**H5和CSS3新增内容**

HTML5新增标签及属性

CSS3新增样式

Web API

<http://www.open-open.com/lib/view/open1360135504424.html>

<http://www.cnblogs.com/yuzhongwusan/archive/2011/11/17/2252208.html>

参考w3school中有列表

**选用标签的标准**

如果要支持比较老的浏览器（通常是桌面版网页）则不使用新标签

手机平板等网页开发可以使用新标签

不管什么情况下，都应该优先根据语义来选用标签

根据搜索引擎优化的要求和建议选用标签

**布局**

流式布局

定位

浮动/清除

Flex

布局框架（如Bootstrap栅格）

**CSS单位与计算**

px

%

rem/em

vw/vh

mm/cm

calc()

**CSS半透明/透明**

大部分标签元素默认是背景透明的，但也有不透明的（如iframe、button、图片等）

transparent

RGBA

HSLA

opacity

**动画**

transition：transition能过渡大部分样式（<http://dwz.cn/3NvjUN>）而且支持多样式同时过渡，注意标签元素上有多个样式类时，与过渡样式相关的最后2个样式会形成过渡效果。

animation：注意使用animation-play-state:paused可以使动画就地暂停，animation-play-state: ‘’可以使动画在暂停的状态继续进行

jQuery动画：jQuery动画与CSS3动画不同，jQuery动画是js驱动的，jQuery动画与CSS3动画是互补关系（而非谁取代谁）。jQuery内置了一些动画，如（fadeIn/slideUp等），也支持通过animate()方法自定义动画。而且提供了强大的动画控制API，如动画延迟、链式动画（上一个动画结束后开始下一下动画）、控制动画停止、关闭动画效果、动画频率调整、还可以指定回调函数参与每一个动画的每一步等。jQuery中的动画返回Deferred对象，因此支持Promise异步编程。jQuery默认不能过渡颜色，使用jQuery.Color插件可以解决问题。

触发动画：CSS一般在页面加载时会自动进行transition和animation，可以在浏览器的查看器（动画窗格）中观察到标签元素上的动画帧并对动画进行控制及设置不同的速度进行预览。使用:hover、:active伪类可以配合鼠标指针触发动画效果。通过js添加删除标签元素的样式类时也会触发相应的动画。jQuery动画的触发和控制由jQuery动画API控制。

动画结束事件/回调：CSS3动画支持DOM事件：animationstart（动画开始时）、animationend（动画结束时）、animationiteration（动画重复执行时），可以像普通click事件一样使用它们。jQuery动画通过回调函数获取执行完成通知，形如：$(..).animate({css}<, duration><, easing><, callbakc>)，因为jQuery动画返回Deferred对象，所以也可以使用.then(success,fail)或.done(function)等方法得到通知。

同时进行多个动画：在关键帧定义中可以定义多个样式，animation-name也可以指定多个关键帧动画名称。transition-property也可以指定多个样式。jQuery的animate方法中也可以指定多个样式。还可以在同一个标签元素上调多次animate方法（分开调用，而不是链式调用）

顺序进行多个动画：可以通过animation-delay、transition-delay延迟，安排出顺序执行效果。还可以通过监听动画事件在动画结束后再触发下一个动画。jQuery中animate()的链式调用本身就是顺序执行的。

requestAnimationFrame：会与浏览器呈现页面同步配合执行，因此比使用setInterval要更加流畅。requestAnimationFrame是H5新增的方法，与setInterval相比，没有提供频率的控制方法，因为它的频率与页面呈现频率是同步的，通常60次/s，相当于16.66667ms一次。

矩阵与图形变换

贝塞尔曲线与时间函数

**媒体查询与响应式页面**

媒体查询@media赋予CSS探测设备类型和设备属性（如屏幕宽高等）的能力，之前媒体查询主要用来控制页面打印效果（即针对页面上打印部分定义打印专用样式）；在移动互联网时代媒体查询对于页面适配各种大小的屏幕至关重要，可以说媒体查询是响应式页面的核心技术。Bootstrap就是以媒体查询技术为基础的响应式UI框架。

**浏览器兼容性和Web App页面**

浏览器兼容性问题涉及的面比较广，从HTML标签、CSS样式到JS都存在兼容性问题。兼容性问题给Web开发造成了额外的成本支出，在兼容性上要求越高，付出的代价越大。

浏览器的兼容性主要通过测试解决，最为困难的是桌面网页兼容IE6/7/8/等老浏览器（可以使用IE Tester测试）！解决浏览器兼容性没有什么难度，繁锁而已，如果要兼容老浏览器，则应放弃使用新功能或者对老浏览器显示友好的提示信息（使开发测试时间增长、成本增加）。

直接使用框架可以有效地避免兼容性问题，如各种UI框架和jQuery框架都会考虑兼容性。另外使用重置样式表也可以抹平各浏览器的差异。

在开发Web App时需要注意，移动设备，无论是iOS还是Android都使用Webkit内核，尤其是iOS，需要使用以下样式：

-webkit-appearance: none; 禁用浏览器特定外观（如按扭等表单元素）

-webkit-tap-highlight-color: rgba(0,0,0,0); 将触摸时的高亮颜色设为完全透明

-webkit-touch-callout: none; 禁用页面上的上下文菜单（上面有复制、粘贴等功能）

-webkit-text-size-adjust: none; 禁用页面自动调整文字大小

-webkit-user-select: none; 禁止用户选择页面上的文字

回弹滚动

overflow: auto; /\* auto | scroll \*/

-webkit-overflow-scrolling: touch;

