSql 方法被调用时，将会触发 executeSql 中设定的回调函数并返回一个 SQLResultSet 对象。下面介绍一下其中所涉及的属性。

- insertId：表示 SQLResultSet 对象通过 SQL 语句插入到数据库后，该行数据的 ID（如果插入多行数据，则返回最后一行的 ID）。
- rowAffected：被 SQL 语句改变的记录行数，如果语句没有影响任何行则设置为 0。
- rows：是一个 SQLResultSetRowList 对象，表示返回的多条记录。如果没有返回任何记录，则此对象为空。

当调用 SQLTransaction 对象的 executeSql 方法时，将会触发 executeSql 中设定的回调函数并返回一个 SQLResultSet 对象。该结果对象包含 3 个属性：第一个是 insertId 返回成功的 SQL 插入语句所插入行的 ID，如果 SQL 语句不是插入语句则 insertId 将不被设定；第二个是 rowAffected，在 SQL 执行查询操作时此属性总是 0，当插入或更新操作时此属性返回受到影响的行数；最后一个属性是 SQLResultSetList 类型，返回 SQL 查询语句的返回数据。

SQLResultSetList 是 SQLResultSet 对象的一个属性，包含 SQL 查询所返回的所有行数据。

其属性 Length 表示 SQL 查询所返回的记录行数。其涉及的方法 Item，作用是根据指定索引返回一个行记录的 JavaScript 对象。

SQLResultSetList 包含一个 SQL 查询语句所返回的数据。该对象包含一个长度属性，可告知用户有多少符合查询条件的行记录数被返回。通过传递指定的索引给该对象的 item 方法，可获取指定的行记录数据，此 item 方法返回一个 JavaScript 对象，其属性包含前述查询语句所针对的数据库的所有列。

```
function successCB() {
    alert("success!");
    db.transaction(queryDB,errorCB);
}
```

```
function queryDB(tx){
   tx.executeSql("SELECT * FROM DEMO",[],querySuccess,errorCB);
}
function querySuccess(tx,results){
   alert("Row affected="+results.rowsAffected);
   alert("返回的行数："+results.rows.length);//rows对应的是SQLResultSetList对象
   //输出查询到的行
   var len=results.rows.length;
   for(var i=0;i<len;i++){
     alert("Row="+i+" ID="+results.rows.item(i).id+" Data="+results.rows.item(i).data);
   }
}
```

2，localStorage

localStorage 提供对 W3C Storage 接口（http://dev.w3.org/html5/webstorage/#the-localstorage-attribute）的访问。相关代码如下所示：

```
var storage = window.localStorage;
```

下面介绍一下其中涉及的方法。

- key：返回指定位置的键的名称。
- getItem：返回指定键所对应的记录。
- setItem：存储一个键值对。
- removeItem：删除指定键对应的记录。
- clear：删除所有的键值对。

示例如下：

```
window.localStorage.setItem("renxiuhu", "shidandan");
var keyname = window.localStorage.key(0);
alert(keyname);
var value = window.localStorage.getItem("renxiuhu");
alert(value);
window.localStorage.removeItem("renxiuhu");
window.localStorage.setItem("key2", "value2");
window.localStorage.clear();
```

2，Jquery Mobile

Jquery Mobile构建与Jquery和Jquery UI类库之上，使用了极少的HTML5，CSS3，js和Ajax脚本代码来完成页面的布局渲染。

你是否想知道为什么在 **<script>** 标签中 没有插入 **type="text/javascript"** ？

在 HTML5 已经不需要该属性。 JavaScript 在所有现代浏览器中是 HTML5 的默认脚本语言！

要使用JM首先需要在html文件中加入如下的引用：

```
<link rel="stylesheet" type="text/css" href="../css/jquery.mobile-1.4.5.min.css">

<script type="text/javascript" src="../js/jquery.js"></script>

<script type="text/javascript"  src="jquery.mobile-1.4.5.min.js"></script>
```

1，页面基本结构

```
<div data-role="page">
```
——page代表显示的页面

```
<div data-role="header">
```
——header代表顶部工具栏，常用于标题和按钮

```
<h1>欢迎来到我的主页</h1>

</div>

<div data-role="content">
```
——content定义了页面内容

