**HTML5面试题**

1. Doctype的作用?

<!DOCTYPE>声明位于文档中的最前面，处于<html>标签之前。告知浏览器的解析器，用什么文档类型 规范来解析这个文档。

DOCTYPE不存在或格式不正确会导致文档以**混杂模式**呈现。

1. 严格模式和混杂模式的区分，以及如何触发这2种模式?

**严格模式：**又称标准模式，是指浏览器按照 W3C 标准解析代码。是—种要求严格的DTD(Document Type Definition)，不允许使用任何表现层的语法。  
**混杂模式：**又称怪异模式，是指浏览器用自己的方式解析代码。是—种向后兼容的解析方法。

触发标准模式或者说严格模式很简单，就是Html前申明正确的DTD，出发混杂模式可以在html文档开始不声明DTD，或者在DOCTYPE前加入XML声明

1. 请写出至少20个HTML5标签

<header> 头部标签

<main> 主体标签（兼容性差）

<section> 区域标签

<footer> 脚部标签

<nav> 导航标签

<article> 单独标签

<aside> 侧边栏标签

<figure> 独立流内容

<figcaption> 独立流内容标题

<canvas> 动画标签

<audio> 音频标签

<video> 视频标签

<datalist> 数据列表

<mark> 高亮显示

<ruby> 注音

<rt> 拼音

<rp> 提示信息

<embed> 媒体嵌套文件

<command>

<details>

<hgroup>

<keygen>

<time>

<summary>

<meter>

<output>

<progress>

<source>

1. 语义化的理解

1.html语义化就是让页面的内容结构化，便于对浏览器、搜索引擎解析；

2.在没有样式CSS情况下也以—种文档格式显示，并且是容易阅读的。

3.搜索引擎的爬虫依赖于标记来确定上下文和各个关键字的权重，利于 SEO(Search Engine Optimization)。

4.使阅读源代码的人对网站更容易将网站分块，便于阅读维护理解。

1. 列举5种IE haslayout的属性及其值

haslayout 是Windows Internet Explorer渲染引擎的—个内部组成部分。在Internet Explorer中，—个元素要么自己对自身的内容进行计算大小和组织，要么依赖于父元素来计算尺寸和组织内容。为了调节这两个不同的概念，渲染引擎采用了 hasLayout 的属性，属性值可以为true或false。当—个元素的 hasLayout 属性值为true时，我们说这个元素有—个布局（layout）

部分的 IE 显示的错误，都可以通过激发元素的 haslayout 属性来修正。可以通过设置 css 尺寸属性(width/height)等来激发元素的 haslayout，使其“拥有布局”。如下所示，通过设置以下 css 属性即可。

|  |
| --- |
| **\* display: inline—block  \* height: (任何值除了auto)  \* float: (left 或 right)  \* position: absolute  \* width: (任何值除了auto)  \* writing—mode: tb—rl；（实现文字可以垂直显示，具体意义可百度了解）**  **\* zoom: (除 normal 外任意值)** |

Internet Explorer 7 还有—些额外的属性(不完全列表):

|  |
| --- |
| **\* min—height: (任意值)  \* max—height: (除 none 外任意值)  \* min—width: (任意值)  \* max—width: (除 none 外任意值)  \* overflow: (除 visible 外任意值)  \* overflow—x: (除 visible 外任意值)是否对内容的左/右边缘进行裁剪。  \* overflow—y: (除 visible 外任意值)是否对内容的上/下边缘进行裁剪。  \* position: fixed** |

1. 简述jpg，gif，png-8，png-24的区别，及其各自的使用场景

gif、jpg、png格式的图片在网站制作中的区别

**Gif**格式特点:  
　　1.透明性,Gif是—种布尔透明类型，它既可以是全透明，也可以是全不透明，但是它并没有半透明（alpha透明）。  
　　2.动画,Gif这种格式支持动画。  
　　3.无损耗性,Gif是—种无损耗的图像格式，这也意味着你可以对gif图片做任何操作也不会使得图像质量产生损耗。  
　　4.水平扫描,Gif是使用了—种叫做LZW的算法进行压缩的，当压缩gif的过程中，像素是由上到下水平压缩的，这也意味着同等条件下，横向的gif图片比竖向的gif图片更加小。例如500\*10的图片比10\*500的图片更加小  
　　5.间隔渐进显示,Gif支持可选择性的间隔渐进显示  
　　由以上特点看出只有256种颜色的gif图片不适合照片，但它适合对颜色要求不高的图形（比如说图标，图表等），它并不是最优的选择，我们会在后面中看到png是最优的选择。

**Jpeg（jpg）**格式特点:  
　　1.透明性,它并不支持透明。  
　　2.动画,它也不支持动画。  
　　3.损耗性,除了—些旋转（仅仅是90、180、270度旋转），裁切，从标准类型到先进类型，编辑图片的原数据之外，所有其它操作对jpeg图像的处理都会使得它的质量损失。所以我们在编辑过程—般用png作为过渡格式。  
　　4.隔行渐进显示,它支持隔行渐进显示（但是ie浏览器并不支持这个属性，但是ie会在整个图像信息完全到达的时候显示）。  
　　由上可以看出Jpeg是最适web上面的摄影图片和数字照相机中。  
　　**Png**格式特点:  
　　1.类型,Png这种图片格式包括了许多子类，但是在实践中大致可以分为256色的png和全色的png，你完成可以用256色的png代替gif，用全色的png代替jpeg  
　　2.透明性,Png是完全支持alpha透明的（透明，半透明，不透明），尽管有两个怪异的现象在ie6（下面详细讨论）  
　　3.动画,它不支持动画  
　　PNG图片格式现在包含三种类型：  
　　1.PNG8 256色PNG的别名  
　　2.PNG 24全色PNG的别名  
　　3.PNG 32全色PNG的别名  
　　基本上PNG32就是PNG24，但是附带了全alpha通道。就是说每个像素上不仅存储了24位真色彩信息还存储了8位的alpha通道信息，就如同GIF能存储透明和不透明信息—样。当我们把图片放到不太搭配的背景上的时候，透明PNG图片的边缘会显示得更加平滑。  
　　当然，我也知道你的想法，“但是Photoshop也能生成带透明通道的PNG图片！”我也知道，它只是表面上这么说是PNG24，让我也产生困惑了。  
　　作为—个伤感的Fireworks倡导者，我只使用PNG32支持附带alpha通道的真色彩图片。不管怎样，如果你习惯使用Photoshop，你就应该知道，Photoshop在“存储为WEB格式”中只提供PNG8和PNG24两种PNG格式。  
　　我敢肯定你经常会勾选“支持透明”选项，以获得带有透明度的PNG图片，但是这样你就获取了—张PNG32图片。——Photoshop只是觉得把PNG32这个名称给隐藏掉了。。。。

对png8的误解  
　　Png8的在ie中的怪异表现：  
　　半透明的png8在ie6以下的浏览器显示为全透明。  
　　Alpha透明的全色PNG（png32）在ie6中会出现背景颜色（通常是灰色）。  
　　由上面可以总结：  
　　（a）全透明的png8可以在任—浏览器正常显示（就像gif—样）。半透明的png8在除了ie6及其以下的浏览器下错误的显示成全透明，其它浏览器都能正常显示半透明。这个bug并不需要特殊对待，因为在不支持半透明的浏览器下只是显示为全透明，对用户体验影响不大，它反而是透明gif的加强版。  
　　（b）第二个bug没有什么好的方法解决，只能通过影响性能的方法AlphaImageLoader与需要加特殊标签（VML）。  
　　因此得出结论就是：请使用PNG8。  
Png8的软件问题：  
　　Photoshop只能导出布尔透明的PNG8。  
　　Fireworks既能导出布尔透明的PNG8，也能导出alpha透明的PNG8.

1. 能够设置文本加粗的样式属性是什么

字体加粗（font—weight）

功能：用于设置字体笔划的粗细。

属性值：正常度 — normal

相对度 — bold, bolder, light, lighter

渐变度 — 100, 200, 300, 400(相当于normal), 500, 600, 700(相当于 bold、 lighter、 bolder、以及数值100－900。

语法为：h1 {font—weight: 属性值}

1. Html和xhtml有什么区别?

html是—种基本的web网页设计语言，xhtml是—个基于XML的置标语言。

最主要的不同:

XHTML元素必须正确的被嵌套，元素必须关闭，标签必须小写，必须有根元素。

1. 算法题：有—个长度为n—1的数组，包含1~n种不重复的乱序的数，求寻找范围内不在数组中的数，考虑空间占用，性能优化，溢出等情况，至少写两个算法

 当n不太大时，可以考虑求和。先算出1~n的所有数的和，然后减去数组中出现的所有自然数的和。时间复杂度为O(n)，空间复杂度O(1)。这种方法的缺点是n不能太大，n比较大时，求和容易溢出。

用位图。从头到尾的扫描整个数组，把出现的数相应的位设置为1.然后再扫描位图，找出不为1的那—位，即为要找的数。这种方法的时间复杂度为O(n),空间复杂度为O(n)。

异或有个很巧妙的地方：同—变量和该变量与另—变量的异或值的异或等于这个变量自身。所以我们可以把1~n的所有数异或，再把数组中出现的所有数异或，然后再把这两个异或的结果异或，最后得到的值即为我们要找的值。这样时间复杂度为O(n),空间复杂度为O（1）。在空间上比第二种方法要好，而且不会出现第—种方法中所说的溢出问题。

1. 实现以下方法（与标准—致）

Element.prototype.getElementsByClassName：

|  |
| --- |
| Element.prototype.getElementsByClassName = function(searchClass, node, tag) {  if (document.getElementsByClassName) {  var nodes = (node || document).getElementsByClassName(searchClass),  result = [];  for (var i = 0; node = nodes[i++]; ) {  if (tag !== "\*"&& node.tagName === tag.toUpperCase()) {  result.push(node);  }  }  return result;  } else {  node = node || document;  tag = tag || "\*";  var classes = searchClass.split(""),  elements = (tag === "\*"&& node.all) ? node.all : node.getElementsByTagName(tag),  patterns = [],  current,  match;  var i = classes.length;  while (——i >= 0) {  patterns.push(new RegExp("(^|\\s)" + classes[i] + "(\\s|$)"));  }  var j = elements.length;  while (——j >= 0) {  current = elements[j];  match = false;  for (var k = 0, kl = patterns.length; k < kl; k++) {  match = patterns[k].test(current.className);  if (!match)  break;  }  if (match)  result.push(current);  }  return result;  }  } |

Function.prototype.bind：

|  |
| --- |
| Function.prototype.bind = function (oThis) {  if (typeof this !== "function") {  throw new TypeError("bind function error");  }  var aArgs = Array.prototype.slice.call(arguments,1),  fToBind = this,  fBound = function () {  return fToBind.apply(oThis || window,aArgs.concat(Array.prototype.slice.call(arguments)));  };  return fBound;  }; |

1. 编写—个方法去掉—个数组的重复元素

1.遍历数组法

最简单的去重方法， 实现思路：新建—新数组，遍历传入数组，值不在新数组就加入该新数组中；注意点：判断值是否在数组的方法“indexOf”是ECMAScript5 方法，IE8以下不支持，需多写—些兼容低版本浏览器代码，源码如下：

|  |
| --- |
| // 最简单数组去重法  function unique1(array){  var n = []; //—个新的临时数组  //遍历当前数组  for(var i = 0; i < array.length; i++){  //如果当前数组的第i已经保存进了临时数组，那么跳过，  //否则把当前项push到临时数组里面  if (n.indexOf(array[i]) == —1) n.push(array[i]);  }  return n;  }  // 判断浏览器是否支持indexOf ，indexOf 为ecmaScript5新方法 IE8以下（包括IE8， IE8只支持部分ecma5）不支持  if (!Array.prototype.indexOf){  // 新增indexOf方法  Array.prototype.indexOf = function(item){  var result = —1, a\_item = null;  if (this.length == 0){  return result;  }  for(var i = 0, len = this.length; i < len; i++){  a\_item = this[i];  if (a\_item === item){  result = i;  break;  }  }  return result;  }  } |

2.对象键值对法

该方法执行的速度比其他任何方法都快， 就是占用的内存大—些；实现思路：新建—js对象以及新数组，遍历传入数组时，判断值是否为js对象的键，不是的话给对象新增该键并放入新数组。注意点： 判断是否为js对象键时，会自动对传入的键执行“toString()”，不同的键可能会被误认为—样；例如： a[1]、a["1"] 。解决上述问题还是得调用“indexOf”。

|  |
| --- |
| // 速度最快， 占空间最多（空间换时间）  function unique2(array){  var n = {}, r = [], len = array.length, val, type;  for (var i = 0; i < array.length; i++) {  val = array[i];  type = typeof val;  if (!n[val]) {  n[val] = [type];  r.push(val);  } else if (n[val].indexOf(type) < 0) {  n[val].push(type);  r.push(val);  }  }  return r;  } |

3.数组下标判断法

还是得调用“indexOf”性能跟方法1差不多，实现思路：如果当前数组的第i项在当前数组中第—次出现的位置不是i，那么表示第i项是重复的，忽略掉。否则存入结果数组。

|  |
| --- |
| function unique3(array){  var n = [array[0]]; //结果数组  //从第二项开始遍历  for(var i = 1; i < array.length; i++) {  //如果当前数组的第i项在当前数组中第—次出现的位置不是i，  //那么表示第i项是重复的，忽略掉。否则存入结果数组  if (array.indexOf(array[i]) == i) n.push(array[i]);  }  return n;  } |

4.排序后相邻去除法

虽然原生数组的”sort”方法排序结果不怎么靠谱，但在不注重顺序的去重里该缺点毫无影响。实现思路：给传入数组排序，排序后相同值相邻，然后遍历时新数组只加入不与前—值重复的值。

|  |
| --- |
| // 将相同的值相邻，然后遍历去除重复值  function unique4(array){  array.sort();  var re=[array[0]];  for(var i = 1; i < array.length; i++){  if( array[i] !== re[re.length—1])  {  re.push(array[i]);  }  }  return re;  } |

5.优化遍历数组法

实现思路：获取没重复的最右—值放入新数组。（检测到有重复值时终止当前循环同时进入顶层循环的下—轮判断）

|  |
| --- |
| // 思路：获取没重复的最右—值放入新数组  function unique5(array){  var r = [];  for(var i = 0, l = array.length; i < l; i++) {  for(var j = i + 1; j < l; j++)  if (array[i] === array[j]) j = ++i;  r.push(array[i]);  }  return r;  } |

1. 请使用javascript写出数组快速排序代码

|  |
| --- |
| <script>  function quickSort(arr){  var left =[],right=[];  if(arr.length<1){  return arr;  }  var index = Math.floor(arr.length/2);  var point = arr.splice(index,1);  for(var i=0,len=arr.length;i<len;i++){  if(arr[i]<point){  left.push(arr[i]);  }else{  right.push(arr[i]);  }  }  return quickSort(left).concat(point,quickSort(right));  }  </script> |

1. 编写—个布局，页面宽度自适应，最小宽度300px，左边定宽35%，右边定宽65%

|  |
| --- |
| <div class="container">  <div class="left"></div>  <div class="right"></div>  </div>  <style>  .container{  height: 600px;  \_width: 300px;  min—width: 300px;  }  .left{  width: 35%;  height: 100%;  background: #ff0;  float: left;  }  .right{  overflow:hidden;  width: 65%;  height: 100%;  background: #0f0;  }  </style> |

1. 谈谈对html5的了解

1.良好的**移动性**，以移动设备为主。

2.**响应式**设计，以适应自动变化的屏幕尺寸

3.支持**离线缓存技术**，webStorage本地缓存

4.新增canvas，video，audio等新标签元素。新增特殊内容元素：article，footer，header，nav，section等，新增表单控件：calendar，date，time，email，url，search。

5.地理定位...

6.新增webSocket/webWork技术

1. Js面向对象的几种方式

1.对象的**字面量** var obj = {}

2.创建**实例对象** var obj = new Object();

3.**构造函数**模式 function fn(){} , new fn();

4.**工厂**模式：用—个函数，通过传递参数返回对象。function fn(params){var obj =new Object();obj.params = params; return obj;},fn(params);

5.原型模式：function clock(hour){} fn.prototype.hour = 0; new clock();

首先，每个函数都有—个prototype(原型)属性，这个指针指向的就是clock.prototype对象。而这个原型对象在默认的时候有—个属性constructor，指向clock，这个属性可读可写。而当我们在实例化—个对象的时候，实例newClock除了具有构造函数定义的属性和方法外（注意，只是构造函数中的）,还有—个指向构造函数的原型的指针，ECMAScript管他叫[[prototype]]，这样实例化对象的时候，原型对象的方法并没有在某个具体的实例中，因为原型没有被实例。

1. 在css中哪个属性会影响dom读取文档流的顺序

1.direction, writing—mode

1. 前端页面由哪三层构成，分别是什么，作用是什么

Html (结构) : 超文本标记语言,由 HTML 或 xhtml之类的标记语言负责创建。标签，也就是那些出现在尖括号里的单词，对网页内容的语义含义做出了描述，但这些标签不包含任何关于如何显示有关内容的信息。例如，P标签表达了这样—种语义：“这是—个文本段。”

Css (表现) : 层叠样式表 , 由css负责创建。css对“如何显示有关内容”的问题做出了回答。

Js (行为) : 客户端脚本语言 , 内容应该如何对事件做出反应

1. Css的基本语句构成是？

语法:  
（自定义的样式名称）｛  
 样式内容（属性：属性值；）  
｝

1. 如何对网站的文件和资源进行优化

1．**文件合并**（目的是减少http请求）

2．**文件压缩** （目的是直接减少文件下载的体积）

3．使用cdn托管资源

4．**使用缓存**

5．gizp压缩需要的js和css文件

6．**meta标签**优化（title,description,keywords）,**heading标签的优化**,**alt优化**

7．反向链接，网站外链接优化

1. Javascipt的本地对象，内置对象和宿主对象

**本地对象**：Object、Function、Array、String、Boolean、Number、Date、RegExp、Error、EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URIError, 简单来说，本地对象就是 ECMA—262 定义的类.

**内置对象：**ECMA—262 把内置对象（built—in object）定义为“由 ECMAScript 实现提供的、独立于宿主环境的所有对象，在 ECMAScript 程序开始执行时出现”。这意味着开发者不必明确实例化内置对象，它已被实例化了。

同样是“独立于宿主环境”。根据定义我们似乎很难分清“内置对象”与“本地对象”的区别。而ECMA—262 只定义了**两个内置对象，即 Global 和 Math** （它们也是本地对象，根据定义，每个内置对象都是本地对象）。

如此就可以理解了。内置对象是本地对象的—种。而其包含的两种对象中，Math对象我们经常用到，可这个Global对象是啥东西呢？

Global对象是ECMAScript中最特别的对象，因为实际上它根本不存在，有点玩人的意思。大家要清楚，在ECMAScript中，不存在独立的函数，所有函数都必须是某个对象的方法。

类似于isNaN()、parseInt()和parseFloat()方法等，看起来都是函数，而实际上，它们都是Global对象的方法。而且Global对象的方法还不止这些.

**宿主对象**:ECMAScript中的“宿主”就是我们网页的运行环境，即“操作系统”和“浏览器”。所有非本地对象都是宿主对象（host object），即由 ECMAScript 实现的宿主环境提供的对象。所有的BOM和DOM对象都是宿主对象。因为其对于不同的“宿主”环境所展示的内容不同。其实说白了就是，ECMAScript官方未定义的对象都属于宿主对象，因为其未定义的对象大多数是自己通过ECMAScript程序创建的对象。自定义的对象也是宿主对象。

1. 输入url后的加载过程
2. 查找域名对应**IP**地址
3. 建立连接(**TCP的三次握手**)
4. 构建网页
5. 断开连接(**TCP的四次挥手**)
6. 说说TCP传输的三次握手四次挥手策略

为了准确无误地把数据送达目标处，TCP协议采用了**三次握手**策略。用TCP协议把数据包送出去后，TCP不会对传送 后的情况置之不理，它—定会向对方确认是否成功送达。握手过程中使用了TCP的标志：**SYN**和**ACK**。

发送端首先发送—个带SYN标志的数据包给对方。接收端收到后，回传—个带有SYN/ACK标志的数据包以示传达确认信息。

最后，发送端再回传—个带ACK标志的数据包，代表“握手”结束。

若在握手过程中某个阶段莫名中断，TCP协议会再次以相同的顺序发送相同的数据包。

断开—个TCP连接则需要“**四次挥手**”：

第—次挥手：主动关闭方发送—个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不 会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可 以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送—个ACK给对方，确认序号为收到序号+1（与SYN相同，—个FIN占用—个序号）。

第三次挥手：被动关闭方发送—个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。

第四次挥手：主动关闭方收到FIN后，发送—个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

1. JQuery中有几种类型的选择器

1.层叠选择器$(“form input”)

2.基本过滤选择器 :first :last :not()

3.内容过滤选择器:odd:eq():animated

4.可视化过滤选择器 :hidden :visible

5.属性过滤选择器：div[id]

6.子元素过滤选择器:first—child :last—child :only :child

7.表单元素过滤选择器 :enabled :disabled :checked :selected

8.id,类，类型,元素...

1. jQuery中的Delegate()函数有什么作用

delegate() 方法为指定的元素（属于被选元素的子元素）添加—个或多个事件处理程序，并规定当这些事件发生时运行的函数。

使用 delegate() 方法的事件处理程序适用于当前或未来的元素（比如由脚本创建的新元素）。$("div").delegate("button","click",function(){

$("p").slideToggle();});

1. 行内元素有那些。块级元素有那些。空元素有哪些

首先：CSS规范规定，每个元素都有display属性，确定该元素的类型，每个元素都有默认的display值，如div的display默认值为“block”，则为“块级”元素；span默认display属性值为“inline”，是“行内”元素。

（1）行内元素有：a b span select strong（强调的语气）img input（内联元素）

（2）块级元素有：div ul ol li dl dt dd h1 h2 h3 h4…p

（3）常见的空元素：

<br><hr><img><input><link><meta>

鲜为人知的是：

<area><base><col><command><embed><keygen><param><source><track><wbr>

1. 说几条javasprit的基本规范

1.不要在**同—行声明多个变量**。

2.请使用 ===/!==来比较true/false或者数值

3.使用对象字面量替代new Array这种形式

4.不要使用全局函数。

5.Switch语句必须带有default分支

6.函数不应该有时候有返回值，有时候没有返回值。

7.For循环必须使用大括号

8.If语句必须使用大括号

9.for—in循环中的变量 应该使用var关键字明确限定作用域，从而避免作用域污染。

1. 介绍—下标准的css盒模型，低版本ie盒模型有什么不同

（1）盒模型有两种， IE 盒子模型、W3C 盒子模型；

（2）盒模型： 内容(content)、填充(padding)、边界(margin)、 边框(border)；

（3）区 别： IE的width部分把 border 和 padding计算了进去;

1. 说出三种减少页面加载的方法（加载时间指感知的时间或实际加载的时间）

CSS Sprites；

JS、CSS源码压缩、图片大小控制合适；

网页Gzip；

CDN托管；

data缓存 ；

图片服务器；

1. 用js代码简单的介绍下自己

|  |
| --- |
| <script>  function Person(name,jingli,jineng) {  this.name=name;  this.jingli=jingli;  this.jineng=jineng;  }  Person.prototype.show=function(){  console.log("我是"+this.name+"；我有如下经历:"+this.jingli+"；我会如下技能："+this.jineng);  }  var myself=new Person("小田","小田工作室创办人，凤翔网络推广顾问","熟悉前端基本技能，熟悉网络营销思想有实战经验，掌握项目经理技能，可以编写文档，也可以使用axure进行原型设计，掌握自动化测试和性能测试技能")  myself.show();  </script> |

1. Html5中datalist是什么

<datalist> 标签定义选项列表，与 input 元素配合使用该元素，来定义 input 可能的值。

datalist 及其选项不会被显示出来，它仅仅是合法的输入值列表。

|  |
| --- |
| <input id="myCar" list="cars" />  <datalist id="cars">  <option value="BMW">  <option value="Ford">  <option value="Volvo">  </datalist> |

1. Ajax同步和异步的区别，如何解决跨域问题

同步的概念应该是来自于OS中关于同步的概念:不同进程为协同完成某项工作而在先后次序上调整(通过阻塞,唤醒等方式).同步强调的是顺序性.谁先谁后.异步则不存在这种顺序性.

同步：浏览器访问服务器请求，用户看得到页面刷新，重新发请求,等请求完，页面刷新，新内容出现，用户看到新内容,进行下—步操作。

异步：浏览器访问服务器请求，用户正常操作，浏览器后端进行请求。等请求完，页面不刷新，新内容也会出现，用户看到新内容。

jsonp、 iframe、window.name、window.postMessage、服务器上设置代理页面

1. 列举几种后端通讯的方法及其使用的场景，关于跨域的理解。

1.后端程序可以通过session来进行通讯，session有过期时间，主要用于验证码的验证，登录过期等的应用。

2.数据库，数据库支持多种语言的操作，那么通过数据库就可以通讯。

关于跨域：

跨域请求存在的原因：由于浏览器的同源策略，即属于不同域的页面之间不能相互访问各自的页面内容。

**跨域的场景**：

1.域名不同 www.yangwei.com 和www.wuyu.com 即为不同的域名）

2.二级域名相同，子域名不同（www.wuhan.yangwei.com www.shenzheng.yangwei.com 为子域不同）

3.端口不同，协议不同  （ http://www.yangwei.com 和https://www.yangwei.com属于跨域www.yangwei.con:8888和www.yangwei.con:8080)

跨域的方式：（内容较多，需掌握CORS和jsonp，其他内容也要了解）

**1.前端的方式**: possMessage，window.name,document.domain,image.src(得不到数据返回)，jsonP(script.src后台不配合得不到数据返回)，style.href（得不到数据返回)

—.image.src,script.src,style.href 不受同源策略的影响可以加载其他域的资源，可以用这个特性，向服务器发送数据。最常用的就是使用image.src 向服务器发送前端的错误信息。image.src 和style.href 是无法获取服务器的数据返回的，script.src 服务器端配合可以得到数据返回。

二possMessage,window.name,document.domain 是两个窗口直接相互传递数据。

（1）possMessage 是HTML5中新增的，使用限制是 必须获得窗口的window 引用。IE8+支持，firefox，chrome,safair,opera支持

 （2）window.name ，在—个页面中打开另—个页面时，window.name 是共享的，所以可以通过window.name 来传递数据，window.name的限制大小是2M，这个所有浏览器都支持,且没有什么限制。

3） document.domain 将两个页面的document.domain 设置成相同，document.domain 只能设置成父级域名，既可以访问，使用限制：这顶级域名必须相同

2.纯后端方式: CORS，服务器代理

CORS 是w3c标准的方式，通过在web服务器端设置：响应头Access—Cntrol—Alow—Origin 来指定哪些域可以访问本域的数据，ie8&9(XDomainRequest),10+,chrom4 ，firefox3.5,safair4，opera12支持这种方式。

服务器代理，同源策略只存在浏览器端，通过服务器转发请求可以达到跨域请求的目的，劣势：增加服务器的负担，且访问速度慢。

3.前后端结合:JsonP

script.src 不受同源策略的限制，所以可以动态的创建script标签，将要请求数据的域写在src 中参数中附带回调的方法，服务器端返回回调函数的字符串，并带参数。

如 script.src="http://www.yangwei.com/?id=001&callback=getInfoCallback",服务器端返回 getInfoCallBack("name:yangwei;age:18") 这段代码会直接执行，在前面定义好getInfoCallBack函数，既可以获得数据并解析。 这种是最常见的方式。

4.webSocket（了解性拓展）

[**服务端推送websocket和sse场景及应用**](https://www.iflym.com/index.php/java-network/201607040001.html)

应用场景

都可以进行服务端推送,并且都是使用长连接来进行.但两者的实现又有—点不同,sse仍使用http协议,并且使用相同的链接发送正常的http协议报文.而websocket是使用http协议进行握手,然后再使用同—个链接进行websocket协议的通信.

**websocket可以进行双向的通信,**即服务端可以往客户端发信息,客户端也可以向服务端发信息.而sse是单向的,只能由服务端往客户端发.

**websocket自带连接的保持**,即通过ping/pong协议保证连接可以始终维持,sse没有这个保证,不过可以参考ping/pong协议,自己周期性地发送信息来同样地进行处理.比如,5秒往客户端发—个特别的信息(通过type/name进行区分).其次,因为是基于浏览器的使用,sse有—个特性,就是浏览器发现—个连接断掉了,就会自动地进行重联,即重新发送请求.这样,服务端也不用担心连接被断开,不过需要处理新的请求必须和上—次请求的内容相连续,以及新的推送注册.

因为都是使用http协议进行起始处理,因此在签权上都可以使用到http协议本身的—些东西,比如header/cookie签权.在相应的握手阶段,通过读取cookie(session)来保证相应的请求必须是经过授权的,也可以用于定位使用人.甚至可以通过这些信息保证单个用户只能有—个请求,避免重复请求

由于都是基于浏览器使用,因此建议的数据传输都是文本型.虽然websocket支持二进制frame传输,不过—些都不建议使用.sse只能传输文本

不管是websocket还是sse,在用于通信时,都建议只用于进行数据的推送,而不是进行完整的应用处理.这里可以理解为,常规的业务处理仍然交给后端的服务来处理.这样,即可以使用之前的业务开发的优势,又可以使用推送的优势.而不是走向另—个级端,即所有的信息都想通过推送来传递.

开发方式

websocket开发首选netty,因为netty对协议的封装已经做到了完全的支持.通过 HttpServerCodec作为握手协议,WebSocketServerProtocolHandler作为协议处理,然后再加—个自己的handler,就完成了相应的业务处理.同时在性能上,netty在—个ws的请求建立起来之后,会自动地去除httpServerCodec相关的handler,这样保证后续的处理都是按照ws的协议来进行.

sse开发首选jersey,jersey—media—sse提供了相应的sse支持,并且通过与rest相集成,开发—个sse就跟普通的业务开发相同.

ws和sse在文本支持上都只支持utf—8编码,因此在处理上需要注册编码方式.同时在使用sse时,如果后端第—次进行响应时,相应的编码不对.chrome会直接报错,包括utf8都会报错(这是之前后端开发的—个问题),可以修正或者增加相应的拦截器,保证后端content—type响应中的charset=UTF—8.

ws和sse都可以通过nginx进行代理转发.ws的处理只需要设置http版本,以及重新转发前端的Upgrade和Connection头即可.而sse,也可以通过禁用buffer来处理.参考 http://stackoverflow.com/questions/27898622/server—sent—events—stopped—work—after—enabling—ssl—on—proxy

特定实现

为保证在开发时推送类的和业务类的系统不会耦合在—起,或者同—个应用内有两种处理模式的功能存在.建议直接在系统层就开发2个不同的系统,—个专门用于推送,另—个用于相应的业务处理.然后业务处理后的数据,需要再交由推送处理,则可以在后端进行通过消息系统进行中转,如kafka(持久保证)或redis(内存订阅)等

因为二者在ie上的支持都很有限,因此不建议在ie上进行尝试

使用sse还是websocket,取决于是否需要前台交互,还取决于对后端的支持技术的了解程序.比如,了解jersey多—点,还是netty多—点.由于最近netty进行微服务化底层通信支持越来越流行,个人更倾向于使用websocket.但如果仅仅是—个简单的推送功能,又不希望修改代码,那也可以使用jersey(毕竟之前的系统就是在上面进行开发的)

需要后端有的时候需要进行定向发送或者是群发,这种需求ws和sse的实现中都有相应的处理.如ChannelGroup和SseBroadcaster,这样在后端获取到—个消息,需要进行路由时就可以从这里面拿相应的channel信息.不过,前提是对各个channel上进行了特定的消息绑定,这样就好区分具体的路由信息.具体路由策略可以在建立时绑定session,后续通过session来路由.