另：有一些样式还需要使用-webkit前缀，如：

CSS滤镜要写成-webkit-filter

或者要兼容特别老的iOS或Android系统时CSS3样式要加-webkit前缀

另：如果需要对某些标签进行深度控制，则需要使用-webkit伪类，如：

input[type=range]::-webkit-slider-runnable-track{ }

input[type=range]::-webkit-slider-thumb{ }

可以控制滑动条的样式

progress::-webkit-progress-bar { }

progress::-webkit-progress-value { }

可以控制进度条的样式

另：使用meta标签可以要求浏览器支持或关闭某些行为

<meta name="format-detection" content="telephone=no" /> 禁用手机号探测

<meta name="format-detection" content="email=no"> 禁用email探测

固定到手机桌面（桌面书签）

<meta name="apple-mobile-web-app-title" content="常伟">

<link rel="apple-touch-icon" sizes="57x57" href="icon57.png" />

<link rel="apple-touch-startup-image" href="launch6.png">

<meta name="mobile-web-app-title" content="常伟">

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

关闭iOS键盘首字母自动大写，自动修正

<input type="text" autocapitalize="off" />

<input type="text" autocorrect="off" />

**LESS/SASS/SCSS**

Less 是 CSS 预处理语言，在CSS中引入了变量、函数、混入等编程机制，从而使CSS获得更高的灵活性和可扩展性。Less可以使用js直接在页面上转换成CSS，也可以使用命令行工具Lessc完成从less文件到css文件的转换，还有相应的自动化构建插件。注意Less中使用calc()或滤镜等函数时需要使用 ~””或e(””)进行转义，否则会被误识别为less函数，而less中并没有这些函数。

Sass （Sass3时提供了兼容CSS的语法，被称为SCSS）是对 CSS 的扩展，除支持变量、函数、混入外还支持条件判断和循环语句，Sass让 CSS 语言更强大、更优雅。

值得一提的是，Less是基于Node.js和js的，而Sass/SCSS是基于Ruby的（也需要Node.js），Sass/SCSS不能直接在页面上通过js转换成CSS。

**JS基本数据类型（typeof的返回结果）**

number（Infinity/NaN）

string

boolean

function

object（null、各种值装箱对象、内置对象、自定义对象）

undefined

判断对象是否为某个【类/构造函数】的实例（注意继承）用instanceof（对象与函数的关系）

获取对象的原型用Object.getPrototypeOf(obj)，如果原型中有constructor则可获得构造函数[获得的是函数而不是函数名字]

判断A对象是否为B对象原型链上的一环A.isPrototypeOf(B) （对象与对象之间的关系）

**原型与原型链**

如果在构造函数中直接为对象添加属性和方法，则会导致每个对象都包含一份自己的属性和方法，当大量创建对象时就会浪费很多存储空间。而将属性和方法添加到构造函数的原型中就可以使通过该构造函数创建出来对象共用这些属性和方法。即原型实现了属性和方法的**共享**。

为构造函数指定原型：

ObjFunc.prototype = object.create(baseFunc)

ObjFunc.prototype.constructor = baseFunc

ES6中支持 class 和 extends 关键字，就不需要用上面的方式了（但本质上还是一样的）。

当访问对象的属性和方法时，JS运行时会首先在该对象自身中查找要使用的属性和方法，找到就调用；如果找不到，则会到原型中查找。因此原型中的属性和方法虽然是共享的，但它们的优先级要低一点。

JS运行时在查找可调用的属性和方法时并不只查找一层原型，因为原型本身也是对象，因此JS运行时还会查找原型对象的原型，一直查找到Object的原型为止。这样就构成了一个链式结构，称为原型链。通过原型链可以形成类似**继承**的效果。

但JS中的继承与其它编程语言中的继承还是有区别的！在其它编程语言中，任何类的对象都是由一个实例构成的，属性值也只有一个值，不管继承多少层。而JS通过原型链形成的继承是由多个实例构成的链，属性值也可以有多个值（每一层原型上都可以有一个不同的值），继承的层次越多，涉及的实例就越多，值也越多！

JS的原型链在Angular中还被用于构造作用域链。

**this**

js中的this与其它编程语言中的不同，js中的this指向的对象不是固定的，而是可以改变的（通过function对象的call、apply、bind，jQuery的proxy等可以改变this指向）。以下是this的默认指向：

1. 定义在全局作用域中的普通函数中的this指向window对象（严格模式下本条无效，参见后面严格模式的说明）
2. 事件处理函数中的this指向触发事件的标签元素
3. 构造函数中的this指向当前正在创建的对象（严格模式下必须使用new）
4. 原型中的函数内的this指向当前实例
5. 其它框架中的函数内的this都有特定的指向，如jQuery.each(func)中func中的this指向当前迭代的标签元素对象。

**DOM**

文档对象模型，包括document本身和页面上的各种标签元素。DOM是访问和控制文档及标签元素的API，通过DOM可以查找、修改、添加、删除标签元素，还可以添加事件监听函数以响应页面上的各种事件。H5新增了一些DOM API（如querySelector）在一程度上增强了DOM，但实际开发中，使用jQuery进行开发还是非常普遍的。因为jQuery不仅提供了强大的DOM操作能力，而且能够解决浏览器兼容性问题。此外jQuery还提供了Ajax请求、动画等实用功能，还提供了Callbacks、Deferred等高级API。

**BOM**

浏览器对象模型，包括window、location、history、navigator等

window对象是全局作用域，window对象会随页面的加载或刷新而重置，即全局作用域生命周期相关的内容都会被销毁。window对象提供了窗口相关的API，用于控制窗口及窗口与窗口之间的关系。window对象还提供了XHR、setTimeout、alert、WebWorker等各种编程接口和功能支持。实际上从window对象出发可以访问几乎所有的Web API。

location提供了与url相关的API，通过location提供的属性可以获取url的各个组成部分。location还提供了对页面进行导航控制的基本功能，如对href赋值、assign()、reload()、replace()方法等。值得注意的是location提供了hashchange事件，可以监听hash fragment的变化，这对于单页Web应用中的路由机制提供了重要的支持。

history对象提供了浏览历史记录控制功能，如back()、forward()、go()等，H5还新增了状态相关的API：state、pushState、replaceState配合window的popstate事件可以为路由机制提供重要的支持。

navigator对象提供了浏览器和操作系统信息的描述，如userAgent、H5还在navigator中加入了很多设备访问功能，如geolocation、battery、vibrate等（浏览器目前支持不太好）。

