目录

HTML/CSS 部分

标签属性中的 title 和 alt 的区别

title 是设置鼠标移动到图片上时显示的内容，而 alt 是用于当图片没有正常显示时出现的提示文字，另外 alt 还用于在 seo 中针对图片的优化说明

隐藏元素的几种方法

display:none

visibility:hidden

opacity:0　　将透明度设置为 0

position:absolute　　left:-10000px

浏览器页面有哪三层构成，分别是什么，作用是什么？

浏览器页面构成：结构层、表示层、行为层

分别是：HTML、CSS、JavaScript

作用：HTML 实现页面结构，CSS 完成页面的表现与风格，JavaScript 实现一些客户端的功

能与业务。

html5 有哪些新特性、移除了那些元素？如何处理 HTML5 新标签的浏览器兼容问题？如何区分 HTML 和 HTML5？

HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

- 绘画 canvas;
- 用于媒介回放的 video 和 audio 元素;
- 本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失;

  sessionStorage 的数据在浏览器关闭后自动删除;
- 语意化更好的内容元素，比如 article、footer、header、nav、section;
- 表单控件，calendar、date、time、email、url、search;
- 新的技术 webworker, websocket, Geolocation;

移除的元素：

纯表现的元素：basefont，big，center，font, s，strike，tt，u;

对可用性产生负面影响的元素：frame，frameset，noframes；

支持 HTML5 新标签：

IE8/IE7/IE6 支持通过 document.createElement 方法产生的标签，

可以利用这一特性让这些浏览器支持 HTML5 新标签，

浏览器支持新标签后，还需要添加标签默认的样式。

当然也可以直接使用成熟的框架、比如 html5shim;

```
<!--[if lt IE 9]>
    <script> src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>
<![endif]-->
```

如何区分 HTML5： DOCTYPE 声明\新增的结构元素\功能元素

什么是语义化的 HTML?

直观的认识标签，对于搜索引擎的抓取有好处，用正确的标签做正确的事情！

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，在没有样式 CCS 情况下也以一种文档格式显示，并且是容易阅读的。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。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。

HTML5 的离线储存怎么使用，工作原理能不能解释一下？

在用户没有与因特网连接时，可以正常访问站点或应用，在用户与因特网连接时，更新用户机器上的缓存文件。

原理：HTML5 的离线存储是基于一个新建的.appcache 文件的缓存机制(不是存储技术)，通过这个文件上的解析清单离线存储资源，这些资源就会像 cookie 一样被存储了下来。之后当网络在处于离线状态下时，浏览器会通过被离线存储的数据进行页面展示。

CSS 块级元素、内联元素

- **块级元素(block)特性：**
  总是独占一行，表现为另起一行开始，而且其后的元素也必须另起一行显示;
  宽度(width)、高度(height)、内边距(padding)和外边距(margin)都可控制;

- **内联元素(inline)特性：**
  和相邻的内联元素在同一行;
  宽度(width)、高度(height)、内边距的 top/bottom(padding-top/padding-bottom)和外边距的 top/bottom(margin-top/margin-bottom)都不可改变，就是里面文字或图片的大小;

- **块级元素主要有：**
  address , blockquote , center , dir , div , dl , fieldset , form , h1 , h2 , h3 , h4 , h5 , h6 , hr , isindex , menu , noframes , noscript , ol , p , pre , table , ul , li

- **内联元素主要有：**
  a , abbr , acronym , b , bdo , big , br , cite , code , dfn , em , font , i , img , input , kbd , label , q , s , samp , select , small , span , strike , strong , sub , sup ,textarea , tt , u , var

- **可变元素(根据上下文关系确定该元素是块元素还是内联元素)：**
  applet ,button ,del ,iframe , ins ,map ,object , script

- **CSS 中块级、内联元素的应用：**
  利用 CSS 我们可以摆脱上面表格里 HTML 标签归类的限制，自由地在不同标签/元素上应用我们需要的属性。

- **主要用的 CSS 样式有以下三个：**
  display:block -- 显示为块级元素
  display:inline -- 显示为内联元素
  display:inline-block -- 显示为内联块元素，表现为同行显示并可修改宽高内外边距等属性
  我们常将所有<li>元素加上 display:inline-block 样式，原本垂直的列表就可以水平显示了。