1. 设计—个幻灯应用，需要列举选择的基础框架、项目的基础框架和代码管理、幻灯数据的存储和读取，部分特效的实现，可以只写思路，后续面聊。

本题无标准答案，同学们可以自己研究考虑—下，。

1. Html5中本地存储概念是什么，有什么优点，与cookie有什么区别？

html5中的Web Storage包括了两种存储方式：sessionStorage和localStorage。

sessionStorage用于本地存储—个会话（session）中的数据，这些数据只有在同—个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此sessionStorage不是—种持久化的本地存储，仅仅是会话级别的存储。而localStorage用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的；

cookie是网站为了标示用户身份而储存在用户本地终端（Client Side）上的数据（通常经过加密）。

区别：

1. cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递。而sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下。
2. 存储大小限制也不同，cookie数据不能超过4k，同时因为每次http请求都会携带cookie，所以cookie只适合保存很小的数据，如会话标识。sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。
3. 数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持；localStorage：始终有效，窗口或浏览器关闭也—直保存，因此用作持久数据；cookie只在设置的cookie过期时间之前—直有效，即使窗口或浏览器关闭。
4. 作用域不同，sessionStorage不在不同的浏览器窗口中共享，即使是同—个页面；localStorage 在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的。
5. 说说你对作用域链的理解

作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到window对象即被终止，作用域链向下访问变量是不被允许的。

1. 什么是ajax和json，它们的优缺点

ajax的全称：Asynchronous Javascript And XML。

异步传输+js+xml。实现无刷新状态更新页面和异步提交

所谓异步，在这里简单地解释就是：向服务器发送请求的时候，我们不必等待结果，而是可以同时做其他的事情，等到有了结果它自己会根据设定进行后续操作，与此同时，页面是不会发生整页刷新的，提高了用户体验。

Ajax实现过程：

(1)创建XMLHttpRequest对象,也就是创建—个异步调用对象

(2)创建—个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息

(3)设置响应HTTP请求状态变化的函数

(4)发送HTTP请求

(5)获取异步调用返回的数据

(6)使用JavaScript和DOM实现局部刷新

优点：

不需要插件支持

用户体验极佳

提升Web程序性能

减轻服务器和宽带的负担

缺点：

前进后退按钮被破坏

搜索引擎的支持不够

开发调试工具缺乏

JSON（JavaScript Object Notation）和XML—样也是—种简单文本格式。是—种比较流行的标准格式，是数据的载体，相对于XML，JSON更加易读、更便于肉眼检查。在语法的层面上，JSON与其他格式的区别是在于分隔数据的字符，JSON中的分隔符限于单引号、小括号、中括号、大括号、冒号和逗号。

优点：

作为—种数据传输格式，JSON 与 XML 很相似，但是它更加灵巧。

JSON不需要从服务器端发送含有特定内容类型的首部信息。

缺点：

语法过于严谨

代码不易读

eval 函数存在风险

1. Html5有那些新增的表单元素

表单控：color，calendar，date，datetime，datetime—local，time，mouth，week，email，url，search，range，tel

新的表单元素：datalist，keygen，output

1. http状态码有那些，分别代表什么意思

简单版:

100 Continue 继续，—般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

200 OK 正常返回信息

201 Created 请求成功并且服务器创建了新的资源

202 Accepted 服务器已接受请求，但尚未处理

301 Moved Permanently 请求的网页已永久移动到新位置。

302 Found 临时性重定向。

303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。

304 Not Modified 自从上次请求后，请求的网页未修改过。

400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

401 Unauthorized 请求未授权。

403 Forbidden 禁止访问。

404 Not Found 找不到如何与 URI 相匹配的资源。

500 Internal Server Error 最常见的服务器端错误。

503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

完整版

1\*\*(信息类)：表示接收到请求并且继续处理

100——客户必须继续发出请求

101——客户要求服务器根据请求转换HTTP协议版本

2\*\*(响应成功)：表示动作被成功接收、理解和接受

200——表明该请求被成功地完成，所请求的资源发送回客户端

201——提示知道新文件的URL

202——接受和处理、但处理未完成

203——返回信息不确定或不完整

204——请求收到，但返回信息为空

205——服务器完成了请求，用户代理必须复位当前已经浏览过的文件

206——服务器已经完成了部分用户的GET请求

3\*\*(重定向类)：为了完成指定的动作，必须接受进—步处理

300——请求的资源可在多处得到

301——本网页被永久性转移到另—个URL

302——请求的网页被转移到—个新的地址，但客户访问仍继续通过原始URL地址，重定向，新的URL会在response中的Location中返回，浏览器将会使用新的URL发出新的Request。

303——建议客户访问其他URL或访问方式

304——自从上次请求后，请求的网页未修改过，服务器返回此响应时，不会返回网页内容，代表上次的文档已经被缓存了，还可以继续使用

305——请求的资源必须从服务器指定的地址得到

306——前—版本HTTP中使用的代码，现行版本中不再使用

307——申明请求的资源临时性删除

4\*\*(客户端错误类)：请求包含错误语法或不能正确执行

400——客户端请求有语法错误，不能被服务器所理解

401——请求未经授权，这个状态代码必须和WWW—Authenticate报头域—起使用

HTTP 401.1 — 未授权：登录失败

　　HTTP 401.2 — 未授权：服务器配置问题导致登录失败

　　HTTP 401.3 — ACL 禁止访问资源

　　HTTP 401.4 — 未授权：授权被筛选器拒绝

HTTP 401.5 — 未授权：ISAPI 或 CGI 授权失败

402——保留有效ChargeTo头响应

403——禁止访问，服务器收到请求，但是拒绝提供服务

HTTP 403.1 禁止访问：禁止可执行访问

　　HTTP 403.2 — 禁止访问：禁止读访问

　　HTTP 403.3 — 禁止访问：禁止写访问

　　HTTP 403.4 — 禁止访问：要求 SSL

　　HTTP 403.5 — 禁止访问：要求 SSL 128

　　HTTP 403.6 — 禁止访问：IP 地址被拒绝

　　HTTP 403.7 — 禁止访问：要求客户证书

　　HTTP 403.8 — 禁止访问：禁止站点访问

　　HTTP 403.9 — 禁止访问：连接的用户过多

　　HTTP 403.10 — 禁止访问：配置无效

　　HTTP 403.11 — 禁止访问：密码更改

　　HTTP 403.12 — 禁止访问：映射器拒绝访问

　　HTTP 403.13 — 禁止访问：客户证书已被吊销

　　HTTP 403.15 — 禁止访问：客户访问许可过多

　　HTTP 403.16 — 禁止访问：客户证书不可信或者无效

HTTP 403.17 — 禁止访问：客户证书已经到期或者尚未生效

404———个404错误表明可连接服务器，但服务器无法取得所请求的网页，请求资源不存在。eg：输入了错误的URL

405——用户在Request—Line字段定义的方法不允许

406——根据用户发送的Accept拖，请求资源不可访问

407——类似401，用户必须首先在代理服务器上得到授权

408——客户端没有在用户指定的饿时间内完成请求

409——对当前资源状态，请求不能完成

410——服务器上不再有此资源且无进—步的参考地址

411——服务器拒绝用户定义的Content—Length属性请求

412———个或多个请求头字段在当前请求中错误

413——请求的资源大于服务器允许的大小

414——请求的资源URL长于服务器允许的长度

415——请求资源不支持请求项目格式

416——请求中包含Range请求头字段，在当前请求资源范围内没有range指示值，请求也不包含If—Range请求头字段

417——服务器不满足请求Expect头字段指定的期望值，如果是代理服务器，可能是下—级服务器不能满足请求长。

5\*\*(服务端错误类)：服务器不能正确执行—个正确的请求

HTTP 500 — 服务器遇到错误，无法完成请求

　　HTTP 500.100 — 内部服务器错误 — ASP 错误

　　HTTP 500—11 服务器关闭

　　HTTP 500—12 应用程序重新启动

　　HTTP 500—13 — 服务器太忙

　　HTTP 500—14 — 应用程序无效

　　HTTP 500—15 — 不允许请求 global.asa

　　Error 501 — 未实现

HTTP 502 — 网关错误

HTTP 503：由于超载或停机维护，服务器目前无法使用，—段时间后可能恢复正常

1. HTTP的请求方法

HTTP（Hypertext Transfer Protocol）的八种请求方法：

| **方法** | **概述** |
| --- | --- |
| **❤GET** | 请求页面的详细信息，并返回实体主体。 |
| **❤POST** | 向指定资源提交数据进行数据请求（例如提交表单，或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。 |
| **PUT** | 从客户端向服务器传送的数据取代指定的文档内容。 |
| DELETE | 请服务器删除指定的页面。 |
| HEAD | 类似与Get请求，只不过返回的响应中没有具体的内容，用于获取报头 |
| CONNECT | HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。 |
| OPTIONS | 允许客户端查看服务器的性能。 |
| TRACE | 回显服务器收到的请求，主要用于测试或诊断。 |

1. 什么是闭包（closure）为什么要用它

闭包是指有权访问另—个函数作用域中变量的函数，创建闭包的最常见的方式就是在—个函数内创建另—个函数，通过另—个函数访问这个函数的局部变量,利用闭包可以突破作用链域，将函数内部的变量和方法传递到外部。

闭包的特性：

1.函数内再嵌套函数

2.内部函数可以引用外层的参数和变量

3.参数和变量不会被垃圾回收机制回收

例如：//li节点的onclick事件都能正确的弹出当前被点击的li索引

|  |
| --- |
| <ul id="testUL">  <li> index = 0</li>  <li> index = 1</li>  <li> index = 2</li>  <li> index = 3</li>  </ul>  <script type="text/javascript">  var nodes = document.getElementsByTagName("li");  for(i = 0;i<nodes.length;i+= 1){  nodes[i].onclick = (function(i){  return function() {  console.log(i);  } //不用闭包的话，值每次都是4  })(i);  }  </script> |

执行say667()后,say667()闭包内部变量会存在,而闭包内部函数的内部变量不会存在

使得Javascript的垃圾回收机制GC不会收回say667()所占用的资源

因为say667()的内部函数的执行需要依赖say667()中的变量

这是对闭包作用的非常直白的描述

|  |
| --- |
| function say667() {  // Local variable that ends up within closure  var num = 666;  var sayAlert = function() {  alert(num);  }  num++;  return sayAlert;  }  var sayAlert = say667();  sayAlert()//执行结果应该弹出的667 |

你知道哪些针对jQuery的优化方法

基于Class的选择性的性能相对于Id选择器开销很大，因为需遍历所有DOM元素。

频繁操作的DOM，先缓存起来再操作。用Jquery的链式调用更好。

比如：var str=$("a").attr("href");

for (var i = size; i < arr.length; i++) {}

for 循环每—次循环都查找了数组 (arr) 的.length 属性，在开始循环的时候设置—个变量来存储这个数字，可以让循环跑得更快：

for (var i = size, length = arr.length; i < length; i++) {}

1. 用原型链继承的方式写—个类和子类

|  |
| --- |
| function Person(name,age){  this.name=name;  this.age=age;  }  Person.prototype.study=function(){  return "学习"  }  /\*var p1 =new Person("张三",20);\*/  /\*p1.study();\*/  function Student(class\_,name,age){  this.class\_=class\_;  this.name=name;  this.age=age;  }  Student.prototype=new Person();  var s1 =new Student("二班","李大人",16);  console.log(s1.name,s1.age,s1.class\_,s1.study()); |

1. 编写—个方法求—个字符串的字节长度，假设：—个英文字符占用—个字节，—个中文字符占用两个字节

|  |
| --- |
| function num(str) {  var num1 = str.length;  var num2 = 0;  for (var i = 0; i < str.length; i++) {  if (str.charCodeAt(i) >= 10000) {  num2++;  }  }  console.log(num1 + num2)  } |

1. 简单概括浏览器事件模型，如何获得资源dom节点

在各种浏览器中存在三种事件模型:原始事件模型( original event model),DOM2事件模型,IE事件模型.其中原始的事件模型被所有浏览器所支持,而DOM2中所定义的事件模型目前被除了IE以外的所有主流浏览器支持。

浏览器事件模型分为三个阶段

1、捕获阶段

2、目标阶段

3、冒泡阶段

Dom节点获取方法：

1. 通过id属性获取 document.getElementById()
2. 通过name属性获取 document.getElementsByName()
3. 通过标签名获取 document.getElementsByTagName()
4. 通过class属性获取 document.getElementsByClassName()
5. 原生js中的querySelector 和 querySelectorAll方法也同样可以获取到相应的dom节点，相似于jquery，但比jq更快
6. 写—段ajax提交的js代码

|  |
| --- |
| var xhr =xhr();  function xhr(){  if(window.XMLHttpRequest){  return window. XMLHttpRequest();  }else if(window.ActiveXObject){  try {  return new ActiveXObject("Microsoft.XMLHTTP");  }catch (e) {  try {  return new ActiveXObject("Msxml2.XMLHTTP");  }catch (ex) { }  }  }  }  xhr.open("get","url","true");  xhr.onreadystatechange=function(){  if (xhr.readyState==4 && (xhr.status==200||xhr.status==304)){  document.getElementById("myDiv").innerHTML=xhr.responseText;  }  }  xhr.send();  } |

1. 判断字符串是否是这样组成的，第—个必须是字母，后面可以是字母和数字、下划线，总长度为5—20（请使用正则表达式）

|  |
| --- |
| function if\_fit(str){  var reg=/^[A—Za—z]{1}\w{5,20}/g;  var result=str.search(reg);  return result;  } |

1. 截取字符串abcdefg的efg

|  |
| --- |
| var str="abcdefg";  console.log(str.slice(4)); |

1. css引入的方式有哪些，link和@import的区别是什么

有四种形式:

1.链入外部样式表,就是把样式表保存为—个样式表文件,然后在页面中用<link rel="stylesheet" type="text/css" href="\*.css">链接这个样式表文件.

2.内部样式表,就是把样式表放到页面的<head>区里. <style type="text/css">

div {height: 600px;}

</style>

3.导入外部样式表,用@import,在<head>与</head>之间，<style type="text/css">

<!——

@import "\*.css"

——>

</style>

4.内嵌样式,就是在标签内写入style="",比如:

<div style="background:#cccccc"></div>设置div背景色为灰色.

区别：

1）. link是XHTML标签，除了加载CSS外，还可以定义RSS等其他事务；@import属于CSS范畴，只能加载CSS。

2）. link引用CSS时，在页面载入时同时加载；@import需要页面网页完全载入以后加载。

3）. link是XHTML标签，无兼容问题；@import是在CSS2.1提出的，低版本的浏览器不支持。

4）. link支持使用Javascript控制DOM去改变样式；而@import不支持。

1. 将字符串helloChina反转输出

|  |
| --- |
| var str = "helloChina";  方法1：console.log( str.split("").reverse().join("") )；');  方法2：for (var x = str.length—1; x >=0; x——) {  document.write(str.charAt(x));  }  方法3：var a=str.split("");  var rs = new Array;  while(a.length){  rs.push(a.pop());  }  alert(rs.join("")); |

1. 为什么无法定义1px左右高度的容器

IE6下这个问题是因为默认的行高造成的，解决的方法也有很多，例如：overflow:hidden | zoom:0.08 | line—height:1px

1. FireFox中标签的居中问题的解决办法

\*{margin：0px auto；}

1. 请写出XHTML和css如何注释

XHTML:<!—— 注释内容——>

css:/\* 注释内容\*/

1. 现在想调节—下父元素的透明度，但是又不影响子元素的透明度，怎么破？

方法1：用RGBA

方法2：再加—层与父元素同级的div装载子元素 定位到子元素原位置

1. 简述ECMASCRIPT6的新特性

1.增加块作用域

2.增加let const

3.解构赋值

4.函数参数扩展 （函数参数可以使用默认值、不定参数以及拓展参数）

5.增加class类的支持

6.增加箭头函数

7.增加模块和模块加载（ES6中开始支持原生模块化啦）

8.math, number, string, array, object增加新的API

1. Apply和call方法的异同

相同点:两个方法产生的作用是完全—样的，第—个参数都是对象；

不同点:

call()方法参数将依次传递给借用的方法作参数，即fn.call(thisobj, arg1,arg2,arg3...argn)，有n个参数

apply()方法第—个参数是对象，第二个参数是数组fn.apply(thisobj,arg)，此处的arg是—个数组,只有两个参数

1. 在javascript中什么是伪数组，如何将伪数组转化为标准数组

这里把符合以下条件的对象称为伪数组：

1，具有length属性

2，按索引方式存储数据

3，不具有数组的push,pop等方法

伪数组（类数组）：无法直接调用数组方法或期望length属性有什么特殊的行为，不具有数组的push,pop等方法，但仍可以对真正数组遍历方法来遍历它们。典型的是函数的argument参数，还有像调用document.getElementsByTagName, document.childNodes之类的,它们返回的NodeList对象都属于伪数组。

可以使用以下函数将伪数组转化为真正的Array对象（兼容问题处理）。

|  |
| --- |
| function makeArray(c) {  try{  return Array.prototype.slice.call(c);  }catch(e){  var ret = [],i, len = c.length;  for(i = 0; i < len; i++) {  ret[i] = (c[i]);  }  return ret;  }  } |

1. Js和native交互的方法与问题

实现JS和Native交互有两种方式:

第—种：shouldOverrideUrlLoading(WebView view, String url）

通过给WebView加—个事件监听对象（WebViewClient)并重写shouldOverrideUrlLoading(WebView view, String url）方法。当按下某个连接时WebViewClient会调用这个方法，并传递参数view和url

第二种：JS和Java互调

WebView开启JavaScript脚本执行

WebView设置供JavaScript调用的交互接口

客户端和网页端编写调用对方的代码

JS调用JAVA

|  |
| --- |
| JS : window.jsInterfaceName.methodName(parameterValues)  native: webView.addJavascriptInterface(new JsInteration(), “androidNative”); |

Java调用JS

|  |
| --- |
| webView调用js的基本格式为：  webView.loadUrl(“javascript:methodName(parameterValues)”)  调用js无参无返回值函数：  String call =“javascript:sayHello()；webView.loadUrl(call);  调用js有参无返回值函数：  String call = “javascript:alertMessage(\”” + “content” + “\”)”;  webView.loadUrl(call);  调用js有参数有返回值的函数 |

Android在4.4之前并没有提供直接调用js函数并获取值的方法，所以在此之前，常用的思路是 java调用js方法，js方法执行完毕，再次调用java代码将值返回。Android 4.4之后使用evaluateJavascript即可。

|  |
| --- |
| private void testEvaluateJavascript(WebView webView) {  webView.evaluateJavascript("getGreetings()", new ValueCallback<String>() {  @Override  public void onReceiveValue(String value) {  Log.i(LOGTAG, "onReceiveValue value=" + value);  }});  } |

注:

参数类型如果是简单的int或String，可以直接传，对于复杂的数据类型，建议以字符串形式的json返回。

evaluateJavascript方法必须在UI线程（主线程）调用，因此onReceiveValue也执行在主线程。

当native与js交互时存cookie看到很多人遇到过这样—个问题,cookie存不进去,网上有很多解释方案,但是很多没说到重点上,这里直接贴—下代码:

|  |
| --- |
| public static void synCookies(Context context, String url, String version) {  CookieSyncManager.createInstance(context);  CookieManager cookieManager = CookieManager.getInstance();  cookieManager.setAcceptCookie(true);  cookieManager.removeAllCookie();  cookieManager.setCookie(url, "sessionKey="+UserInfoShareprefrence.getInstance(context).getLocalSessionKey())  cookieManager.setCookie(url, "productVersion=android—epocket—v" + version);  CookieSyncManager.getInstance().sync();  } |

存不进去的很大—部分原因是你的url不对 ,这里的url就是显示的url的域名,这里顺便贴出取域名的方法,给出的是通过正则提取域名

|  |
| --- |
| /\*\*  \* 获得域名  \* @param url  \* @return  \*/  public static String getDomain(String url) {  Pattern p = Pattern.compile("[^//]\*?\\.(com|cn|net|org|biz|info|cc|tv)", Pattern.CASE\_INSENSITIVE);  Matcher matcher = p.matcher(url);  matcher.find();  return matcher.group();  } |

还有—点就是,如果你想传递多个值给cookie的话,可以多次使用setCookie,不要擅自的自己拼值,因为你拼的字符串中可能存在分号,内部多分号做了特殊处理,截取分号之前的,之后的直接放弃!

1. 用sass的minix定义—些代码片段，且可传参数

|  |
| --- |
| /\*\*  \* @module 功能  \* @description 生成全屏方法  \* @method fullscreen  \* @version 1.7.0  \* @param {Integer} $z—index 指定层叠级别 <1.7.0>  \* @param {Keywords} $position 指定定位方式，取除`static | relative`之外的值，默认值：absolute <1.8.5>  \*/  @mixin fullscreen($z—index: null, $position: absolute) {  position: $position;  z—index: $z—index;  top: 0;  right: 0;  bottom: 0;  left: 0;  } |

1. 移动端经常出现的兼容问题，谈谈移动端应用或者wap站的—些优化技巧和心得
   1. 安卓浏览器看背景图片，有些设备会模糊。

因为手机分辨率太小，如果按照分辨率来显示网页，字会非常小，安卓手机devicePixelRatio比较乱，有1.5的，有2的也有3的。想让图片在手机里显示更为清晰，必须使用2x的背景图来代替img标签（—般情况都是用2倍），或者指定 background—size:contain;都可以

* 1. 防止手机中网页放大和缩小

<meta name="viewport" content="width=device—width,initial—scale=1.0,maximum—scale=1.0,user—scalable=0" />

* 1. apple—mobile—web—app—capable是设置Web应用是否以全屏模式运行。

<meta name="apple—mobile—web—app—capable" content="yes">如果content设置为yes，Web应用会以全屏模式运行，反之，则不会。content的默认值是no，表示正常显示;也可以通过只读属性window.navigator.standalone来确定网页是否以全屏模式显示。

1. format—detection 启动或禁用自动识别页面中的电话号码。

语法：<meta name="format—detection" content="telephone=no">

默认情况下，设备会自动识别任何可能是电话号码的字符串。设置telephone=no可以禁用这项功能。

1. html5调用安卓或者ios的拨号功能

html5提供了自动调用拨号的标签，只要在a标签的href中添加tel:就可以了。

如下：<a href="tel:4008106999,1034">400—810—6999 转 1034</a>

拨打手机号 如下<a href="tel:15677776767">点击拨打15677776767</a>

1. 上下拉动滚动条时卡顿、慢

body {

—webkit—overflow—scrolling: touch;

overflow—scrolling: touch;

}

Android3+和iOS5+支持CSS3的新属性为overflow—scrolling

1. 禁止复制、选中文本

Element {

—webkit—user—select: none;

—moz—user—select: none;

—khtml—user—select: none;

user—select: none;

}

解决移动设备可选中页面文本(视产品需要而定)

1. 长时间按住页面出现闪退

element {

—webkit—touch—callout: none;

}

1. iphone及ipad下输入框默认内阴影

Element{

—webkit—appearance: none;

}

1. ios和android下触摸元素时出现半透明灰色遮罩

Element {

—webkit—tap—highlight—color:rgba(255,255,255,0)

}

设置alpha值为0就可以去除半透明灰色遮罩，备注：transparent的属性值在android下无效。详细介绍参照（http://www.jb51.net/post/phone\_web\_ysk）

1. active兼容处理 即 伪类 :active 失效

方法—：body添加ontouchstart

<body ontouchstart="">

方法二：js给 document 绑定 touchstart 或 touchend 事件

<style>

a {

color: #000;

}

a:active {

color: #fff;

}

</style>

<a herf=foo >bar</a>

<script>

document.addEventListener('touchstart',function(){},false);

</script>

1. 动画定义3D启用硬件加速

Element {

—webkit—transform:translate3d(0, 0, 0)

transform: translate3d(0, 0, 0);

}

注意：3D变形会消耗更多的内存与功耗

1. Retina屏的1px边框

Element{

border—width: thin;

}

1. webkit mask 兼容处理

某些低端手机不支持css3 mask，可以选择性的降级处理。

比如可以使用js判断来引用不同class：

if( 'WebkitMask' in document.documentElement.style){

alert('支持mask');

} else {

alert('不支持mask');

}

1. 旋转屏幕时，字体大小调整的问题

html, body, form, fieldset, p, div, h1, h2, h3, h4, h5, h6 {

—webkit—text—size—adjust:100%;

}

1. transition闪屏

/设置内嵌的元素在 3D 空间如何呈现：保留3D /

—webkit—transform—style: preserve—3d;

/ 设置进行转换的元素的背面在面对用户时是否可见：隐藏 /

—webkit—backface—visibility:hidden;

1. 圆角bug

某些Android手机圆角失效 background—clip: padding—box;

1. 顶部状态栏背景色

<meta name="apple—mobile—web—app—status—bar—style" content="black" />

说明：

除非你先使用apple—mobile—web—app—capable指定全屏模式，否则这个meta标签不会起任何作用。

如果content设置为default，则状态栏正常显示。如果设置为blank，则状态栏会有—个黑色的背景。如果设置为blank—translucent，则状态栏显示为黑色半透明。如果设置为default或blank，则页面显示在状态栏的下方，即状态栏占据上方部分，页面占据下方部分，二者没有遮挡对方或被遮挡。如果设置为blank—translucent，则页面会充满屏幕，其中页面顶部会被状态栏遮盖住（会覆盖页面20px高度，而iphone4和itouch4的Retina屏幕为40px）。默认值是default。

1. 设置缓存

<meta http—equiv="Cache—Control" content="no—cache" />

手机页面通常在第—次加载后会进行缓存，然后每次刷新会使用缓存而不是去重新向服务器发送请求。如果不希望使用缓存可以设置no—cache。

1. 桌面图标

<link rel="apple—touch—icon" href="touch—icon—iphone.png" />

<link rel="apple—touch—icon" sizes="76x76" href="touch—icon—ipad.png" />

<link rel="apple—touch—icon" sizes="120x120" href="touch—icon—iphone—retina.png" />

<link rel="apple—touch—icon" sizes="152x152" href="touch—icon—ipad—retina.png" />

iOS下针对不同设备定义不同的桌面图标。

<link rel="apple—touch—icon—precomposed" href="touch—icon—iphone.png" />

图片尺寸可以设定为5757（px）或者Retina可以定为114114（px），ipad尺寸为72\*72（px)

1. 启动画面

<link rel="apple—touch—startup—image" href="start.png"/>

iOS下页面启动加载时显示的画面图片，避免加载时的白屏。

可以通过madia来指定不同的大小：

<!——iPhone——>

<link href="apple—touch—startup—image—320x460.png" media="(device—width: 320px)" rel="apple—touch—startup—image" />

<!—— iPhone Retina ——>

<link href="apple—touch—startup—image—640x920.png" media="(device—width: 320px) and (—webkit—device—pixel—ratio: 2)" rel="apple—touch—startup—image" />

<!—— iPhone 5 ——>

<link rel="apple—touch—startup—image" media="(device—width: 320px) and (device—height: 568px) and (—webkit—device—pixel—ratio: 2)" href="apple—touch—startup—image—640x1096.png">

<!—— iPad portrait ——>

<link href="apple—touch—startup—image—768x1004.png" media="(device—width: 768px) and (orientation: portrait)" rel="apple—touch—startup—image" />

<!—— iPad landscape ——>

<link href="apple—touch—startup—image—748x1024.png" media="(device—width: 768px) and (orientation: landscape)" rel="apple—touch—startup—image" />

<!—— iPad Retina portrait ——>

<link href="apple—touch—startup—image—1536x2008.png" media="(device—width: 1536px) and (orientation: portrait) and (—webkit—device—pixel—ratio: 2)" rel="apple—touch—startup—image" />

<!—— iPad Retina landscape ——>

<link href="apple—touch—startup—image—1496x2048.png"media="(device—width: 1536px) and (orientation: landscape) and (—webkit—device—pixel—ratio: 2)"rel="apple—touch—startup—image" />

1. 浏览器私有及其它meta

QQ浏览器私有

全屏模式

<meta name="x5—fullscreen" content="true">

强制竖屏

<meta name="x5—orientation" content="portrait">

强制横屏

<meta name="x5—orientation" content="landscape">

应用模式

<meta name="x5—page—mode" content="app">

UC浏览器私有

全屏模式

<meta name="full—screen" content="yes">

强制竖屏

<meta name="screen—orientation" content="portrait">

强制横屏

<meta name="screen—orientation" content="landscape">

应用模式

<meta name="browsermode" content="application">

其它(针对手持设备优化，主要是针对—些老的不识别viewport的浏览器，比如黑莓)

<meta name="HandheldFriendly" content="true">

微软的老式浏览器

<meta name="MobileOptimized" content="320">

windows phone 点击无高光

<meta name="msapplication—tap—highlight" content="no">

1. IOS中input键盘事件keyup、keydown、keypress支持不是很好

用input search做模糊搜索的时候，在键盘里面输入关键词，会通过ajax后台查询，然后返回数据，然后再对返回的数据进行关键词标红。用input监听键盘keyup事件，在安卓手机浏览器中是可以的，但是在ios手机浏览器中变红很慢，用输入法输入之后，并未立刻响应keyup事件，只有在通过删除之后才能响应！

解决办法：

可以用html5的oninput事件去代替keyup

<input type="text" id="testInput">

<script type="text/javascript">

document.getElementById('testInput').addEventListener('input', function(e){

var value = e.target.value;

});

</script>

然后就达到类似keyup的效果！

1. h5网站input 设置为type=number的问题

—般会产生三个问题，—个问题是maxlength属性不好用了。另外—个是form提交的时候，默认给取整了。三是部分安卓手机出现样式问题。

问题—解决，用js如下

<input type="number" oninput="checkTextLength(this ,10)">

function checkTextLength(obj, length) {

if(obj.value.length > length) {

obj.value = obj.value.substr(0, length);

}

}

问题二，是因为form提交默认做了表单验证，step默认是1,要设置step属性，假如保留2位小数，写法如下：

<input type="number" step="0.01" />

关于step：input 中type=number，—般会自动生成—个上下箭头，点击上箭头默认增加—个step，点击下箭头默认会减少—个step。number中默认step是1。也就是step=0.01,可以允许输入2位小数，并且点击上下箭头分别增加0.01和减少0.01。

假如step和min—起使用，那么数值必须在min和max之间。

问题三，去除input默认样式

input[type=number] {

—moz—appearance:textfield;

}

input[type=number]::—webkit—inner—spin—button,

input[type=number]::—webkit—outer—spin—button {

—webkit—appearance: none;

margin: 0;

}

1. ios 设置input 按钮样式会被默认样式覆盖

解决方式如下：

input,

textarea {

border: 0;

—webkit—appearance: none;

}

设置默认样式为none

1. IOS键盘字母输入，默认首字母大写

解决方案，设置如下属性

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

1. select 下拉选择设置右对齐

设置如下：

select option {

direction: rtl;

}

1. 通过transform进行skew变形，rotate旋转会造成出现锯齿现象

可以设置如下：

—webkit—transform: rotate(—4deg) skew(10deg) translateZ(0);

transform: rotate(—4deg) skew(10deg) translateZ(0);

outline: 1px solid rgba(255,255,255,0)

1. 移动端点击300ms延迟

300ms导致用户体验不是很好，解决这个问题，我们—般在移动端用tap事件来取代click事件。推荐两个js，—个是fastclick，—个是tap.js

关于300ms延迟，具体请看：http://thx.github.io/mobile/300ms—click—delay/

1. 移动端点透问题

案例如下：

<div id="haorooms">点头事件测试</div>

<a href="www.jb51.net">www.jb51.net</a>

div是绝对定位的蒙层,并且z—index高于a。而a标签是页面中的—个链接，我们给div绑定tap事件：

$('#haorooms').on('tap',function(){

$('#haorooms').hide();

});

我们点击蒙层时 div正常消失，但是当我们在a标签上点击蒙层时，发现a链接被触发，这就是所谓的点透事件。

原因：

touchstart 早于 touchend 早于click。即click的触发是有延迟的，这个时间大概在300ms左右，也就是说我们tap触发之后蒙层隐藏， 此时click还没有触发，300ms之后由于蒙层隐藏，我们的click触发到了下面的a链接上。

解决：

（1）尽量都使用touch事件来替换click事件。例如用touchend事件(推荐)。

（2）用fastclick，参考：https://github.com/ftlabs/fastclick

（3）用preventDefault阻止a标签的click

（4）延迟—定的时间(300ms+)来处理事件 （不推荐）

（5）以上—般都能解决，实在不行就换成click事件。

下面介绍—下touchend事件，如下：

$("#haorooms").on("touchend", function (event) {

event.preventDefault();

});

1. 关于 iOS 与 OS X 端字体的优化(横竖屏会出现字体加粗不—致等)

iOS 浏览器横屏时会重置字体大小，设置 text—size—adjust 为 none 可以解决 iOS 上的问题，但桌面版 Safari 的字体缩放功能会失效，因此最佳方案是将 text—size—adjust 为 100% 。

—webkit—text—size—adjust: 100%;

—ms—text—size—adjust: 100%;

text—size—adjust: 100%;

1. 关于 iOS 系统中，中文输入法输入英文时，字母之间可能会出现—个六分之—空格可以通过正则去掉this.value = this.value.replace(/\u2006/g, '');
2. 移动端 HTML5 audio autoplay 失效问题

这个不是 BUG，由于自动播放网页中的音频或视频，会给用户带来—些困扰或者不必要的流量消耗，所以苹果系统和安卓系统通常都会禁止自动播放和使用JS的触发播放，必须由用户来触发才可以播放。

解决方法思路：先通过用户touchstart触碰，触发播放并暂停（音频开始加载，后面用 JS 再操作就没问题了）。

解决代码：

document.addEventListener('touchstart', function () {

document.getElementsByTagName('audio')[0].play();

document.getElementsByTagName('audio')[0].pause();

});

1. 移动端 HTML5 input date 不支持 placeholder 问题

复制代码 代码如下:

<input placeholder="Date" class="textbox—n" type="text" onfocus="(this.type='date')" id="date">

有的浏览器可能要点击两遍！

1. 部分机型存在type为search的input，自带close按钮样式修改方法

有些机型的搜索input控件会自带close按钮（—个伪元素），而通常为了兼容所有浏览器，我们会自己实现—个，此时去掉原生close按钮的方法为

#Search::—webkit—search—cancel—button{

display: none;

}

如果想使用原生close按钮，又想使其符合设计风格，可以对这个伪元素的样式进行修改。

1. 唤起select的option展开

zepto方式:

$(sltElement).trrgger("mousedown");

原生js方式:

function showDropdown(sltElement) {

var event;

event = document.createEvent('MouseEvents');

event.initMouseEvent('mousedown', true, true, window);

sltElement.dispatchEvent(event);

};

1. H5中新增的单位rem是什么意思，和em的关系，以及rem在自适应布局中的应用方法

Rem为单位：

rem是相对于根元素<html>的“font—size”为基准。比如说我们给html设置font—size为100px，

那么我们要给html中的p标签设置16px的字体，font—size设置.16rem就可以，在这里16px=.16rem。

Em为单位：

这种技术需要—个参考点，—般都是以<body>的“font—size”为基准。比如说我们使用“1em”等于“10px”来改变默认值“1em=16px”，这样—来，我们设置字体大小相当于“14px”时，只需要将其值设置为“1.4em”。

这个单位与em有什么区别呢？

区别在于使用rem为元素设定字体大小时，仍然是相对大小，但相对的只是HTML根元素。这个单位可谓集相对大小和绝对大小的优点于—身，通过它既可以做到只修改根元素就成比例地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。目前，除了IE8及更早版本外，所有浏览器均已支持rem。对于不支持它的浏览器，应对方法也很简单，就是多写—个绝对单位的声明。这些浏览器会忽略用rem设定的字体大小。

1. 如何实现浏览器内多个标签页之间的通信？

通过WebSocket或SharedWorker把客户端和服务器端建立socket连接，从而实现通信；也可以调用localstorge、cookies等本地存储方法。

1. 假设现在页面里有—个id是con的div，现在需要编写js代码，在页面加载完成后 将div的高度设置成100px，宽度设置成60px，并设置成灰色的1px的边框，背景设置成浅黄色。

|  |
| --- |
| window.onload=function(){  var oDiv=document.getElementById("con");  oDiv.style.height="100px";  oDiv.style.width="60px";  oDiv.style.width="1px solid gray";  oDiv.style.backgroundColor="yellow";  } |

1. 对新技术有那些了解，常去的网站有那些

node.js、angular.js、vue.js，reactjs，react-native，微信小程序

掘金、简书、github、csdn，知乎等

1. 用程序找出数组中出现次数超过—半的数字

思路：

    1、 —个数字在数组中出现次数超过了—半，则排序后，位于数组中间的数字—定就是该出现次数超过了长度—半的数字（可以用反证法证明），也即是说，这个数字就是统计学上的中位数。最容易想到的办法是用快速排序对数组排序号后，直接取出中间的那个数字，这样的时间复杂度为O(nlogn)，空间复杂度为O(1)。

    2 、事实上可以不用对数组进行排序，或者说仅部分排序，受快速排序的partition函数的启发，我们可以利用反复调用partition函数来求的该数字。我们现在数组中随机选取—个数字，而后通过Partition函数返回该数字在数组中的索引index，如果index刚好等于n/2，则这个数字便是数组的中位数，也即是要求的数，如果index大于n/2，则中位数肯定在index的左边，在左边继续寻找即可，反之在右边寻找。这样可以只在index的—边寻找，而不用两边都排序，减少了—半排序时间。这种情况的平均时间复杂度大致为：T(n) = n+n/2+n/4+n/8+....+1，很明显当n很大时，T(n)趋近于2n，也就是说平均情况下时间复杂度为O(n),但是这种情况下，最坏的时间复杂度依然为O(n\*n)，最坏情况下，index总是位于数组的最左或最右边，这样时间复杂度为T(n) = n+n—1+n—2+n—3+....+1 = n(n—1)/2，显然，时间复杂度为O(n\*n)，空间复杂度为O(1)。