**JS语言核心对象**

包括Object、Function、Array、与值类型对应的包装对象（Boolean、Number、String）、Math、Date、Error、Regexp、JSON，这些对象是JS编程语言自带的，无论JS在何处（如Photoshop、Mongodb等）运行都会包含这些对象。

**Map/Reduce**

js中的数组支持map/reduce（ES6）。map可以对数组中的每一个元素进行加工，然后返回一个同样长度的**数组**，reduce可以将数组中的每一个元素都拼在一起生成**一个**最终结果。

Map/Reduce是一个非常经典的编程思想，广泛应用于数据处理（如大数据技术），Map/Reduce的好处是能够将数据映射与数据归结分离开，有利于简化问题，在大数据技术中还可以分布式计算提高数据处理速度。

因为Map返回的还是数组，因此可以构成如下的链式调用：

data.map(function(i){ }) .map(function(i){ }).reduce(function(prev, current, seed){ })

**图片预加载和懒加载**

图片的可以为页面带来更好的展示效果，同时也能向用户传达更丰富的信息。在页面上展示图片时有两类典型问题：

* 1. 图片最初是隐藏的，但在需要显示时应能立即显示出来（如轮播图、相册展示、幻灯片）
  2. 网页很长，而且网页上有大量的图片（如电商网站中的商品列表），但这些图片未必用户都会看到，如果用户根本就不滚动页面来查看这些图片，那这些图片的加载就会成为一种浪费。

解决这两类问题就需要使用图片预加载和懒加载技术。

图片预加载是在图片显示之前就让浏览器加载这个图片，这样当浏览器真正需要显示这个图片时就能瞬间将它显示出来，不需要再去下载和解码。图片预加载通常使用Image对象将图片载入内存。Image对象实际上就是一个<img>标签，只不过没有append到页面上而已。

图片懒加载则是让网页下部分的图片都先显示同一个loading图片，当页面向下滚动，这些图片露出来时，才将img标签的src改为正确的图片地址，这时浏览器才去下载图片并解码显示到页面上。图片懒加载的关键是监视页面的滚动及窗口大小变量并计算图片是否处于窗口工作区内（即用户能看到）。有很多图片懒加载插件或独立脚本，如：

jQuery.lazyload.js

echo.js

**Ajax/XHR/HTTP/jQuery Ajax**

Ajax即通过XHR API使用js发起的异步网络请求，它不会导致页面刷新，因此是现代Web App的关键技术。

HTTP协议是Web开发中最重要的网络协议，HTTP协议详细规定了请求和响应报文。

请求报文由4个部分构成：

* 1. 请求行：包含请求方法和URL
  2. 请求头：包含HTTP协议中定义的请求头和自定义请求头（每个头1行）
  3. 空行：分隔请求头与请求体（由协议实现方控制，没有提供API）
  4. 请求体：随请求发送到服务端的数据（可多行）

常用的请求方法是GET和POST。GET没有请求体，数据只能放在URL中，因些可以发送的数据很少，而且用户能够通过浏览器地址栏看到发送的数据。POST请求可以通过请求体发送大量的数据，可以支持各种数据编码方案（需要用Content-Type头指明），常见的有urlencoded、json、xml、text等。其它请求方法在RESTful中常用。

请求头中有一些与协议相关的重要数据，如Content-Type、Content-Length、支持断点续传的Range、携带浏览器信息的UserAgent、还有常用来存储用户身份信息和偏好设置的Cookie等

响应报文也由4个部分构成：

1. 响应行：包括响应状态码、和状态描述、HTTP版本（最新2.x只用于https，http最新1.x，广泛使用的是1.1）
2. 响应头
3. 空行
4. 响应体

XHR能够灵活地控制HTTP的请求和响应报文：请求行在open方法中控制，请求头通过xhr.setRequestHeader('name', 'value')设置，请求体通过send方法发送。通过xhr对象的status属性和statusText可以获取响应行信息，通过xhr.getResponseHeader('name')可获取单个响应头，通过xhr.getAllResponseHeaders()可获取所有响应头。通过xhr.response、xhr.responseText、xhr.responseXML可获取响应体。

XHR还支持很多事件，可以监控请求/响应状态的变化，如：

readystatechange最常用

loadstart

progress

abort

error

load

timeout

loadend

jQuery中也包含了对Ajax的强大支持，jQuery框架能获得巨大的成功与其对Ajax的卓越支持是分不开的。在jQuery Ajax是对XHR的封装，它提供了丰富的高级API，可以很方便地发起各种请求，很方便地控制和获取请求/响应报文的各个方面，监听请求/响应状态的变化（包括单次请求和全局监视）

$.ajax({请求配置})

$.get(url, data, success, type)

$.post(url, data, success, type)

此外，jQuery的Ajax还支持JSONP（$.getJSON(url, data, success)），支持直接加载HTML到页面$('div').load(url, data, success)，支持通过代码加载js脚本$.getScript(url, success)。更值得称赞的是，jQuery的Ajax返回的是Deferred对象，支持Promise异步编程！

**REST**

在常规网络请求中，通过在URL中插入【动词】来【表示】不同的功能

获取数据： /api/card/id

保存数据： /api/card/【save】/id

添加数据： /api/card/【add】

删除数据： /api/card/【remove】/id

更新数据： /api/card/【update】/id

REST 表达（表示）状态迁移【转移】

把上面表示不同功能的动词，从URL中转移到请求行的请求方法中

原先的请求行：

POST /api/card/【save】/id

上面这种写法没有注意到HTTP协议中请求方法的作用，到处使用POST/GET这两个请求，把这2个请求方法当成是万能的，在URL中插入各种各样的动词来表示本应该由请求方法表示的功能，这种写法还对URL产生了不良的影响：