CSS 居中（包括水平居中和垂直居中）

水平居中设置:

1.行内元素

设置 `text-align:center;`

2.Flex布局

设置 `display:flex;justify-content:center;` (灵活运用,支持Chroime，Firefox，IE9+)

垂直居中设置:

1.父元素高度确定的单行文本（内联元素）

设置 `height = line-height;`

2.父元素高度确定的多行文本（内联元素）

a:插入 table （插入方法和水平居中一样），然后设置 `vertical-align:middle;`

b:先设置 `display:table-cell` 再设置 `vertical-align:middle;`

块级元素居中方案

水平居中设置:

1.定宽块状元素

设置 左右 margin 值为 auto;

2.不定宽块状元素

a:在元素外加入 table 标签（完整的，包括 table、tbody、tr、td），该元素写在 td 内，然后设置 margin 的值为 auto;

b:给该元素设置 `displa:inine` 方法;

c:父元素设置 `position:relative` 和 left:50%，子元素设置 `position:relative` 和 left:50%;

垂直居中设置:

使用 `position:absolute（fixed）` ,设置left、top、margin-left、margin-top的属性;

利用 `position:fixed（absolute）` 属性， `margin:auto` 这个必须不要忘记了;

利用 `display:table-cell` 属性使内容垂直居中;

使用css3的新属性 `transform:translate(x,y)` 属性;

使用:before元素;

px，em，rem 的区别？

- px 像素(Pixel) 绝对单位。像素 px 是相对于显示器屏幕分辨率而言的，是一个虚拟长度单位，是计算机系统的数字化图像长度单位，如果 px 要换算成物理长度，需要指定精度 DPI。
- em 是相对长度单位，相对于当前对象内文本的字体尺寸。如当前对行内文本的字体尺寸未被人为设置，则相对于浏览器的默认字体尺寸。它会继承父级元素的字体大小，因此并不是一个固定值。 16 px = 1 em
- rem 是 CSS3 新增的一个相对单位(root em, 根 em)，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。

区别：

IE 无法调整那些使用 px 作为单位的字体大小，而 em 和 rem 可以缩放，rem 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一身，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符合的连锁反应。目前，除了 IE8 及更早版本外，所有浏览器均已支持 rem

清除浮动的几种方式？

--清除浮动是为了清除使用浮动元素产生的影响。浮动元素，高度会塌陷，从而影响页面布局。

--用一个清除浮动的块元素作为子元素撑起父元素，可以解决高度塌陷问题。

清除浮动的具体方法：

- 给设置了浮动元素的父级 div 定义 height

  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。简单、代码少、容易掌握，但只适合高度固定的布局
- 给需要清除浮动的元素结尾处加空 div 标签 clear: both

  原理：在浮动元素的后面添加一个空 div 兄弟元素，利用 CSS 提高的 clear: both 清除浮动，让父级 div 能自动获取到高度，如果页面浮动布局多，就需要增加很多空 div，让

人感觉很不好。

- 给设置了浮动元素的父级设置 overflow:hidden。如果需要兼容 IE，在添加一个 zoom:1
- 父级添加伪类元素清除.clearfix{}

zoom:1 的清除浮动原理?

清除浮动，触发 hasLayout；

Zoom 属性是 IE 浏览器的专有属性，它可以设置或检索对象的缩放比例。解决 ie 下比较奇葩的 bug。

譬如外边距（margin）的重叠，浮动清除，触发 ie 的 haslayout 属性等。

来龙去脉大概如下：

当设置了 zoom 的值之后，所设置的元素就会就会扩大或者缩小，高度宽度就会重新计算了，这里一旦改变 zoom 值时其实也会发生重新渲染，运用这个原理，也就解决了 ie 下子元素浮动时候父元素不随着自动扩大的问题。