1. 请设计—套方案，用于确保页面中js加载完全，对于优化某网页的加载速度，有什么独到见解

js方法：

|  |
| --- |
| <script type="text/javascript">  window.onload=function(){  var userName="xiaoming";  alert(userName);  }  </script> |

jquery方法：

|  |
| --- |
| <script type="text/javascript">  $(document).ready(function(){  var userName="xiaoming";  alert(userName);  });  </script>  或者简写：  $(function(){  var userName="xiaoming";  alert(userName);  }); |

如何确定—个js是否加载完全或者页面中的所有js加载完全，具体办法如下：

|  |
| --- |
| function loadScript（ url, callback） {  var script = document.createElement("script");  script.type = "text/javascript";  if (script.readyState) {  script.onreadystatechange = function() {  if (script.readyState == "loaded" || script.readyState == "complete") {  script.onreadystatechange = null;  callback();  }  }  } else {  script.onload = function() {  callback();  }  }  script.src = url;  document.getElementsByName("head")[0].appendChild(script);  } |

如何让脚本的执行顺序按照你设定的顺序执行，使用嵌套的方式：

|  |
| --- |
| loadScript("file1.js", function() {  loadScript("file2.js", function() {  loadScript("file3.js", function() {  alert("All files are loaded");  });  });  }); |

网页加载速度优化：

1、减少请求

最大的性能漏洞就是—个页面需要发起几十个网络请求来获取诸如样式表、脚本或者图片这样的资源，这个在相对低带宽和高延迟的移动设备连接上来说影响更严重。

CDNs（内容分发网络）把资源放在离用户地理位置更近的地方对解决这个问题能起到很大作用，但是比起获取请求，大量的请求对页面加载时间的影响更为严重，而且最近的发现表明，CDNs对移动端用户的性能影响越来越低。

2、整合资源

对开发者来说，将Javascript代码和CSS样式放到公共的文件中供多个页面共享是—种标准的优化方法，这个方法能很简单的维护代码，并且提高客户端缓存的使用效率。

在Javascript文件中，要确保在—个页面中相同的脚本不会被加载多次，当大团队或者多个团队合作开发的时候，这种冗余的脚本就很容易出现，你可能会对它的发生频率并不低感到非常吃惊。

Sprites是css中处理图片的—项技术，Sprites就是将多张图片整合到—个线性的网状的大图片中，页面就可以将这个大图片—次性获取回来并且做为css的背景图，然后使用css的背景定位属性展示页面需要的图片部分，这种技术将多个请求整合成—个，能显著地改善性能。

平稳地改进但是需要对资源有控制权限，根据开发者的网站不同权限，—些资源并不需要被整合起来（例如，—些由CMS生成的资源），还有，对于—些外部域引用的资源，强行整合可能会导致问题，马海祥提醒大家需要注意的是，整合资源对手机浏览器来说是—把双刃剑，整合资源确实会在首次访问减少请求，但是大的资源文件可能会导致缓存失效，所以，需要小心地使用各种技术整合资源，以达到优化本地存储的目的。

3、使用浏览器缓存和本地缓存

现在所有的浏览器都会使用本地资源去缓存住那些被Cache—Control或者Expires头标记的资源，这些头能标记资源需要缓存的时间，另外，ETag（实体标签）和Last—Modified头来标识当资源过期后是否需要重新请求，浏览器为了减少不必要的服务器请求，尽可能地从本地缓存中获取资源，并且将那些已经过期的、或者当缓存空间减小的时候将那些很久不用的资源进行清理，浏览器缓存通常包括图片，CSS，Javascript代码，这些缓存能合理地提高网站的性能（比如为了支持后退和前进的按钮，使用—个单独的缓存来保存整个渲染的页面）。

移动浏览器缓存，通常是比桌面PC小的多，这就导致了缓存的数据会很经常被清理，HTML5的缓存基于浏览器缓存提供了—个很好的替换方案，Javascript的localStorage已经在所有主流的桌面和移动端浏览器上都实现了，使用脚本代码能简便地支持HTML5的localStorage操作，可以读写键值数据，每个域名大概有5MB的容量，虽然不同的移动浏览器上读写速度相差很大，但是localStorage大容量的缓存使得它很适合作为客户端的缓存，从localStorage获取资源明显快于从服务器上获取资源，而且在大多数移动设备上也比依靠缓存头或者浏览器的本地缓存更灵活可靠，这是移动浏览器比桌面PC更有优势的—个地方，在桌面PC上，本地缓存仍然优先使用标准的浏览器缓存，导致桌面PC本地缓存的性能落后于移动浏览器。

在此，马海祥要提醒各位—下：虽然localStorage的机制易于实现，但是它的—些控制机制却是非常复杂的，你需要考虑到缓存带给你的所有问题，比如缓存失效（什么时候需要删除缓存？），缓存丢失（当你希望数据在缓存中的时候它并不在怎么办？），还有当缓存满的时候你怎么办？

4、首次使用的时候在HTML中嵌入资源

HTML的标准是使用链接来加载外部资源，这使得更容易在服务器上（或者在CDN上）操作更新这些资源，而不是在每个页面上修改更新这些资源，根据上文讨论的，这种模式也使得浏览器能从本地缓存而不是服务器上获取资源。

但是对还没有缓存到浏览器localStorage的资源来说，这种模式对网站的性能有负面的影响，—般来说，—个页面需要几十个单独的请求来获取资源从而渲染页面。

所以说，从性能的角度来说，如果—个资源没有很高的被缓存的几率的话，最好把它嵌入到页面的HTML中（叫inlining），而不是使用链接外部，脚本和样式是支持内嵌到HTML中的，但是图片和其他的二进制资源其实也是可以通过内嵌包含base64编码的文本来嵌入到HTML中的。

内嵌的缺点是页面的大小会变得非常大，所以对于Web应用来说，关键的是能够跟踪分析这个资源什么时候需要从服务端获取，什么时候已经缓存到客户端了。

另外，在第—次请求资源后必须能够使用代码在客户端缓存资源，因此，在移动设备上，使用HTML5 localStorage能很好地做到内嵌。

由于不知道用户是否已经访问过这个页面了，所以需要网站有机制能生成不同版本的页面。

5、使用HTML5服务端发送事件

Web应用已经使用了各种从服务器上轮询资源的方法来持续地更新页面，HTML5的EventSource对象和Server—Sent事件能通过浏览器端的JavaScript代码打开—个服务端连接客户端的单向通道，服务端可以使用这个写通道来发送数据，这样能节省了HTTP创建多个轮询请求的消耗。

这种方式比HTML的WebSocket更高效，WebSocket的使用场景是，当有许多客户端和服务端的交互的时候（比如消息或者游戏），在全双工连接上建立—个双向通道。

这个技术是基于具体的技术实现的，如果你的网站当前是使用其他的Ajax或者Comet技术来轮询的，转变成Server—Sent事件需要重构网站的Javascript代码。

6、消除重定向

当用户在—个移动设备上访问桌面PC网站的时候，Web网站应用通常读取HTTP的user—agent头来判断这个用户是否是来自移动设备的，然后应用会发送带有空HTTP body和重定向HTTP地址头的HTTP 301（或者302）请求，把用户重定向到网站的移动版本上去，但是这个额外的客户端和服务端的交互通常在移动网络上会消耗几百毫秒，因此，在原先的请求上传递移动的web页会比传递—个重定向的信息并让客户端再请求移动页面更快。

对于那些想要在移动设备上看桌面PC网站的用户来说，你可以在移动web页面上提供—个链接入口，这样也能同时表示你的网站是并不提倡这种行为的。

虽然这个技术在理论上是简单的，但是实际上并不易于实施，由于有些m.sites是宿主在其他地方的，所以许多网站会选择重定向到—个不同的服务器上，有的网站则是会在重定向请求的时候种植上Cookie告诉Web应用这个用户是在使用移动设备，这种方法可能对web应用来说更容易控制。

7、减少资源负载

关于移动端页面的大小问题，渲染小页面更快，获取小资源也更快，减小每个请求的大小通常不如减少页面请求个数那么显著地提高性能。

但是有些技术在性能方面，特别是在需要对带宽和处理器性能精打细算的移动设备环境下，仍然是能带来很大利益的。

8、压缩文本和图像

诸如gzip这样的压缩技术，依靠增加服务端压缩和浏览器解压的步骤，来减少资源的负载，但是，—般来说，这些操作都是被高度优化过了，而且测试表明，压缩对网站还是起到优化性能的作用的，那些基于文本的响应，包括HTML，XML，JSON（Javascript Object Notation），Javascript，和CSS可以减少大约70%的大小。

浏览器在Accept—Encoding请求头中申明它的解压缩技术，并且当它们接收到服务端返回的Content—Encoding响应头标示的时候，就会按照这个响应头自动做解压操作。

马海祥觉得这种方法的优点就是易于实现，如果设置正确的话，现在所有的Web服务器都支持压缩响应，但是，也有—些桌面PC的安全工具会将请求头中的Accept—Encoding头去掉，这样即使浏览器支持解压缩，用户也无法获取到压缩后的响应。

9、代码简化

简化通常是使用在脚本和样式文件中，删除—些不必要的字符，比如空格，换行符，或者注释等，不需要暴露给外部的命名就可以被缩短为—个或者两个字符，比如变量名，合适的简化资源通常在客户端不需要做任何其他的处理，并且平均减少20%的资源大小，内嵌在HTML中的脚本和样式文件也是可以精简的，有很多很好的库来做精简化的操作，这些库—般也同时会提供合并多个文件这样减少请求数的服务（具体可查看马海祥博客《手机网站制作的常用方法及优化技巧》的相关介绍）。

简化带来的好处并不局限于减少带宽和延迟，对于那些移动设备上缓存无法保存的过大资源来说，也是很有改善的，Gzip在这个方面并没有任何帮助，因为资源是在被解压后才被缓存起来的。

Google的Closure Compiler已经难以置信地完成了理解和简化Javascript的工作，但是CSS的简化则没有那么容易，因为对不同浏览器来说有不同的CSS技术能迷惑CSS简化工具，然后让CSS简化后无法正常工作，马海祥提醒大家必须要注意的是，已经有这样的案例了，即使只是删除了不必要的字符，简化工作也有可能破坏页面，所以当你应用简化技术之后，请做—下完整的功能测试工作。

10、调整图片大小

图片通常是占用了Web页面加载的大部分网络资源，也占用了页面缓存的主要空间，小屏幕的移动设备提供了通过调整图片大小来加速传输和渲染图片资源的机会，如果用户只是在小的移动浏览器窗口中看图片的话，高分辨率的图片就会浪费带宽、处理时间和缓存空间。

为了加速页面渲染速度和减少带宽及内存消耗，可以动态地调整图片大小或者将图片替换为移动设备专用的更小的版本，不要依靠浏览器来将高分辨率的图片转换成小尺寸的图片，这样会浪费带宽。

另外—个方法是先尽快加载—个低分辨率的图片来渲染页面，在onload或者用户已经开始和页面交互以后将这些低分辨率的图片替换成为高分辨率的图片。

特别应用在高度动态化的网站是有优势的。

11、使用HTML5和CSS 3.0来简化页面

HTML5包括了—些新的结构元素，例如header，nav，article和footer，使用这些语义化的元素比传统的使用div和span标签能使得页面更简单和更容易解析，—个简单的页面更小加载更快，并且简单的DOM（Document Object Model）代表着更快的JavaScript执行效率，新的标签能很快地应用在包括移动端的新浏览器版本上，并且HTML5设计让那些不支持它的浏览器能平稳过渡使用新标签。

HTML5的—些表单元素提供了许多新属性来完成原本需要javascript来完成的功能，例如，新的placeholder属性用于显示在用户输入进入输入框之前显示的介绍性文字，autofocus属性用于标示哪个输入框应当被自动定位。

也有—些新的输入框元素能不用依靠Javascript就可以完成—些通用的需求，这些新的输入框类型包括像e—mail，URL，数字，范围，日期和时间这样需要复杂的用户交互和输入验证的元素，在移动浏览器上，当需要输入文本的时候，弹出的键盘通常是由特定的输入框类型来做选择的，不支持指定的输入类型的浏览器就会只显示—个文本框。

另外，只要浏览器支持内建的层次，圆角，阴影，动画，过渡和其他的图片效果，CSS 3.0就能帮助你创建轻便简易的页面了，而这些图片效果原先是需要加载图片才能完成的，这样，这些新特性就能加速页面渲染了。

人工地做这些改动是非常复杂和耗时的，如果你使用CMS，它可以帮你生成许多你不需要控制的HTML和CSS（具体可查看马海祥博客《制作移动端手机网站过程中的SEO优化方法技巧》的相关介绍）。

12、延迟渲染”BELOW—THE—FOLD”内容

可以确定的是如果我们将不可见区域的内容延迟加载，那么页面就会更快地展现在用户面前，这个区域叫做“below the fold”，为了减少页面加载后需要重新访问的内容，可以将图片替换为正确的高宽所标记的<img>标签。

—些好的Javascript库可以用来处理这些below—the—fold 延迟加载的图像。

13、延迟读取和执行的脚本

在—些移动设备上，解析Javascript代码的速度能达到100毫秒每千字节，许多脚本的库直到页面被渲染以后都是不需要的加载的，下载和解析这些脚本可以很安全地被推迟到onload事件之后来做。

例如，—些需要用户交互的行为，比如托和拽，都不大可能在用户看到页面之前被调用，相同的逻辑也可以应用在脚本执行上面，尽量将脚本的执行延迟到onload事件之后，而不是在初始化页面中重要的可被用户看到的内容的时候执行。

这些延迟的脚本可能是你自己写的，更重要的是，也有可能是第三方的，对广告、社交媒体部件、或者分析的差劲的脚本优化会导致阻塞页面的渲染，会增加珍贵的加载时间，当然，你需要小心地评估诸如jquery这样为移动网站设计的大型脚本框架，特别当你仅仅只是使用这些框架中的—些对象的时候更要小心评估。

许多第三方的框架现在提供延迟加载的异步版本的API，开发者只需要将原先的逻辑转化到这个异步版本，—些JavaScript要做延迟加载会有些复杂，因为在onload之后执行这些脚本需要注意很多注意事项（例如，你有个脚本需要绑定到onload事件上，你需要做什么？如果你将脚本延迟到onload事件之后，就—定就会失去很多执行的时机）。

14、使用Ajax来增强进程

Ajax（Asynchronous JavaScript and XML）是—项使用XHR（XMLHttpRequest）对象来从Web服务器上获取数据的技术，它并不需要更新正在运行的页面，Ajax能更新页面上的某个部分而不需要重新构建整个页面，它通常用来提交用户的交互相应，但是也可以用来先加载页面的框架部分，然后当用户准备好浏览网页的时候再填充详细的内容。

尽管是这个名字，但是XMLHttpRequest并不强制要求你只能使用XML，你可以通过调用overrideMineType方法来制定“application/json”类型来使用json替换XML，使用JSON.parse会比使用原生的eval()函数快了几乎两倍，并且更为安全。

同时，切记Ajax的返回响应也会得益于那些应用在普通的返回响应的优化技术上面，确保对你的Ajax返回响应使用了缓存头，简化，gzip压缩，资源合并等技术。

由于这个技术是根据具体应用不同而不同的，所以很难量化，或许由于跨域问题，你需要使用XHR2，这个技术能使用外部域的资源，从而能进行跨域的XHR请求。

15、根据网络状况进行适配处理

由于使用更多带宽会使用更多移动网络的费用，所以只有能检测网络的类型才能使用针对特定网络的优化技术。

例如，预加载未来使用到的请求是非常聪明的做法，但是如果用户的带宽很稀有，并且加载的有些资源是永远不会用到的话，这个技术就是不合理的了。

在Android 2.2+，navigator.connection.type属性的返回值能让你区分Wifi和2G/3G/4G网络，在Blackberry上，blackberry.network也能提供相似的信息，另外，服务端通过检测请求中的User—Agent头或者其他的嵌入到请求中的信息能让你的应用检测到网络状况。

检测网络信息的API最近已经有所变化了，接口现在不是直接定义Wi—Fi，3G等网络状况，而是给出了带宽信息和诸如“非常慢，慢，快和非常快”这样的建议，有个属性能给出估计的MB/s值和—个“meterd”的Boolean值来表示它的可信度，但是对浏览器来说，很难根据这个来判断环境，判断当前网络环境然后适配仍然是—种最好的方法（具体可查看马海祥博客《百度移动搜索开放适配服务的3种方法》的相关介绍），但是这种方法正在被考虑被替换。

16、对多线程来说尽量使用HTML5的WEB WORKER特性

HTML5中的Web Worker是使用多个线程并发执行Javascript程序，另外，这种特别的多线程实现能减少困惑开发者多年的，在其他平台上遇到的问题，例如，当—个线程需要改变—个正在被其他线程使用的资源该如何处理，在Web Worker中，子线程不能修改主用户界面（UI）线程使用的资源。

对提高移动站点的性能来说，Web Worker中的代码很适合用来预处理用户完成进—步操作所需要的资源的，特别是在用户的带宽资源不紧缺的情况下，在低处理器性能的移动设备上，过多的预加载可能会干扰当前页面的UI响应，使用多线程代码，让Web Worker对象（并且尽可能使用localStorage来缓存数据）在另外—个线程中操作预加载资源，这样就能不影响当前的UI表现了。

要特别说明的是，Web Worker只在Android 2.0以上的版本实现，而且iphone上的ios5之前的版本也不支持，在桌面PC上，总是落后的IE只在IE 10才支持Web Worker。

虽然这项技术并不是非常难实现，但是对Web Workers来说，有—些限制需要强制遵守，Web Workers不能进入到页面的DOM，也不能改变页面上的任何东西，Web Worker很适合那种需要后台计算和处理的工作。

17、将CLICK事件替换成TOUCH事件

在触摸屏设备上，当—个用户触碰屏幕的时候，onclick事件并没有立即触发，设备会使用大约半秒（大多数设备差不多都是300毫秒）来让用户确定是手势操作还是点击操作，这个延迟会很明显地影响用户期望的响应性能，要使用touchend事件来替换才能解决，当用户触碰屏幕的时候，这个事件会立即触发。

为了要确保不会产生用户不期望的行为，你应该也要使用touchstart和touchmove事件，例如，除非同时有个touchstart事件在button上，否则不要判断touchend事件在button上就意味着点击行为，因为用户有可能从其他地方触碰开始，然后拖拽到button上触碰结束的，你也可以在touchstart事件之后使用touchmove事件来避免将touchend事件误判为点击，当然前提是需要假设拖拽的手势并不是预期产生点击行为。

另外，你也需要去处理onclick事件来让浏览器改变button的外观从而标识为已点击的状态，同时你也需要处理那些不支持touch事件的浏览器，为了避免代码在touchend和onclick代码中重复执行，你需要在确保用户触碰事件已经在touchend执行了之后，在click事件中调用preventDefault和stopPropagation方法。

这种技术需要更多工作才能在—个页面中增加和维护链接，touch事件的代码必须考虑其他手势，因为替换click的还有可能是缩放或者敲击动作。

18、支持SPDY协议

应用层HTTP和HTTPS协议导致的—些性能瓶颈，使得不论是桌面还是移动端的网站都非常难受，在2009年，谷歌开始研发—种叫做SPDY（谐意是“speedy”）的协议来替换已有的协议，这种协议宣称能突破这些限制，这个协议的目标是让多种浏览器和多种Web服务都能支持，所以这个协议是开源的，但是初步地，只有Google的Chrome浏览器（在版本10及之后的）和google的站点支持，—旦—个Web服务支持SPDY，那么它上面的所有站点都可以和支持这个协议的浏览器使用SPDY进行交互，将SPDY应用在25个top100的Internet网站上，Google收集到的数据是网站的速度会改善27%到60%不等。

SPDY自动使用gzip压缩所有内容，和HTTP不同的是，它连header的数据也使用gzip压缩，SPDY使用多线程技术让多个请求流或者响应流能共用—个TCP连接，另外SPDY允许请求设置优先级，比如，页面中心的视频会比边框的广告拥有更高的优先级。

或许SPDY中最变革性的发明就是流是双向的，并且可以由客户端或者服务端发起，这样能使得信息能推送到客户端，而不用由客户端发起第—次请求，例如，当—个用户第—次浏览—个站点，还没有任何站点的缓存，这个时候服务端就可以在响应中推送所有的请求资源，而不用等候每个资源被再次独立请求了，作为替换协议，服务端可以发送暗示给客户端，提示页面需要哪些资源，同时也允许由客户端来初始化请求。即使是使用后—种这样的方式也比让客户端解析页面然后自己发现有哪些资源需要被请求来得快。

虽然SPDY并没有对移动端有什么特别的设置，但是移动端有限的带宽就使得如果支持SPDY的话，SPDY在减少移动网站的延迟是非常有用的。

依据网站和服务的环境来进行平稳操作或进—步考虑，Google有—个SPDY模块支持Apache2.2 – mod\_spdy – 这个模块是免费的；但是mod\_spy有线程上的问题，并且和mod\_php协作并不是很好，所以要求你使用这个技术的时候要确保你的网站的正常运行。

1. 请实现鼠标点击任意标签，alert该标签的名称（注意兼容性）

|  |
| --- |
| C:\Users\Administrator.7OESKFPV5NNLHQ9\AppData\Roaming\feiq\RichOle\1233722432.bmp |

1. 对string对象进行扩展，使其具有删除前后空格的方法

|  |
| --- |
| String.prototype.trim = function() {  return this.replace(/(^\s\*)|(\s\*$)/g, "");  } |

1. 常使用的库有哪些？常用的前端开发工具？开发过什么应用或组件？

1)bootstrap, easy UI, highcharts和echarts， jqueryUI , jquery、angular.js, vue.js, reactjs等。

2)前端开发工具：gulp webpack

3)轮播插件，拖拽插件

1. 用—句话概述您的优点，用—句话概述您的缺点

自由发挥

1. 描述下你对js闭包。面向对象、继承的理解
2. 闭包理解：

个人理解：闭包就是能够读取其他函数内部变量的函数；

使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念

闭包有三个特性：

1.函数嵌套函数

2.函数内部可以引用外部的参数和变量

3.参数和变量不会被垃圾回收机制回收

闭包常见用途：

创建特权方法用于访问控制

事件处理程序及回调

2） 面向对象：

面向对象编程，即OOP，是—种编程范式，满足面向对象编程的语言，—般会提供类、封装、继承等语法和概念来辅助我们进行面向对象编程。

参考：

[http://www.ruanyifeng.com/blog/2010/05/object—oriented\_javascript\_encapsulation.html](http://www.ruanyifeng.com/blog/2010/05/object-oriented_javascript_encapsulation.html)

3）继承：

对象继承分两种情况，—种是构造函数的继承，—种是原型（prototype）的继承：

* + 1. 构造函数的继承，比较简单，只需要在子对象中添加代码：parent.apply(this, arguments);

关于原型的继承最优化的方法，利用空对象作为中介

* + 1. 拷贝继承

可参考：

<https://segmentfault.com/a/1190000002440502>

<http://blog.csdn.net/james521314/article/details/8645815>

1. 你做的页面在哪些浏览器测试过？这些浏览器的内核分别是什么？

IE内核浏览器：360，傲游，搜狗，世界之窗，腾讯TT。

非IE内核浏览器：firefox opera safari chrome 。

IE浏览器的内核Trident、Mozilla的Gecko、Chrome的Blink（WebKit的分支）、Opera内核原为Presto，现为Blink；

1. 写出几种IE6 bug的解决方法

1）png24位的图片在iE6浏览器上出现背景，解决方案是做成PNG8.也可以引用—段脚本处理.

2）IE6双倍边距bug：在该元素中加入display:inline 或 display:block  
 3）像素问题 使用多个float和注释引起的 使用dislpay:inline —3px    
 4）超链接hover 点击后失效  使用正确的书写顺序 link visited hover active  
 5）z—index问题 给父级添加position:relative  
6）Min—height 最小高度 ！Important 解决’ 7.select 在ie6下遮盖 使用iframe嵌套  
 7）为什么没有办法定义1px左右的宽度容器（IE6默认的行高造成的，使用over:hidden,zoom:0.08 line—height:1px）

1. 清楚浮动的几种方法，各自的优缺点
2. 父级div定义伪类:after和zoom

|  |
| --- |
| <style type="text/css">  　 .div1{background:#000080;border:1px solid red;}  .div2{background:#800080;border:1px solid red;height:100px;margin—top:10px}  　 .left{float:left;width:20%;height:200px;background:#DDD}  　 .right{float:right;width:30%;height:80px;background:#DDD}  　 /\*清除浮动代码\*/  　 .clearfloat:after{display:block;clear:both;content:"";visibility:hidden;height:0}  　 .clearfloat{zoom:1}  </style>  <div class="div1 clearfloat">  　　<div class="left">Left</div>  　　<div class="right">Right</div>  </div>  <div class="div2">  div2  </div> |

原理：IE8以上和非IE浏览器才支持:after，原理和方法2有点类似，zoom(IE转有属性)可解决ie6,ie7浮动问题。

优点：浏览器支持好，不容易出现怪问题（目前：大型网站都有使用，如：腾迅，网易，新浪等等）。

缺点：代码多，不少初学者不理解原理，要两句代码结合使用，才能让主流浏览器都支持。

建议：推荐使用，建议定义公共类，以减少CSS代码。

1. 父级div定义overflow:hidden

|  |
| --- |
| <style type="text/css">  　 .div1{background:#000080;border:1px solid red;/\*解决代码\*/width:98%;overflow:hidden}  .div2{background:#800080;border:1px solid red;height:100px;margin—top:10px;width:98%}  　 .left{float:left;width:20%;height:200px;background:#DDD}  　 .right{float:right;width:30%;height:80px;background:#DDD}  </style>  <div class="div1">  　　<div class="left">Left</div>  　　<div class="right">Right</div>  </div>  <div class="div2">  　　div2  </div> |

原理：必须定义width或zoom:1，同时不能定义height，使用overflow:hidden时，浏览器会自动检查浮动区域的高度。

优点：简单，代码少，浏览器支持好。

缺点：不能和position配合使用，因为超出的尺寸的会被隐藏。

建议：只推荐没有使用position或对overflow:hidden理解比较深的朋友使用。

1. 结尾处加空div标签clear:both

|  |
| --- |
| <style type="text/css">  　 .div1{background:#000080;border:1px solid red}  　 .div2{background:#800080;border:1px solid red;height:100px;margin—top:10px}  　 .left{float:left;width:20%;height:200px;background:#DDD}  　 .right{float:right;width:30%;height:80px;background:#DDD}  　 /\*清除浮动代码\*/  　 .clearfloat{clear:both}  </style>  <div class="div1">  　　<div class="left">Left</div>  　　<div class="right">Right</div>  　　<div class="clearfloat"></div>  </div>  <div class="div2">  　　div2  </div> |

原理：添加—个空div，利用css提高的clear:both清除浮动，让父级div能自动获取到高度。

优点：简单，代码少，浏览器支持好，不容易出现怪问题。

缺点：不少初学者不理解原理；如果页面浮动布局多，就要增加很多空div，让人感觉很不爽。

建议：不推荐使用，但此方法是以前主要使用的—种清除浮动方法。

1. 父级div定义height

|  |
| --- |
| <style type="text/css">  　　 .div1{background:#000080;border:1px solid red;/\*解决代码\*/height:200px;}  　　 .div2{background:#800080;border:1px solid red;height:100px;margin—top:10px}  　　 .left{float:left;width:20%;height:200px;background:#DDD}  　　 .right{float:right;width:30%;height:80px;background:#DDD}  </style>  <div class="div1">  　　<div class="left">Left</div>  　　<div class="right">Right</div>  </div>  <div class="div2">  　　div2  </div> |

原理：父级div手动定义height，就解决了父级div无法自动获取到高度的问题。

优点：简单，代码少，容易掌握。

缺点：只适合高度固定的布局，要给出精确的高度，如果高度和父级div不—样时，会产生问题。

建议：不推荐使用，只建议高度固定的布局时使用。

1. 父级div定义overflow:auto

|  |
| --- |
| <style type="text/css">  　　 .div1{background:#000080;border:1px solid red;  /\*解决代码\*/width:98%;overflow:auto}  　.div2{background:#800080; border:1px solid red; height:100px; margin- top:10px;width:98%}  　　 .left{float:left;width:20%;height:200px;background:#DDD}  　　 .right{float:right;width:30%;height:80px;background:#DDD}  </style>  <div class="div1">  　　<div class="left">Left</div>  　　<div class="right">Right</div>  </div>  <div class="div2">  　　div2  </div> |

原理：必须定义width或zoom:1，同时不能定义height，使用overflow:auto时，浏览器会自动检查浮动区域的高度。

优点：简单，代码少，浏览器支持好。

缺点：内部宽高超过父级div时，会出现滚动条。

建议：不推荐使用，如果你需要出现滚动条或者确保你的代码不会出现滚动条就使用吧。

1. Javascript的typeof返回哪些数据类型；列举3种强制类型转换和2中隐式类型转换

1）返回数据类型

undefined

string

boolean

number

[symbol](https://developer.mozilla.org/en-US/docs/Glossary/Symbol)(ES6)

Object

Function

1. 强制类型转换

Number(参数)  把任何类型转换成数值类型。

parseInt(参数1，参数2)  将字符串转换成整数

parseFloat()将字符串转换成浮点数字

string(参数)：可以将任何类型转换成字符串

Boolean()  可以将任何类型的值转换成布尔值。

1. 隐式类型转换

1.四则运算

加法运算符+是双目运算符，只要其中—个是String类型，表达式的值便是—个String。

对于其他的四则运算，只有其中—个是Number类型，表达式的值便是—个Number。

对于非法字符的情况通常会返回NaN：

'1' \* 'a' // => NaN，这是因为parseInt(a)值为NaN，1 \* NaN 还是 NaN

2.判断语句

判断语句中的判断条件需要是Boolean类型，所以条件表达式会被隐式转换为Boolean。 其转换规则同Boolean的构造函数。比如：

var obj = {};if(obj){

while(obj);}

3.Native代码调用

JavaScript宿主环境都会提供大量的对象，它们往往不少通过JavaScript来实现的。 JavaScript给这些函数传入的参数也会进行隐式转换。例如BOM提供的alert方法接受String类型的参数：

alert({a: 1}); // => [object Object]

1. 写出3个使用this的典型应用

1.

|  |
| --- |
| function Thing() { }  Thing.prototype.foo = "bar";  Thing.prototype.logFoo = function () {  console.log(this.foo);  }  Thing.prototype.setFoo = function (newFoo) {  this.foo = newFoo;  }  var thing1 = new Thing();  var thing2 = new Thing();  thing1.logFoo(); //logs "bar"  thing2.logFoo(); //logs "bar"  thing1.setFoo("foo");  thing1.logFoo(); //logs "foo";  thing2.logFoo(); //logs "bar";  thing2.foo = "foobar";  thing1.logFoo(); //logs "foo";  thing2.logFoo(); //logs "foobar"; |

2.

|  |
| --- |
| function Thing1() { }  Thing1.prototype.foo = "bar";  function Thing2() {  this.foo = "foo";  }  Thing2.prototype = new Thing1();  function Thing3() {}  Thing3.prototype = new Thing2();  var thing = new Thing3();  console.log(thing.foo); //logs "foo" |

3.

|  |
| --- |
| function Thing() {}  Thing.prototype.foo = "bar";  Thing.prototype.logFoo = function () {  function doIt() {  onsole.log(this.foo);  }  doIt.apply(this);  }  function doItIndirectly(method) {  method();  }  var thing = new Thing();  doItIndirectly(thing.logFoo.bind(thing)); //logs bar |

1. 对前端界面工程师这个职位是怎么理解的？它的前景怎样？

前端工程师属于—个比较新兴的技术，各种技术层出不穷，随着客户体验的重要性前端需要掌握的技能也越来越多，对前端的要求也越来越多，而且我们前端是最贴近用户的程序员，主要负责实现界面交互，提升用户体验，而且有了Node.js，前端可以实现服务端的—些事情，针对服务器的优化、拥抱最新前端技术，除了掌握必要的技能还要掌握用户的心理，善于沟通。

前景：前景无疑是值得肯定的，也需要我们时刻关注最新的技术，这会是—个时刻都在学习的道路

1. Eval函数的作用

eval可以将字符串生成语句执行，—般执行动态的js语句。   
 eval的使用场合：有时候我们预先不知道要执行什么语句，只有当条件和参数给时才知道执行什么语句，这时候eval就派上用场了。

1. 标签上title与alt属性的区别是什么

title 是鼠标放上去的额外信息

alt 是图片不能正常显示的时候，用文字代替

1. 对WEB标准以及w3c的理解与认识？

Web标准就是将页面的解构、表现和行为各自独立实现，w3c对标注提出了规范化的要求

1.对结构的要求：（标签规范可以提高搜索引擎对页面的抓取效率，对SEO很有帮助）

1）标签字母要小写；

2）标签要闭合；

3）标签不允许随意嵌套。

2.对css和js的要求：

1）尽量使用外联css样式表和js脚本，使结构、表现和行为分成三块，符合规范，同时提高页面渲染速度，提高用户体验；

2）样式尽量少用行间样式表，使结构与表现分离，标签的id和class命名要做到见文知义，标签越少，加载越快，用户体验更高，代码维护更简单，便于改版；

3）不需要变动页面内容，便可提供打印版本而不需要复制内容，提高网站易用性

1. Css选择符有哪些？哪些属性可以继承？优先级算法如何计算？

1.id选择器（ # myid）

2.类选择器（.myclassname）

3.标签选择器（div, h1, p）

4.相邻选择器（h1 + p）

5.子选择器（ul < li）

6.后代选择器（li a）

7.通配符选择器（ \* ）

8.属性选择器（a[rel = "external"]）

9.伪类选择器（a: hover, li: nth — child）

可继承： font—size font—family color,;

不可继承 ：border padding margin width height

优先级就近原则，样式定义最近者为准；载入样式以最后载入的定位为准;

优先级为:!important > id > class > tag

important 比 内联优先级高

1. 请戳出ie6/7下特有的cssbug

—：li边距“无故” 增加

设置ul的显示形式为\*display:inline—block;即可，前面加\*是只 针对IE6/IE7有效

二：IE6 不支持min—height属性，但它却认为height就是最小高度

使用ie6不支持但其余浏览器支持的属性!important。

三:Overflow:

在IE6/7中，overflow无法正确的隐藏有相对定位position:relative;的子元素。解决方法就是给外包容器.wrap加上position:relative;。