1. URL是一个地址，地址中应该使用名词，而不应该出现动词！
2. 插入动词之后，本来是同一个资源，只需一个地址，现在变成一堆地址！

按照REST风格改造之后：

获取数据： GET /api/card/id

添加数据： POST /api/card

更新数据： PUT /api/card/id

删除数据： DELETE /api/card/id

URL中的动词被删除了，一个资源只有一个URL【表示功能的动词从URL中转移到了请求方法中】，请求方法是HTTP协议规范定义的，所以大家都使用一组一致的动词，不需再自己创造各种各样只有自己能看懂的动词了。

现代的应用程序框架中普遍支持RESTful，如Backbone和Angular($http、ngResource)。

**跨域访问与同源策略**

因为在同一个浏览器窗口中能够同时打开多个网站的页面，而且它们都处于同一个会话中，如果不禁止跨域访问则会造成用户隐私数据泄露和登录身份冒用的问题，所以浏览器会使用同源策略限制跨域访问。

在浏览器中，通过JS代码访问不同域名下的URL（JS的XHR/AJAX）或者iframe（JS访问iframe内部的DOM）时，会被禁止访问。而不是通过JS代码进行的跨域访问不存在跨域问题！比如可以跨域加载图片、引用JS文件、下载各种文件、使用iframe跨域嵌入其它网站的页面都是允许的。

跨域访问被禁止有时会给应用开发带来阻碍，但在符合特定条件时也有相应的方法在保证安全的情况下能够解决跨域访问的问题。

1. 在对方的服务器中的响应头中添加 Access-Control-Allow-Origin 允许哪些域进行跨域访问它是值可以是 域名，或者 \* 。这种方案只有在对方信任或不在乎（安全）的情况下才能使用。
2. 如果域名都是同一个根域名的子域名，则可以使用document.domain = "根域名" 来统一JS执行环境中的域名。这种方案只能在同一个公司或组织的内部使用。
3. JSONP JSON Padding，原理是：浏览器不限制通过script标签引入其它网站的脚本，所以可以通过JS向页面上动态添加一个script标签并且指定其 src 为一个特殊的url，对方的服务器针对这个url的请求，会进行特殊处理，如：

向head标签中动态添加以下script标签

<script src="http://api.baidu.com/weather/zhengzhou/functionName"></script>

会导致浏览器向上述URL发起一个GET请求（JSONP只能是GET请求）

对方的服务器收到这个请求后，会返回一个特殊的JS文件:

functionName && functionName({

天气数据

})

如果此时在页面中定义了functionName函数，则functionName函数会被调用，并且能够得到天气数据！

这种将JSON数据放入指定函数参数位置的跨域访问解决方案被称为JSONP。即使用JSON填充函数后面的（）内部的空白（padding）

这种方案可以跨域任意域名，但是必是对方有意这样设计才能使用。如果对方不支持将JSON数据padding到函数名后面的()中，则JSONP无法使用。

**在jQuery中 $.getJSON() 这个方法支持 JSONP !!!**

在url后面加 callback=? 即可，jQuery会自动生成函数名并将调用转交给getJSON中的回调函数。

1. 将要请求的URL发送给自己的服务端，让服务端发起请求（服务端没有跨域限制），服务端请求成功后，将数据再回传给浏览中的JS----服务端代理请求。

这种方式只要自己的服务端支持一下就可以了，是比较常用的方案，没有任何限制，而且这种方安还可以实现其它方案无法实现的功能：

通过服务端抓取别人的网页，将网页上的数据提取出来，变成JSON返回

在Node.js中，使用cheerio模块可以像使用jquery一样从HTML字符串中筛选并提取想要的数据。

1. 使用任何可以利用的浏览器端中间机制实现跨域交换数据，如：

window.name在代码中使用name变量时实际上使用的是window对象的name属性，但是name属性是window对象的内部属性。它只接受字符串值，如果给它赋其它值，将会直接被转换成字符串！！！！尤其是赋一个对象给name变量的时候，会导致数据丢失！！！(对象toString()后是[object Object])。但是name有一特别性质可以被用来做跨域数据交换name值不会随全局作用域被销毁，不管窗口跳转到哪个页面，不管窗口打开了多少个页面，name的值都是通用的。其它的，诸如 location.hash 也可以用来做跨域数据交换（主要是iframe）

**路由**

在传统的由多个页面构成的网站中不存在路由的需求，那时通过链接进行跳转（导航）就能满足需求，但是随着Ajax和移动互联网的发展，单页应用越来越多，在单页（通过1个URL打开一个页面）中模拟多页面切换没有问题，但问题时怎样构造出一个能够通过URL实现“页面切换”并且这个URL还要满足可链接、可收藏、可分享的特性就成了一个有难度的技术问题。现在成熟的解决方案是利用url中的hash fragment来的变化来触发“页面切换”，这种将url(hash fragment)与“页面”映射起来的机制被称为路由（Router）,路由中的关键技术是监听url(hash fragment)的变化，支持hashchange事件的浏览器很容易做到这一点，不支持hashchange事件的浏览器通常需要使用setInterval大概以100ms/次的频率轮询检查url的变化。很多应用框架都支持路由，如Backbone和Angular（除ngRouter外，还有uiRouter支持更强大的功能），另外还有很多支持路由功能的独立脚本库如：router.js、director.js等