我现在是一个移动端开发者

——footer代表底部工具条

<h1>底部文本</h1>

使用id可以唯一的标识一个html标签，然后可以通过#id的方式引用标签，如下实现了页面切换的功能：

——添加dialog可以在点击链接时弹出对话框

进入第二页

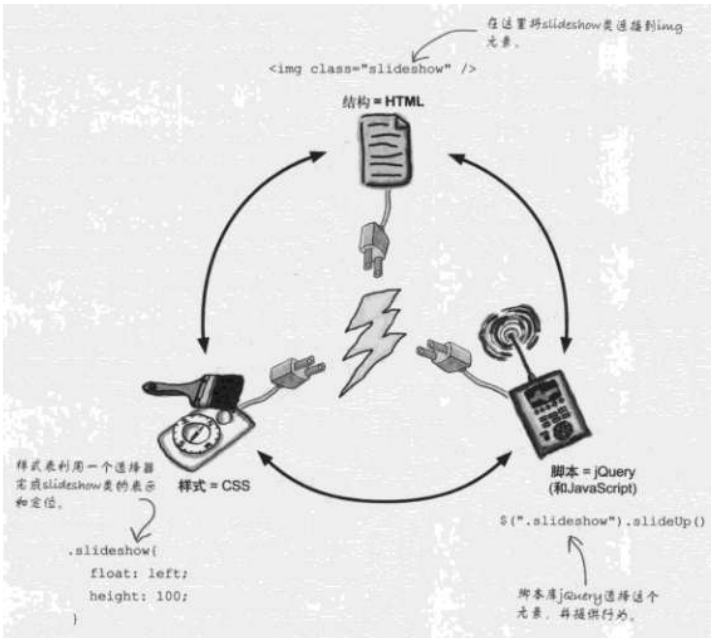
进入第一页

3，Jquery

1，基础入门

Html只是用来处理文档结构的标记语言；CSS用来控制这些元素的外观和位置；DOM时html文档对象模型。Jquery时一个专门用来动态改变web页面文档的javascript库，javascript并不改变原来的html和css文件，只是通过DOM改变浏览器内存中的页面。

Html，css和JS就是结构，样式和脚本的关系：



——注意在选择指定元素进行动画效果操作的时候，一定要注意顺序，从最小节点到最外层节点进

行设置。如下所示：

```
<div id="picframe">
<img src="../images/wife.png"></img>
</div>
```

```
$("#clickme").click(function(){
$("img").fadeIn(1000);
$("#picframe").slideToggle("slow");
});
```

——<script>代码块放在</body>前的原因:

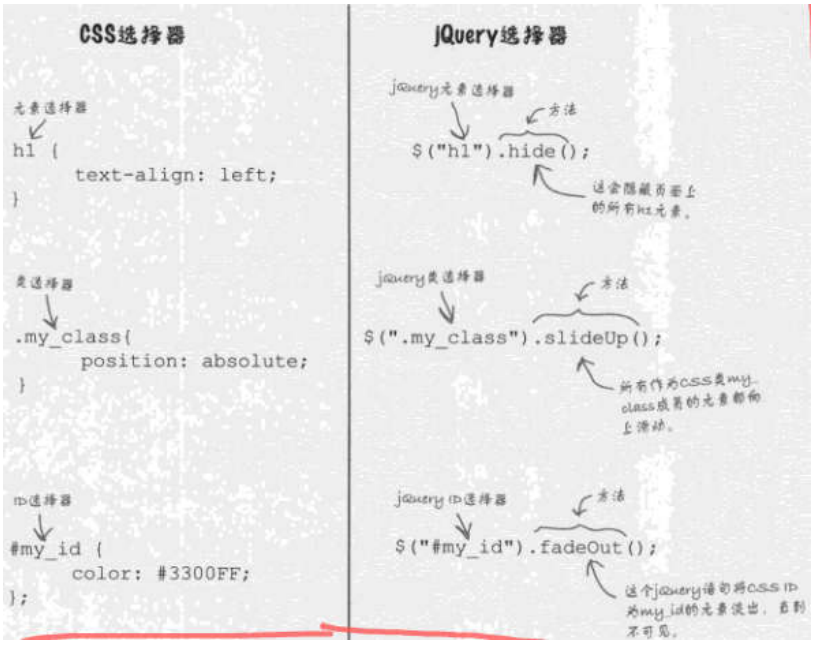
Javascript脚本放在head中会阻塞浏览器中的并行下载，不同服务器的多个图像可以同时下载，但是一旦遇到<script>标签就不能并行下载了，所以放在最后面有助于提高页面的加载速度。