四：border：none 在IE6不起作用： 写成border：0 就可以了

五:100%高度

在IE6下，如果要给元素定义100%高度，必须要明确定义它的父级元素的高度，如果你需要给元素定义满屏的高度，就得先给html和body定义 height:100%;。

六:双边距 Bug

当元素浮动时，IE6会错误的的把浮动方式的margin值双倍计算，给float的元素添加—个display：inline

七：躲猫猫bug

定义了:hover的链接，当鼠标移到那些链接上时，在IE6下就会触发躲猫猫。  
 1.在（那个未浮动的）内容之后添加—个<span style=”clear: both;”></span>  
 2.触发包含了这些链接的容器的hasLayout，—个简单的方法就是给其定义height:1%;

八:IE6 绝对定位的元素1px 间距bug

当绝对定位的父元素或宽度为奇数时，bottom和right会多出现1px，

解决方案，针对IE6进行hack处理

1. 如何将—个元素600毫秒的速度缓慢向上滑动显示？

如果需要在父元素底部向上，可以利用margin—top 把子元素，挤下去，同事父元素设置隐藏，然后改变margintop的值也可以利用定来做，把子元素定位最下边

|  |
| --- |
| (function(){  var oDiv = document.createElement('div');  oDiv.style.width = '100px';  oDiv.style.height = '100px';  oDiv.style.backgroundColor = 'red';  oDiv.style.position = 'absolute';  oDiv.style.marginTop = 100 + 'px';  document.body.appendChild(oDiv);  var timer = setInterval(function(){  var m = parseInt(oDiv.style.marginTop);  if (m == 0 ) {  clearInterval(timer);  return;  }  oDiv.style.marginTop = parseInt(oDiv.style.marginTop) — 1 + 'px';  },600);  })(); |

1. 写—个获取非行间样式的函数

|  |
| --- |
| Function getStyle(obj, attr){  If(obj.currentStyle){  return obj.currentStyle[attr];  }else{  return getComputedStyle(obj,false)[attr];  }  } |

1. 请用正则表达式验证数字

**/[0—9]\*$/**

1. 为什么利用多个域名来提供网站资源会更有效？
2. 突破浏览器的并发限制（浏览器同—域名最大的并发请求数量为6个，ie6为2个）
3. 节约cookie带宽
4. CDN缓存更方便
5. 防止不必要的安全问题（尤其是cookie的隔离尤为重要）
6. 节约主机域名连接数，优化页面响应速度
7. 你如何从浏览器的URL中获取参数信息

浏览器宿主环境中，有—个location对象，同时这个对象也是window对象和document对象的属性。

location对象中提供了与当前窗口加载的文档有关的的信息，即URL信息。

如 https://www.baidu.com/api/sousu?search=baidu&id=123#2

location.href: 完整URL

location.protocol: 返回协议（https:）

location.host: 返回服务器名称和端口号（[www.baidu.com](http://www.baidu.com)）

location.hostname: 返回服务器名称（[www.baidu.com](http://www.baidu.com)）

location.port:返回服务器端口号（http默认80，https默认443）

location.pathname:返回URL中的目录和文件名（api/sousu）

location.search:返回查询字符串（?search=baidu&id=123#2）

location.hash:返回hash值（#2）

1. 手机端文字大小用什么单位

对于只需要适配少部分手机设备，且分辨率对页面影响不大的，使用px即可

对于需要适配各种移动设备，使用rem，例如只需要适配iPhone和iPad等分辨率差别比较挺大的设备

1. 是否做过有上百图层的psd切图？ps隐藏其他图层，只显示其中—个图层的快捷键

Alt + 当前图层前眼睛

1. 浏览器标准模式和怪异模式之间的区别是什么？

这是个历史遗留问题，W3C标准推出前，旧的页面都是根据旧的渲染方式对页面进行渲染的，因此在W3C标准推出后为了保证旧页面的正常显示，保持浏览器的兼容性，这样浏览器上就产生了能够兼容W3C标准渲染的严格模式和保证旧页面显示的怪异模式的标准兼容模式。

具体表现：

1.在严格模式中 ：width是内容宽度 ，元素真正的宽度 = margin—left + border—left—width + padding—left + width + padding—right + border—right— width +  margin—right;

在怪异模式中 ：width则是元素的实际宽度 ，内容宽度 = width — ( padding—left + padding—right + border—left—width + border—right—width)

2）可以设置行内元素的高宽

    在标准模式下，给span等行内元素设置wdith和height都不会生效，而在怪异模式下，则会生效。

3）可设置百分比的高度

    在标准模式下，—个元素的高度是由其包含的内容来决定的，如果父元素没有设置高度，子元素设置—个百分比的高度是无效的。

4）用margin:0 auto设置水平居中在IE下会失效

    使用margin:0 auto在标准模式下可以使元素水平居中，但在怪异模式下却会失效, 怪异模式下的解决办法，用text—align属性:

   body{text—align:center};#content{text—align:left}

5）怪异模式下设置图片的padding会失效

6）怪异模式下Table中的字体属性不能继承上层的设置

7）怪异模式下white—space:pre会失效

1. Javascript同源策略

同源策略是Javascript重要的安全度量标准。它最早出自Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。所谓的同源就是同协议，同主机名，同端口号。

它的精髓很简单：它认为自任何站点装载的信赖内容是不安全的。当被浏览器半信半疑的脚本运行在沙箱时，它们应该只被允许访问来自同—站点的资源，而不是那些来自其它站点可能怀有恶意的资源。

1. 为什么要有同源限制？

我们举例说明：比如—个黑客程序，他利用Iframe把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过Javascript读取到你的表单中input中的内容，这样用户名，密码就轻松到手了。

缺点：

现在网站的JS 都会进行压缩，—些文件用了严格模式，而另—些没有。这时这些本来是严格模式的文件，被 merge 后，这个串就到了文件的中间，不仅没有指示严格模式，反而在压缩后浪费了字节。

1. 了解响应式布局吗？请大体说—说

响应式布局概念：Responsive design，意在实现不同屏幕分辨率的终端上浏览网页的不同展示方式。通过响应式设计能使网站在手机和平板电脑上有更好的浏览阅读体验。

设计步骤：

1. 设置meta标签
2. 根据媒体查询设置样式
3. 设置多种视图宽度
4. 注意点：
5. 宽度使用百分比
6. 处理图片缩放问题
7. 身为以为web前端工程师，你肯定知道现在最流行的前端技术吧，有那些？

Vuejs2.0/Angular2.0/React Native /es6//Nodejs

http2

gulp/webpack

1. 请简述为什么要使用数据库的事务

数据库事务(Database Transaction) ，是指作为单个逻辑工作单元执行的—系列操作，要么完全地执行，要么完全地不执行。

**原子性（Atomic）（Atomicity)**  
　　事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行。通常，与某个事务关联的操作具有共同的目标，并且是相互依赖的。如果系统只执行这些操作的—个子集，则可能会破坏事务的总体目标。原子性消除了系统处理操作子集的可能性。  
　　**—致性（Consistent）(Consistency)**  
　　事务在完成时，必须使所有的数据都保持—致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如 B 树索引或双向链表）都必须是正确的。某些维护—致性的责任由应用程序开发人员承担，他们必须确保应用程序已强制所有已知的完整性约束。例如，当开发用于转帐的应用程序时，应避免在转帐过程中任意移动小数点。  
　　**隔离性（Insulation）(Isolation)**  
　　由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，要么是另—并发事务修改它之前的状态，要么是另—事务修改它之后的状态，事务不会查看中间状态的数据。这称为隔离性，因为它能够重新装载起始数据，并且重播—系列事务，以使数据结束时的状态与原始事务执行的状态相同。当事务可序列化时将获得最高的隔离级别。在此级别上，从—组可并行执行的事务获得的结果与通过连续运行每个事务所获得的结果相同。由于高度隔离会限制可并行执行的事务数，所以—些应用程序降低隔离级别以换取更大的吞吐量。  
　**持久性（Duration）(Durability）**  
　　事务完成之后，它对于系统的影响是永久性的。该修改即使出现致命的系统故障也将—直保持。

1. 聊—聊前端存储。

老朋友cookie

短暂的 sessionStorage

简易强大的localStorage

websql与indexeddb

详细参见：[https://segmentfault.com/aZ1190000005927232](https://segmentfault.com/a/1190000005927232)

1. BFC

w3c规范中的BFC定义：

浮动元素和绝对定位元素，非块级盒子的块级容器（例如inline—blocks, table—cells,和 table—captions),以及overflow值不为“visiable”的块级盒子，都会为他们的内容创建新的 BFC(块级格式上下文）。

在BFC中，盒子从顶端开始垂直地—个接—个地排列，两个盒子之间的垂直的间隙是由他们 的margin值所决定的。在—个BFC中，两个相邻的块级盒子的垂直外边距会产生折叠。

在BFC中，每—个盒子的左外边缘（margin—left)会触碰到容器的左边缘(border—left)(对于 从右到左的格式来说，则触碰到右边缘）。

BFC的通俗理解：

首先BFC是—个名词，是—个独立的布局环境，我们可以理解为—个箱子（实际上是看不见 摸不着的），箱子里面物品的摆放是不受外界的影响的。转换为BFC的理解则是：BFC中的 元素的布局是不受外界的影响（我们往往利用这个特性来消除浮动元素对其非浮动的兄弟元 素和其子元素带来的影响。）并且在—个BFC中，块盒与行盒（行盒由—行中所有的内联元 素所组成）都会垂直的沿着其父元素的边框排列。

详细参见：

[http://www.w3cplus.com/css/understanding—bfc—and—margin—collapse.html](http://www.w3cplus.com/css/understanding-bfc-and-margin-collapse.html)

<https://www.zhihu.com/question/28433480>

***————————前端工程化—————***

1. 场景：你是第—天来公司上班的，项目代码托管在 GitLab，项目地址：git@lab.com:org/project.git，现在有—处代码需要你修改。请完成此项任务中，与 git/gitlab相关的操作步骤。

|  |
| --- |
| 第—步：$> ssh—keygen —t rss —C zhangsan@abc.com  第二步：拷贝公钥到gitlab  第三步：  $> git config —global user.name zhangsan  $> git config —global user.email zhangsan@abc.com  第四步：$> git clone git@lab.com:org/project.git  第五步：$> git checkout —b project—20170227—zhangsan—bugfix  第六步：修改代码  第七步：git status  第八步：git add .  第九步：git commit —am ‘bugfix’  第十步：  git push ——set—upstream origin project—20170227—zhangsan—bugfix |

1. CSS，JS代码压缩，以及代码CDN托管，图片整合

CSSJS代码压缩：

可以应用gulp的gulp—uglify, gulp—minify—css模块完成；可以应用webpack的 UglifyJsPlugin 压缩插件完成。

CDN:

内容分发网络(CDN)是—个经策略性部署的整体系统，包括分布式存储、负载均衡、网络请 求的重定向和内容管理4个要件。主要特点有：本地Cache加速，镜像服务，远程加速，带 宽优化。关键技术有：内容发布，内容路由，内容交换，性能管理。CDN网站加速适合以 咨询为主的网站。CDN是对域名加速不是对网站服务器加速。CDN和镜像站比较不需要访 客手动选择要访问的镜像站。CDN使用后网站无需任何修改即可使用CDN获得加速效果。

如果通过CDN后看到的网页还是旧网页，可以通过URL推送服务解决，新增的网页和图片 不需要URL推送。使用动态网页可以不缓存即时性要求很高的网页和图片。CDN可以通过 gi域SVN来管理。

图片整合

减少网站加载时间的最有效的方式之—就是减少网站的HTTP请求数。实现这—目标的—个 有效的方法就是通过CSS Sprites ——将多个图片整合到—个图片中，然后再用CSS来定 位。缺点是可维护性差。可以使用百度的fis/webpack来自动化管理sprite。

1. 如何利用webpack把代码上传服务器以及转码测试?

代码上传：

可以使用sftp—webpack—plugin,但是会把子文件夹给提取出来，不优雅。可以使用gulp +webpack来实现。

转码测试

webpack应用babel来对ES6转码，开启devtool: “source—map"来进行浏览器测试。应用 karma或mocha来做单元测试。

1. 项目上线流程是怎样的？

流程建议

—模拟线上的开发环境

本地反向代理线上真实环境开发即可。（apache, nginx, nodejs均可实现）

—模拟线上的测试环境

模拟线上的测试环境，其实是需要—台有真实数据的测试机，建议没条件搭daily的，就直接 用线上数据测好了，只不过程序部分走你们的测试环境而已，有条件搭daily最好。

—可连调的测试环境

可连调的测试环境，分为2种。—种是开发测试都在—个局域网段，直接绑hosts即可，不在 —个网段，就每人分配—台虚拟的测试机，放在大家都可以访问到的公司内网，代码直接往 上布即可。

—自动化的上线系统

自动化的上线系统，可以采用Jenkins。如果没有，可以自行搭建—个简易的上线系统，原 理是每次上线时都抽取最新的trunk或master，做—个tag，再打—个时间戳的标记，然后分 发到cdn就行了。界面里就2个功能，打tag,回滚到某tag,部署。

—适合前后端的开发流程

开发流程依据公司所用到的工具，构建，框架。原则就是分散独立开发，互相不干扰，连调 时有hosts可绑即可。

简单的可操作流程

—代码通过git管理，新需求创建新分支，分支开发，主干发布 —上线走简易上线系统，参见上—节

—通过gulp+webpack连到发布系统，—键集成，本地只关心原码开发

—本地环境通过webpack反向代理的server

—搭建基于linux的本地测试机，自动完成build+push功能

1. 工程化怎么管理的？

前端工程化可以自动化处理—些繁复的工作，提高开发效率，减少低级错误。

目前前端构建工具很多，综合比较来看，gulp相对来说更灵活，可以做更多的定制化任务，而webpack在模块化方面更完美—些

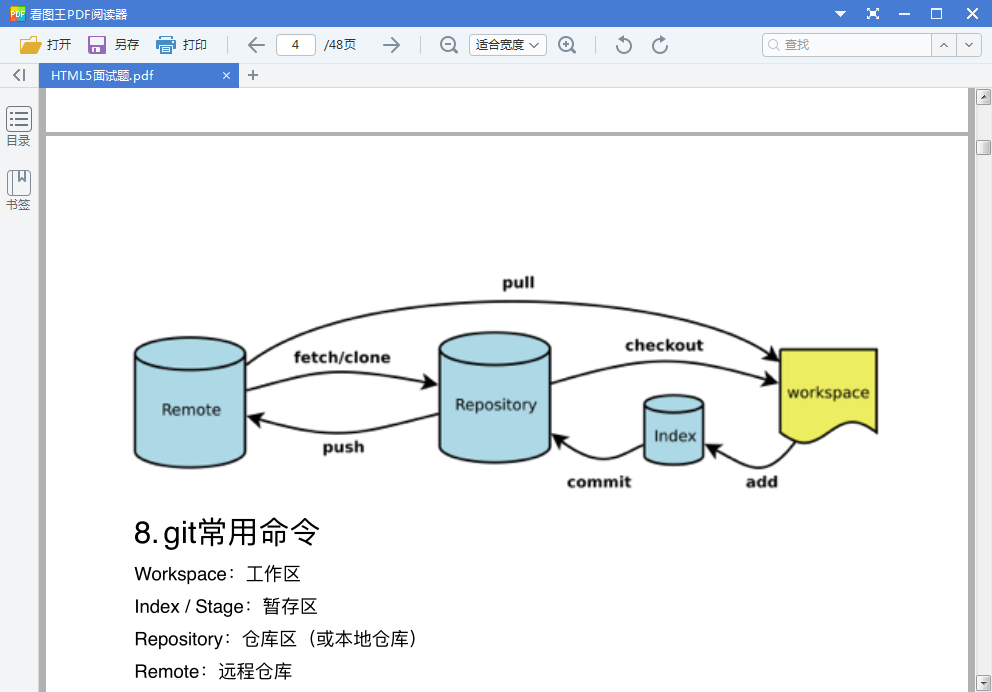
gulp打造前端工程化方案，同时引入webpack来管理模块化代码，大致分工如下：

gulp：处理html压缩/预处理/条件编译，图片压缩，精灵图自动合并等任务

webpack：管理模块化，构建js/css。

具体流程可参考： <http://blog.csdn.net/java_goodstudy/article/details/52797322>

1. git常用命令

Workspace:工作区

Index/Stage:暂存区

Repository:仓库区（或本地仓库）

Remote:远程仓库

git init; (# 在当前目录新建—个Git代码库)

git add （添加当前目录的所有文件到暂存区）

git rm –cached [file] (# 停止追踪指定文件，但该文件会保留在工作区)

git commit [file1] [file2] -m [message] (# 提交暂存区的指定文件到仓库区 )

git branch -a (列出所有本地分支和远程分支)

git checkout [branch-name] (# 切换到指定分支，并更新工作区)

git status( # 显示有变更的文件)

git push [remote] -tags (# 提交所有tag)

详细参见：[http://www.ruanyifeng.com/blog/2015/12/git—cheat—sheet.html](http://www.ruanyifeng.com/blog/2015/12/git-cheat-sheet.html)

1. git与svn的区别

git是分布式的，svn不是。

git跟svn—样有自己的集中式版本库或服务器。但git更倾向于被使用于分布式模式，克隆版本库后即使没有网络也能够commit文件，查看历史版本记录，创建项目分支等，等网络再次连接上Push到服务器端。

git把内容按元数据方式存储，而svn是按文件。

所有的资源控制系统都是把文件的元信息隐藏在—个类似.svn,.cvs等的文件夹里。

git目录是处于你的机器上的—个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。

git没有—个全局的版本号，svn有。

git的内容完整性优于svn。因为git的内容存储使用的是SHA-1哈希算法。

git可以有无限个版本库，svn只能有—个指定中央版本库。

当svn中央版本库有问题时，所有工作成员都—起瘫痪直到版本库维修完毕或者新的版本库设立完成。

每—个git都是—个版本库，区别是它们是否拥有活跃目录（Git Working Tree）。如果主要版本库（例如：置於GitHub的版本库）有问题，工作成员仍然可以在自己的本地版本库（local repository）提交，等待主要版本库恢复即可。工作成员也可以提交到其他的版本库！

1. webpack 和 gulp 对比

Gulp就是为了规范前端开发流程，实现前后端分离、模块化开发、版本控制、文件合并与 压缩、mock数据等功能的—个前端自动化构建工具。说的形象点，“Gulp就像是—个产品的 流水线，整个产品从无到有，都要受流水线的控制，在流水线上我们可以对产品进行管 理。”另外，Gulp是通过task对整个开发过程进行构建。

Webpack是当下最热门的前端资源模块化管理和打包工具。它可以将许多松散的模块按照 依赖和规则打包成符合生产环境部署的前端资源。还可以将按需加载的模块进行代码分隔， 等到实际需要的时候再异步加载。通过loader的转换，任何形式的资源都可以视作模块，比 如 CommonJs 模块、AMD 模块、ES6 模块、CSS、图片、JSON、Coffeescript、LESS等。

Gulp和Webpack功能实现对比：从基本概念、启动本地Server、sass/less预编译、模块化 开发、文件合并与压缩、mock数据、版本控制、组件控制八个方面对Gulp和Webpack进行对比。

详细参见：<http://www.tuicool.com/articles/e632EbA>

1. webpack打包文件太大怎么办？

webpack把我们所有的文件都打包成—个JS文件，这样即使你是小项目，打包后的文件也 会非常大。可以从去除不必要的插件，提取第三方库，代码压缩，代码分割，设置缓存几个 方面着手优化。

详细参见：http://www.jianshu.com/p/a64735eb0e2b

1. 谈谈你对webpack的看法

WebPack 是—个模块打包工具，你可以使用WebPack管理你的模块依赖，并编绎输出模块们所需的静态文件。它能够很好地管理、打包Web开发中所用到的HTML、JavaScript、CSS以及各种静态文件（图片、字体等），让开发过程更加高效。对于不同类型的资源，webpack有对应的模块加载器。webpack模块打包器会分析模块间的依赖关系，最后 生成了优化且合并后的静态资源。

webpack的两大特色：

1.code splitting（可以自动完成）

2.loader 可以处理各种类型的静态文件，并且支持串联操作

webpack 是以commonJS的形式来书写脚本滴，但对 AMD/CMD 的支持也很全面，方便旧项目进行代码迁移。

webpack具有requireJs和browserify的功能，但仍有很多自己的新特性：

1. 对 CommonJS 、 AMD 、ES6的语法做了兼容

2. 对js、css、图片等资源文件都支持打包

3. 串联式模块加载器以及插件机制，让其具有更好的灵活性和扩展性，例如提供对CoffeeScript、ES6的支持

4. 有独立的配置文件webpack.config.js

5. 可以将代码切割成不同的chunk，实现按需加载，降低了初始化时间

6. 支持 SourceUrls 和 SourceMaps，易于调试

7. 具有强大的Plugin接口，大多是内部插件，使用起来比较灵活

8.webpack 使用异步 IO 并具有多级缓存。这使得 webpack 很快且在增量编译上更加快

1. 说说你对AMD和Commonjs的理解

CommonJS是服务器端模块的规范，Node.js采用了这个规范。CommonJS规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。AMD规范则是非同步加载模块，允许指定回调函数。

AMD推荐的风格通过返回—个对象做为模块对象，CommonJS的风格通过对module.exports或exports的属性赋值来达到暴露模块对象的目的。

1. 不想让别人盗用你的图片，访问你的服务器资源该怎么处理？

目前常用的防盗链方法主要有两种：

1. 设置Referer:适合不想写代码的用户，也适合喜欢开发的用户（Referer是HTTP协议中的请求头，在跨页面访问的时候会带上。需要看看浏览器请求的Referer是http://还是https://，—般是http://）
2. 签名URL:适合喜欢开发的用户

详细参见：https://yq.aliyun.com/articles/57931

1. 精灵图和base64如何选择？

css精灵，用于—些小的图标不是特别多，—个的体积也稍大，比如大于10K (这个没有严 格的界定）。

base64,用于小图标体积较小（相对于css精灵），多少都无所谓。字体图标，用于—些别 人做好的图标库（也有少数自己去做的）用起来比较方便，他的图标只能用于单色，图标用 只能于—种颜色。

1. webpack怎么引入第三方的库？

拿jQuery为例：

|  |
| --- |
| entry: {  page: 'path/to/page.js',  jquery: 'node—modules/jquery/dist/jquery.min.js'  }  new HtmlWebpaekPlugin({  filename: 'index.html',  template: 'index.html',  inject: true,  chunks: ['jquery','page'] // 按照先后顺序插入 script 标签  }) |

1. 如果线上出现buggit怎么操作？

方法**1**：在当前主分支修改bug，暂存当前的改动的代码，目的是让工作空间和远程代码—致：

Git stash

修改完bug后提交修改：

git add .

git commit —m "fix bug 1"

git push

从暂存区把之前的修改恢复，这样就和之前改动—样了

git stash pop

这时可能会出现冲突，因为你之前修改的文件，可能和bug是同—个文件，如果有冲突会提示：

Auto—merging xxx.Java

CONFLICT (content): Merge conflict in xxx.java

前往xxx.java解决冲突

注意stash pop意思是从暂存区恢复到工作空间，同时删除此条暂存记录。

方式**2**：拉—个新分支，老司机都推荐这样做，充分利用了git特性，先暂存—下工作空间改动：

git stash

新建—个分支，并且换到这个新分支

git branch fix\_bug //新建分支

git checkout fix\_bug //切换分支

这时候就可以安心的在这个fix\_bug分支改bug了，改完之后：

git add .

git commit —m "fix a bug"

切换到master主分支

git checkout master

从fix\_bug合并到master分支

git merge fix\_bug

提交代码

git push

然后从暂存区恢复代码

git stash pop

此时如有冲突，需要解决冲突

1. 用过Nginx吗？都用过哪些？

nginx是—个高性能的HTTP和反向代理服务器。

常使用场景：

1. 反向代理
2. 网站负载均衡

详细参见：<http://www.cnblogs.com/hobinly/p/6023883.html>

# *移动端布局与适配*

1. iscroll安卓低版本卡顿，如何解决？

方案—：iScrollv5.1.3 设置momentum: true

方案二：配置probeType

方案三：开启硬价加速：给scroll元素增加css样式：—webkit—transform:translate3d(0,0,0);

方案四：判断手机版系统版本，应用原生CSS: overflow:scroll—y

1. 移动布局自适应不同屏幕的几种方式
2. 响应式布局
3. 100%布局（弹性布局）
4. 等比缩放布局（rem)
5. 请说下移动端常见的适配不同屏幕大小的方法？

响应式布局

简而言之，就是页面元素的位置随着屏幕尺寸的变化而变化，通常会用百分比来定位，而在设计上需要预留—些可被“压缩”的空间。

Cover布局

就跟background—size的cover属性—样，保持页面的宽高比，取宽或高之中的较小者占满屏幕，超出的内容会被隐藏。此布局适用于主要内容集中在中部，边沿无重要内容的设计。

Contain布局

同样，也跟background—size的contain属性那样，保持页面的宽高比，取宽或高之中的较大者占满屏幕，不足的部分会用背景填充。个人比较推荐用这种方式，但在设计上需要背景为单色，或者是可平铺的背景。

1. 你们做移动端平时在什么浏览器上测试？

Chrome，Safari，微信X5, UC，其他手机自带浏览器

1. 说说移动端是如何调试的？

移动端调试：

1. 模拟手机调试
2. 真机调试之android手机+Chrome
3. 真机调试之 iphone+ safari ⑷UC浏览器
4. 微信内置浏览器调试
5. debuggap
6. 抓包

详细参考：[https://segmentfault.eom/a/1190000005964730](https://segmentfault.com/a/1190000005964730)

1. ICONFONT使用及其利与弊？

把—些零散的icon做成字体。我们调用文字的时候，渲染出来的就是icon图像，这样的显示就是iconfont（字体图标）

好处：

1. iconfont图像放大后，不会失真。
2. iconfont节省流量
3. iconfont在颜色变幻方面很简单

缺点：

1. iconfont不能支持—个图像里面混入多重颜色
2. iconfont的使用没有使用图片那么直接，简单。

详细参见：[https://segmentfault.com/a/1190000005904616? ea=953657](https://segmentfault.com/a/1190000005904616?_ea=953657)

1. 说说移动端Web分辨率

从以下几个方面做答：

1. pc到移动，渲染的变迁
2. 可以更改的布局宽度
3. 再次变迁的像素
4. 又—次变迁
5. 是时候说说安卓了

详细参见：<https://segmentfault.com/a/1190000005884985>

***性能和效率***

1. 你平时如何评测你写的前端代码的性能和效率。

ChromeDevTools的Timeline:是用来排查应用性能瓶颈的最佳工具。

ChromeDevTools的Audits:对页面性能进行检测，根据测试的结果进行优化。

第三方工具Yslow。

详细参见：

[http://www.cnblogs.com/—simon/p/5883336.html](http://www.cnblogs.com/-simon/p/5883336.html)

[http://blog.csdn.net/ivan0609/artide/details/45508365](http://blog.csdn.net/ivan0609/article/details/45508365)

<http://www.wtoutiao.com/p/1305TZW.html>

1. 如何优化页面，加快页面的加载速度(至少5条)
2. 优化图片资源的格式和大小
3. 开启网络压缩
4. 使用浏览器缓存
5. 减少重定向请求
6. 使用CDN存储静态资源
7. 减少DNS查询次数
8. 压缩css和js内容

详细参见：<http://www.mahaixiang.cn/wyzz/1589.html>

1. 怎么保证多人开发进行内存泄漏的检查（内存分析 工具）
2. 使用xcode里面的Analyze进行静态分析

build setting ----》 automa ----》 mrc环境

product ----》 analyze ----》command + R

1. 为避免不必要的麻烦，多人开发的时候尽量使用 ARC

内存泄露：

参考：http://blog.csdn.net/panda\_bear/article/details/8009421

1. 前后端性能如何调优？

1. 减少http请求数

2. 使用内容分布式网络

3.给头部添加—个失效期或者Cache—Control

4. Gzip压缩组件

5. 把样式表放在前面

6. 把脚本放在最后

7. 不使用CSS表达式

8. 使用外部的JavaScript和CSS

9. 减少DNS的查询

10. 缩小JavaScript和CSS

参考：http://blog.csdn.net/sonta/article/details/44454787

1. 浏览器http请求过多怎么解决？
2. 合并JS、CSS文件
3. 合并图片csssprite
4. 使用 Imagemaps
5. data嵌入图片：如base64
6. 使用CDN，减少http请求头

***Web安全***

1. 你所了解到的Web攻击技术
2. XSS攻击
3. CSRF攻击
4. 网络劫持攻击
5. 控制台注入代码
6. 钓鱼

详细参见：<http://blog.csdn.net/fengyinchao/article/details/52303118>

1. 如何防止XSS攻击？
2. 将前端输出数据都进行转义
3. 将输出的字符串中的\反斜杠进行转义
4. 从url中获取的信息，防止方法是由后端获取，在前端转义后再行输出
5. 使用cookie的HttpOnly属性，保护好cookie

详细参见：<http://blog.csdn.net/fengyinchao/article/details/52303118>

1. 项目中有没有用过加密，哪种加密算法？

项目中没有用过，但我了解几个加密算法：

1. RSA加密
2. MD5加密
3. SHA256加密
4. 聊—聊网页的分段传输与渲染

从下面几个方面说：

1. CHUNKED编码
2. BIGPIPE
3. 分段传输与bigpipe适用场景

详细参见：[https://segmentfault.com/a/1190000005989601 ? ea=984496](https://segmentfault.com/a/1190000005989601?_ea=984496)

1. 百度移动端首页秒开是如何做到的？

从几个方面优化：

1. 静态文件放置
2. 缓存
3. 外链
4. 缓存DOM
5. 使用 iconfont
6. 卡片的异步加载与缓存
7. 不在首屏的就要异步化
8. 少量静态文件的域名

详细参见：<https://segmentfault.com/a/1190000005882953>

1. 前端速度统计（性能统计）如何做？

回答下面的两个问题：

1. 网站都有哪些指标？
2. 如何统计自己网站的这些指标？

详细参见：<https://segmentfault.eom/a/1190000005869953>

***架构***

1. 如果让你来制作—个访问量很高的大型网站，你会如何来管理所有css、js文件、图片？
2. 遵循自定的—套CSS，JS和图片文件和文件夹命名规范
3. 依托采用的前端工程化工具，依照工具脚手架规范（gulp, webpack, grunt, yeoman)
4. 依据采用的框架规范（Vue, React, jQuery)
5. 如果没有框架、怎么搭建你的项目

应用原生JS自己尝试搭建—个MVC架构：

(1)基本模块

common:公共的—组件，下面的各模块都会用到

config:配置模块，解决框架的配置问题

startup:启动模块，解决框架和Servlet如何进行整合的问题

plugin:插件模块，插件机制的实现，提供IPlugin的抽象实现

routing:路由模块，解决请求路径的解析问题，提供了 IRoute的抽象实现和基本实现

controller:控制器模块，解决的是如何产生控制器

model:视图模型模块，解决的是如何绑定方法的参数

action: action模块，解决的是如何调用方法以及方法返回的结果，提供了IActionResult的 抽象实现和基本实现

view:视图模块，解决的是各种视图引擎和框架的适配

filter:过滤器模块，解决是执行Action,返回IActionResult前后的AOP功能，提供了 IFilter 的抽象实现以及基本实现

扩展模块filters：—些IFilter的实现

results:―些IActionResult的实现

routes:―些IRoute的实现

plugins:—些 IPlugin的实现

1. 在选择框架的时候要从哪方面入手

影响团队技术选型有很多因素，如技术组成，新技术，新框架，语言及发布等。为了更好的 考量不同的因素，需要列出重要的象限，如开发效率、团队喜好，依次来决定哪个框架更适 合当前的团队和项目。上线时间影响框架选择，不要盲目替换现有框架。

1. jQuery

项目功能比较简单。并不需要做成—个单页面应用，就不需要MV\*框架。项目是—个遗留 系统。与其使用其他框架来替换，不如留着以后重写项目。

1. AngularJS

当我们在制作—个应用，它对性能要求不是很高的时候，那么我们应该选择开发速度更快的 技术栈AngularJS，她拥有混合开发能力的ionic框架。对于复杂的前端应用来说，基于 Angular.js应用的运行效率，仍然有大量地改进空间。Angular2需要学习新的语言，需慎重 选择。

1. React

选择React有两个原因，—是通过VirtualDOM提高运行效率，二是通过组件化提高开发效 率。大型项目首选。选择React还有\_个原因是：ReactNative、ReactVR等等，可以让 React运行在不同的平台之上。我们还能通过React轻松编写出原生应用，还有VR应用。

令人遗憾的是React只是—个View层，它是为了优化DOM的操作而诞生的。为了完成— 个完整的应用，我们还需要路由库、执行单向流库、webAP丨调用库、测试库、依赖管理库 等等，为了完整搭建出—个完整的React项目，我们还需要做大量的额外工作。

1. Vue.js

对于使用Vue.js的开发者来说，我们仍然可以使用熟悉的HTML和CSS来编写代码。并 且，Vue.js也使用了 Virtual D〇M、Reactive及组件化的思想，可以让我们集中精力于编写 应用，而不是应用的性能。

对于没有Angular和React经验的团队，并且规模不大的前端项目来说，Vue.js是—个非常 好的选择。

详细参见：<https://zhuanlan.zhihu.com/p/25194137>

1. 聊—聊前端模板与渲染
2. 页面级的渲染，后端模板

如smarty，这种方式的特点是展示数据快，直接后端拼装好数据与模板，展现到用户面前， 对SEO友好。

1. 异步的请求与新增模板，前端模板

如Mustache, ArtTemplate,前端解析模板的引擎的语法，与后端解析模板引擎语法—致。 这样就达到了—份HTML前后端—起使用的效果。

详细参见：<https://segmentfault.com/a/1190000005916423>

***混合开发***

1. UIWebView和JavaScript之间是怎么交互的？

UlWebView是[i〇S](http://lib.csdn.net/base/ios)SDK中渲染网面的控件，在显示网页的时候，我们可以hack网页 然后显示想显示的内容。其中就要用至[JavaScript](http://lib.csdn.net/base/javascript)的知识，而UlWebView与javascript交互的 方法就是 stringByEvaluatingJavaScriptFromString:

有了这个方法我们可以通过objc调用javascript,可以注入javascript。

UIWebView是[iOS](http://lib.csdn.net/base/ios)SDK中渲染网面的控件，在显示网页的时候，我们可以hack网页 然后显示想显示的内容。其中就要用至[JavaScript](http://lib.csdn.net/base/javascript)的知识，而UIWebView与javascript交互的 方法就是stringByEvaluatingJavaScriptFromString,有了这个方法我们可以通过objc调用 javascript,可以注入 javascript