**触控与手势**

随着移动设备的广泛应用，对触屏的支持势在必行。

H5中新增了Touch API来支持触控，包括：

TouchEvent表示触控事件

Touch表示一个触控点

TouchList表示一组触控点

TouchEvent是触控事件对象，比普通事件对象多了touches、targetTouches、 changedTouches属性

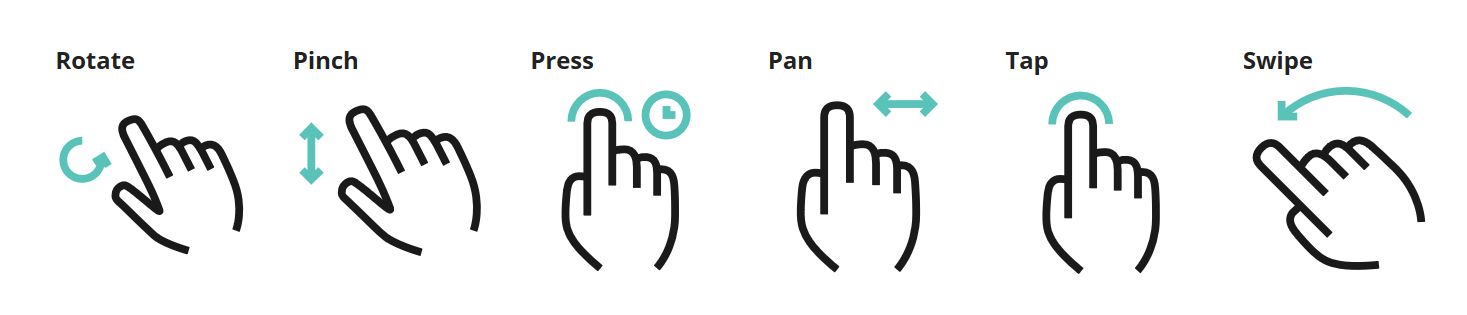
触控事件类型有touchstart、touchend、touchmove、touchcancel

Touch对象包含identifier、screenX、clientX、pageX、target等，甚至还包含radiusX、rotationAngle、force等属性

TouchList包含length、item(index)

上面的基本的触控事件比较底层，使用起来不是很方便，尤其是应用开发中如果能够直接使用手势则更加高效、除了一些常用的框架提供触控和手势支持之外，还有一些脚本库也能单独提供触控和手势的支持，如：

Hammer.js <http://hammerjs.github.io/>



Touch.js <http://touch.code.baidu.com/> （来自百度）

**ES6**

ECMA于2015年6月发布的JS标准（2016年6月小幅修订）、目前在Node.js和桌面浏览器上已经基本完成了支持（主流的现代浏览器都支持了95%以上的特性），移动端的支持较弱（支持50%的特性），目前获得最广泛支持的是ES5.1。ES7预计在2017年发布。

ES6为JS带来了大量的新特性，是JS升级最大的一个版本：

* 1. 支持let块级变量和const常量
  2. 支持变量解构赋值
  3. 扩展了JS的核心对象
  4. 增强了语言反射能力
  5. 增加了Promise
  6. 支持Class定义类
  7. 支持Module模块化
  8. 支持迭代接口和生成器等

由于ES6现在还没有被所有浏览器支持，如果想在项目中使用ES6，而且还想保证浏览器兼容性，就需要使用转换工具将ES6转码成ES5。常用的转换工具有：

1. Traceur：由谷歌提供，可以在网页上就地转换，也可以在线转换、还可以使用Node.js命令行本地转换；
2. babel：支持ES6语法转换，如果要支持新API，则还需要使用babel-core

上述两个工具的使用具体参考：<http://t.im/16ax8>

**模块化**

JS的作用域隔离机制较弱，因此使用不当时会产生命名冲突的问题，尤其是在单页应用日益复杂、使用的第三方框架越来越多的情况下更容易产生冲突问题。而且JS本身没有直接提供使用代码加载其它脚本文件的机制。因此产生了JS模块化概念。

JS模块化通常要提供2个方案：

1. 定义JS模块的方案
2. 按需加载模块及其依赖模块的方案

常用的JS模块化方案有以下几种：

Node.js中的模块化方案（最简单易用，通过约定取消了配置）

浏览器中使用中RequireJs和SeaJs（都需要配置）

ES6中的模块化方案（无需配置）

目前在浏览器端使用最广泛的是RequrieJs，RequireJs通过AMD规范描述了定义和使用模块的方法，本身支持JS文件的按需加载，另外RequireJs还有很多官方和第三方插件可以实现各种文件（如HTML、CSS、其它文本文件等）的动态加载。

**异步编程Promise/Deferred、多线程WebWorker**

长期以来JS都是以单线程的模式运行的，而JS又通常应用在操作用户界面和网络请求这些任务上。操作用户界面时不能进行耗时较长的操作否则会导致界面卡死，而网络请求和动画等就是耗时较长的操作。所以在JS中经常要进行异步编程。而最基本的异步编程方法是事件和回调函数。但无论是事件还是回调函数在遇到稍微复杂一点的场景时都会变得难以使用。如时机问题、等待问题等。这时就产生了Promise的概念。

Promise可以保证无论什么时候添加回调函数，都能使回调函数得到恰当的调用；还能保证异步任务的状态不会被篡改。JS中的Promise有多种实现方案，它们的API各有不同，但核心概念都是相似的。

ES6支持Promise、提供了resolve、reject、then、catch、race、all等最基本的API。

jQuery则通过Deferred额外提供了进度通知及在外部改变状态的API，支持resolve、reject、then、done、fail、always、progress、notify、state等，还有通过$.when()支持类似all的功能，支持通过promise()转换成Promise对象（不是ES6中的Promise，而是表示一种不可从外部更改状态的Deferred）

Angular则支持$q，它即兼有ES6中Promise和jQuery Defferred的特点。

使用Promise可以将异步任务本身与后续业务完全分离，因此可以简化异步编程。

JS在H5时增加了多线程API，即WebWorker。WebWorker是一个真正的分线程，与其它系统线程一样。但与其它编程技术中的多线程不同，它是通过消息机制与主线程交互的。因此可以理解为是放入沙盒中的线程。因为没有开放其它API，避免了产生线程死锁的可能，但功能上要弱一些。

**数据结构**

在程序中表示简单的数据很简单，但要表示一系列有特定关系的简单数据就不那么简单了。要存储和使用一系列有特定关系的数据就需要构建和使用数据结构。常见的数据结构有：

线性链表：用来表示一串数据的结构，如我们经常使用的数组、队列、栈都属于这种类型。数组可以通过索引随机访问，队列和栈都有严格的限制；队列是先进先出的，栈是后进先出的。

