H5 和 CSS3 新增内容（自己总结）

HTML5 新增标签及属性
CSS3 新增样式
Web API

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，桌面网页最为困难的是兼容 IE6/7/8/等老浏览器（可以使用 IE Tester 或 IE11 的开发者工具测试）！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）。

手机浏览器中最为困难的是 Android 版的 UC 浏览器，Android 版的 UC 几乎不支持 H5 和 CSS3，甚至是常用的 CSS 样式有些都不支持，如果项目要兼容 UC 手机浏览器，那写 CSS 时，就只能使用最基本的样式！也不要指望页面能做多漂亮多复杂了。

Android 系统上除了 UC 这个奇葩之外，本身的碎片化、更新慢也是大问题。如很多手机现在的 Android 系统还是 4.4，甚至更古老（尽管 Adnroid7 都发布了）。如果开发应用内嵌页面，可以建议 Android 工程师采用腾讯出的 WebView 控件，能够使页面对 H5 的支持和微信及手机 QQ 一样好，从而达到和 iOS 同级别的效果。

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 sned 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 也比较常用，尤其是针对现代浏览器（它的使用更简单）

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。

   在用 Ionic 开发 Web App 时，通常会遇到浏览器端和服务端分离的情况，这时可使用 CORS 浏览器插件自动在响应头中添加 Access-Control-Allow-Origin 头。这样可以轻松实现开发时的跨域。

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。
$.ajax()方法也支持 JSONP，设置 type:'GET', dataType:'jsonp'即可
Angular 中的$http 服务也提供了 jsonp()方法支持 JSONP

4. 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。
5. 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），React 也通过 React-Router 提供路由支持，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，包括以下几个类（对象）：

TouchEvent 表示触控事件，类似于普通的事件（如 click）对象，只不过比普通事件对象多了 touches、targetTouches、 changedTouches 属性。

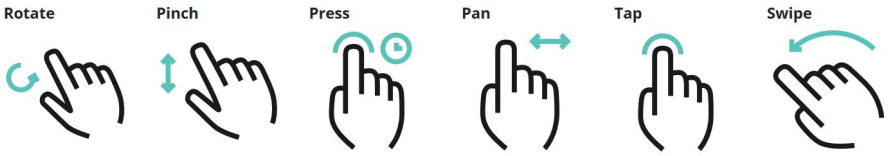
Touch 表示一个触控点（一个手指与屏幕的接触）包含 identifier（唯一识别号）、screenX、clientX、pageX、target 等，甚至还包含 radiusX（接触半径）、rotationAngle（旋转角度）、force（力度）等属性。

TouchList 表示一组触控点（多点触控）其中有多个 Touch 对象，它自身有 length（触控点个数）、item(index)（获取指定触控点）等属性。

触控事件类型有 touchstart（手指开始接触）、touchmove（手指在屏幕上移动）、touchend（手指离开屏幕）、touchcancel（受来电话或电量不足提示等影响而导致触控被中断）。监听触控事件和监听普通事件一样，使用 addEventListener()即可。

对于触摸屏来说，触控事件是优先的，而传统的点击事件是由触控事件转换而来的，即用户接触屏幕后迅速触发 touchstart 事件，300ms 之后，系统才会触发 click 事件。因此在开发手机页面时，如果简单地使用 click 事件则会带来一种轻微的延迟感。很多框架都专门针对这种延迟给予了优化，还有一个叫 faskclick.js 的脚本专门解决这个问题。

上面的基本的触控事件比较底层，使用起来不是很方便（需要监听多个事件），尤其是应用开发中如果能够直接使用**手势**则更加高效。手势是一些常用的触控操作手法，是单点或多点触摸组合加上特定的移动过程构成的。常见的手势有：



常规的框架只能支持部分手势。如：

angular 应用框架中官方提供了一个 ngTouch 模块，这个模块提供了 ngSwipeLeft 和 ngSwipeRight 指令可以像普通 HTML 中的 onclick 一样使用，分别表示向左轻扫屏幕和向右轻扫屏幕。另外该模块还有一个$touchProvider，可以解决 ngClick 的 300ms 延迟问题（与 faskclick.js 作用相同）。

ionic UI 框架也提供了一些手势支持，比 angular 支持的多一些，不过还是以单点触控为主。如 swipe 轻扫可以支持上、下、左、右 4 个方向。还支持 4 个方向拖拽 drag。支持长按 hold，轻点 tap，双击 double-tap 等。ionic 中的列表（List 和 Item）已经集成的常用的左右滑动显示操作按扭的功能，ionic 的侧滑菜单本身也支持手势操作。