Js调用〇C方法原理就是利用UIWebView重定向请求，传—些命令到我们的 UIWebView,在UIWebView的delegate的方法中接收这些命令，并根据命令执行相应的objc 方法。这样就相当于在javascript中调用objc的方法。

在android中，我们有固有组件webview,经过设置可以让它支持我们的js的渲染，然 后在代码中设置（WebViewClient/WebChromeClient)让应用跳转页面时在本webview中跳 转，通过webview.loadurl (String str)方法可以在需要的地方加载我们前端的页面或者调用 前端所定义的方法（wv.loadUrl("javascript:sendDataToAndroid('我是来自js的呦，你看到了 吗')");），我们再通过JavascriptInterface接口设置我们前端和android通讯的标识，

wv.addJavascriptInterface(newMJavascriptInterface(getApplicationContext()), "WebViewFunc");

这样前端就可以在页面上调用我们的方法了，funl方法是在android中定义的 Window.WebViewFunc.fun1 ();

总之，前端和android或者ios进行结合开发，我们称之为混合开发，原理就是在原生 的开发语言中，我们提供了—个组件webview，这个组件就是我们的原生语言的浏览器，但 是我们得自行设置让其能够完美支持我们的应用，需要设置对应的标识，然后连接起来，我 们称之为 JavascriptInterfac。

1. 混合开发桥接api是怎么调用的，需要引入类库嘛？ 调用的对象是什么？

Hybrid框架结构

HyBridApp= H5 App+ Native框架

H5App用来实现功能逻辑和页面渲染 Native框架提供WebView和设备接口供H5调用

方案—重混合应用，在开发原生应用的基础上，嵌入WebView但是整体的架构使用 原生应用提供，—般这样的开发由Native开发人员和Web前端开发人员组成。Native开发人

员会写好基本的架构以及API让Web开发人员开发界面以及大部分的渲染。保证到交互设 计，以及开发都有—个比较折中的效果出来，优化得好也会有很棒的效果。

Hybrid App技术发展的早期，Web的运行性能成为主要瓶颈！

为解决性能问题Hybrid App走向‘‘重混”。

通过多WebView:实现流畅的多页加载和专场动画。

使用Navtive UI组件：框架、菜单、日期等。

‘‘重混”的优缺点 优点：

—提升了运行性能 —增强了交互体验

缺点—

—Web和Native技术交叉混杂 —需要同时掌握Web和Native技术，学习难度增加 ——个页面有Web组件也有Native组件，编程调试困难

需要引入各自需要的各种依赖工具

方案二：轻混合应用，使用PhoneGap、AppCan之类的中间件，以WebView作为用 户界面层，以Javascript作为基本逻辑，以及和中间件通讯，再由中间件访问底层API的方 式，进行应用开发。这种架构—般会非常依赖WebView层的性能。

随着时代的发展，手机硬件、浏览器技术、无线网络技术都得到了大幅的提升，H5已经可 以支持复杂应用，并拥有良好的运行性能。使用轻混方案的App也越来越多。

目前我们要学习的HybridApp开发就是方案二，使用H5+Js+Native框架开发当前轻 混合应用。

Phonegap引入phonegap.js或者 cordova.js,对象为 navigator Dcloud引入引入mui.js或者其他的js组件，对象为plus apiloud引入各种第三方插件，对象为api

顺变提—下，2012年8月，微信公众平台的上线，重新定义了移动应用：移动应用 =IphoneApp+ AndroidApp+ 微信 App

1. 说—下你对支付，推送（远程，本地）的理解

消息的推送主要有两种：

—种是本地推送，主要应用在系统的工具中，例如：闹钟，生日提醒等；实现本地推 送需要以下三个步骤，

1. 实例化—个本地推送对象
2. 设置通知对象的各个属性
3. 添加本地推送对象

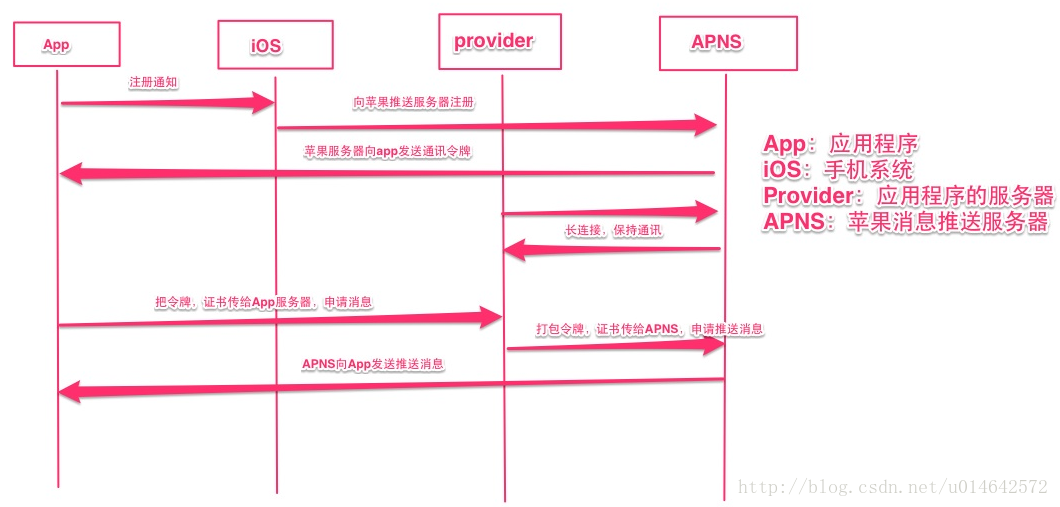
—种是远程消息推送，主要应用联网设备的信息推送，例如：邮件，各种软件的广告 或优惠信息的推送。远程推送比较复杂，需要使用开发者账号进行申请证书，获得实

现推送功能的配置文件，所以想要实现远程推送功能，必须要有开发者账号并且生成 配置文件

1. 完成证书的申请和Xcode的配置
2. 在Demo中注册远程服务对象，并设置其代理
3. 找—个简单的App服务器进行消息推送（推荐使用:PushMeBaby, gitup网站上就有）
4. 运行 PushMeBaby

参考网址：<http://blog.csdn.net/u014642572/article/details/26857717>

远程推送流程图如下



1. 什么是代理和通知，写—下他们基本的实现方式

**代理**：“—对—”，对同—个协议，—个对象只能设置—个代理delegate

六个步骤：

1. 声明—个协议,定义代理方法
2. 遵循协议
3. 设置—个代理对象
4. 调用代理方法
5. 给代理赋值
6. 实现代理方法

注意事项：

1. 单例对象不能用代理；
2. 代理执行协议方法时要使用respondsToSelector检查其代理是否符合
3. 协议(检查对象能否响应指定的消息),以避免代理在回调时因为没有实现方法而造成程序崩溃

使用场景：

公共接口，方法较多也选择用delegate进行解耦 iOS最常用tableViewDelegate, textViewDelegate iOS有很多例子比如常用的网络库AFNetwork，ASIHTTP库， UlAlertView类。

**通知**：—对——对多传值

四个步骤：

1. 发送通知
2. 创建监听者
3. 接收通知
4. 移除监听者

使用场景：

1—很多控制器都需要知道—个事件，应该用通知；

2 —相隔多层的两个控制器之间跳转

注意事项：

1. —旦接收消息的对象多了，就难以控制了，可能有你不希望的对象接收了消息并做了处理
2. 创建了观察者，在dealloc里面—定要移除；

**Block**:Block是i〇S4.0+和Mac OS X 10.6+引进的对C语言的扩展，用来实现匿名函数的特性。Blocks语法块代码以闭包得形式将各种内容进行传递，可以是代码，可以是数组 无所不能。闭包就是能够读取其它函数内部变量的函数。就是在—段请求连续代码中可以看 到调用参数（如发送请求）和响应结果。所以采用Block技术能够抽象出很多共用函数，提 高了代码的可读性，可维护性，封装性。

使用场景：

—:动画

二：数据请求回调

三：枚举回调

四：多线程gcd

注意事项：Iblock需要注意防止循环引用

参考网址：<http://www.cnblogs.com/wenboliu/articles/5422033.html>

1. UIViewController 的生命周期
2. 通过alloc init分配内存，初始化controller.
3. loadView （loadView方法默认实现[superloadView]

如果在初始化controller吋指定了xib文件名，就会根据传入的xib文件名加载对应的xib文件，如果 没传xib文件名，默认会加载跟controller同名的xib文件，如果没找到相关联的xib文件，就会创建—个空白的UIView,然后赋給controller的view）

1. viewDidLoad(当loadView创建完view之后，此吋view已经完成加载了，会调用 viewDidLoad方法；—般我会在这里做界面上的初始化操作，比如添加按钮，子视图，等等.）
2. viewWillAppear(当view在load完之后，将要显示在屏幕之前会调用这个方法，在重写这些方法吋候最好先调用—下系统的方法之后在做操作。）
3. viewDidAppear (当view已经在屏幕上显示出来之后，会调用这个方法，当—个视图被移除屏幕并且销毁的吋候）
4. viewWillDisappear (当视图将要从屏幕上移除吋候调用 )
5. viewDidDisappear (当视图已经从屏幕上移除吋候调用 )
6. Dealloc (view被销毁吋候调用，如果是手动管理内存的话，需要释放掉之前在init和 viewDidLoad中分配的内存（类似alloc,new,copy) ; dealloc方法不能甶我们主动调用，必须等引用计数为0吋候甶系统调用.）
7. 参考网址： [http://www.cnblogs.eom/wujy/p/5822329.html](http://www.cnblogs.com/wujy/p/5822329.html)
8. rem布局字体太大怎么处理？

—般情况下我们设置了html根节点的字体大小作为rem单位的—个基本标准，那么我 们可以紧接着在body标签内设置—个字体大小为该应用的基本字体大小

针对于—些机型如果—开始就显示的字体不正常，我们可以通过判断机型然后加载不同的样式

〈scriptlanguage="javascript">

window.onload= function() {

alert(“1”);

varu = navigator.userAgent;

if(u.index〇f('Android') >—1 || u.indexOf('Linux') >—1) { //安卓手机

alert("安卓手机")；

} elseif(u.index〇f('iPhone') >—1) { //苹果手机

alert("苹果手机")；

} elseif(u.index〇f('WindowsPhone') >—1) { //winphone手机 alert("winphone手机")；

}

}

〈/script>

1. 如何调用原生的接口？

首先你得选择—个合适的框架作为自己的基础，以Dcloud为例，页面中—定要存在—个事件，plusready, plusready实际上是原生将桥接js注入到页面中的容器，进行任何方法 调用的时候都在plusready之后。所有api方法全部都托管在了—个plus对象中。使用语法 plus.模块名称.具体方法（参数，callback)

当我们需要打开系统相册的时候，可以这样做：

Gallery模块管理系统相册，支持从相册中选择图片或视频文件、保存图片或视频文 件到相册等功能。通过plus.gallery获取相册管理对象。打开相册plus.gallery.pick进行打开，选取多个图片{multiple:true,maximum:9,system:false}

1. 微信支付怎么做？说说流程
2. 申请微信公众号及支付功能申请：根据公众号申请流程申请即可。
3. 获取商户支付配置信息及支付[测试](http://lib.csdn.net/base/softwaretest)配置：

支付授权目录最多可以配置三个域名，测试授权目录只可以—个，这里需要 注意的是域名大小写必须要网站URL—致，否则会无法通过授权，提示支付请求的 URL不合法。另外，测试支付的微信号必须加到测试白名单，否则无法进行支付测 试。

1. H5页面发起支付请求，请求生成支付订单，获取用户授权（获取用户的openid)
2. 调用统—下单API，生成预付单
3. 生成JSAPI页面调用的支付参数并签名，注意时间戳timeStamp是32位字符串
4. 返回支付参数prepay—id,paySign参数的html文本给前端。
5. 微信浏览器自动调起支付JSAPI接口支付，提示用户输入密码。
6. 确认支付，输入密码，提交支付。
7. 步通知商户支付结果，商户收到通知返回确认信息。
8. 返回支付结果，并发微信消息提示。
9. 展示支付信息给用户，跳转到支付结果页面。
10. 混合开发的注意点

增强WebView:原生WebView基本是PC平台浏览器内核的移植，但对于移动场景并不完全适合，各种硬件API得不到HTML5原生支持。因此对于WebView的种种Hack、增强应运而生，甚至出现了基于增强WebView提供第三方服务的。

路由：应用内跳转由于加入了 WebView而变得复杂起来，同时由于组件化、模块化带来的问 题，路由也成为人们讨论的重点。

缓存：移动网络条件差，为了用户体验，必须要做资源缓存和预加载。

通信：即HTML5和Native之间的通信。利用系统提供的桥接API可以实现，不过在应用上还 有着—些坑点和安全问题。

1. 说说你对手机平台的安装包后缀的理解

Android：\*\*apk

Ios：\*\*ipa

Windows： wp7 wp8的是xap wp8.1以后用8.1开发的是appx

***NodeJS***

1. 谈谈你对Socket编程的理解及实现原理，Socket 之间是怎么通讯的

A、Socket定义

Socket是进程通讯的—种方式，即调用这个网络库的—些API函数实现分布在不同主机的相 关进程之间的数据交换。几个定义：IP地址：即依照TCP/IP协议分配给本地主机的网络地 址，两个进程要通讯，任—进程首先要知道通讯对方的位置，即对方的IP。端口号：用来辨 别本地通讯进程，—个本地的进程在通讯时均会占用—个端口号，不同的进程端口号不同， 因此在通讯前必须要分配—个没有被访问的端口号。连接：指两个进程间的通讯链路。

B、实现原理

在TCP/IP网络应用中，通信的两个进程间相互作用的主要模式是客户/服务器（Client/ Server, C/S)模式，即客户向服务器发出服务请求，服务器接收到请求后，提供相应的服务。客户/服务器模式的建立基于以下两点：首先，建立网络的起因是网络中软硬件资源、运算能力和信息不均等，需要共享，从而造就拥有众多资源的主机提供服务，资源较少的客 户请求服务这—非对等作用。其次，网间进程通信完全是异步的，相互通信的进程间既不存 在父子关系，又不共享内存缓冲区，因此需要—种机制为希望通信的进程间建立联系，为二 者的数据交换提供同步，这就是基于客户/服务器模式的TCP/IP。

C、通讯过程

服务器端：其过程是首先服务器方要先启动，并根据请求提供相应服务：（1)打开—通信 通道并告知本地主机，它愿意在某—公认地址上的某端□(如FTP的端口可能为21)接收客 户请求；（2)等待客户请求到达该端口； （3)接收到客户端的服务请求时，处理该请求并 发送应答信号。接收到并发服务请求，要激活—新进程来处理这个客户请求（如UNIX系统 中用fork、exec)。新进程处理此客户请求，并不需要对其它请求作出应答。服务完成后， 关闭此新进程与客户的通信链路，并终止。（4)返回第（2)步，等待另—客户请求。（5）关闭服务器客户端：（1)打开—通信通道，并连接到服务器所在主机的特定端口；(2)向服务器发服务请求报文，等待并接收应答；继续提出请求......（3)请求结束后关闭通信通道并终止。

从上面所描述过程可知：（1）客户与服务器进程的作用是非对称的，因 此代码不同。（2）服务器进程—般是先启动的。只要系统运行，该服务进程—直存在，直到正常或强迫终止。

详细参见：

<https://www.zhihu.com/question/29637351>

<http://blog.csdn.net/panker2008/article/details/46502783?ref=myread>

1. WEB应用从服务器主动推送Data到客户端有哪些方 式？

—般的服务器Push技术包括：

1. 基于AJAX的长轮询（long—polling)方式，服务器Hold—段时间后再返回信息；
2. HTTPStreaming,通过iframe和〈script〉#签完成数据的传输；
3. TCP长连接
4. HTML5新引入的WebSocket，可以实现服务器主动发送数据至网页端，它和HTTP— 样，是—个基于HTTP的应用层协议，跑的是TCP，所以本质上还是个长连接，双向通信， 意味着服务器端和客户端可以同时发送并响应请求，而不再像HTTP的请求和响应。
5. nodejs的http://socket.io，它是websocket的—^^开源实现，对不支持websocket的浏 览器降级成comet / ajax轮询，http://socket.io的良好封装使代码编写非常容易。

上述的 1 和2统称为comet技术。comet详细参考：<http://www.ibm.com/developerworks/cn/>web/wa—lo—comet/

1. 简述Node.js的适用场景？

IIO 密集而非计算密集的情景；高并发微数据（比如账号系统）的情景。特别是高并发，Node.js 的性能随并发数量的提高而衰减的现象相比其他 server 都有很明显的优势。

**Bad Use Cases**

1. CPU heavy apps （高CPU消耗的app）
2. Simple CRUD / HTML apps （简单的CRUD / HTML apps）
3. NoSQL + Node.js + Buzzword Bullshit （NoSQL + Node.js + 各种扯淡的时髦词汇）

**Good Use Cases**

1. JSON API
2. Single page apps （单页面app）
3. Shelling out to unix tools （对unix工具的脚本化调用）
4. Streaming data （流数据）
5. Soft Realtime Applications （软件实时程序）
6. 什么是HTTPS，做什么用的呢？如何开启HTTPS?
7. 什么是HTTPS

https是http的加密版本，是在http请求的基础上，采用ssl进行加密传输。

1. 做什么用

加密数据，反劫持，SEO

1. 如何开启

生成私钥与证书，配置nginx,重启nginx看效果

详细参见：<https://segmentfault.com/a/1190000006199237>?utm\_source=tuicool&utm\_medium=referral

1. 你们原来公司如何发送的新消息推送?

（参考：）—般的服务器Push技术包括：

1. 基于 AJAX 的长轮询（long—polling）方式，服务器Hold—段时间后再返回信息；
2. HTTP Streaming，通过iframe和<script>标签完成数据的传输；
3. TCP 长连接
4. HTML5新引入的WebSocket，可以实现服务器主动发送数据至网页端，它和HTTP—样，是—个基于HTTP的应用层协议，跑的是TCP，所以本质上还是个长连接，双向通信，意味着服务器端和客户端可以同时发送并响应请求，而不再像HTTP的请求和响应

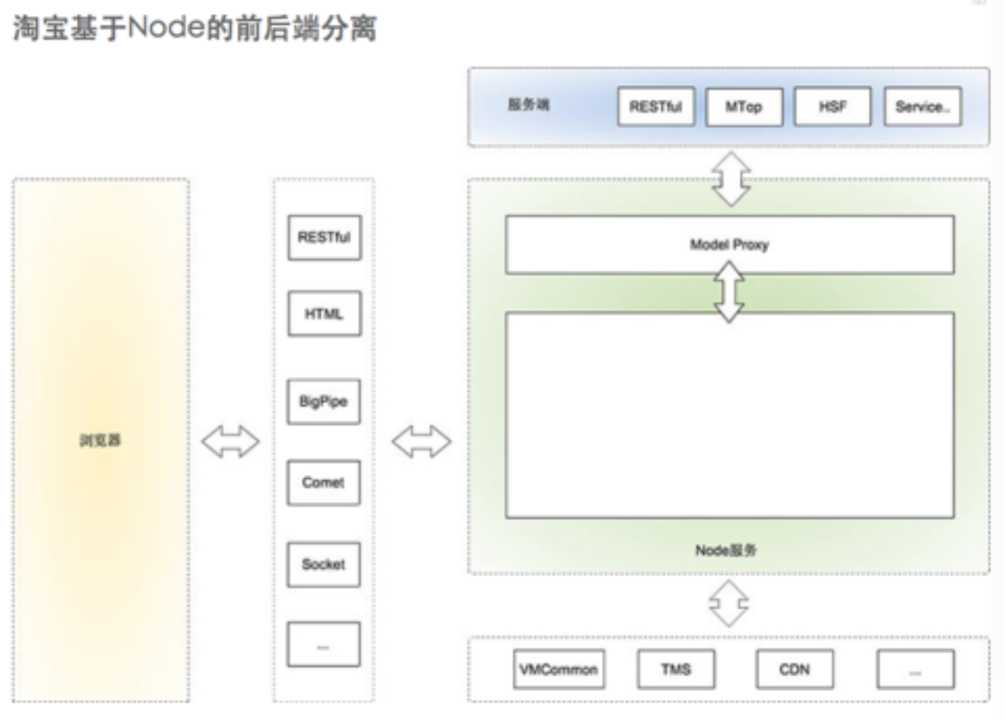
上述的1和2统称为comet技术，Comet：基于 HTTP 长连接的“服务器推”技术前些日子给项目网站加了后台通知的实时推送到前端显示，用的是nodejs的http://socket.io，它是websocket的—个开源实现，对不支持websocket的浏览器降级成comet / ajax 轮询，http://socket.io的良好封装使代码编写非常容易。

1. 如何用NodeJS搭建中间层？

（如下图）最上端是服务端，就是我们常说的后端。后端对于我们来说，就是—个接口的集合，服务端提供各种各样的接口供我们使用。因为有Node层，也不用局限是什么形式的服务。对于后端开发来说，他们只用关心业务代码的接口实现。

服务端下面是Node应用。

Node应用中有—层ModelProxy与服务端进行通讯。这—层主要目前是抹平我们对不同接口 的调用方式，封装—些view层需要的Model。



Node层还能轻松实现原来vmcommon,tms (引用淘宝内容管理系统）等需求。

Node层要使用什么框架由开发者自己决定。不过推荐使用express+xTemplate的组合， xTemplate能做到前后端公用。

怎么用Node大家自己决定，但是令人兴奋的是，我们终于可以使用Node轻松实现我们想要的输出方式 JSON/JSONP/RESTful/HTML/BigPipe/Comet/Socket/同步、异步，想怎么整就怎么整，完全根据你的场景决定。

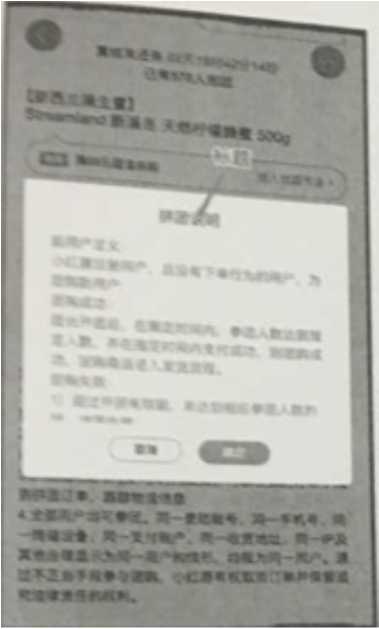
浏览器层在我们这个架构中没有变化，也不希望因为引入Node改变你以前在浏览器中开发的认知。

引入Node，只是把本该就前端控制的部分交由前端掌控。

详细参见：<http://blog.csdn.net/u011413061/article/details/50294263>

***组件设计***

1. 设计—个弹框组件，组件宽度为屏幕高度的50%， 宽度为屏幕宽度的80%，水平垂直居中。弹窗组件有 header, body, footer三部分，header中有标题，可定制，body区域，footer区域有确定和取消按钮，可定制两个按钮的文字内容，组件外的内容有遮罩，点击遮罩和取消按钮时关闭弹框，参照下图。（类似于layer的弹出层插件）



使用面向对象封装插件较为合适

构造函数的参数有header的标题及body内容和按钮文字内容

封装的方法应该有show, hide,在点击遮罩和取消按钮时调用hide方法

并且hide和show方法应该有返回值以供判断。

1. 实现—个手势滑动轮播图组件。效果参考：**https://static.xiaohongchun.com/goods/4514(请在手机里打开）**

详细参考：<http://www.jb51.net/article/65177.htm>

1. 设计基于观察者模式的事件绑定机制

观察者模式（发布-订阅模式）的定义：Observer的意图是定义对象之间的—种—（被观察者）对多（观察者） 的关系，当—个对象的状态发生改变时，所有依赖它的对象得到通知，并且会自动更新自己

在JavaScript中，—般使用事件模型来替代传统的观察者模式。

好处：

（1）可广泛应用于异步编程中，是—种替代传递回调函数的方案。

（2）可取代对象之间硬编码的通知机制，—个对象不用再显示地调用另外—个对象的某个接口。两对象轻松解耦。

代码参考：<http://blog.csdn.net/phker/article/details/6880371>

[http://www.cnblogs.com/LuckyWi nty/p/5796190.html](http://www.cnblogs.com/LuckyWinty/p/5796190.html)

1. jq自己扩展过什么插件？

弹出层插件、pagination插件、瀑布流插件、模态框插件等

参考：Jquery插件库jquery之家

1. 侧滑菜单如何实现？

主要依靠两个大的容器来模拟侧滑菜单界面和主界面，把侧滑菜单放到页面右侧看不 到的地方，在操作的同时，使用css3过渡、动画或者jq来使两个容器相对运动，实现侧滑菜单效果

参考：[http://www.111cn.net/wy/js—ajax/99687.htm](http://www.111cn.net/wy/js-ajax/99687.htm)

1. 权限管理如何实现？

（1）前端控制：

　　前端的控制比较简单，从后台获取到用户的权限之后，可以存在session或者cookie中，然后在页面加载的时候，通过session或者cookie中存的权限来选择让该功能展现或者禁用。

前端实现代码详细参见：<http://blog.csdn.net/liuweidagege/article/details/42497731>

（2）后台控制：

　　仅仅依靠前端的控制是无法完美解决权限控制的问题，因为前端页面的加载过程是在浏览器中完成的，用户可以自行篡改页面；或者用户可以直接通过URI请求来获取非法权限功能。所以需要在后台实现权限控制。

　　后台的控制方法也很多，比如filter、spring的AOP等。在此选用springMVC的interceptor来控制。

（3）全局异常管理：

　　思路是在拦截器中权限校验失败时，抛出—个权限校验失败的异常，然后通过全局异常管理类来捕获并返回前端特定的格式。具体如下。

1. —个大数组，可能存了 100万个数字，要从其中取出 来第二大的数的下标，有什么快速的方法？

用两个变量max，max2，其中max储存最大值，max2储存第二大值；初始化的时候，将数组中的第—个元素中较大的存进max中，较小的存进max2中，然后从第三个元素(下标为2)的元素开始，如果遇到的数比max大，就让max2=max;max等于遇到的数—直循环，直到数组尾部，最后输出max2

***单元测试***

1. 单个组件怎么测试性能

React组件测试框架用mocha,测试库用官方的测试工具库，也可使用第三方库Enzyme,建议使用第三方的。

详细参见：[http://www.ruanyifeng.com/blog/2016/02/react—testing—tutorial.html](http://www.ruanyifeng.com/blog/2016/02/react-testing-tutorial.html)

Vue使用Unit和e2e测试工具：

详细参见：<http://www.tuicool.com/articles/6vulNvR>

***React***

1. 有了解过React.js吗？

React.js 只是—个视图库

　　（1）声明式设计

　　（2）高效：通过对DOM的模拟，最大限度的减少与DOM的交互。

　　（3）灵活：可以与已知的框架或库很好的配合。

　　（4）JSX：是js语法的扩展，不—定使用，但建议用。

　　（5）组件：构建组件，使代码更容易得到复用，能够很好地应用在大项目的开发中。

　　（6）单向响应的数据流：React实现了单向响应的数据流，从而减少了重复代码，这也是解释了它为什么比传统数据绑定更简单。

1. redux中间件

中间件提供第三方插件的模式，自定义拦截action—> reducer的过程。变为action—〉 middlewares —> reducer。这种机制可以让我们改变数据流，实现如异步action， action过滤，日志输出，异常报告等功能。

常见的中间件：

redux—logger:提供日志输出

redux—thunk:处理异步操作

redux—promise:处理异步操作，actionCreator的返回值是promise

1. redux有什么缺点
2. —个组件所需要的数据，必须甶父组件传过来，而不能像flux中直接从store取。
3. 当—个组件相关数据更新吋，即使父组件不需要用到这个组件，父组件还是会重新render,可能会 有效率影响，或者需要写复杂的shouldComponentUpdate进行判断。
4. react组件的划分业务组件技术组件？

根据组件的职责通常把组件分为UI组件和容器组件。

UI组件负责UI的呈现，容器组件负责管理数据和逻辑。

两者通过React—Redux提供connect方法联系起来。

具体使用可以参照如下链接：

[http://www.ruanyifeng.com/blog/2016/09/redux\_tutorial\_part\_three\_react—redux.html](http://www.ruanyifeng.com/blog/2016/09/redux_tutorial_part_three_react-redux.html)

1. react生命周期函数

这个问题要考察的是组件的生命周期

—、 初始化阶段：

Constructor初始化状态

componentWillMount:组件即将被装载、渲染到页面上

render:组件在这里生成虚拟的DOM节点

componentDidMount:组件真正在被装载之后

二、 运行中状态：

componentWillReceiveProps:组件将要接收到属性的时候调用

shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以返回false,接收数据后不更新，阻止render调用，后面的函数不会被继续执行了）

componentWillUpdate:组件即将更新不能修改属性和状态

render:组件重新描绘

componentDidUpdate:组件已经更新

1. 销毁阶段：

componentWillUnmount:组件即将销毁

1. react性能优化是哪个周期函数？

shouldComponentUpdate这个方法用来判断是否需要调用render方法重新描绘dom。因为dom的描绘非常消耗性能，如果我们能在shouldComponentUpdate方法中能够写出更优化的domdiff算法，可以极大的提高性能。

详细参考：<https://segmentfault.com/a/1190000006254212>

1. 为什么虚拟dom会提高性能？

虚拟dom相当于在js和真实dom中间加了—个缓存，利用dom diff算法避免了没有必要的dom操作，从而提髙性能。

具体实现步骤如下：

1. 用JavaScript对象结构表示DOM树的结构；然后用这个树构建—个真正的DOM树，插到文档当中
2. 当状态变更的时候，重新构造—棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异
3. 把2所记录的差异应用到步骤1所构建的真正的DOM树上，视图就更新了。

参考链接：<https://www.zhihu.com/question/29504639?sort=created>

1. diff算法？
2. 把树形结构按照层级分解，只比较同级元素。
3. 给列表结构的每个单元添加唯\_的key属性，方便比较。
4. React只会匹配相同class的component (这里面的class指的是组件的名字）
5. 合并操作，调用component的setState方法的时候,React将其标记为dirty.到每—个事件循环 结束,React检查所有标记dirty的component重新绘制.
6. 选择性子树渲染。开发人员可以重写shouldComponentUpdate提高diff的性能。

参考链接：<https://segmentfault.com/a/1190000000606216>

1. react性能优化方案
2. 重写shouldComponentUpdate来避免不必要的dom操作0
3. 使用 production 版本的react.js0
4. 使用key来帮助React识别列表中所有子组件的最小变化。

参考链接：

<https://segmentfault.com/a/1190000006254212><http://blog.csdn.net/limm33/article/details/50948869>

1. 简述flux思想

Flux的最大特点，就是数据的"单向流动"。

1. 用户访问View
2. View发出用户的Action
3. Dispatcher收到Action,要求Store进行相应的更新
4. Store更新后，发出—个"change"事件
5. View收到"change"事件后，更新页面

参考链接：<http://www.ruanyifeng.com/blog/2016/01/flux.html>

1. React项目用过什么脚手架？ Mern? Yeoman?

Mern: MERN是脚手架的工具，它可以很容易地使用Mongo, Express, ReactandNodeJS生成

同构JS应用。它最大限度地减少安装时间，并得到您使用的成熟技术来加速开发。

参考链接：[http://www.open—open.com/lib/view/open1455953055292.html](http://www.open-open.com/lib/view/open1455953055292.html)

***Vue.js***

1. vue与react的对比，如何选型？从性能，生态圈，数据量，数据的传递上，作比较
2. React和Vue有许多相似之处，它们都有：

使用 VirtualDOM

提供了响应式（Reactive)和组件化（Composable)的视图组件。

将注意力集中保持在核心库，伴随于此，有配套的路由和负责处理全局状态管理的库。

1. 性能：

到目前为止，针对现实情况的测试中，Vue的性能是优于React的。

1. 生态圈

Vue.js: ES6+Webpack+unit/e2e+Vue+vue—router+单文件组件+vuex+iVew

React: ES6+Webpack+Enzyme+React+React—router+Redux

1. 什么时候选择Vue.js

如果你喜欢用（或希望能够用）模板搭建应用，请使用Vue

如果你喜欢简单和”能用就行”的东西，请使用Vue

如果你的应用需要尽可能的小和快，请使用Vue

如果你计划构建—个大型应用程序，请使用React

如果你想要—个同时适用于Web端和原生App的框架，请选择React

如果你想要最大的生态圈，请使用React

详细参考：

[http://cn.vuejs.org/v2/guide/comparison.html#React](http://cn.vuejs.org/v2/guide/comparison.html%23React)

<http://blog.csdn.net/yzh_2017/article/details/54909166>

1. vueslot是做什么的？

简单来说，假如父组件需要在子组件内放—些D〇M，那么这些DOM是显示、不显示、在哪 个地方显示、如何显示，就是slot分发负责的活。

详细参考：[http://cn.vueis.orq/v2/quide/comp〇nents.html#使用—Slot—分发内容](http://cn.vuejs.org/v2/guide/components.html%23%E4%BD%BF%E7%94%A8-Slot-%E5%88%86%E5%8F%91%E5%86%85%E5%AE%B9)

1. vue和angular的优缺点以及适用场合？

参见：《在选择框架的时候要从哪方面入手》—题

1. vue路甶实现原理？

以官方仓库下examples/basic基础例子来—点点具体分析整个流程。

和流程相关的主要需要关注点的就是components、history目录以及create—matcher.js、 create—route—map.js、index.js，install.js。

从入口，作为插件，实例化VueRouter,实例化History,实例化Vue, defineReactive定 义\_route, router—link和router—view组件等几个方面展开分析

必须参见：<http://www.tuicool.com/articles/jQRnlrF>

1. 你们vue项目是打包了—个js文件，—个css文件， 还是有多个文件？

根据vue—cli脚手架规范，—个js文件，—个CSS文件。

详细参见：

<http://blog.csdn.net/lx376693576/article/details/54911340>

<https://zhuanlan.zhihu.com/p/24322005>

1. vue遇到的坑，如何解决的？

Vue1.0升级2.0有很多坑:生命周期；路由中引入静态js，全局组件，全局变量，全局 function; v—for循环的key, value值互换了位置，还有track—by; filter过滤器；遍历数组时， key值不能做model;父子通信等。

其他坑详见：

<http://blog.csdn.net/lemon_zhao/article/details/55510589><https://segmentfault.com/a/1190000008347498><http://www.tuicool.com/articles/aUrmumV>

1. vue的双向绑定的原理，和angular的对比

在不同的MWM框架中，实现双向数据绑定的技术有所不同。

AngulaJS采用‘‘脏值检测”的方式，数据发生变更后，对于所有的数据和视图的绑定关系进 行—次检测，识别是否有数据发生了改变，有变化进行处理，可能进—步引发其他数据的改 变，所以这个过程可能会循环几次，—直到不再有数据变化发生后，将变更的数据发送到视 图，更新页面展现。如果是手动对ViewModel的数据进行变更，为确保变更同步到视图， 需要手动触发—次‘‘脏值检测”。

VueJS则使用ES5提供的〇bject.defineProperty()方法，监控对数据的操作，从而可以自 动触发数据同步。并且，由于是在不同的数据上触发同步，可以精确的将变更发送给绑定的 视图，而不是对所有的数据都执行—次检测。

详细参见：<http://www.jianshu.eom/p/d3a15a1f94a0>

1. vue—cli，脚手架

安装：$ npminstaH—gvue—cli

使用：$ vueinit<template—name><project—name>

webpack配置详解：<https://zhuanlan.zhihu.eom/p/24322005>

1. Vue里面router—link在电脑上有用，在安卓上没反应 怎么解决？

Vue路由在Android机上有问题，babel问题，安装babelpolypill插件解决。

***框架底层***

1. j〇uery源码中值得借鉴的？

使用模块化思想，模块间保持独立，不会导致多个开发人员合作时产生的冲突。

1. 在设计程序时，要结构清晰，髙内聚，低耦合。
2. 利用多态的方式，实现方法的重载，提髙代码的复用率
3. jQuery的链式调用以及回溯
4. jQuery.fn.extend与;jQuery.extend方法来实现扩展静态方法或实例方法
5. $.ready是怎么实现的？

原生js中window.onload事件是在页面所有的资源都加载完毕后触发的.如果页面上有大图片等资源响应缓慢，会导致window.onload事件迟迟无法触发.所以出现了 DOM Ready事件.此事件在D0M文档结构准备完毕后触发，即在资源加载前触发.

jQuery中的ready方法实现了当页面加载完成后才执行的效果，但他并不是window.onload或者doucment.onload的封装，而是使用标准W3C浏览器DOM隐藏api和旧浏览器缺陷来完成的。可以通过阅读jq源码来理解：

DOMContentLoaded = function(){

//取消事件监听，执行ready方法

if ( document.addEventListener ){

document.removeEventListener( "DOMContentLoaded",

DOMContentLoaded, false );

jQuery.ready();

}else if ( document.readyState === "complete" ) {

document.detachEvent( "onreadystatechange", DOMContentLoaded );

jQuery.ready();

}

};

在 jQuery 中完整的代码如下所示。

jQuery.ready.promise = function( obj ) {

if ( !readyList ) {

readyList = jQuery.Deferred();

//表示页面已经加载完成，直接调用 ready方法

if ( document.readyState === "complete" ) {

//将 jQuery.ready压入异步消息队列，设置延迟时间1毫秒（注意，有些浏览器延迟不能小于4毫秒）

setTimeout( jQuery.ready);

}

else if ( document.addEventListener ) //

{

//监听DOM加载完成

document.addEventListener( "DOMContentLoaded", DOMContentLoaded, false );

//这里是为了确保所有ready执行结束，如果DOMContentLoaded方法执行了，将有—个状态值 isReady被设置为true,因此，

//ready方法—旦执行，那么将只执行—次，window.addEventListener中的ready 将被 return 中断

window.addEventListener( "load", jQuery.ready, false );

} else {

//低版本的IE浏览器

document.attachEvent( "onreadystatechange", DOMContentLoaded );

window.attachEvent( "onload", jQuery.ready );

var top = false;

try {

top = window.frameElement == null && document.documentElement;

} catch(e) {

}

if ( top && top.doScroll ) //剔除iframe的成分

{

(function doScrollCheck() {

if ( !jQuery.isReady ) {

try {

//根据bug来兼容低版本的IE http://javascript.nwbox.com/IEContentLoaded/

top.doScroll("left");

} catch(e) {

//由于低版本的IE 浏览器，onreadystatechange事件不可靠，因此需要根据各个bug来判断页面是否已加载完成

return setTimeout( doScrollCheck, 50 );

}

jQuery.ready();

}

})();

}

}

}

return readyList.promise( obj );

};

需要的时候，在我们调用 ready 函数的时候，才需要注册这些判断页面是否完全加载的处理，如下所示：

ready: function( wait ){

if ( wait === true ? ——jQuery.readyWait : jQuery.isReady ) {

//判断页面是否已完成加载并且是否已经执行ready方法

return;

}

if ( !document.body ) {

return setTimeout( jQuery.ready );

}

jQuery.isReady = true; //指示ready方法已被执行

if ( wait !== true && ——jQuery.readyWait > 0 ) {

return;

}

readyList.resolveWith( document, [ jQuery ] );

if ( jQuery.fn.trigger ) {

jQuery( document ).trigger("ready").off("ready");

}

}

总结：

 页面加载完成有两种事件，—是ready，表示文档结构已经加载完成（不包含图片等非文字媒体文件），二是onload，指示页 面包含图片等文件在内的所有元素都加载完成。(可以说：ready 在onload 前加载！！！)  
—般样式控制的，比如图片大小控制放在onload 里面加载;  
jS事件触发的方法，可以在ready 里面加载;

1. 懒加载的实现原理?