树：用来表示有子分支数据的结构（有分枝但无环），树在用户界面（页面上的标签元素就是以DOM树的形式存在的）和文件系统中有广泛的使用，树在组织机构等业务中也有广泛的使用。树由节点构成，节点通过parent连接上级节点就能构成树，有时为了能够更方便地使用树，还会提供children、level、path等API。B树（平衡多叉树）可用于二分查找（数据库索引常用方法）。遍历所有树节点有2种方法，即深度优先法和广度优先法。

表：用来存储有名称（或哈希值）的数据的结构。表有广泛的应用，如我们使用的Collection、Hashtable等就是表，JS中的对象也可以认为是表。

图：用来存储有环形分支数据的结构，在地图、路径规划类场景中有广泛应用，在图中求最优路径是最典型的需求。

**算法**

解决问题的操作步骤。如排序算法、查找算法、最短路径算法、图形识别算法、围棋算法等。计算机算法要求必须在有限步内可解（否则相当于死循环）。算法通常追求使用时间最短和占用内存最少，但2者往往不能兼有。用时短的算法往往占用空间较多，而占用空间少的往往需要运算的时间较长。算法除了不断追求性能外，还逐渐向着智能化的方向发展。

**加密**

数字摘要：从一个较大的数据（如一个文件或一个长字符串）中提取出一部分信息用来作为该数据的指纹（也叫哈希值）。通常用来防篡改或验证数据是否有损坏。数字摘要算法可以达到在几万字的文件中哪怕只修改一个标点就会产生截然不同的摘要的效果。常见的MD5、SHA都是数字摘要算法。数字摘要值通常是几十个字符的长度，通过数字摘要无法反向算出原数据，即在摘要时，信息会丢失。**但是，对于密码这样的短字符串，可以通过巨大的密码字典反向映射查出原字符串，因此短密码通过数字摘要的方式存储保密效果不佳！6个字符生成的数字摘要可以在1秒内查出原字符串，6位纯数字则可以缩短到毫秒内破解！**

对称加密：加密和解密都使用的是相同的密钥。通过密钥可以从密文还原出原文。对称加密最大的问题是如何保存密钥。战争时期，各种间谍努力窃取的就是密钥。DES、3DES、AES、RC5等都是对称加密算法。我们使用无线网络上网时输入的密码就需要以加密的方式发送，因为Wifi信号充满了空间，任何人都可以监听到。但用来加密的密钥本身也需要通过Wifi从无线路由传到我们的电脑上，这时就会造成密钥泄露，因此通过较长时间监听Wifi数据就能从数据中找到密钥，进而破解出Wifi密码。

非对称加密：加密和解密使用的是不同的密钥。使用非对称加密时，首先会计算出一对密钥，分别称为公钥和私钥。公钥可以分发给其他人，甚至完全公开；私钥则由自己保存。通过私钥加密的数据只能使用公钥解密，而使用公钥加密的数据只能使用私钥解密。非对称加密的这个特性非常有趣，也有很大的应用价值。RSA即是一种非对称加密。

数字签名：非对称加密技术的一种应用。可以用来保证数据发送者的身份。因为私钥只有创建密钥对的人才有，因此可以认为只要是使用某个私钥加密过的数据就一定是这个私钥的拥有者发出的数据。因为非对称加密比较慢，不适合加密大量数据，所以通常是先使用数字摘要计算出一个很短的摘要值，再对摘要值进行私钥加密。

数字证书：非对称加密技术的另一种应用。虽然公钥是可以公开的，但怎样确保我们得到的公钥是某人或某个机构的公钥而不是别人冒充的公钥呢？这时就需要有一个可以信任的机构为公钥提供证明。这种包含了公钥拥有者名称和可靠证明的公钥（通常是证明机构的数字签名）就是证书。所以在实际的通信和数据交换时，都不会直接使用裸公钥，而是要求对方提供证书。如HTTPS请求就要求网站的服务器提供证书证明公钥是可靠的（以防冒充钓鱼），我们在手机上安装的App也需要证书才能提交到App Store。值得一提的是，证书是有有效期的，证书过期将使HTTPS网站受到影响，向CA证书颁发机申请证书是需要交费的（按年计费）。

**Angular中$rootScope、$scope、$watch()、$digest()及$apply()**

$rootScope是Angular中的一项内置服务，通过它可以获得根作用域。根作用域是Angular应用启动时创建的第一个作用域（$id等于1）其它作用域都是它的子孙作用域。作用域中有$parent属性指向父作用域从而将作用域连成一颗树。大部分作用域还会将原型（prototype）指向父作用域从而形成作用域链，以继承上级作用域的属性和方法。

在控制器构造函数中通过$scope可以获取当前作用域。某些指令或组件中还存在隔离作用域，隔离作用域也在作用域树中，只是没有与上级作用域通过原型链连接起来，从而不受上级作用域的影响。使用$new()方法可以创建新的作用域，需指定是否是隔离的，及指定上级作用域。

作用域有自己单独的事件传播机制，通过作用域$emit()的事件会向作用域树的上级传播（含自己），通过作用域$broadcast()的事件会向作用域树的下级传播（也包含自己）。

作用域最重要的是提供了数据的修改监视和通知功能，通过$watch()可以监视作用域中数据的变化（当发现变化时可以更新界面或作用域中的其它数据）。$watch()通常在模版编译时被Angular调用为模版插值添加监视器，也可以由程序员调用。$watch()可以监视复杂的表达式。

$watch()的监视功能是由$digest()完成的，$digest会遍历整个作用域树循环处理每一个watcher、计算watcher中被监视表达式的值，如果表达式的值与原值（缓存的）相比发生了变化则调用watcher中的函数处理变化，因为watcher变化处理函数有可能修改作用域中的其它值，因此$digest会遍历多次（至少2次）直到作用域中的数据稳定（检测不到任何变化）为止。

Angular控制器及自带的指令（如ngClick等）、服务（如$timeout等）都会自动调用$digest()以触发监视过程，如果在非Angular执行环境中修改作用域中属性的值则不会自动激活$digest()过程，这时需要使用$apply()而不要自己调用$digest()，$apply(func)会执行func，并且在执行之后会调用$digest()，而且会恰当处理func执行过程中的错误。