如果要支持更复杂的多点触控手势，就需要使用功能更多的脚本库了，如：

Hammer.js　http://hammerjs.github.io/（最强大，它还提供了 jQuery 插件、angular 插件）

Touch.js http://touch.code.baidu.com/ （来自百度，比 Hammer 弱一点）

ES6

ECMA 于 2015 年 6 月发布的 JS 标准（也被称作 ECMA2015，该版本于 2016 年 6 月小幅修订）、目前在 Node.js 和桌面浏览器上已经完成了支持（主流的现代浏览器都支持），移动端的支持较弱（支持约 50%的特性，Android 由于碎片化严重，老机型支持更差），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 语言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(通过代码获取代码自身信息的能力，如一个类包含哪些方法)
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，如果要支持新增加的 API，则还需要使用 babel-core。Babel 目前由 Facebook 支持（因为 Facebook 的 React 给 JS 扩展了一套新的组件语法 jsx，而这个语法需要 babel 来转换成普通 js）。Babel 推荐使用 webpack 之类打包工具进行转换，Babel 也提供了 babel-standalone 支持浏览器转换，但不应在商业正式环境中使用。

上述两个工具的使用可参考：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 模块化通常要解决 2 个问题：

1. 提供定义 JS 模块的办法
2. 提供按需加载模块及其依赖模块的办法

常用的 JS 模块化方案有以下几种：

1. Node.js 中的模块化方案（通常称为 CommonJS 规范，最简单易用，通过约定取消

了配置）

2. 浏览器中使用中 RequireJs（称为 AMD 规范）和 SeaJs（称为 CMD 规范），这两个都需要配置
3. 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（jQuery、Underscore、Backbone、Angular、ionic 等流行的库和框架都支持），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。

另外还需要注意的是 Webpack 打包工具支持混合使用 Nodejs 中的 require()导及 ES6 中的 import … from …，还支持直接从 npm 模块包中导入 js，因此使用 Webpack + npm 也越来越流行，尤其是在 React 阵营中几乎成为主流。

组件化技术

自从 HTML 产生之后，就以其简单易出成果而被广泛应用，HTML 的功能也越来越强大，标签数量也越来越多，到 H5 标签已经超过 100 个。但是希望自定义 HTML 标签的人并不会因为 HTML 标签越来越多而止步，他们的期望更高，那就是可以自由地增加 HTML 标签。于是产生了各种支持自定义标签的技术，都可以称之为 Web 组件化技术，常见的有：

1. 现代浏览器支持的直接自定义新标签的技术（可以理解为是 XML），在 HTML 代码中直接写新标签，默认是 inline 的，通过 CSS 可以给予更多的控制。老浏览器不支持。
2. 通过 Angular 的指令或组件自定义标签。成熟，功能强大，商业应用广泛。ionic 中就通过 Angular 定义了大量的 UI 控件。
3. 通过 Vue.js 的组件定义标签。Vue.js 中可以定义 componet（组件），然后在模版中可以像标签一样使用（Vue.js 见下文第三方库及框架列表）。
4. 通过 React 组件定义标签。这种其实算不上自定义标签，因为它不是 HTML，而是 JS（所谓的标签其实是 jsx 组件，编译后变成了 js，与 HTML 关系不大）。
5. Web Components 技术，由 W3C 提出的技术，有望成为未来的标准。允许直接从 DOM 的 HTMLElement 类继承出新标签。这种技术是多种 Web 新技术的综合，包含自定义标签类、template 模板、shadow 封装、import 导入等，目前还处于试验阶段，除了谷歌外大多数公司都还没有提供支持，离商业应用还很远。

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（jQuery3.x 的 Deferred 已经向 ES6 靠拢）。

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。通过 .then() 的链式调用还可以减少回调函数嵌套的层级，使代码更加易于阅读和易于理解。

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。

在某一些应用框架中，为了简化开发避免复杂应用带来混乱，会强制采用单向数据流的方式解决应用内部数据传递和方法相互调用问题。典型的如 React + Redux（或者 Flux）。在这种应用框架中，表示数据的 state 是不可变的，如果要改变，只能生成一个全新的数据对象替换原对象。这时就需要不可变的数据结构。专门有一个叫 immutable.js 的脚本库（也是 Facebook 的人开发的）为 JS 提供了不可变的数据结构，其中提供了 Collection、List、Map、Set、Record、Seq 等数据结构。

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（在地图上规划路线）、图形识别算法（识别车牌号、花朵、人脸）、围棋算法（阿尔法狗）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（人工智能 AI、深度学习）。

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。（但自定义指令和组件中的**隔离作用域**只有$parent 指向父作用域而不会将自己的原型指向父作用域，因些隔离作用域不会受上级作用域干扰，这样自定义指令和组件就可以放在页面上任何地方使用了。隔离作用域的原型指向作用域类的 prototype）

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()推动执行的，$digest()会遍历整个作用域树循环处理每一个 watcher、计算 watcher 中被监视表达式的值，如果表达式的值与原值（缓存的）相比发生了变化则调用 watcher 中的函数处理变化，因为 watcher 变化处理函数有可能修改作用域中的其它值，因此$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