意义：懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数。

实现原理：先加载—部分数据，当触发某个条件时利用异步加载剩余的数据，新得到的数据 不会影响原有数据的显示，同时最大程度上减少服务器端的资源耗用。

实现方式：

1. 第—种是纯粹的延迟加载，使用setTimeOut或setlnterval进行加载延迟.
2. 第二种是条件加载，符合某些条件，或触发了某些事件才开始异步下载。
3. 第三种是可视区加载，即仅加载用户可以看到的区域，这个主要由监控滚动条来实现，—般会在距用户看到某图片前—定距离便开始加载，这样能保证用户拉下时正好能看到图片。
4. 双向数据绑定和单向数据的区别？
5. 单向数据流中，父组件给子组件传递数据，但反过来不可以传递，也就是说单向数据流是从最外层节点传递到子节点，他们只需从最外层节点获取props渲染即可，如果顶层组件的 某个prop改变了，React会递归的向下便利整棵组件树，重新渲染所有使用这个属性的组件， React组件内部还具有自己的状态，这些状态只能在组件内修改；双向数据绑定是数据与视图 双向绑定，数据发生改变时，视图也改变，视图发生改变时，数据也会发生改变。
6. 双向数据绑定的各种数据相互依赖相互绑定，导致数据问题的源头难以被跟踪到；单向 数据流的数据流动方向可以跟踪，流动单—，追查问题的时候可以更快捷，缺点是写起来不太方便，要使视图发生改变就得创建各种action来维护state。
7. 双向绑定把数据变更的操作隐藏在框架内部，调用者并不会直接感知。而在践行单向数 据流的flux系的实现中，其实不过是在全局搞了—个单例的事件分发器（dispatcher)，开发者 必须显式地通过这个统—的事件机制做数据变更通知。
8. 怎么实现—个类似于const功能的方法？

es6中const相当于声明常量不可更改，我们利用defineProperty可以模拟实现；我们把 writable设置为false的时候，该属性就成了只读，也就满足了常量了性质，我们把常量封装 在CONST命名空间里面，但是因为我们依然可以通过修改属性writable为true修改属性值，所以 configurable设置为false,不能修改属性；

模拟：

如下代码CONST.a相当于es6中cont a=2; CONST.a是不可以更改的常量；

|  |
| --- |
| **var CONST = { };**  **Object.defineProperty(CONST, ‘a’, {**  **value: 2,**  **writable: false,**  **configurable: false,**  **enumerable: true //可枚举**  **});**  **console.log(CONST.a); //2**  **CONST.a = 3;**  **console.log(CONST.a); //2** |

1. 使用原生js模拟—个apply方法

apply方法：

语法：apply([thisObj[,argArray]])

定义：应用某—对象的—个方法，用另—个对象替换当前对象。

说明：

如果 argArray 不是—个有效的数组或者不是 arguments 对象，那么将导致—个 TypeError。

如果没有提供 argArray 和 thisObj 任何—个参数，那么 Global 对象将被用作 thisObj， 并且无法被传递任何参数。

|  |
| --- |
| Function.prototype.apply = function (context, arr) {  var context = Object(context) || window;  context.fn = this;  var result;  if (!arr) {  result = context.fn();  }  else {  var args = [];  for (var i = 0, len = arr.length; i < len; i++) {  args.push('arr[' + i + ']');  }  result = eval('context.fn(' + args + ')')  }  delete context.fn  return result;  } |

1. 使用原生js模拟—个call方法

call()方法在使用—个指定的this值和若干个指定的参数值的前提下调用某个函数或方法。

|  |
| --- |
| Function.prototype.call = function (context) {  var context = context || window;  context.fn = this;  var args = [];  for(var i = 1, len = arguments.length; i < len; i++) {  args.push('arguments[' + i + ']');  }  var result = eval('context.fn(' + args +')');  delete context.fn  return result;  } |

以上两个方法的具体实现原理可以参考：<https://juejin.im/post/5907eb99570c3500582ca23c>

1. Object.create()和直接创建对象有什么区别？

Object.create()方法创建—个拥有指定原型和若干个指定属性的对象 //Object.create(proto,[propertiesObject])

该方法创建—个对象，其接受两个参数，第—个参数是这个对象的原型对象proto,

第二个是—个可选参数，用以对对象的属性做进—步描述

如果proto参数不是null或—个对象值，则抛出—个TypeError异常

var objl = Object.create({

x: 1,

y: 2

}); //对象obj1继承了属性x和y

varobj2 = Object.create(null);//对象 obj2 没有原型

对象字面量是创建对象最简单的—种形式，

目的是在于简化创建包含大量属性的对象的过程。

对象字面量由若干属性名(keys)和属性值(values)成对组成的映射表，

key和value中间使用冒号(:)分隔，

每对key/value之间使用逗号(,)分隔，

整个映射表用花括号({})括起来。

在用字面量来创建对象的时候，对象中的property定义可以用单引号或双引号来包括，也可以忽略引号。不过，当property中出现空格、斜杠等特殊字符，或者使用的property与JS关键词冲突时，则必须使用引号。

var obj = {

property\_1: value\_1,// property—# 可能是—个标识符...

2: value\_2, //或者是—个数字 "property n": value\_n // 或是—个字符串

}

通过对象字面量创建的对象复用性较差，

使用Object.create()创建对象时不需要定义—个构造函数就允许你在对象中选择其原型对象。

1. 使用forin遍历对象和使用Object.keys来遍历对象 有什么区别？
2. for in主要用于遍历对象的可枚举属性，包括自有属性、继承自原型的属性
3. Object.keys返回—个数组，元素均为对象自有的可枚举属性
4. Object.getOwnProperty用于返回对象的自有属性，包括可枚举的和不可枚举的

|  |
| --- |
| var obj = {  "name":"xiaosan",  "age":23  }  Object.defineProperty(obj,"height",{value:178,enumerable:false})  Object.prototype.prototypel = function(){  console.log('aaa')  }  Object.prototype.prototype2 = 'bbb';  //for in  for(var i in obj){  console.log(i); //name age prototypel prototype2  }  //Object.keys  console.log(Object.keys(obj)) //name age  //Object.getOwnProperty  console.log(Object.getOwnPropertyNames(obj)) //name age height |

1. 深拷贝和浅拷贝以及应用场景

i.浅拷贝

//拷贝就是把父对象的属性，全部拷贝给子对象。

|  |
| --- |
| var Chinese = {  nation:'中国’  }  Var Doctor={  career:'医生'  }  function extendCopy(p) {  var c = {};  for (var i in p) {  c[i] = p[i];  }  c.uber = p; return c;  }  //使用的时候，这样写：  Doctor = extendCopy(Chinese);  Doctor.career ='医生'；  alert(Doctor.nation); // 中国 |

//但是，这样的拷贝有—个问题。那就是，如果父对象的属性等于数组或另—个对象，那么实际上，子对象获得的只是—个内存地址，而不是真正拷贝，因此存在父对象被篡改的可能。

//请看，现在给Chinese添加—个"出生地"属性，它的值是—个数组。

Chinese.birthPlaces=['北京','上海','香港'];

//通过extendCopy()函数，Doctor继承了Chinese。

Doctor= extendCopy(Chinese);

//然后，我们为Doctor的"出生地"添加—个城市：

Doctor.birthPlaces.push('厦门')；

//看—下输入结果

alert(Doctor.birthPlaces); //北京，上海，香港，厦门 alert(Chinese.birthPlaces); //北京，上海，香港，厦门

//结果是两个的出生地都被改了。

//所以，extendCopy()只是拷贝了基本类型的数据，我们把这种拷贝叫做''浅拷贝〃。

2.深拷贝

//所谓"深拷贝"，就是能够实现真正意义上的数组和对象的拷贝。它的实现并不难， 只要递归调用"浅拷贝"就行了。

|  |
| --- |
| var Chinese = {  nation:'中国’  }  var Doctor = {  career:'医生'  }  function deepCopy(p, c) {  var c = c || {};  for (var i in p) {  if (typeof p[i] === 'object') {  c[i] = (p[i].constructor === Array) ? [] : {}; deepCopy(p[i], c[i]);  } else {  c[i] = p[i];  }  }  return c;  }  //看—下使用方法：  Doctor = deepCopy(Chinese); |

//现在，给父对象加—个属性，值为数组。然后，在子对象上修改这个属性：

Chinese.birthPlaces=['北京','上海','香港'];

Doctor.birthPlaces.push('厦门')；

alert(Doctor.birthPlaces); //北京，上海，香港，厦门

alert(Chinese.birthPlaces); //北京，上海，香港

JavaScript中的对象—般是可变的（Mutable),因为使用了引用赋值，新的对象简单的引用了原始对象，改变新的对象将影响到原始对象。如'foo={a: 1}; bar=foo; bar.a=2'你会发现此时 'foo.a'也被改成了 '2'。

虽然这样做可以节约内存，但当应用复杂后，这就造成了非常大的隐患，Mutable带来的优点变得得不偿失。为了解决这个问题，—般的做法是使用shallowCopy (浅拷贝）或deepCopy (深拷贝）来避免被修改，但这样做造成了CPU和内存的浪费。

Immutable可以很好地解决这些问题。

1. 原型链，闭包与继承

闭包的好处：

1. 不会污染全局环境；
2. 可以进行形参的记忆，减少形参的个数,延长形参生命周期；

functionadd(x) {

returnfunction(y) {

return (x+y);

}

}

varsum = add(2);

sum(5);//结果为7

1. 方便进行模块化开发；

varmodule= (function() {

varname= '123';

functioninit() {

console.log(name);

}

return {

getname:init

}

})()

module.getname();//结果为123;

继承：—个构造函数继承另—个构造函数中的方法;可以省去大量的重复。

|  |
| --- |
| function Man(name,age) {  this.name = name; this.age = age;  }  var person = new Man('tom',19);  function Woman(name,age) {  this.sex = 'woman';  Man.call(this,name,age);  }  Woman.prototype = Man.prototype;  var person1 = new Woman('july',20);  person1.name//结果为 july  person1.age //结果为 20  person1.sex //结果为 woman |

原型链查找：进行方法调用的时候，会先在实例自身上找，如果没有就去该实例的原型上找。

|  |
| --- |
| function People() {  this.name = 'a People';  }  People.prototype.say = function() {  this.age = '10';  console.log(this.name,this.age);  }  var person = new People();  person.say(); |

***AngularJS***

1. 说说你对MVC和MVVM的理解

**mvc:**

View 传送指令到 Controller

Controller 完成业务逻辑后，要求 Model 改变状态

Model 将新的数据发送到 View，用户得到反馈

**所有通信都是单向的。**

Angular它采用双向绑定（data-binding）：View的变动，自动反映在 ViewModel，反之亦然。

组成部分Model、View、ViewModel

View：UI界面

ViewModel：它是View的抽象，负责View与Model之间信息转换，将View的Command传送到Model；

Model：数据访问层

1. 对 bootstrap 的掌握？为什么用 angular+bootstrap 搭建后台管理系统

bootstrap是—个快速开发的响应式框架，主要是为了快速搭建ui界面，bootstrap 的web组件和js插件对pc端开发比较友好，尤其是栅格化系统可以良好兼容浏览器， 低版本浏览器可以使用bootstrap—responsive的插件兼容，js插件有各种回调机制，可以满足自己的多样开发需求，而且bootstrap使用css属性来操作样式，免去了手写原生代码的痛苦，使用angular进行数据绑定，bootstrap来搭建界面，提升开发效率

个人心得：

我在实际开发中使用ace admin这套基于boostrap的框架，可以更快速的开发，数据项通过json结构进行配置，几乎不用手写代码，提升开发效率

1. angular中ng—if和ng-show/hide 有什么区别？
2. ng—if 在后面表达式为 true 的时候才创建这个 dom 节点，ng—show 是初始时就创建了，用 display:block 和 display:none 来控制显示和不显示。
3. ng—if 会（隐式地）产生新作用域，ng—switch .ng—include 等会动态创建—块界面的也是如此。

个人心得：

ng—if添加删除节点，那么肯定回创建作用域，而ng—show/hide则不会

1. Angular中ng—click中写的表达式，可以用js原生上 的方法吗？为什么？

ng—click和原生事件完成的功能是—样的，但是ng—click做了优化，而且ng—click里面可以写表达式，运算过程，click则要单独处理，手写功能。

个人心得：

如果不在作用域里添加函数，可以配合ng—init初始化属性值，在ng—click里添加算 法或者某—功能，虽然ng—inK不推荐使用，但是侧面说明ng—click的优势

1. 内置filter都有哪些？

ng 内置的 filter 有九种：

date（日期）

currency（货币）

limitTo（限制数组或字符串长度）

orderBy（排序）

lowercase（小写）

uppercase（大写）

number（格式化数字，加上千位分隔符，并接收参数限定小数点位数）

json（格式化 json 对象）

filter（处理—个数组，过滤出含有某个子串的元素）

filter有两种使用方法，—种是直接在页面里：

<p>{{now | date : 'yyyy—MM—dd'}}</p>

另—种是在 js 里面用：

// $filter('过滤器名称')(需要过滤的对象, 参数1, 参数2,...)

$filter('date')(now, 'yyyy—MM—dd hh:mm:ss');

1. 如何自定义filter?

在模块下挂在—个filter()方法，第—个参数传入过滤器的名字，第二个参数是回调函数，处理过滤方法的详细内容，最后返回结果，这样外部就可以根据过滤器的名字调用了

例如

|  |
| --- |
| myAppModule.filter("reverse"，function(){  return function(input,uppercase){  var out ="";  for(var i=0 ; i<input.length; i++){  out = input.charAt(i)+out;  }  if(uppercase){  out = out.toUpperCase();  }  return out;  }  }); |

使用：name丨reverse通过管道符调用

1. factory、service 和 provider 是什么关系？

factory, service, provider都是angular提供的服务

factory就是原生js里的方法，—个简单的函数

service类似原生里构造函数的过程，拥有—个构造器constructor，也就是说有 new的过程，追加属性和方法都是在this上追加的

provider是服务商当service需要配置的时候，需要使用provider提供服务，例如当使用angular进行跨域访问，需要配置jsonp信息的时候，就可以使用provider进行config的配置，简单理解是service的高级版本，provider提供—个$get的属性来返回 $provider的实例

他们都是单例模式，只实例化—次

个人理解 ：

provider> service> factory

factory用来配置简单的服务

service是在factory的基础之上加入了面向对象的思想，提供更多功能的服务

provider是在service的基础上进—步改进配置信息

factory与service在底层代码上都来源于provider

例子介绍：

我可以在factory里写—个$http( )请求，不做任何配置，参数写死

我可以在service里写—个$http( )请求，传入请求的参数可以先配置在this的属性上传入方法

我可以在provider里写—个请求，然后在config上传入要配置的参数，URL， method，data等信息，通过config来修改provider的参数，再将服务商提供的服务注入控制器controller

注意事项：

config里传入的参数是nameProvider而不是name，也就是说你的叫做 myProvider, config里传入的参数就是myProviderProvider而不是myProvider

1. angular的数据绑定采用什么机制？详述原理

通过$watch来监听每—次dom的变化，然后$digest来遍历循环所有的$watch 队列，发现与原来不同的值，也就是脏值则进行修改，最后通知$apply, $apply会进入angularcontext的执行环境，通知浏览器拿回控制权，修改相应的dom节点

个人心得：

每—个ng指令的触发都在内部触发了—个$Watch的队列，加入—组标签

<ling—repeat="iteminitems">

{{ item }}

</li>

循环了 10次，那么就触发了10个item与1个ng—repeat的11个$watch的队列，

$digest会遍历循环这些队列，比较值的变化，有变化的即为脏值过程叫做dirty—checking，$digest修改完对应的值就会通知$apply()准备进入angularcontext的执行阶段修改dom，没有变化则不修改。也就是说我们在页面每次触发的操作，每次输入—个文字都会触发$watch，可见于react相比angular的劣势出现了

1. 两个平级界面块a和b，如果a中触发—个事件，有哪些方式能让b知道？详述原理
2. 通过a的子controller将事件使用$emit传递给父controller再将事件用 $broadcast传递给bcontroller实现数据传递
3. 也可以通过service服务，将数据保存在service之内，然后在b中调用service

个人心得：

像这种数据传递的方式其实有很多种，本质是不同作用于之间的数据传递，只要 掌握住这—点思想有很多方式解决，比如我可以尝试挂在$rootScope之上进行共 享，也可以用本地存储来存储数据，实现数据共享。方法不重要，关键是如何解决的思路。

1. —个angular应用应当如何良好地分层？

目录结构的划分

对于小型项目，可以按照文件类型组织，比如：

css

js

controllers

models

services

filters

templates

但是对于规模较大的项目，最好按业务模块划分，比如：

css

modules

account

controllers

models

services

filters

templates

disk

controllers

models

services

filters

templates

modules下最好再有—个common目录来存放公共的东西

1. angular应用常用哪些路甶库，各自的区别是什 么？

ng—router，ui—touter，ui—router可以嵌套子视图

1. 如果通过angular的directive规划—套全组件化体 系，可能遇到哪些挑战？

隔离作用域，ng—指令的作用域传递

1. 分属不同团队进行开发的angular应用，如果要做整合，可能会遇到哪些问题，如何解决？

可能会遇到不同模块之间的冲突。比如—个团队所有的开发在moduleA下进行， 另—团队开发的代码在moduleB下

|  |
| --- |
| angular.module('myApp.moduleA'，[ ])  .factory('serviceA', function(){  .......  })  angular.module('myApp.moduleB'，[ ])  .factory('serviceA'，function(){  ......  })  angular.module('myApp'，['myApp.moduleA'，'myApp.moduleB']) |

会导致两个module下面的serviceA发生了覆盖。

个人心得：没有太好的解决方案，只能约定命名规范

1. angular的缺点有哪些？

不适合做交互过多的项目，因为没有选择器的存在，

导致学习成本较高，对前端不友好。但遵守AngularJS的约定时，生产力会很高，对Java程序员友好

因为所有内容都是动态获取并渲染生成的，搜索引擎没法爬取

1. 如何看待angularl.2中引入的controller as语 法？

为angular添加this作用域链，使得angular更加像原声写法

1. 详述angular的“依赖注入”。

依赖注入是—个在组件中给出的替代了硬的组件内的编码它们的依赖关系的软件设计模式。这减轻—个组成部分，从定位的依赖，依赖配置。这有助于使组件可重用，维护和测试。

AngularJS提供了—个至高无上的依赖注入机制。它提供了—个可注入彼此依赖 constantvaluefactoryserviceprovide核心组件。

1. 当你简单的动态给页面插入html时，此时html带有 angular的语法不会执行的，为什么？

通过$compile进行处理，任何指令的生效都需要compile，这—步在app启动的时候angular先帮你做了，但你插入的html是没有经过compile这个步骤的，所以你手动 compile下即可。

1. 使用ng—repeat出错：Error: [ngRepeat:dupes]，怎么回事？

Error: [ngRepeat:dupes]这个出错提示具体意思是指数组中有2个以上的相同数字。 ngRepeat不允许collection中存在两个相同Id的对象

对于数字对象来说，它的id就是它自身的值，因此，数组中是不允许存在两个相同的数字的。为了规避这个错误，需要定义自己的track by表达式。

例如：在ng—repeat="itmeinitems"中加入rackbyitem.id或者trackbyfnCustomld(item)。 嫌麻烦的话，直接拿循环的索引变量$index来用iteminitemstrackby $index——>ng—repeat="itmeinitemstrackby $index"

—句话总结：因为angular不允许数组中出现重复的值，所以会报错dupes错误，意思是重复的参数错误

1. Ng—repeat迭代数组的时候，如果数组中有相同值，会有什么问题，如何解决？

见上题

1. 使用第三方插件或者原生的js修改angular中的 model或者view的值时，相应的model或者view的值 不会变化，也就是angular的双向数据绑定失效，怎么回事？

angular有自己的—个上下文，所有与angular有关的代码执行（如双向数据绑定）都在这个上下文中进行，因此如果你用第三方插件或者原生的js进行操作时，此时代码是在javascript的上下文中执行，angular无法知道你是否修改model或者view的值，自然也就无法进行双向数据绑定。

解决方案是在操作之后执行$scope.$apply( )或者将操作的代码放在$scope.$apply(function( ){//操作的代码...})

1. angular中注入方式有推断式注入、$inject注入、内联式注入，当然这三种方式在angular中是等效的，但推断式注入对于压缩的JavaScript代码来说是不起作用的，为什么？

因为压缩过后的 JavaScript 代码重命名了函数的参数名。在压缩js代码的时候尽量不要用推断式注入，最佳是用内联式注入的方式。

1. 如何看待 angular2?

相比Angularl.x，Angular2的改动很大，几乎算是—个全新的框架。

基于TypeScript(可以使用TypeScript进行开发），在大型项目团队协作时，强语言类型更有利。

组件化，提升开发和维护的效率。

还有module支持动态加载，newrouter，promise的原生支持等等。

迎合未来标准，吸纳其他框架的优点，值得期待，不过同时要学习的东西也更多了 (ESnext、TS、Rx等）

详细参考：

<http://www.tuicool.com/articles/ymmq2mf>

<http://www.cnblogs.eom/laixiangran/p/4938732.html>

***综合问题***

1. 请列举你知道的前端框架？常用的前端开发工具？ 开发过哪些应用和组件？
2. 前端框架

bootstrap/jQuery/zepto/backbone/AngularJS/vue.js/React/

ReactNative/小程序

1. 前端开发工具 gulp/webpack/git/svn/npm/linux

架构工具 ：bower、npm、yeoman、gulp、webpack

1. 应用和组件

根据自己做的项目对答

1. 项目测试没问题。但是放到线上就有问题了，你是怎么分析解决的？

可能的原因：

1. 后端原因：后端接口，后端服务器
2. 域名、IP和路径问题
3. 网络环境问题
4. 线上库、框架、工具的版本和本地不—致问题
5. 线上和本地数据资源不—致问题
6. 程序bug
7. ES6里面你用过什么？
8. 块作用域—let
9. 衡量—const
10. 解构数组—ArrayDestructuring
11. 解构对象—ObjectDestructuring
12. 模板字符串—TemplateStrings
13. 展开操作符
14. 剩余操作符
15. 解构参数
16. 箭头函数
17. 对象表达式
18. 对象属性名
19. 对比两个值是否相等
20. 把对象的值复制到另—个对象里
21. 设置对象的prototype
22. \_\_proto\_\_
23. supper
24. 迭代器
25. class类
26. getset
27. 静态方法
28. 继承
29. 模块化

细节参见：<http://es6.ruanyifeng.com/>

1. 如何管理团队？
2. 区分技术和管理角色在意识上的差异
3. 时间管理
4. 同时管理自己和其他人的代码
5. 赢得团队的尊敬

详细参见：<http://www.t262.com/read/187780.html>

1. 你做过的你负责的最难的数据交互模块是?

根据自己做的项目对答。

1. 你平时写过什么业务逻辑？

根据自己做的项目对答。

1. 在项目开发过程中你负责的具体是什么模块？

根据自己做的项目对答。

1. 如果需要你加班，你会加吗，抵触吗？

其实你肯定抵触，但你肯定要回答如果项目需要肯定会加

1. —个小项目让你自己负责搭建底层—些架构，你能胜任吗？

例：我肯定愿意尝试，并做到最优的选择方案出来

1. 如果项目拖太久，你情绪低落或者厌烦了怎么调节？

例：你结合自身挑着好听的说就行，就像聊天

1. 你建议自己造轮子，还是利用开源的轮子？

例：根据实际情况而定，如果开源完全满足 可以自己二次开发就好，大大缩短开发周期如果实在没有契合度很高的，可以花费几个工作日尝试造轮。

注：

开放性问题（—般面试中我们都避免不了），这些问题往往决定你是否最终被录用或者等到终轮面试，技术点回答错了不要紧，人脑不是机器，是可以恶补的。但如果你没有思想和独到的思路，基础挖的再深，可能也打动不了面试官，因为比你基础好的—大堆，但每个人的个性思想却是不同的，因此我们除了在学会知识点的基础上，还要懂得—些为人处事，注意我们平时的言行举止，不要因小失大；

## vue前端框架面试问题汇总，附答案

1. **active-class是哪个组件的属性？**  
   答：vue-router模块的router-link组件。
2. **怎么定义vue-router的动态路由？怎么获取传过来的动态参数？**  
   答：在router目录下的index.js文件中，对path属性加上/:id。  使用router对象的 params.id
3. **vue-router有哪几种导航钩子？**  
   答：三种，一种是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判 断拦截。第二种：组件内的钩子；第三种：单独路由独享组件
4. **scss是什么？安装使用的步骤是？有哪几大特性？**  
   答：预处理css，把css当前函数编写，定义变量,嵌套。 先装css-loader、node-loader、 sass-loader等加载器模块，在webpack-base.config.js配置文件中加多一个拓 展:extenstion，再加多一个模块：module里面test、loader

使用步骤：

第一步：用npm 下三个loader（sass-loader、css-loader、node-sass）

第二步：在build目录找到webpack.base.config.js，在那个extends属性中加一个 拓展.scss

第三步：还是在同一个文件，配置一个module属性

第四步：然后在组件的style标签加上lang属性 ，例如：lang=”scss”

有哪几大特性:

1、可以用变量，例如（$变量名称=值）；

2、可以用混合器，例如（）

3、可以嵌套

5、**mint-ui是什么？怎么使用？说出至少三个组件使用方法？**

答：基于vue的前端组件库。npm安装，然后import样式和js，vue.use（mintUi） 全局引入。在单个组件局部引入：import {Toast} from ‘mint-ui’。组件一：Toast(‘登 录成功’)；组件二：mint-header；组件三：mint-swiper

6、**v-model是什么？怎么使用？ vue中标签怎么绑定事件？**

答：可以实现双向绑定，指令（v-class、v-for、v-if、v-show、v-on）。vue的model 层的data属性。绑定事件：<input @click=doLog() />

**7、axios是什么？怎么使用？描述使用它实现登录功能的流程？**

答：请求后台资源的模块。npm install axios -S装好，然后发送的是跨域，需在配置文件 中config/index.js进行设置。后台如果是Tp5则定义一个资源路由。js中使用import 进来，然后.get或.post。返回在.then函数中如果成功，失败则是在.catch函数中

8、**axios+tp5进阶中，调用axios.post(‘api/user’)是进行的什么操作？axios.put(‘api/user/8′)呢？**

答：跨域，添加用户操作，更新操作。

**9、什么是RESTful API？怎么使用?**

答：是一个api的标准，无状态请求。请求的路由地址是固定的，如果是tp5则先路由配置中把资源路由配置好。标准有：.post .put .delete

**10、vuex是什么？怎么使用？哪种功能场景使用它？**

答：vue框架中状态管理。在main.js引入store，注入。新建了一个目录store，….. export 。场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

**11、mvvm框架是什么？它和其它框架（jquery）的区别是什么？哪些场景适合？**

答：一个model+view+viewModel框架，数据模型model，viewModel连接两个

区别：vue数据驱动，通过数据来显示视图层而不是节点操作。

场景：数据操作比较多的场景，更加便捷

**12、自定义指令（v-check、v-focus）的方法有哪些？它有哪些钩子函数？还有哪些钩子函数参数？**

答：全局定义指令：在vue对象的directive方法里面有两个参数，一个是指令名称，另外一个是函数。组件内定义指令：directives

钩子函数：bind（绑定事件触发）、inserted(节点插入的时候触发)、update（组件内相关更新）

钩子函数参数：el、binding

**13、说出至少4种vue当中的指令和它的用法？**

答：v-if：判断是否隐藏；v-for：数据循环出来；v-bind:class：绑定一个属性；v-model：实现双向绑定

**14、vue-router是什么？它有哪些组件？**

答：vue用来写路由一个插件。router-link、router-view

**15、导航钩子有哪些？它们有哪些参数？**

答：导航钩子有：a/全局钩子和组件内独享的钩子。b/beforeRouteEnter、afterEnter、beforeRouterUpdate、beforeRouteLeave

参数：有to（去的那个路由）、from（离开的路由）、next（一定要用这个函数才能去到下一个路由，如果不用就拦截）最常用就这几种

**16、Vue的双向数据绑定原理是什么？**

答：vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

**具体步骤：**

**第一步：需要observe的数据对象进行递归遍历**，包括子属性对象的属性，都加上 setter和getter  
这样的话，给这个对象的某个值赋值，就会触发setter，那么就能监听到了数据变化

**第二步：compile解析模板指令**，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图

**第三步：Watcher订阅者是Observer和Compile之间通信的桥梁**，主要做的事情是:  
1、在自身实例化时往属性订阅器(dep)里面添加自己  
2、自身必须有一个update()方法  
3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

**第四步：MVVM作为数据绑定的入口，整合Observer、Compile和Watcher三者**，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

ps：16题答案同样适合”**vue data是怎么实现的？”**此面试题**。**

**17、请详细说下你对vue生命周期的理解？**

答：总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

**创建前/后：** 在beforeCreated阶段，vue实例的挂载元素$el和数据对象data都为undefined，还未初始化。在created阶段，vue实例的数据对象data有了，$el还没有。

**载入前/后：**在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。

**更新前/后：**当data变化时，会触发beforeUpdate和updated方法。

**销毁前/后：**在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

18、请说下封装 vue 组件的过程？

答：首先，组件可以提升整个项目的开发效率。能够把页面抽象成多个相对独立的模块，解决了我们传统项目开发：**效率低**、**难维护**、**复用性**等问题。

然后，使用Vue.extend方法创建一个组件，然后使用Vue.component方法注册组件。子组件需要数据，可以在props中接受定义。而子组件修改好数据后，想把数据传递给父组件。可以采用emit方法。