具体可参考：http://dwz.cn/3Npuio

**各种代码库及框架**

|  |  |  |
| --- | --- | --- |
| Bootstrap | UI框架 | 响应式页面、基本排版、表单、常用UI组件、插件 |
| jQuery | 脚本库 | DOM/Ajax/回调函数/Deferred等工具 |
| jQuery UI | 桌面UI库 | jQuery官方推出的，用于桌面网页开发的UI库，有丰富的控件、主题较多且可以自由定制，它可以向页面上添加丰富的功能，而不会对原页面产生影响 |
| jQuery Mobile | 移动UI框架 | jQuery官方推出的，用于移动Web应用开发的UI，控件丰富，动画效果很好。 |
| Underscore.js | 脚本库 | 数组/对象/函数类工具，还包含一个HTML模板引擎 |
| Require.js | JS模块化 | JS模块定义、使用代码异步加载JS、依赖解决、其它各种文件的代码加载。Require.js有很多插件，用来实现各种各样的异步加载功能 |
| Sea.js | JS模块化 | 与Require.js大同小异，来自阿里（作者王保平、阿里昵称玉伯，现供职于支付宝前端技术部）、主要在国内应用 |
| Backbone | 应用框架  类似MVC架构 | 非常小（压缩后7K多），但功能完善的前端应用开发框架，支持模型（集合）、视图、路由等，支持RESTful风格的服务端交互，是一种思路比较经典的MVC框架 |
| Angular | 应用框架  MVVM架构  MVC架构  MVW（Angular自创词） | 非常好地体现了快速应用开发（RAD）思想的框架，通过对HTML的扩展，借助声明式语言的优势实现快速应用开发。内置大量指令，支持双向绑定，支持表单输入状态管理，表单验证，表单辅助提交，支持路由、动画、RESTful数据模型。广泛使用了依赖注入实现松耦合。Angular有大量插件，逐步形成了一个生态系统 |
| APP Framework | 移动UI框架 | 原来叫JqMobi后被Intel收购，简单、支持多种系统主题，有很好的动画效果（侧滑、转场），适合开发小产品，资料较少，应用不多 |
| Ext.js/Sencha | UI框架  应用框架  MVC架构 | 整合了树、列表、面板、布局控制、图表等全部的Windows应用控件。在所有JS的UI框架中，仿Windows风格质量最高，功能最强大的。通常用来开发各种内部使用的管理系统/业务系统。是很多类似UI框架的鼻祖。商业使用是收费的 |
| Prototype.js | 脚本库 | DOM/BOM/Ajax/数组/对象/函数工具，一整套面向对象的实现机制。历史很早的一个脚本库，侵入式的，jQuery产生后逐渐淡出 |
| Highchart.js | 统计图库 | 基于SVG技术，支持折线图、曲线图、柱形图、条形图、饼图、环形图、雷达图、极地图、蛛网图、仪表图等，有丰富的动画和交互效果，提供了简单易用、控制能力完善的API。产生时间较早，很成熟。商业使用收费。 |
| Chart.js | 统计图库 | 基于Canvas技术，支持折线、曲线、柱、饼等，因为使用Canvas技术，交互性较弱。 |
| artTemplate | HTML模板化 | 性能很高，语法简洁，支持编译，支持浏览器端使用，支持express。来自腾讯 |
| EJS | HTML模板化 | 默认使用<%%>的HTML模板语言，支持浏览器端使用，支持express |
| Jade | HTML模板化 | 非常独特的HTML模板语言，完全取消了标签中的<>和标签配对，过于独特限制了它的应用，支持express |
| ionic | 移动UI框架 | ionic是一个非常漂亮的移动UI框架、并且它集成了Cordova（参看后面的说明）和Angular，可以快速开发出与原生应用相媲美的基于Web技术的App |

**设计原则/设计模式**

面向对象编程（OOP）中最重要的是面向对象设计（OOD）。单独设计一个类并不难，难的是设计很多类，并使用这些类及它们的实例构成复杂的应用程序来满足业务需求！当使用面向对象的思想对需求进行分析（OOA，面向对象分析）逐步构成设计成果时，如果确定类的范围、如何解决类与类、包括它们的实例之间的各种关系必须需要一些设计原则作为指导才能较好地完成设计。

基本的面向对象设计原则有：

1. 单一职责原则（封装的范围和粒度）

一个类、方法应该只负责一项功能

1. 替换原则（正确使用继承和多态）

将父类看成接口，子类完全可以替换父类，

1. 依赖倒置原则（解除依赖）

面向抽象<接口/依赖注入>编程，而不是面向实现编程

1. 接口隔离原则（抽象的粒度）

接口的范围和粒度

1. 迪米特法则（尽可能减少耦合）

一个对象应该尽可能少地了解其它对象

1. 开闭原则（保持可扩展性）

对扩展开放，对修改关闭

具体可参考http://dwz.cn/3NfHS9

设计模式是一些常用的设计结构，是从大量的面向对象设计中识别并总结出来的。设计模式可以看成是特定设计场景下的经典设计方案。是一些被命名、分类、说明了使用场景的固定设计公式。由GOF<4人邦，即提出设计模式的4个人>最先总结出23种模式并出书论述而流行起来。学习设计模式可以丰富我们的设计思路，提升我们对设计的理解，可以使我们在设计上快速成长和成熟起来，也能在一定程度上提升设计的质量和速度。但不同的编程语言有不同的特点，也有不同的框架支持，因此对设计模式不可僵化理解和应用，而应该根据情况灵活变通取舍；也不可过度应用，为了应用设计模式而设计。

GOF提出的设计模式可分为3大类：

创建型模式：用于创建实例，如单例模式、工厂模式等

结构型模式：用于解决实例间的关系，如适配器模式、外观模式、代理模式等

行为型模式：用于实现某些功能：迭代器模式、观察者模式、模版方法模式等

**JS严格模式**

ES5引入的一种新的JS运行模式，在严格模式下可以改变JS历史上遗留的一些不合理的语法或运行行为（这些语法或行为不能在普通模式下直接改变，因为会造成之前的老代码出问题），JS严格模式是为了在JS中引入新特性又不致于破坏老代码而产生的解决方案。