使用 batarang 浏览器插件可以观察 Angular 的性能，每次$digest()的时候都会在图形上显示一条竖线。

影响 Angular 性能的一个重要因素就是数据绑定及很多 Angular 指令都会向作用域中添加 watcher，一个页面上可能有成百上千个 watcher。每次$digest()时都需要执行所有的 watcher，这样 watcher 越多，性能就越受影响。假设页面上显示一个表格显示商品信息，表格有 10 列，分别显示商品名称、进价、销售价、库存等 10 个数据（通过{{}}绑定）。则显示 1000 条商品信息至少会产生 10000 个 watcher，如果 1 个 watcher 的执行需要 1ms，则页面上的任何操作触发$digest()都需要至少 10 秒才能反应过来，页面就像卡死了一样。所以减少 watcher 是关键。如有一个第三方提供了 **bindonce** 指令(bo-开头)，使用 bindonce 只会在数据第一次显示时产生 watcher，数据显示之后 watcher 就移除了，所以可以大大提升性能。

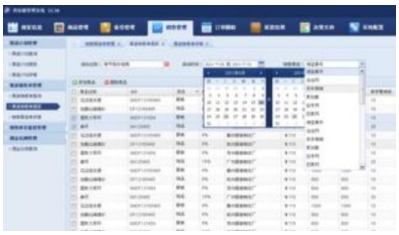
除了使用 bindonce 之外，还可以考虑优化 ngRepeat，比如使用 ionic 的列表（ionList/ionItem）时，如果加载的数据很多，如使用无限加载，应该使用 collection-repeat 替换 ng-repeat。因为 collection-repeat 会把滚动过去的列表项对应的标签元素（DOM）及 watcher 从页面上删除，这样无论列表多长都不会卡了。

如果应用比较复杂，控制器很多，还可以使用异步方式加载控制器，而不是应用一打开就加载所有的控制器（angular 的 ng-controller 及路由默认都是立刻加载控制器的）。这方面的解决方案有：angular-async-loader 模块。

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	增强了 DOM/Ajax/动画等操作，提供了 Callbacks、Deferred 等异步编程工具
Underscore.js	脚本库	增强了 JS 中的数组/对象/函数等，还包含一个 HTML 模板引擎，使用它可以简化数组数据操作，因此 Backbone 框架的集合大大依赖于 Underscore。Undercorce 还提供了函数调用节流（可以降低函数调用频率，如窗口 resize 事件处理函数、鼠标 mousemove 事件处理函数的调用频率都是很高的）等实用功能。
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。但 UI 设计风格显老。
SUI Mobile	移动 UI 框架	淘宝基于 framework7 推出，功能丰富，使用简单，多页开发模式（但可以模仿单页加载）。
Frozen UI	移动 UI 框架	Frozen UI 是一个开源的、简单易用的（与 SUI Mobile 类似），轻量快捷的移动端 UI 框架。基于手 Q 样式规范，选取最常用的组件构成的 UI 框架，目前多应用在腾讯手 Q 增值业务中。
We UI	移动 UI 框架	腾讯推出的，用于微信页面开发的 UI 框架，界面风格与微信相似度最高，功能和组件偏少，适合于简单的微信风格的移动页面开发
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。
ionic	移动 UI 框架 应用框架	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
WeX5	移动 UI 框架 应用框架	国内（起步科技）开发的类似于 ionic 的应用开发框架，在底层同样采用了 Cordova。还基于 Eclipse 提供了快速开发工具（支持拖拽式所见即所得开发）。WeX5 虽然是国内公司开发的，但即使在国内应用的也不多。与 ionic 不同，WeX5 的应用框架采用的是 Knockout.js（下面有介绍），也许这是它未能广泛应用的原因之一。
MUI	移动 UI 框架	国内（DCloud，做 Hbuilder 的那家公司）开发的类似于 SUI 的 UI 框架。这家公司（参加了国内的 HTML5 产业联盟，几家公司拼凑出来的，并没有什么影响力）还提供了一个所谓 HTML5+（几乎就是抄 Cordova)，