1，选择器

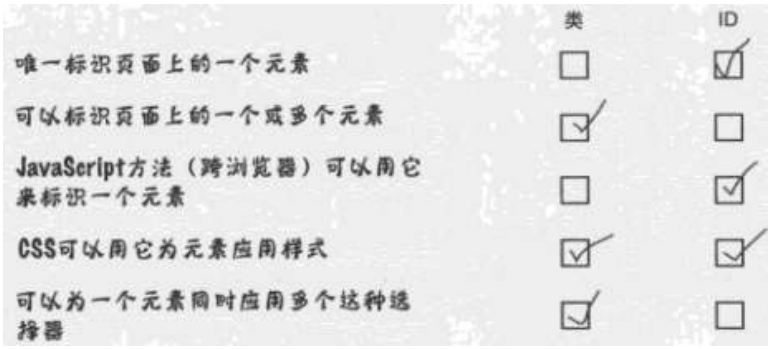
$()其实就是jquery()方法，有一下三种用途：

1. 参数为html串，用于动态增加DOM元素
2. 获取参数制指定的页面元素
3. 用于增加javascript对象

$()获取界面元素和CSS一样，使用的CSS选择器，其选择方式有一下三种：

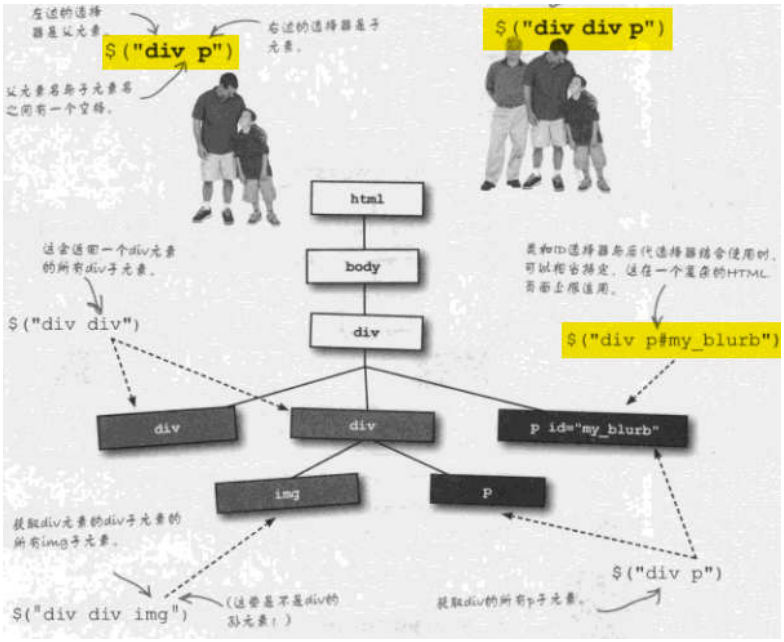


类选择器和ID选择器的异同：



后代选择器：

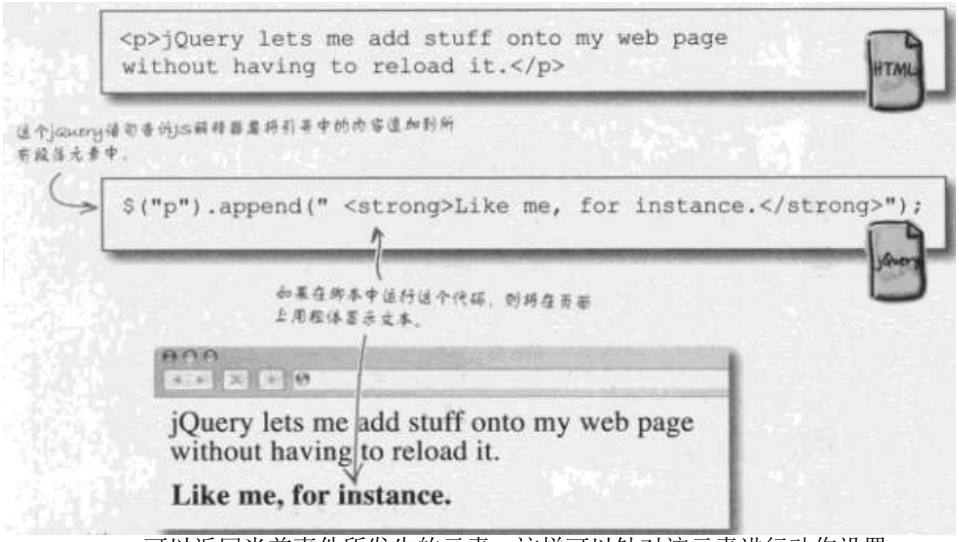
后代选择器是Jquery的另一种选择器，利用后代选择器，可以指定元素之间的关系，可以选择子元素，父元素和兄弟元素。



2，Jquery函数

常用的内置函数：

——append方法：可以向一个已有的元素插入内容：



——$(this)：可以返回当前事件所发生的元素，这样可以针对该元素进行动作设置：

```
$("#myImg").click( function(){
    $(this).slideUp();
});
```