使用字符串字面量 ‘use strict’使脚本、<script>标签或函数进入严格模式

在严格模式下：

1. 全局变量必须先声明才能使用（包括赋值）
2. 禁止使用with
3. eval执行在eval作用域中（这个作用域是新增的），这样eval就不会影响全局作用域或函数作用域了
4. this不会也不能指向全局作用域（window对象），这样构造函数不加new调用就会报错，从而不至于影响全局作用域
5. 禁止访问arguments.caller（表示调用当前函数的函数）、arguments.callee（表示当前函数自已，如用于匿名函数递归调用），也不能通过函数名访问arguments，但对arguments的使它使用（如获取参数）是正常的！
6. arguments不可被赋值，也不再追踪参数值的变化（arguments中的值不会变）
7. 禁止使用delete删除变量
8. 显式报错，而不是沉默忽略错误（如对只读属性赋值、对禁止扩展的对象添加属性、删除不能删除的属性、属性名重复、参数名重复）
9. 不支持八进制，整数第1位是0将报错
10. 函数不能在if{}、for{}等{}内部声明（以后版本会引入块级作用域）
11. 新增mplements, interface, let, package, private, protected, public, static, yield关键字，不能使用这些名称作为变量名，函数名、参数名等

具体可参考http://dwz.cn/A6tGP

**正则表达式**

使用特定字符构成的表达式，可以用来描述字符串模式，用来在字符串中搜索（匹配）符合条件的子字符串（当然也以替换）。使用正则表达式可以方便地构造出各种各样的字符串搜索条件，而且还可以将正则表达式编译成可执行代码，从而可以以极高的性能处理输大的字符串和文本文件。正则表达式常用来检验字符串格式（数据验证）和对大量文本进行数据分析，加工（数据处理）。

**Git/SVN**

源代码管理（SCM）工具，Git是分布式，任何人都可以拥有完整的代码库，几乎所有的源代码管理操作都可以不用连网，而在自己的电脑上完成，所以是开源项目的不二之选，在公司中使用也有很多优势。SVN是集中式的，代码库及历史版本都集中管理在中央服务器上，开发人员只有当前使用的版本，SVN简单、能够实现代码的严格控制，因此在要求较强控制和管理的场景下应用较多。

**PhoneGap/Cordova**

H5具有跨平台的能力，而且包含了与App相关的大量新特性，随着移动浏览器不断增强，使用H5开发（write once）然后使用原生外壳打包成各个平台的App（run anywhere）就成为一种可能。PhoneGap就这样产生了，PhoneGap的目标是弥补手机与Web之间的间隙。

PhoneGap后被Adobe收购，但PhoneGap并没有获得广泛的应用，Adobe想通过PhoneGap赚钱就必须吸引更多的人加入到PhoneGap的阵营，而又必须保留营利的机会。于是Adobe将PhoneGap捐给Apache Software Foundation成为一个开源项目（吸引更多的人开发它<降低成本>和使用它<扩大影响力>），而又保留了PhoneGap的商标。

ASF接收PhoneGap的源代码后无法使用PhoneGap的名字，只能改名，于是产生了Cordova。Adobe则在Cordova的基础上继续使用PhoneGap的名字发展相关的开发工具，如桌面版工具（PhoneGap Desktop）、移动版运行App（PhoneGap App）、和能够营利的构建云服务（PhoneGap Build）。最终的结果是，Adobe把吃力的事（写底层代码Cordova）抛给了开源组织，自己则把精力放在赚钱上（研发基于Cordova的开发工具PhoneGap）。

现在核心代码在Cordova中，PhoneGap则是Adobe研发的一系列Cordova开发工具和云服务。

Cordova提供了很多插件，可以使js获得访问原生API的能力，如控制系统状态栏、获得电池信息、访问手机存储空间等；Cordova还提供了App构建工具，将Web App打包成各种平台上的原生App，以发布到应用市场中。值得一提的是Cordova中不存在Ajax跨域限制，因此可以像原生应用一样访问任何域名下的接口。

**各种Web服务器及服务端技术**

Java系：以Java EE（最早叫J2EE）中的Servlet（Web核心）、JSP（呈现HTML）、EJB（增强并规范Java语言面向对象特性）等为核心Web服务端开发技术，随后产生了很多相关技术和框架如：Structs（MVC）、Spring（解除耦合）、Hibernate（对象和关系型数据库的映射）等。

PHP系：以PHP为核心的Web服务端开发技术，因为PHP语言及开发方式比较杂乱，因此产生了很多框架，如：Laravel、Symfony2、Nette、CodeIgniter、Yii、PHPixie、Zend Framework等，还有国产框架Think PHP。

.NET系：微软公司的Web服务端开发技术。使用C#、VB.NET等编程语言和.NET Framework，分为两大系列，早期的ASP.NET Web Form（组件化、事件驱动、与Windows应用程序的开发方式相似）、 和后来的ASP.NET MVC。另外ASP是微软在.NET之前的Web开发技术，与JSP一样古老。

其它：Node.js、Python、Ruby等

**高并发网页**

减少文件字节数（编码/代码压缩<去除空白，缩短变量名等>）

文件压缩<二进制压缩>后传输

使用客户端及网络缓存（可减少请求次数）技术

将CSS/JS/图片等资源与HTML分离到不同的服务器上（分散请求）

使用CDN（静态网页及各种资源都适用，分散请求）

服务端缓存（Memcache等Redis各种内存数据库/分布式内存数据库，提升处理速度）

集群（负载均衡网络硬件解决方案/反向代理软件解决方案，增强服务端处理能力）

不规律高并发（云计算弹性资源池，按需分配资源以解决平时资源利用不充分的问题）

大规模数据存储和计算（需用大数据技术）

大规模实时数据处理（流式大数据技术）