		通过它可以让页面上的 js 调用手机 API（需要打包成 APP 或运行在它开发的 Webview 浏览器中），不过也没有在国内获得广泛的应用（或许是缺少强大的应用框架的原因）。
React	UI 组件化库	React 提供了组件化 UI 的支持，可以很方便地开发出 UI 组件，并且支持嵌套（比 Angular 简单）。为了简化组件的使用，React 还扩展了 js，提供了 jsx。React 不算是 UI 框架或应用框架，因为它即没有提供现成的 UI 控件可以使用，也没有提供管理数据的模型和控制器机制。要想使用 React 开发一个项目，需要用到大量相关技术支持，如 Redux、Reselect、React-Router、Webpack、Babel 等，React 是一个集成了很多技术的体系，单一使用 React 是很难做出复杂项目的。因为 React 过于激进，开发体系还不是很成熟，目前使用 React 的公司还比较少。React 的远期目标是打造一个学习一次就能通吃 Web 和 iOS、Android 的应用开发技术，但要走的路还很长。
Vue.js	UI 组件化库	与 React 类似，只提供 UI 层的组件化机制，但它参考了 Angular，支持数据绑定。Vue.js 是尤雨溪（中国在美留学生）发布的一个开源项目，这兼有 React 和 Angular 的特点（甚至还有 Backbone 的影子），但从本质上来说，更接近于 React。因为 Angular 是一个完整的应用框架，而 Vue.js 只是 View 层的。 Vue.js 与 React 不同的是 React 是非常激进的，彻底革命性的，React 的目标更宏大，它为原生 App 开发提供了一个新的技术，Vue.js 则只使用了普通的 js 技术（使用模版而不是 JSX，虽然它也支持 JSX）。
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

Knockout.js	应用框架 MVVM 架构	面向对象风格的，支持数据绑定、支持自定义组件（类似于 Angular 的指令）的应用框架。国内应用很少（可能做服务端的比较喜欢）。
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。 移动互联网时代到来之后，Ext 也推出了移动开发框架，并且在应用架构上强化了 MVC，还逐渐推出了自己的应用构建工具，并改名为 Sencha。 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，因此应用较少。
jquery EasyUI	UI 库	仿 Extjs 的一个 UI 库，提供了和 Extjs 高度相似（界面和 API，甚至文档界面都高度相似，几乎可以算是抄袭了）的应用控件，与 jquery 结合更好，使用上也比 ext 后来的版本要简单，相当于 ext3.x/4.x 的程度。在 ext 收费和越来越复杂的情况下，EasyUI 真正是给大家提供了一个 Easy 的 UI 框架，因此国内有一定的应用（常用来做后台管理系统之类的，做服务端的人比较喜欢）。值得大家课余抽时间研究一下。 
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。
eChart	统计图库	百度推出，与 Highchart.js 相似。免费，开源、功能很丰富，对移动端支持也很好。产生稍晚，百度自家产品中大量使用，借助百度影响力应用会越来越多。
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

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
2. 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，
3. 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
4. 接口隔离原则（抽象的粒度）
   接口的范围和粒度
5. 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
6. 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 关键字

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（同时也无法修改全局作用域中的变量）
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，通常在 js 早期版本用于匿名函数递归调用），也不能通过函数名访问 arguments，但对 arguments 的其它使用（如获取实参）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（但删除对象的属性是正常的）
8. 显式报错（程序会停止执行），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 implements, 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。

不管将来入职的公司采用的是 Git 还是 SVN，入职时或入职后都会分配一个帐号，通过这个帐号可以获得公司的源代码并加入到开发团队中。如果是 Git，一般会给一个网址，打开之后与“码云”或 github 类似，找到项目克隆到自己的电脑上就行了（提示权限时输入用户名和密码）。如果是 SVN，则需要使用相应的 SVN 工具（如 TortoiseSVN、苹果系统中还可用 Versions 等；有些开发工具自身就支持 SVN，如 WebStorm，需要配置一下），工具装好后，还需要项目网址（通常与帐号一块提供），然后将项目 checkout（迁出）到自己的电脑上就可以了，SVN 使用比 Git 简单得多，从代码库获取代码就叫 checkout（迁出），代码修改后更新到代码库叫 checkin（迁入），其它的事都是领导或管理员操作。

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（那时 H5 还没发布），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）、移动版运行和测试用的 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 三件套）。

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（Structs、Spring、Hibernate 也常简称为 SSH）。

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

1. 减少 HTTP 请求次数，如将小图片整合成大图片(CSS 雪碧图技术)，将 js、css 打包成一个文件（Grunt/Gulp/Webpack 等）
2. 减少文件字节数（选择适当的编码方式/代码压缩<去除空白，缩短变量名等，使用 Grunt/Gulp>）
3. 文件压缩<二进制压缩>后传输（现代服务器如 IIS 等都支持的功能，属于服务端技术）
4. 使用浏览器端及网络缓存（可减少请求次数）技术
5. 将 CSS/JS/图片等资源与 HTML 分离到不同的服务器上（分散请求）
6. 使用 CDN（静态网页及各种资源都适用，分散请求，还有利于浏览器缓存）
7. 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，属于服务端技术）
8. 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，属

于服务端技术）

9. 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，属于服务端技术）
10. 大规模数据存储和计算（需用大数据技术、属于服务端技术）
    大规模实时数据处理（流式大数据技术、属于服务端技术）