——remove方法：可以将一个元素或一组元素从页面中删除：

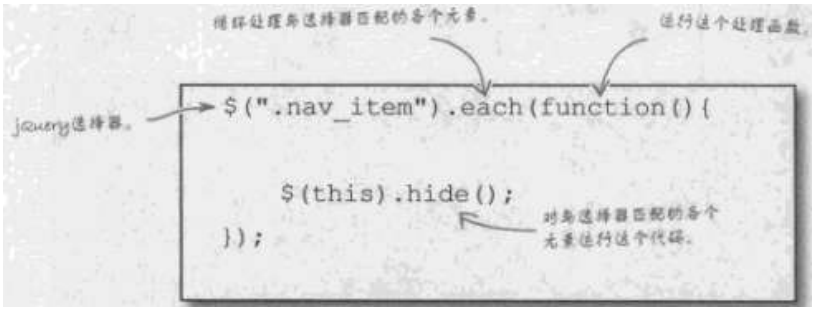
这是按钮的代码，它会删除列表中的所有列表项：

```
$("#btnRemove").click(function(){
    $("li").remove();
});
```

——Math.floor方法：将一个数取整最为接近的整数，并返回结果

——Math.random方法：返回一个介于0和1之间的随机数。

——each方法：可以循环的给一组元素执行function中的代码，达到迭代处理的效果。另外还可以带参数。




```
$(".guess_box").each(function(index, value) {
    if(numRand == index){
        $(this).append("<span id='has_discount'></span>");
        return false;
    }
});
```

——trigger方法：可以在代码中触发元素的某个事件

```
$( " button:first " ).trigger( 'click ');
```

——$.contains方法：一个静态方法，用于检查第二个元素是否包含在第一个参数中：

$.contains(this,document.getElementById("hs_discount"))

——addClass和removeClass：可以为元素添加和删除class属性，如果再在CSS中定义相应的Class的显示效果，就可以达到动态调节CSS效果的功能

CSS文件定义：

```
.hover{
border: 3px solid #f00;
}
.no_hover{
border: 3px solid #000;
}
```

```
$("#btn1").click( function(){
  $("#header").addClass("hover");
  $("#header").removeClass("no_hover");
});
```

自定义函数：

自定义创建函数，指定函数名的两种方式：

```
function myFunc1(){
      $("div").hide();
}
```

```
var myFunc2 = function() {
      $("div").show();
}
```

3，Jquery事件

绑定事件：

元素绑定事件及事件处理函数有两种方法，如下所示：

方法1：

```
$("#myElement").click( function() {
      alert($(this).text());
});
```

方法2：

```
$("#myElement").bind('click', function()
      alert($(this).text());
});
```

方法1被称为便利方法，必须当DOM已经存在的时候才能够使用，如果是通过代码添加的新DOM元素，则只能使用第二种方法添加监听。

解除绑定：

使用unbind方法可以解除绑定在元素上面的事件，可以解绑某个事件，也可以解绑所有事件：

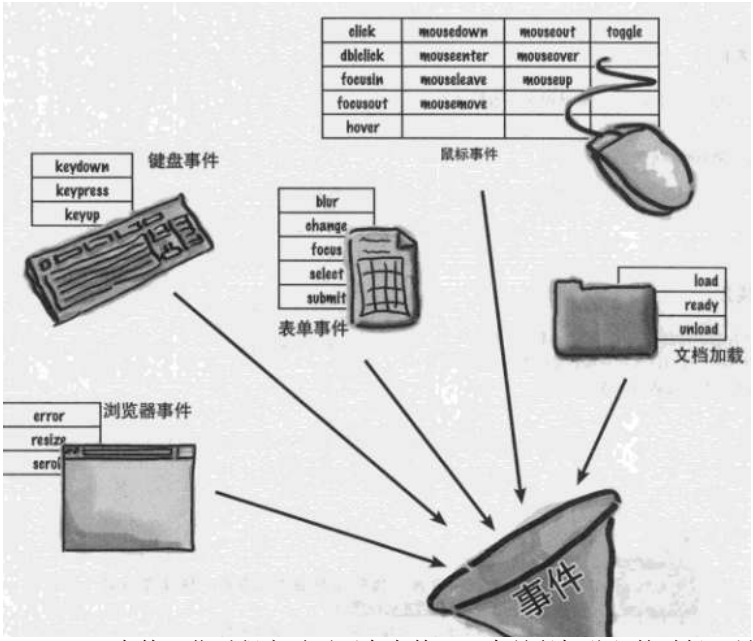
解绑一个事件：

```
$("#myElement").unbind('click');
```

解绑所有事件：

```
$("#myElement").unbind();
```

常用的JS事件如下，所有的事件可以在API文档中查看：



——hover事件：监听程序需要两个事件，一个是鼠标移入的时候，另一个是鼠标移出的时候：

```
$(".guass_box").hover(function(){
$(this).addClass("hover");
},
function(){
$(this).removeClass("hover");
});
```