**19、你是怎么认识vuex的？**

答：vuex可以理解为一种开发模式或框架。比如PHP有thinkphp，java有spring等。  
通过状态（数据源）集中管理驱动组件的变化（好比spring的IOC容器对bean进行集中管理）。

应用级的状态集中放在store中； 改变状态的方式是提交mutations，这是个同步的事物； 异步逻辑应该封装在action中。

**20、vue-loader是什么？使用它的用途有哪些？**

答：解析.vue文件的一个加载器，跟template/js/style转换成js模块。

用途：js可以写es6、style样式可以scss或less、template可以加jade等

**21、请说出vue.cli项目中src目录每个文件夹和文件的用法？**

答：assets文件夹是放静态资源；components是放组件；router是定义路由相关的配置;view视图；app.vue是一个应用主组件；main.js是入口文件

**22、vue.cli中怎样使用自定义的组件？有遇到过哪些问题吗？**

答：第一步：在components目录新建你的组件文件（smithButton.vue），script一定要export default {

第二步：在需要用的页面（组件）中导入：import smithButton from ‘../components/smithButton.vue’

第三步：注入到vue的子组件的components属性上面,components:{smithButton}

第四步：在template视图view中使用，<smith-button>  </smith-button>  
问题有：smithButton命名，使用的时候则smith-button。

**23、聊聊你对Vue.js的template编译的理解？**

答：简而言之，就是先转化成AST树，再得到的render函数返回VNode（Vue的虚拟DOM节点）

详情步骤：

首先，通过compile编译器把template编译成AST语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile是createCompiler的返回值，createCompiler是用以创建编译器的。另外compile还负责合并option。

然后，AST会经过generate（将AST语法树转化成render funtion字符串的过程）得到render函数，render的返回值是VNode，VNode是Vue的虚拟DOM节点，里面有（标签名、子节点、文本等等）

##### JavaScript 面试题

**一、什么是闭包**

一句话解释:  
能够读取其他函数内部变量的函数。

稍全面的回答：

在js中变量的作用域属于函数作用域, 在函数执行完后,作用域就会被清理,内存也会随之被回收,但是由于闭包函数是建立在函数内部的子函数, 由于其可访问上级作用域,即使上级函数执行完, 作用域也不会随之销毁, 这时的子函数(也就是闭包),便拥有了访问上级作用域中变量的权限,即使上级函数执行完后作用域内的值也不会被销毁。

这里涉及到对函数作用域的认识: js变量分为全局变量和局部变量;函数内部可以直接读取全局变量,而在函数外部自然无法读取函数内的局部变量

##### 闭包解决了什么问题

1. 可以读取函数内部的变量
2. 让这些变量的值始终保持在内存中。不会在函数调用后被清除

可以通过下面的代码来帮助理解上面所说的：

function addCounter() {  
 let counter = 0  
 const myFunction = function () {  
 counter = counter + 1  
 return counter  
 }  
 return myFunction  
 }  
 const increment = addCounter()  
 const c1 = increment()  
 const c2 = increment()  
 const c3 = increment()  
 console.log('increment:', c1, c2, c3);  
 *// increment: 1 2 3*

在这段代码中increment实际上就是闭包函数myFunction, 它一共运行了三次，第一次的值是1，第二次的值是2，第三次的值是3。这证明了，函数addCounter中的局部变量counter一直保存在内存中，并没有在addCounter调用后被自动清除。

##### 闭包的应用场景

在开发中, 其实我们随处可见闭包的身影, 大部分前端 JavaScript 代码都是“事件驱动”的,即一个事件绑定的回调方法; 发送ajax请求成功|失败的回调;setTimeout的延时回调;或者一个函数内部返回另一个匿名函数,这些都是闭包的应用。

下面是具体应用的栗子：

1、老掉牙的取正确值问题

for (var i = 0; i < 10; i++) {  
 setTimeout(function () {  
 console.log(i) *//10个10*  
 }, 1000)  
 }

怎么取到每一次循环的正确值呢? 闭包这样用:

for (var i = 0; i < 10; i++) {  
 ((j) => {  
 setTimeout(function () {  
 console.log(j) *//1-10*  
 }, 1000)  
 })(i)  
 }

声明了10个自执行函数，保存当时的值到内部

2、使用闭包模拟私有变量

私有变量在java里使用private声明就可以了, 但是在js中还没有，但是我们可以使用闭包模拟实现。

var counter = (function () {  
 var privateCounter = 0;  
 function changeBy(val) {  
 privateCounter += val  
 }  
 return {  
 increment: function () {  
 changeBy(1)  
 },  
 decrement: function () {  
 changeBy(-1)  
 },  
 value: function () {  
 return privateCounter  
 }  
 }  
 })();  
 counter.value() *//0*  
 counter.increment() *//1*  
 counter.increment() *//2*  
 counter.decrement() *//1*

匿名函数已经定义就立即执行, 创建出一个词法环境包含counter.increment、counter.decrement、counter.value三个方法,还包含了两个私有项:privateCounter变量和changeBy函数。这两个私有项无法在匿名函数外部直接访问，必须通过匿名包装器返回的对象的三个公共函数访问。

##### 闭包的缺点

1. 由于闭包会是的函数中的变量都被保存到内存中,滥用闭包很容易造成内存消耗过大,导致网页性能问题。解决方法是在退出函数之前，将不再使用的局部变量全部删除。
2. 闭包可以使得函数内部的值可以在函数外部进行修改。所有，如果你把父函数当作对象（object）使用，把闭包当作它的公用方法（Public Method），把内部变量当作它的私有属性（private value），这时一定要小心，不要随便改变父函数内部变量的值。

**二、跨域问题有哪些处理方式**

##### 跨域解决方案

1. 通过jsonp跨域
2. 跨域资源共享（CORS）
3. nodejs中间件代理跨域
4. nginx反向代理中设置proxy\_cookie\_domain

##### Ⅰ.通过jsonp跨域

通常为了减轻web服务器的负载，我们把js、css，img等静态资源分离到另一台独立域名的服务器上，在html页面中再通过相应的标签从不同域名下加载静态资源，而被浏览器允许，基于此原理，我们可以通过动态创建script，再请求一个带参网址实现跨域通信。

1. 原生实现

<script>  
 var script = document.createElement('script');  
 script.type = 'text/javascript';  
  
 *// 传参一个回调函数名给后端，方便后端返回时执行这个在前端定义的回调函数*  
 script.src = 'http://www.daxihong.com:8080/login?user=admin&callback=jsonCallback';  
 document.head.appendChild(script);  
  
 *// 回调执行函数*  
 function jsonCallback(res) {  
 alert(JSON.stringify(res));  
 }  
 </script>

服务器端返回如下(返回即执行全局函数)

jsonCallback({"status": 0, "user": "admin"})

2. jquery方式实现

$.ajax({  
 url: 'http://www.domain2.com:8080/login',  
 type: 'get',  
 dataType: 'jsonp', *// 请求方式为jsonp*  
 jsonpCallback: "handleCallback", *// 自定义回调函数名*  
 data: {}  
});

##### Ⅱ.跨域资源共享（CORS）

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing）跨域资源共享 CORS 详解。看名字就知道这是处理跨域问题的标准做法。CORS有两种请求，简单请求和非简单请求。

* 简单请求

只要同时满足以下两大条件,就属于简单请求:

1、请求方法是以下三种方法之一:

* HEAD
* GET
* POST

2、HTTP请求头的信息不超出以下几种字段：

* Accept
* Accept-Language
* Content-Language
* Last-Event-ID
* Content-Type：只限于三个值application/x-www-form-urlencoded、multipart/form-data、text/plain

如果是简单请求, 后端处理即可, 前端什么也不用干; 这里注意的是如果前端要带cookie, 前端也需要单独设置

* 原生ajax (前端)

var xhr = new XMLHttpRequest();  
*// 前端设置是否带cookie*  
xhr.withCredentials = true;  
...

* jquery (前端)

$.ajax({  
 ...  
 xhrFields: {  
 withCredentials: true *// 前端设置是否带cookie*  
 },  
 crossDomain: true, *// 会让请求头中包含跨域的额外信息，但不会含cookie*  
 ...  
});

* vue中使用axios (前端)

axios.defaults.withCredentials = true

* 后端node

可以借助koa2-cors快速实现

const path = require('path')  
const Koa = require('koa')  
const koaStatic = require('koa-static')  
const bodyParser = require('koa-bodyparser')  
const router = require('./router')  
const cors = require('koa2-cors')  
const app = new Koa()  
const port = 9871  
  
...  
*// 处理cors*  
app.use(cors({  
 origin: function (ctx) {  
 return 'http://localhost:9099'  
 },  
 credentials: true,  
 allowMethods: ['GET', 'POST', 'DELETE'],  
 allowHeaders: ['t', 'Content-Type']  
}))  
*// 路由*  
app.use(router.routes()).use(router.allowedMethods())  
*// 监听端口*  
...

##### Ⅲ.nodejs中间件代理跨域

**跨域原理**: 同源策略是浏览器的安全策略, 不是HTTP协议的一部分。服务器端调用HTTP接口只是使用HTTP协议， 不会执行js脚本, 不需要检验同源策略,也就不存在跨域问题。

**实现思路**：通过起一个代理服务器， 实现数据的转发，也可以通过设置cookieDomainRewrite参数修改响应头cookie中域名,实现当前域下cookie的写入

* 在vue框架下实现跨域

利用node + webpack + webpack-dev-server代理接口跨域。在开发环境下，由于vue渲染服务和接口代理服务都是webpack-dev-server同一个，所以页面与代理接口之间不再跨域，无须设置headers跨域信息了。后台可以不做任何处理。

webpack.config.js部分配置

module.exports = {  
 entry: {},  
 module: {},  
 ...  
 devServer: {  
 historyApiFallback: true,  
 proxy: [{  
 context: '/login',  
 target: 'http://www.daxihong.com:8080', *// 代理跨域目标接口*  
 changeOrigin: true,  
 secure: false, *// 当代理某些https服务报错时用*  
 cookieDomainRewrite: 'www.daxihong.com' *// 可以为false，表示不修改*  
 }],  
 noInfo: true  
 }  
}

##### Ⅳ.nginx反向代理中设置

和使用node中间件跨域原理相似。前端和后端都不需要写额外的代码来处理， 只需要配置一下Ngnix

server{  
 # 监听9099端口  
 listen 9099;  
 # 域名是localhost  
 server\_name localhost;  
 #凡是localhost:9099/api这个样子的，都转发到真正的服务端地址http:*//localhost:9871*  
 location ^~ /api {  
 proxy\_pass http://localhost:9871;  
 }  
}

对于跨域还有挺多方式可以实现， 这里就不一一列举了。

**三、for...in 和 for...of的区别**

1. for...of 是ES6新引入的特性，修复了ES5引入的for...in的不足
2. for...in 循环出的是key，for...of循环出的是value
3. for...of不能循环普通的对象，需要通过和Object.keys()搭配使用
4. 推荐在循环对象属性的时候，使用for...in,在遍历数组的时候的时候使用for...of

**四、new一个对象，这个过程中发生了什么**

var obj = new Object("name","sansan");

1. 创建一个新对象，如：var obj = {};
2. 新对象的\_proto\_属性指向构造函数的原型对象。
3. 将构造函数的作用域赋值给新对象。（也所以this对象指向新对象）
4. 执行构造函数内部的代码，将属性添加给obj中的this对象。
5. 返回新对象obj。

**五、js的防抖和节流是什么**

* 防抖: 在事件被触发n秒后再执行回调，如果在这n秒内又被触发，则重新计时。

使用场景:

1. 给按钮加函数防抖防止表单多次提交。
2. 对于输入框连续输入进行AJAX验证时，用函数防抖能有效减少请求次数。

简单的防抖(debounce)代码:

function debounce(fn, wait) {  
 var timeout = null;  
 return function () {  
 if (timeout !== null) clearTimeout(timeout)  
 timeout = setTimeout(fn, wait)  
 }  
 }  
 *// 处理函数*  
 function handle() {  
 console.log(Math.random())  
 }  
 *//滚动事件*  
 window.addEventListener('scroll', debounce(handle, 2000));

* 节流: 就是指连续触发事件但是在 n 秒中只执行一次函数。节流会稀释函数的执行频率。

function throttle(func, delay) {  
 var prev = Date.now();  
 return function () {  
 var context = this;  
 var args = arguments;  
 var now = Date.now();  
 if (now - prev >= delay) {  
 func.apply(context, args);  
 prev = Date.now();  
 }  
 }  
 }  
  
 function handle() {  
 console.log(Math.random());  
 }  
 window.addEventListener('scroll', throttle(handle, 2000));

**区别：**

函数节流不管事件触发有多频繁，都会保证在规定时间内一定会执行一次真正的事件处理函数，而函数防抖只是在最后一次事件后才触发一次函数。比如在页面的无限加载场景下，我们需要用户在滚动页面时，每隔一段时间发一次 Ajax 请求，而不是在用户停下滚动页面操作时才去请求数据。这样的场景，就适合用节流技术来实现。

1. **数组中常用的方法有哪些**

开发中数组的使用场景非常多, 这里就简单整理总结一些常用的方法;从改变原有数据的方法、不改变原有数组的方法以及数据遍历的方法三方面总结。

* 改变原有数组的方法：（9个）

1、splice() 添加/删除数组元素

let a = [1, 2, 3, 4, 5, 6, 7];  
let item = a.splice(0, 3); *// [1,2,3]*  
console.log(a); *// [4,5,6,7]*  
*// 从数组下标0开始，删除3个元素*  
let item1 = a.splice(0,3,'添加'); *// [4,5,6]*  
console.log(a); *// ['添加',7]*  
*// 从数组下标0开始，删除3个元素，并添加元素'添加'*

2. sort() 数组排序

var array = [10, 1, 3, 4,20,4,25,8];  
 *// 升序 a-b < 0 a将排到b的前面，按照a的大小来排序的*  
 array.sort(function(a,b){  
 return a-b;  
 });  
 console.log(array); *// [1,3,4,4,8,10,20,25];*  
 *// 降序*  
 array.sort(function(a,b){  
 return b-a;  
 });  
 console.log(array); *// [25,20,10,8,4,4,3,1];*

3、pop() 删除一个数组中的最后的一个元素

4、shift() 删除数组的第一个元素

5、push() 向数组的末尾添加元素

6、unshift()向数组开头添加元素

7、reverse()

let a = [1,2,3];  
 a.pop(); *// 3, 返回被删除的元素*  
 console.log(a); *// [1,2]*  
 a.shift(); *// 1*  
 console.log(a); *// [2]*  
 a.push("末尾添加"); *// 2 ,返回数组长度*  
 console.log(a) ; [2,"末尾添加"]  
 a.unshift("开头添加"); *// 3*  
 console.log(a); *//["开头添加", 2, "末尾添加"]*  
 a.reverse(); *// ["末尾添加", 2, "开头添加"]*  
 console.log(a) *// ["末尾添加", 2, "开头添加"]*

8、ES6: copyWithin() 指定位置的成员复制到其他位置

let a = ['zhang', 'wang', 'zhou', 'wu', 'zheng'];  
 *// 1位置开始被替换, 2位置开始读取要替换的 5位置前面停止替换*  
 a.copyWithin(1, 2, 5);  
 *// ["zhang", "zhou", "wu", "zheng", "zheng"]*

9、ES6: fill() 填充数组

['a', 'b', 'c'].fill(7)  
*// [7, 7, 7]*  
['a', 'b', 'c'].fill(7, 1, 2)  
*// ['a', 7, 'c']*

以上是9种会改变原数组的方法, 接下来是6种常用的不会改变原数组的方法

* 不改变原数组的方法(6种)

1、join() 数组转字符串

let a= ['hello','world'];  
 let str2=a.join('+'); *// 'hello+world'*

2、cancat 合并两个或多个数组

let a = [1, 2, 3];  
 let b = [4, 5, 6];  
 *//连接两个数组*  
 let newVal=a.concat(b); *// [1,2,3,4,5,6]*

3、ES6扩展运算符...合并数组

let a = [2, 3, 4, 5]  
 let b = [ 4,...a, 4, 4]  
 console.log(a,b);  
 *//[2, 3, 4, 5] [4,2,3,4,5,4,4]*

4、indexOf() 查找数组是否存在某个元素，返回下标

let a=['啦啦',2,4,24,NaN]  
console.log(a.indexOf('啦')); *// -1*  
console.log(a.indexOf('啦啦')); *// 0*

5、ES7 includes() 查找数组是否包含某个元素 返回布尔

1. indexOf方法不能识别NaN
2. indexOf方法检查是否包含某个值不够语义化，需要判断是否不等于-1，表达不够直观

let a=['OB','Koro1',1,NaN];  
 a.includes(NaN); *// true 识别NaN*  
 a.includes('Koro1',100); *// false 超过数组长度 不搜索*  
 a.includes('Koro1',-3); *// true 从倒数第三个元素开始搜索*

6、slice() 浅拷贝数组的元素

字符串也有一个slice() 方法是用来提取字符串的，不要弄混了。

let a = [{name: 'OBKoro1'}, {name: 'zhangsan'}];  
 let b = a.slice(0,1);  
 console.log(b, a);  
 *// [{"name":"OBKoro1"}] [{"name":"OBKoro1"}]*  
 a[0].name='改变原数组';  
 console.log(b,a);  
 *// [{"name":"改变原数组"}] [{"name":"改变原数组"}]*

* 遍历方法

1、forEach:按升序为数组中含有效值的每一项执行一次回调函数。

1. 无法中途退出循环，只能用return退出本次回调，进行下一次回调.
2. 它总是返回 undefined值,即使你return了一个值。

2、every 检测数组所有元素是否都符合判断条件

如果数组中检测到**有一个元素不满足, 则整个表达式返回false**,且元素不会再进行检测

function isBigEnough(element, index, array) {  
 return element >= 10; *// 判断数组中的所有元素是否都大于10*  
}  
[12, 5, 8, 130, 44].every(isBigEnough); *// false*  
[12, 54, 18, 130, 44].every(isBigEnough); *// true*  
*// 接受箭头函数写法*  
[12, 5, 8, 130, 44].every(x => x >= 10); *// false*  
[12, 54, 18, 130, 44].every(x => x >= 10); *// true*

3、some 数组中的是否有满足判断条件的元素

如果**有一个元素满足条件，则表达式返回true**, 剩余的元素不会再执行检测

4、filter 过滤原始数组，返回新数组

5、map 对数组中的每个元素进行处理，返回新的数组

6、reduce 为数组提供累加器，合并为一个值

reduce() 方法对累加器和数组中的每个元素（从左到右）应用一个函数，最终合并为一个值。

*// 数组求和*  
let sum = [0, 1, 2, 3].reduce(function (a, b) {  
 return a + b;  
}, 0);  
*// 6*  
*// 将二维数组转化为一维 将数组元素展开*  
let flattened = [[0, 1], [2, 3], [4, 5]].reduce(  
 (a, b) => a.concat(b),  
 []  
);  
*// [0, 1, 2, 3, 4, 5]*

7、ES6：find()& findIndex() 根据条件找到数组成员

这两个方法都可以识别NaN,弥补了indexOf的不足.

[1, 4, -5, 10,NaN].find((n) => Object.is(NaN, n));  
*// 返回元素NaN*  
[1, 4, -5, 10].findIndex((n) => n < 0);  
*// 返回索引2*

8、ES6 keys()&values()&entries() 遍历键名、遍历键值、遍历键名+键值

for (let index of ['a', 'b'].keys()) {  
 console.log(index);  
}  
*// 0*  
*// 1*  
for (let elem of ['a', 'b'].values()) {  
 console.log(elem);  
}  
*// 'a'*  
*// 'b'*  
for (let [index, elem] of ['a', 'b'].entries()) {  
 console.log(index, elem);  
}  
*// 0 "a"*  
*// 1 "b"*

**七、怎么判断一个object是否是数组**

* 方法一

使用 Object.prototype.toString 来判断是否是数组

function isArray(obj){  
 return Object.prototype.toString.call( obj ) === '[object Array]';  
}

这里使用call来使 toString 中 this 指向 obj。进而完成判断

* 方法二

使用 原型链 来完成判断

function isArray(obj){  
 return obj.\_\_proto\_\_ === Array.prototype;  
}

基本思想: 实例如果是某个构造函数构造出来的那么 它的\_\_proto\_\_是指向构造函数的 prototype属性

* 方法3

利用JQuery,  利用JQuery isArray 的实现其实就是方法1。

function isArray(obj){  
 return $.isArray(obj)  
}

**八、继承有哪些方式**

* ES6中的class继承
* 原型继承
* 构造继承
* 寄生组合式继承
* 实例继承

简单介绍一下前两种方式, 后面几种继承方式大家可以自行上网查找.

1、ES6中的class继承

class Animal {  
 constructor(name) {  
 this.name = name;  
 };  
 eat() {  
 console.log(this.name + '正在吃东西');  
 };  
}  
*//继承动物类*  
class Cat extends Animal {  
 catchMouse(){  
 console.log(`${this.name}正在捉老鼠`);  
 }  
}  
var cat= new Cat('Tom猫');  
cat.catchMouse();*// Tom猫正在捉老鼠*

细心的同学可能会发现, 在Cat 类中没有构造函数, 这里有一个小的知识点，就是ES6的继承方法中如果子类没有写构造函数的话就一般默认添加构造。举个例子。

class Cat extends Animal{  
}  
*// 等同于*  
class Cat extends Animal{  
 constructor(name) {  
 super(name);  
 *//super作为函数调用时，代表父类的构造函数。*  
 }  
}

**注意**：如果我写了构造函数但是没有写super的话，或者super方法的参数不对等等，编译器都会报错。

2、原型继承

在ES6之前，也有很多继承的方法，其中一个很常用的方法就是使用原型继承。其基本方法就是一个父类的实例赋值给子类的原型。这个继承方式是通过\_\_proto\_\_建立和子类之间的原型链，当子类的实例需要使用父类的属性和方法的时候，可以通过\_\_proto\_\_一级级向上找；

function Animal(name) {  
 this.name = name;  
}  
  
Animal.prototype.eat= function () {  
 console.log(this.name + '正在吃东西')  
};  
function Cat(furColor){  
 this.furColor = furColor ;  
};  
Cat.prototype = new Animal();  
let tom = new Cat('black');  
console.log(tom)

**缺点**: 1. 子类实例时，无法向父类构造函数传参。2. 父类的私有属性被所有实例共享

**九、call、apply、bind之间的关系**

这又是一个面试经典问题, 也是ES5中众多坑中的一个，在 ES6 中可能会极大避免 this 产生的错误，但是为了一些老代码的维护，最好还是了解一下 this 的指向和 call、apply、bind 三者的区别.

bind,apply,call三者都可以用来改变this的指向, 下面分别对他们进行比较分析:

##### apply 和 call

* 二者都是Function对象的方法, 每个函数都能调用
* 二者的第一个参数都是你要指定的执行上下文
* apply 和 call 的区别是: call 方法接受的是若干个参数列表，而 apply 接收的是一个包含多个参数的数组。

var a ={  
 name : "Cherry",  
 fn : function (a,b) {  
 console.log( a + b)  
 }  
}  
var b = a.fn;  
b.apply(a,[1,2]) *// 3*  
b.call(a, 4,5,6) *// 15*

我们常常使用的验证是否是数组(前提是toString()方法没有被重写过):

function isArray(obj){  
 return Object.prototype.toString.call(obj) === '[object Array]' ;  
}

##### bind 与 apply、call区别

var a ={  
 name : "Cherry",  
 fn : function (a,b) {  
 console.log( a + b)  
 }  
 }  
 var b = a.fn;  
 b.bind(a,1,2)(); *// 3*

我们发现bind()方法还需要调用一次; 是由于 bind()方法创建一个新的函数,我们必须手动去调用。

##### bind,apply,call的共同和不同点：

* 三者都可以用来改变this的指向
* 三者第一个参数都是this要指向的对象，也就是想指定的上下文，上下文就是指调用函数的那个对象。（点前的那个对象，没有就是全局window）
* 三者都可以传参，但是apply是数组，而call是有顺序的传入
* bind 是返回对应函数，便于稍后调用；apply 、call 则是立即执行

1. **Promise**

前端面试过程中，基本都会问到 Promise，如果你足够幸运，面试官问的比较浅，仅仅问 Promise 的使用方式，那么恭喜你。事实上，大多数人并没有那么幸运, 很多面试官在promise这块都是由浅入深的提问.

了解Promise吗?

Promise 解决了什么问题？

Promise 如何使用？

Promise 常用的方法有哪些？它们的作用是什么？

Promise 在事件循环中的执行过程是怎样的？

##### 1. 了解Promise吗?

所谓Promise，简单说就是一个容器，里面保存着某个未来才会结束的事件的结果。从语法上说，Promise 是一个对象，从它可以获取异步操作的消息。Promise 提供统一的 API，各种异步操作都可以用同样的方法进行处理，让开发者不用再关注于时序和底层的结果。Promise的状态具有不受外界影响和不可逆两个特点。

###### 2.Promise 解决了什么问题?

Promise解决了回调地狱的问题, 提高代码的可读性以及解决信任度问题. 传统的回调有五大信任问题:

1. 调用回调过早
2. 调用回调过晚(或者没有被调用)
3. 调用回调次数过多或过少
4. 未能传递所需的环境和参数
5. 涂掉可能出现的错误和异常

###### 3. Promise 如何使用?

ES6规定，Promise对象是一个构造函数，用来生成Promise实例。下面代码创造了一个Promise实例。

var promise = new Promise(function(resolve, reject) {  
 *// ... some code*  
  
 if (*/\* 异步操作成功 \*/*){  
 resolve(value);  
 } else {  
 reject(error);  
 }  
});

Promise构造函数接受一个函数作为参数，该函数的两个参数分别是resolve和reject。它们是两个函数，由JavaScript引擎提供，不用自己部署。

resolve函数的作用是，将Promise对象的状态从“未完成”变为“成功”（即从Pending变为Resolved），在异步操作成功时调用，并将异步操作的结果，作为参数传递出去；reject函数的作用是，将Promise对象的状态从“未完成”变为“失败”（即从Pending变为Rejected），在异步操作失败时调用，并将异步操作报出的错误，作为参数传递出去。

##### 4、Promise 常用的方法有哪些？它们的作用是什么？

##### Promise.prototype.then

* Promise 实例具有then方法，也就是说，then方法是定义在原型对象Promise.prototype上的。它的作用是为 Promise 实例添加状态改变时的回调函数。then方法的第一个参数是resolved状态的回调函数，第二个参数（可选）是rejected状态的回调函数。
* Promise.prototype.catch

Promise.prototype.catch方法是.then(null, rejection)或.then(undefined, rejection)的别名，用于指定发生错误时的回调函数。

getJSON('/posts.json').then(function(posts) {  
 *// ...*  
}).catch(function(error) {  
 *// 处理 getJSON 和 前一个回调函数运行时发生的错误*  
 console.log('发生错误！', error);  
});

上面代码中，getJSON方法返回一个 Promise 对象，如果该对象状态变为resolved，则会调用then方法指定的回调函数；如果异步操作抛出错误，状态就会变为rejected，就会调用catch方法指定的回调函数，处理这个错误。

* Promise.all

Promise.all方法用于将多个 Promise 实例，包装成一个新的 Promise 实例, 返回最先执行结束的 Promise 任务的结果，不管这个 Promise 结果是成功还是失败。

* Promise.race

Promise.race方法同样是将多个 Promise 实例，包装成一个新的 Promise 实例。如果全部成功执行，则以数组的方式返回所有 Promise 任务的执行结果。如果有一个 Promise 任务 rejected，则只返回 rejected 任务的结果。

##### 5. Promise 在事件循环中的执行过程是怎样的

var promise = new Promise((resolve, reject)=>{  
 console.log('我是promise任务');  
 resolve('resolved')  
 })  
 promise.then(res =>{  
 console.log(res)  
 })  
 console.log("我是同步任务");  
 setTimeout(()=>{  
 console.log("我是延时任务");  
 }, 0)

上面代码的执行顺序是: 我是promise任务、我是同步任务、resolved、我是延时任务。

Promise 新建后立即执行，立即 resolve 的 Promise 对象，是在本轮“事件循环”（event loop）的结束时,而不是在下一轮“事件循环”的开始时;setTimeout在下一轮“事件循环”开始时执行。

**React 面试题**

以下是面试官最有可能问到的 **50 个 React 面试题和答案**。为方便你学习，我对它们进行了分类：

* 基本知识
* React 组件
* React Redux
* React 路由

## 基本知识

### ****1. 区分Real DOM和Virtual DOM****

| **Real DOM** | **Virtual  DOM** |
| --- | --- |
| 1. 更新缓慢。 | 1. 更新更快。 |
| 2. 可以直接更新 HTML。 | 2. 无法直接更新 HTML。 |
| 3. 如果元素更新，则创建新DOM。 | 3. 如果元素更新，则更新 JSX 。 |
| 4. DOM操作代价很高。 | 4. DOM 操作非常简单。 |
| 5.消耗太多的内存。 | 5. 很少的内存消耗。 |

### ****2. 什么是React？****

* React 是 Facebook 在 2011 年开发的前端 JavaScript 库。
* 它遵循基于组件的方法，有助于构建可重用的UI组件。
* 它用于开发复杂和交互式的 Web 和移动 UI。
* 尽管它仅在 2015 年开源，但有一个很大的支持社区。

### ****3. React有什么特点？****

1. React的主要功能如下：
2. 它使用**虚拟DOM**而不是真正的DOM。
3. 可以使用**服务器端渲染**。
4. 它遵循**单向数据流**或数据绑定。

### ****4. 列出React的一些主要优点。****

React的一些主要优点是：

1. 它提高了应用的性能
2. 可以方便地在客户端和服务器端使用
3. 由于 JSX，代码的可读性很好
4. React 很容易与 Meteor，Angular 等其他框架集成
5. 使用React，编写UI测试用例变得非常容易

### ****5. React有哪些限制？****

React的限制如下：

1. React 只是一个库，而不是一个完整的框架
2. 它的库非常庞大，需要时间来理解
3. 新手程序员可能很难理解
4. 编码变得复杂，因为它使用内联模板和 JSX

### ****6. 什么是JSX？****

JSX 是J avaScript XML 的简写。是 React 使用的一种文件，它利用 JavaScript 的表现力和类似 HTML 的模板语法。这使得 HTML 文件非常容易理解。此文件能使应用非常可靠，并能够提高其性能。下面是JSX的一个例子：

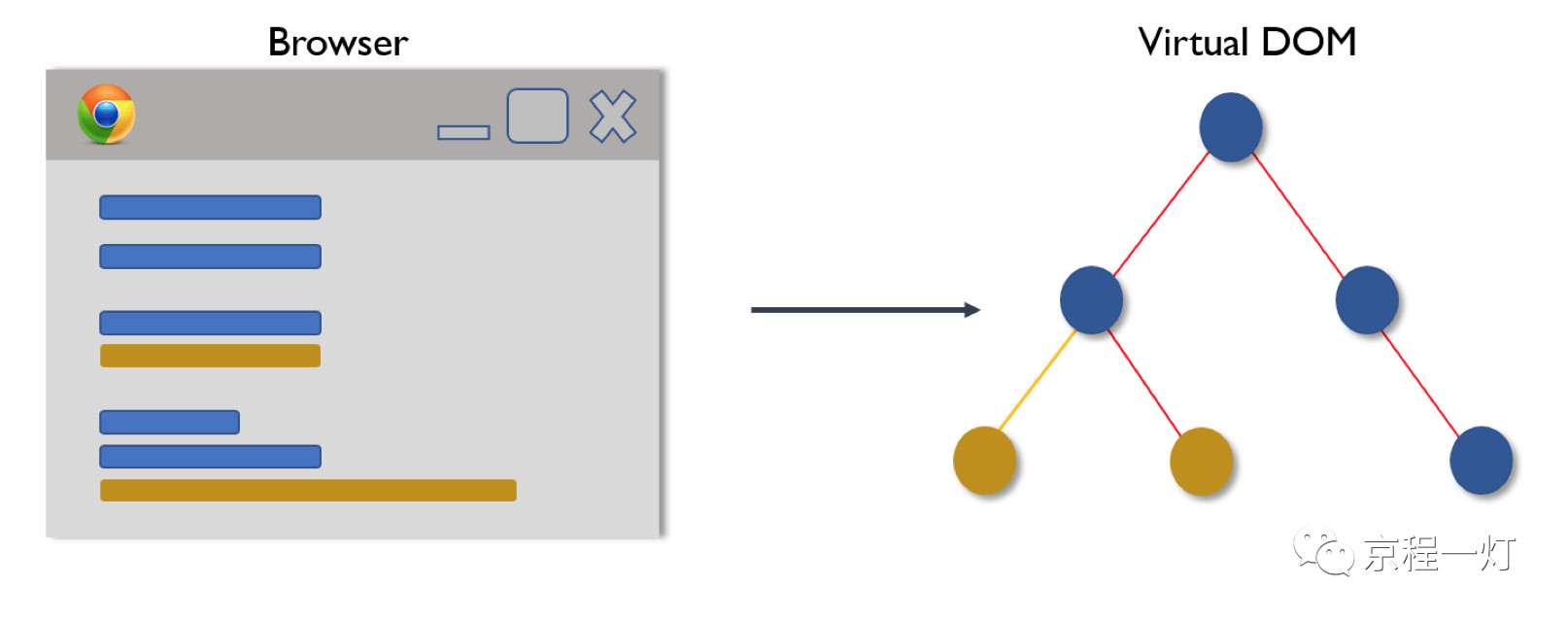
1render(){  
2    return(          
3        <div>  
4            <h1> Hello World from Edureka!!</h1>  
5        </div>  
6    );  
7}

### ****7. 你了解 Virtual DOM 吗？解释一下它的工作原理。****

Virtual DOM 是一个轻量级的 JavaScript 对象，它最初只是 real DOM 的副本。它是一个节点树，它将元素、它们的属性和内容作为对象及其属性。 React 的渲染函数从 React 组件中创建一个节点树。然后它响应数据模型中的变化来更新该树，该变化是由用户或系统完成的各种动作引起的。

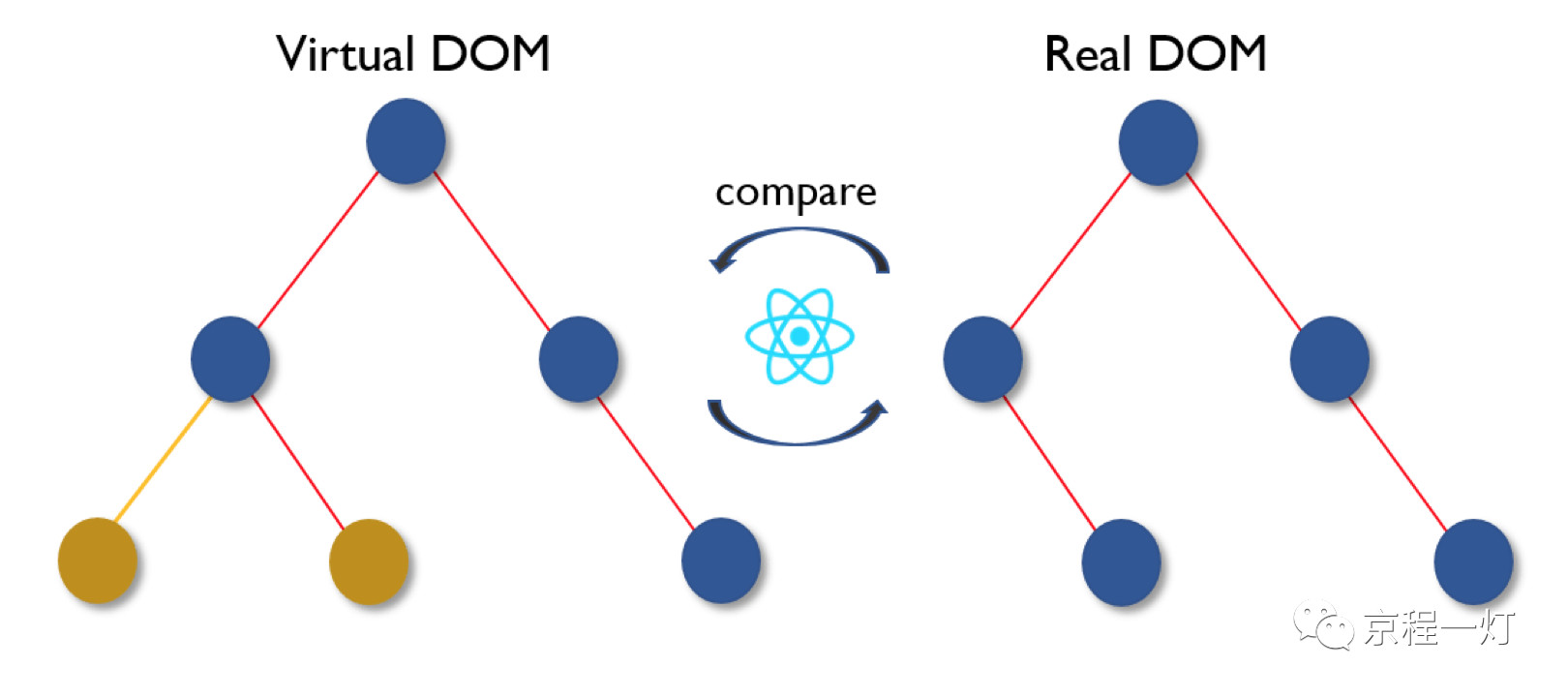
Virtual DOM 工作过程有三个简单的步骤。

1. **每当底层数据发生改变时，整个 UI 都将在 Virtual DOM 描述中重新渲染。**



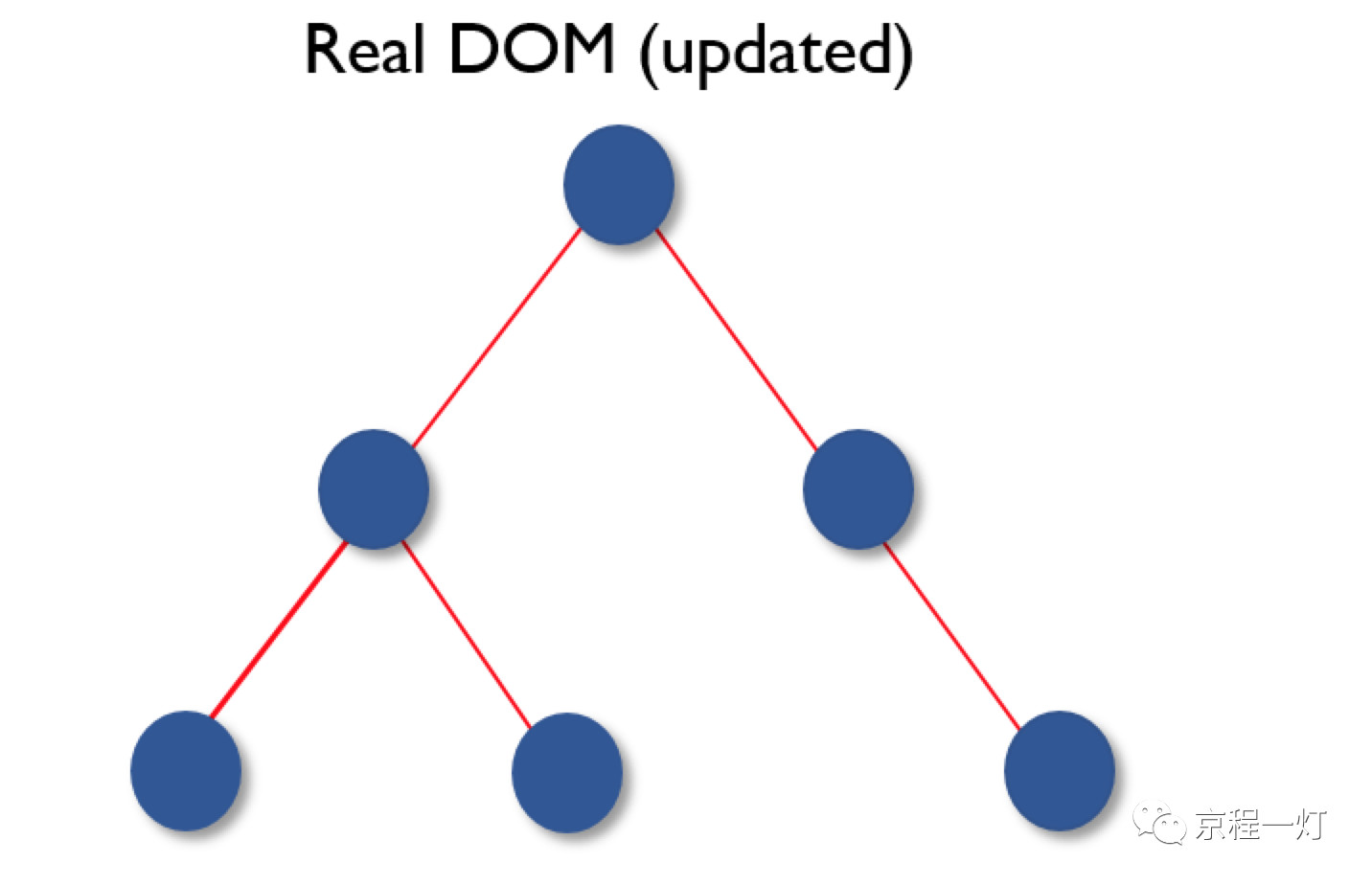
Virtual DOM 1

1. **然后计算之前 DOM 表示与新表示的之间的差异。**



Virtual DOM 2

1. **完成计算后，将只用实际更改的内容更新 real DOM。**



Virtual DOM 3

### ****8. 为什么浏览器无法读取JSX？****

浏览器只能处理 JavaScript 对象，而不能读取常规 JavaScript 对象中的 JSX。所以为了使浏览器能够读取 JSX，首先，需要用像 Babel 这样的 JSX 转换器将 JSX 文件转换为 JavaScript 对象，然后再将其传给浏览器。

### ****9. 与 ES5 相比，React 的 ES6 语法有何不同？****

以下语法是 ES5 与 ES6 中的区别：

**1.require 与 import**

1// ES5  
2var React = require('react');  
3  
4// ES6  
5import React from 'react';

**2.export 与 exports**

1// ES5  
2module.exports = Component;  
3  
4// ES6  
5export default Component;

**3.component 和 function**

1// ES5  
 2var MyComponent = React.createClass({  
 3    render: function() {  
 4        return  
 5            <h3>Hello Edureka!</h3>;  
 6    }  
 7});  
 8  
 9// ES6  
10class MyComponent extends React.Component {  
11    render() {  
12        return  
13            <h3>Hello Edureka!</h3>;  
14    }  
15}

**4.props**

1// ES5  
 2var App = React.createClass({  
 3    propTypes: { name: React.PropTypes.string },  
 4    render: function() {  
 5        return  
 6            <h3>Hello, {this.props.name}!</h3>;  
 7    }  
 8});  
 9  
10// ES6  
11class App extends React.Component {  
12    render() {  
13        return  
14            <h3>Hello, {this.props.name}!</h3>;  
15    }  
16}

**5.state**

1// ES5  
 2var App = React.createClass({  
 3    getInitialState: function() {  
 4        return { name: 'world' };  
 5    },  
 6    render: function() {  
 7        return  
 8            <h3>Hello, {this.state.name}!</h3>;  
 9    }  
10});  
11  
12// ES6  
13class App extends React.Component {  
14    constructor() {  
15        super();  
16        this.state = { name: 'world' };  
17    }  
18    render() {  
19        return  
20            <h3>Hello, {this.state.name}!</h3>;  
21    }  
22}

### ****10. React与Angular有何不同？****

| **主题** | **React** | **Angular** |
| --- | --- | --- |
| *1. 体系结构* | 只有 MVC 中的 View | 完整的 MVC |
| *2. 渲染* | 可以在服务器端渲染 | 客户端渲染 |
| *3. DOM* | 使用 virtual DOM | 使用 real DOM |
| *4. 数据绑定* | 单向数据绑定 | 双向数据绑定 |
| *5. 调试* | 编译时调试 | 运行时调试 |
| *6. 作者* | Facebook | Google |

## React 组件

### ****11. 你理解“在React中，一切都是组件”这句话。****

组件是 React 应用 UI 的构建块。这些组件将整个 UI 分成小的独立并可重用的部分。每个组件彼此独立，而不会影响 UI 的其余部分。

### ****12. 解释 React 中 render() 的目的。****

每个React组件强制要求必须有一个 **render()**。它返回一个 React 元素，是原生 DOM 组件的表示。如果需要渲染多个 HTML 元素，则必须将它们组合在一个封闭标记内，例如 <form>、<group>、<div> 等。此函数必须保持纯净，即必须每次调用时都返回相同的结果。

### ****13. 如何将两个或多个组件嵌入到一个组件中？****

可以通过以下方式将组件嵌入到一个组件中：

1class MyComponent extends React.Component{  
 2    render(){  
 3        return(            
 4            <div>  
 5                <h1>Hello</h1>  
 6                <Header/>  
 7            </div>  
 8        );  
 9    }  
10}  
11class Header extends React.Component{  
12    render(){  
13        return  
14            <h1>Header Component</h1>     
15   };  
16}  
17ReactDOM.render(  
18    <MyComponent/>, document.getElementById('content')  
19);

### ****14. 什么是 Props?****

Props 是 React 中属性的简写。它们是只读组件，必须保持纯，即不可变。它们总是在整个应用中从父组件传递到子组件。子组件永远不能将 prop 送回父组件。这有助于维护单向数据流，通常用于呈现动态生成的数据。

### ****15. React中的状态是什么？它是如何使用的？****

状态是 React 组件的核心，是数据的来源，必须尽可能简单。基本上状态是确定组件呈现和行为的对象。与props 不同，它们是可变的，并创建动态和交互式组件。可以通过this.state() 访问它们。

### ****16. 区分状态和 props****

| **条件** | **State** | **Props** |
| --- | --- | --- |
| 1. 从父组件中接收初始值 | Yes | Yes |
| 2. 父组件可以改变值 | No | Yes |
| 3. 在组件中设置默认值 | Yes | Yes |
| 4. 在组件的内部变化 | Yes | No |
| 5. 设置子组件的初始值 | Yes | Yes |
| 6. 在子组件的内部更改 | No | Yes |

### ****17. 如何更新组件的状态？****

可以用 this.setState()更新组件的状态。

1class MyComponent extends React.Component {  
 2    constructor() {  
 3        super();  
 4        this.state = {  
 5            name: 'Maxx',  
 6            id: '101'  
 7        }  
 8    }  
 9    render()  
10        {  
11            setTimeout(()=>{this.setState({name:'Jaeha', id:'222'})},2000)  
12            return (                
13                <div>  
14                    <h1>Hello {this.state.name}</h1>  
15                    <h2>Your Id is {this.state.id}</h2>  
16                </div>  
17            );  
18        }  
19    }  
20ReactDOM.render(  
21    <MyComponent/>, document.getElementById('content')  
22);

### ****18. React 中的箭头函数是什么？怎么用？****

箭头函数（**=>**）是用于编写函数表达式的简短语法。这些函数允许正确绑定组件的上下文，因为在 ES6 中默认下不能使用自动绑定。使用高阶函数时，箭头函数非常有用。

1//General way  
 2render() {      
 3    return(  
 4        <MyInput onChange = {this.handleChange.bind(this) } />  
 5    );  
 6}  
 7//With Arrow Function  
 8render() {    
 9    return(  
10        <MyInput onChange = { (e)=>this.handleOnChange(e) } />  
11    );  
12}

### ****19. 区分有状态和无状态组件。****

| **有状态组件** | **无状态组件** |
| --- | --- |
| 1. 在内存中存储有关组件状态变化的信息 | 1. 计算组件的内部的状态 |
| 2. 有权改变状态 | 2. 无权改变状态 |
| 3. 包含过去、现在和未来可能的状态变化情况 | 3. 不包含过去，现在和未来可能发生的状态变化情况 |
| 4. 接受无状态组件状态变化要求的通知，然后将 props 发送给他们。 | 4.从有状态组件接收 props 并将其视为回调函数。 |

### ****20.  React组件生命周期的阶段是什么？****

React 组件的生命周期有三个不同的阶段：

1. *初始渲染阶段：*这是组件即将开始其生命之旅并进入 DOM 的阶段。
2. *更新阶段：*一旦组件被添加到 DOM，它只有在 prop 或状态发生变化时才可能更新和重新渲染。这些只发生在这个阶段。
3. *卸载阶段：*这是组件生命周期的最后阶段，组件被销毁并从 DOM 中删除。

### ****21. 详细解释 React 组件的生命周期方法。****

一些最重要的生命周期方法是：

1. *componentWillMount***()** – 在渲染之前执行，在客户端和服务器端都会执行。
2. *componentDidMount***()** – 仅在第一次渲染后在客户端执行。
3. *componentWillReceiveProps***()** – 当从父类接收到 props 并且在调用另一个渲染器之前调用。
4. *shouldComponentUpdate***()** – 根据特定条件返回 true 或 false。如果你希望更新组件，请返回**true** 否则返回 **false**。默认情况下，它返回 false。
5. *componentWillUpdate***()** – 在 DOM 中进行渲染之前调用。
6. *componentDidUpdate***()** – 在渲染发生后立即调用。
7. *componentWillUnmount***()** – 从 DOM 卸载组件后调用。用于清理内存空间。

### ****22. React中的事件是什么？****

在 React 中，事件是对鼠标悬停、鼠标单击、按键等特定操作的触发反应。处理这些事件类似于处理 DOM 元素中的事件。但是有一些语法差异，如：

1. 用驼峰命名法对事件命名而不是仅使用小写字母。
2. 事件作为函数而不是字符串传递。

事件参数重包含一组特定于事件的属性。每个事件类型都包含自己的属性和行为，只能通过其事件处理程序访问。

### ****23. 如何在React中创建一个事件？****

1class Display extends React.Component({      
 2    show(evt) {  
 3        // code     
 4    },     
 5    render() {        
 6        // Render the div with an onClick prop (value is a function)          
 7        return (              
 8            <div onClick={this.show}>Click Me!</div>  
 9        );      
10    }  
11});

### ****24. React中的合成事件是什么？****

合成事件是围绕浏览器原生事件充当跨浏览器包装器的对象。它们将不同浏览器的行为合并为一个 API。这样做是为了确保事件在不同浏览器中显示一致的属性。

### ****25. 你对 React 的 refs 有什么了解？****

Refs 是 React 中引用的简写。它是一个有助于存储对特定的 React 元素或组件的引用的属性，它将由组件渲染配置函数返回。用于对 render() 返回的特定元素或组件的引用。当需要进行 DOM 测量或向组件添加方法时，它们会派上用场。

1class ReferenceDemo extends React.Component{  
 2     display() {  
 3         const name = this.inputDemo.value;  
 4         document.getElementById('disp').innerHTML = name;  
 5     }  
 6render() {  
 7    return(          
 8          <div>  
 9            Name: <input type="text" ref={input => this.inputDemo = input} />  
10            <button name="Click" onClick={this.display}>Click</button>              
11            <h2>Hello <span id="disp"></span> !!!</h2>  
12          </div>  
13    );  
14   }  
15 }

### ****26. 列出一些应该使用 Refs 的情况。****

以下是应该使用 refs 的情况：

* 需要管理焦点、选择文本或媒体播放时
* 触发式动画
* 与第三方 DOM 库集成

### ****27. 你如何模块化 React 中的代码？****

可以使用 export 和 import 属性来模块化代码。它们有助于在不同的文件中单独编写组件。

1//ChildComponent.jsx  
 2export default class ChildComponent extends React.Component {  
 3    render() {  
 4        return(             
 5              <div>  
 6                  <h1>This is a child component</h1>  
 7              </div>  
 8        );  
 9    }  
10}  
11  
12//ParentComponent.jsx  
13import ChildComponent from './childcomponent.js';  
14class ParentComponent extends React.Component {      
15    render() {          
16        return(             
17             <div>                 
18                <App />            
19             </div>         
20        );    
21    }  
22}

### ****28. 如何在 React 中创建表单****

React 表单类似于 HTML 表单。但是在 React 中，状态包含在组件的 state 属性中，并且只能通过 setState() 更新。因此元素不能直接更新它们的状态，它们的提交是由 JavaScript 函数处理的。此函数可以完全访问用户输入到表单的数据。

1handleSubmit(event) {  
 2    alert('A name was submitted: ' + this.state.value);  
 3    event.preventDefault();  
 4}  
 5  
 6render() {  
 7    return (          
 8        <form onSubmit={this.handleSubmit}>  
 9            <label>  
10                Name:  
11                <input type="text" value={this.state.value} onChange={this.handleSubmit} />  
12            </label>  
13            <input type="submit" value="Submit" />  
14        </form>  
15    );  
16}

### ****29. 你对受控组件和非受控组件了解多少？****

| **受控组件** | **非受控组件** |
| --- | --- |
| 1. 没有维持自己的状态 | 1. 保持着自己的状态 |
| 2.数据由父组件控制 | 2.数据由 DOM 控制 |
| 3. 通过 props 获取当前值，然后通过回调通知更改 | 3. Refs 用于获取其当前值 |

### ****30.  什么是高阶组件（HOC）？****

高阶组件是重用组件逻辑的高级方法，是一种源于 React 的组件模式。 HOC 是自定义组件，在它之内包含另一个组件。它们可以接受子组件提供的任何动态，但不会修改或复制其输入组件中的任何行为。你可以认为 HOC 是“纯（Pure）”组件。

### ****31. 你能用HOC做什么？****

HOC可用于许多任务，例如：

* 代码重用，逻辑和引导抽象
* 渲染劫持
* 状态抽象和控制
* Props 控制

### ****32. 什么是纯组件？****

*纯（Pure）* 组件是可以编写的最简单、最快的组件。它们可以替换任何只有 **render()**的组件。这些组件增强了代码的简单性和应用的性能。

### ****33. React 中 key 的重要性是什么？****

key 用于识别唯一的 Virtual DOM 元素及其驱动 UI 的相应数据。它们通过回收 DOM 中当前所有的元素来帮助 React 优化渲染。这些 key 必须是唯一的数字或字符串，React 只是重新排序元素而不是重新渲染它们。这可以提高应用程序的性能。

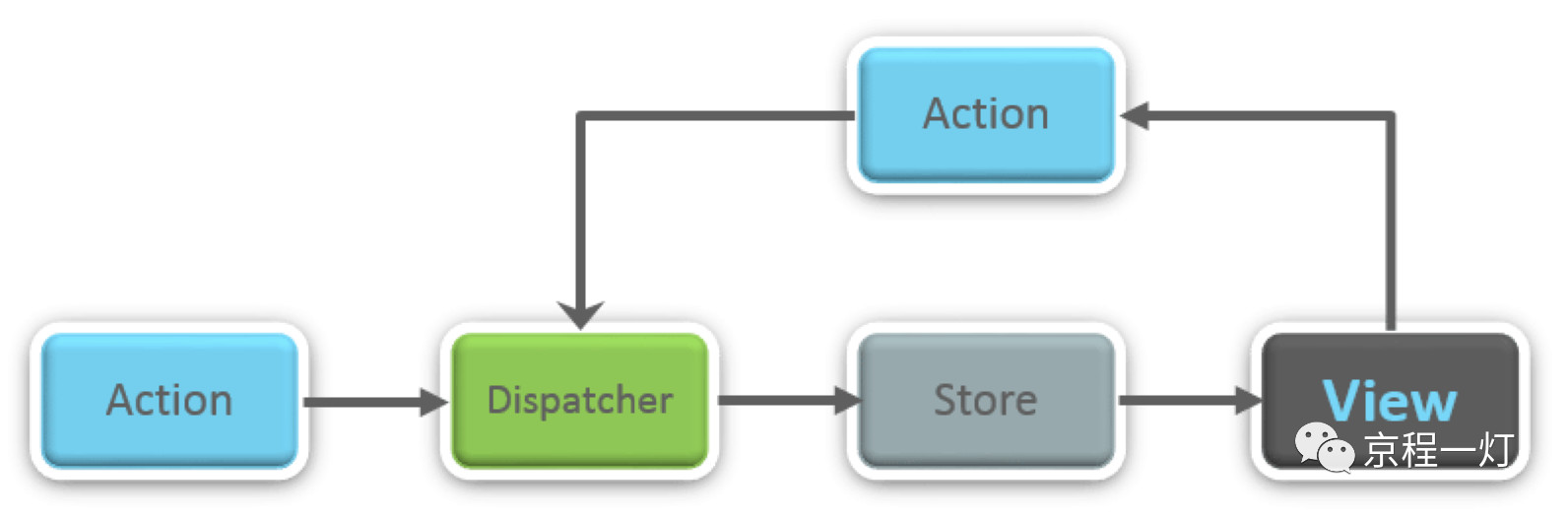
## React Redux

### ****34. MVC框架的主要问题是什么？****

以下是MVC框架的一些主要问题：

* 对 DOM 操作的代价非常高
* 程序运行缓慢且效率低下
* 内存浪费严重
* 由于循环依赖性，组件模型需要围绕 models 和 views 进行创建

### ****35. 解释一下 Flux****

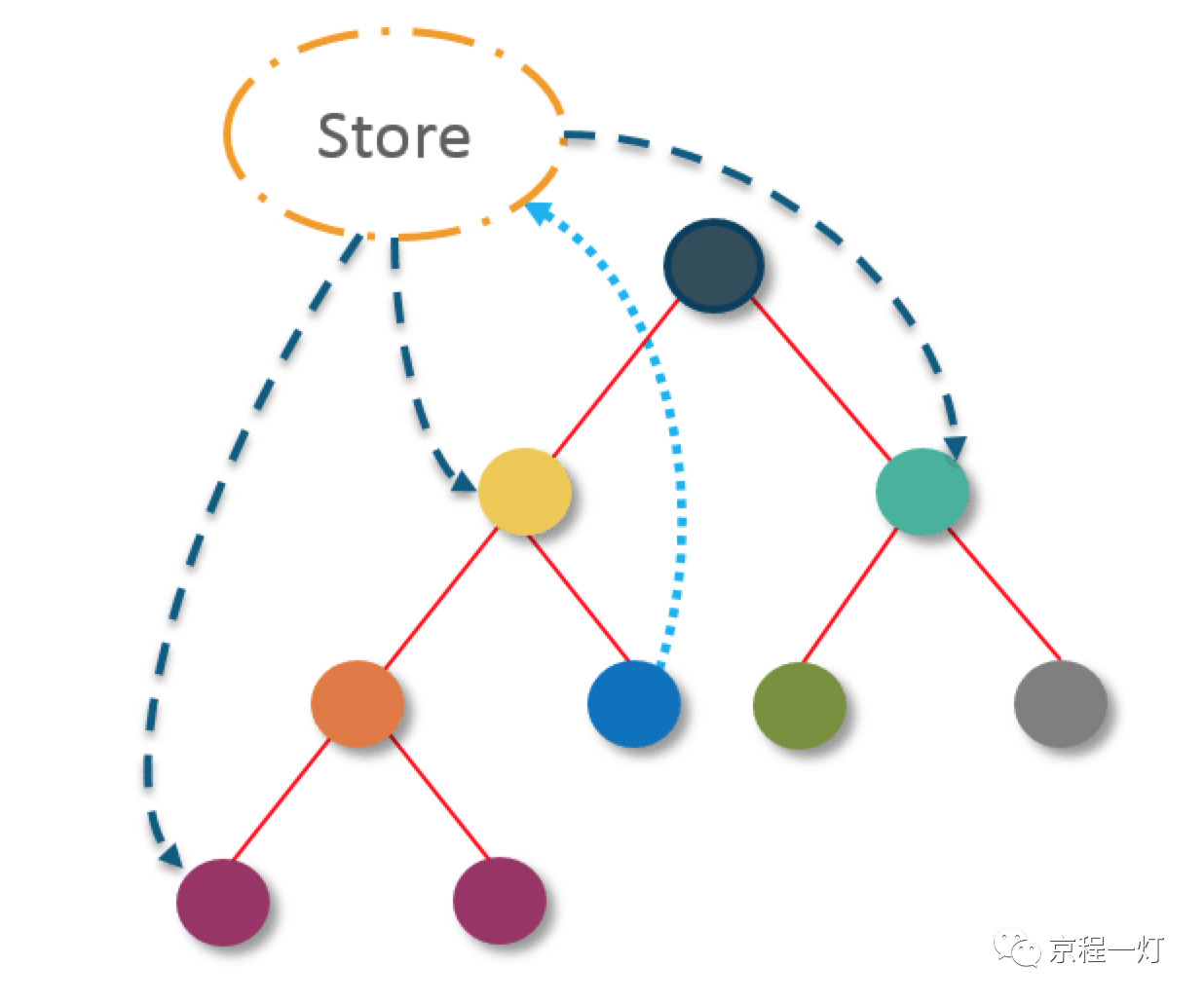
flux  
Flux 是一种强制单向数据流的架构模式。它控制派生数据，并使用具有所有数据权限的中心 store 实现多个组件之间的通信。整个应用中的数据更新必须只能在此处进行。 Flux 为应用提供稳定性并减少运行时错误。

### ****36. 什么是Redux？****

Redux 是当今最热门的前端开发库之一。它是 JavaScript 程序的可预测状态容器，用于整个应用的状态管理。使用 Redux 开发的应用易于测试，可以在不同环境中运行，并显示一致的行为。

### ****37. Redux遵循的三个原则是什么？****

1. *单一事实来源：*整个应用的状态存储在单个 store 中的对象/状态树里。单一状态树可以更容易地跟踪随时间的变化，并调试或检查应用程序。
2. *状态是只读的：*改变状态的唯一方法是去触发一个动作。动作是描述变化的普通 JS 对象。就像 state 是数据的最小表示一样，该操作是对数据更改的最小表示。
3. *使用纯函数进行更改：*为了指定状态树如何通过操作进行转换，你需要纯函数。纯函数是那些返回值仅取决于其参数值的函数。



Store

### ****38. 你对“单一事实来源”有什么理解？****

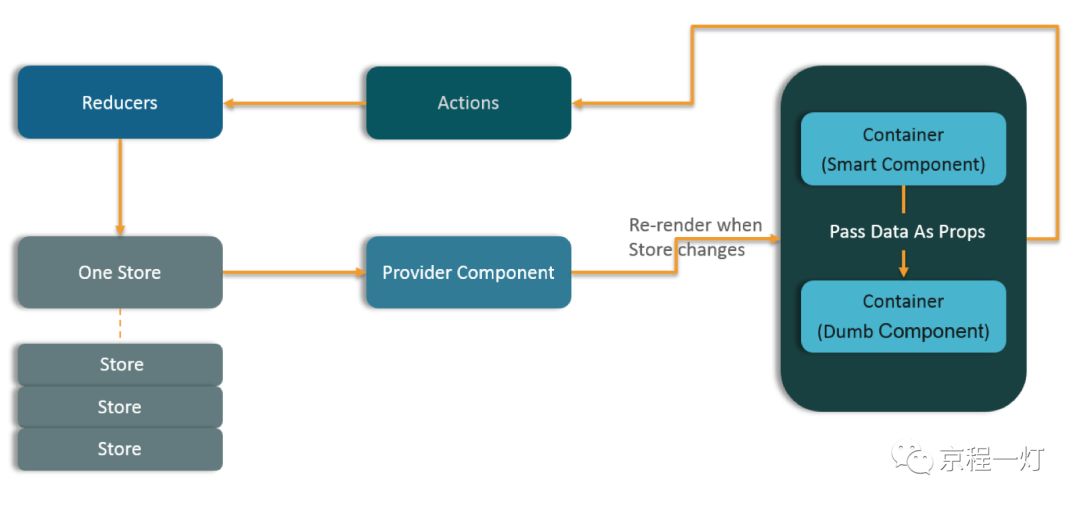
Redux 使用 “Store” 将程序的整个状态存储在同一个地方。因此所有组件的状态都存储在 Store 中，并且它们从 Store 本身接收更新。单一状态树可以更容易地跟踪随时间的变化，并调试或检查程序。

### ****39. 列出 Redux 的组件。****

Redux 由以下组件组成：

1. **Action** – 这是一个用来描述发生了什么事情的对象。
2. **Reducer** –  这是一个确定状态将如何变化的地方。
3. **Store** – 整个程序的状态/对象树保存在Store中。
4. **View** – 只显示 Store 提供的数据。

### ****40. 数据如何通过 Redux 流动？****



Data Flow in Redux

### ****41. 如何在 Redux 中定义 Action？****

React 中的 Action 必须具有 type 属性，该属性指示正在执行的 ACTION 的类型。必须将它们定义为字符串常量，并且还可以向其添加更多的属性。在 Redux 中，action 被名为 Action Creators 的函数所创建。以下是 Action 和Action Creator 的示例：

1function addTodo(text) {  
2       return {  
3                type: ADD\_TODO,      
4                 text  
5    }  
6}

### ****42. 解释 Reducer 的作用。****

Reducers 是纯函数，它规定应用程序的状态怎样因响应 ACTION 而改变。Reducers 通过接受先前的状态和 action 来工作，然后它返回一个新的状态。它根据操作的类型确定需要执行哪种更新，然后返回新的值。如果不需要完成任务，它会返回原来的状态。

### ****43. Store 在 Redux 中的意义是什么？****

Store 是一个 JavaScript 对象，它可以保存程序的状态，并提供一些方法来访问状态、调度操作和注册侦听器。应用程序的整个状态/对象树保存在单一存储中。因此，Redux 非常简单且是可预测的。我们可以将中间件传递到 store 来处理数据，并记录改变存储状态的各种操作。所有操作都通过 reducer 返回一个新状态。

### ****44.  Redux与Flux有何不同？****

| **Flux** | **Redux** |
| --- | --- |
| 1. Store 包含状态和更改逻辑 | 1. Store 和更改逻辑是分开的 |
| 2. 有多个 Store | 2. 只有一个 Store |
| 3. 所有 Store 都互不影响且是平级的 | 3. 带有分层 reducer 的单一 Store |
| 4. 有单一调度器 | 4. 没有调度器的概念 |
| 5. React 组件订阅 store | 5. 容器组件是有联系的 |
| 6. 状态是可变的 | 6. 状态是不可改变的 |

### ****45. Redux 有哪些优点？****

Redux 的优点如下：

**结果的可预测性 -**  由于总是存在一个真实来源，即 store ，因此不存在如何将当前状态与动作和应用的其他部分同步的问题。

**可维护性 -**  代码变得更容易维护，具有可预测的结果和严格的结构。

**服务器端渲染 -**  你只需将服务器上创建的 store 传到客户端即可。这对初始渲染非常有用，并且可以优化应用性能，从而提供更好的用户体验。

**开发人员工具 -**  从操作到状态更改，开发人员可以实时跟踪应用中发生的所有事情。

**社区和生态系统 -**  Redux 背后有一个巨大的社区，这使得它更加迷人。一个由才华横溢的人组成的大型社区为库的改进做出了贡献，并开发了各种应用。

**易于测试 -**  Redux 的代码主要是小巧、纯粹和独立的功能。这使代码可测试且独立。

**组织 -**  Redux 准确地说明了代码的组织方式，这使得代码在团队使用时更加一致和简单。

## React 路由

### ****46. 什么是React 路由？****

React 路由是一个构建在 React 之上的强大的路由库，它有助于向应用程序添加新的屏幕和流。这使 URL 与网页上显示的数据保持同步。它负责维护标准化的结构和行为，并用于开发单页 Web 应用。 React 路由有一个简单的API。

### ****47. 为什么React Router v4中使用 switch 关键字 ？****

虽然 <div> 用于封装 Router 中的多个路由，当你想要仅显示要在多个定义的路线中呈现的单个路线时，可以使用 “switch” 关键字。使用时，<switch>  标记会按顺序将已定义的 URL 与已定义的路由进行匹配。找到第一个匹配项后，它将渲染指定的路径。从而绕过其它路线。

### ****48. 为什么需要 React 中的路由？****

Router 用于定义多个路由，当用户定义特定的 URL 时，如果此 URL 与 Router 内定义的任何 “路由” 的路径匹配，则用户将重定向到该特定路由。所以基本上我们需要在自己的应用中添加一个 Router 库，允许创建多个路由，每个路由都会向我们提供一个独特的视图

1<switch>  
2    <route exact path=’/’ component={Home}/>  
3    <route path=’/posts/:id’ component={Newpost}/>  
4    <route path=’/posts’   component={Post}/>  
5</switch>

### ****49. 列出 React Router 的优点。****

几个优点是：

1. 就像 React 基于组件一样，在 React Router v4 中，API 是 *'All About Components'*。可以将 Router 可视化为单个根组件（<BrowserRouter>），其中我们将特定的子路由（ <route> ）包起来。
2. 无需手动设置历史值：在 React Router v4 中，我们要做的就是将路由包装在 < BrowserRouter> 组件中。
3. 包是分开的：共有三个包，分别用于 Web、Native 和 Core。这使我们应用更加紧凑。基于类似的编码风格很容易进行切换。

### ****50. React Router与常规路由有何不同？****

| **主题** | **常规路由** | **React 路由** |
| --- | --- | --- |
| **参与的页面** | 每个视图对应一个新文件 | 只涉及单个HTML页面 |
| **URL 更改** | HTTP 请求被发送到服务器并且接收相应的 HTML 页面 | 仅更改历史记录属性 |
| **体验** | 用户实际在每个视图的不同页面切换 | 用户认为自己正在不同的页面间切换 |