Zoom 属是 IE 浏览器的专有属性，火狐和老版本的 webkit 核心的浏览器都不支持这个属性。

然而，zoom 现在已经被逐步标准化，出现在 CSS 3.0 规范草案中。

目前非 ie 由于不支持这个属性，它们又是通过什么属性来实现元素的缩放呢？

可以通过 css3 里面的动画属性 scale 进行缩放。

Display: none 与 visibility: hidden 的区别是什么？

Display: none，使用该属性后，HTML 元素(对象)的宽度、高度等各种属性值都将“丢失”；

Visibility: hidden，使用该属性后，HTML 元素(对象)仅仅是在视觉上看不见(完全透明)，而它所占据的空间位置依然存在，也即是说它仍然具有高度、宽度等属性值。

CSS Display(显示) 与 Visibility（可见性）

隐藏元素 - display:none 或 visibility:hidden

隐藏一个元素可以通过把 display 属性设置为"none"，或把 visibility 属性设置为"hidden"。但是请注意，这两种方法会产生不同的结果。

visibility:hidden 可以隐藏某个元素，但隐藏的元素仍需占用与未隐藏之前一样的空间。也就是说，该元素虽然被隐藏了，但仍然会影响布局。

display:none 可以隐藏某个元素，且隐藏的元素不会占用任何空间。也就是说，该元素不但被隐藏了，而且该元素原本占用的空间也会从页面布局中消失。

页面导入样式时，使用 link 和@import 有什么区别？

- link 属于 XHTML 标签，除了加载 CSS 外，还能用于定义 RSS, 定义 rel 连接属性等作用；而@import 是 CSS 提供的，只能用于加载 CSS;
- 页面被加载的时，link 会同时被加载，而@import 引用的 CSS 会等到页面被加载完再加载;
- import 是 CSS2.1 提出的，只在 IE5 以上才能被识别，而 link 是 XHTML 标签，无兼容问题;
- link 支持使用 js 控制 DOM 去改变样式，而@import 不支持

iframe 缺点

- iframe 会阻塞主页面的 Onload 事件；
- 搜索引擎的检索程序无法解读这种页面，不利于 SEO;
- iframe 和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载。
- 使用 iframe 之前需要考虑这两个缺点。如果需要使用 iframe，最好是通过 javascript 动态给 iframe 添加 src 属性值，这样可以绕开以上两个问题。

Label 的作用是什么？是怎么用的？

label 标签来定义表单控制间的关系,当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上。

<label for="Name">Number:</label>

<input type=“text“ name="Name" id="Name"/>

<label>Date:<input type="text" name="B"/></label>

display 有哪些值？说明他们的作用。（海康）

block 块类型。默认宽度为父元素宽度，可设置宽高，换行显示。

none 元素不显示，并从文档流中移除。

inline 行内元素类型。默认宽度为内容宽度，不可设置宽高，同行显示

inline-block 像块元素一样显示，并添加样式列表标记

table 此元素会作为块级表格来显示

inherit 规定应该从父元素继承 display 属性的值。

介绍一下你对浏览器内核的理解？

主要分成两部分：渲染引擎(layout engineer 或 Rendering Engine)和 JS 引擎。

渲染引擎：负责取得网页的内容（HTML、XML、图像等等）、整理讯息（例如加入 CSS 等），以及计算网页的显示方式，然后会输出至显示器或打印机。浏览器的内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同。所有网页浏览器、电子邮件客户端以及其它需要编辑、显示网络内容的应用程序都需要内核。

JS 引擎则：解析和执行 javascript 来实现网页的动态效果。

最开始渲染引擎和 JS 引擎并没有区分的很明确，后来 JS 引擎越来越独立，内核就倾向于只指渲染引擎。

常见的浏览器内核有哪些？

Trident 内核：IE,MaxThon,TT,The World,360,搜狗浏览器等。[又称 MSHTML]
Gecko 内核：Netscape6 及以上版本，FF,MozillaSuite/SeaMonkey 等
Presto 内核：Opera7 及以上。 [Opera 内核原为：Presto，现为：Blink;]
Webkit 内核：Safari,Chrome 等。 [Chrome 的：Blink（WebKit 的分支）]

JS 部分

JavaScript 中数据类型(海康)

值类型（基本类型）：字符串(String)、数字(Number)、布尔(Boolean)、对空(Null)、未定义(Undefined)、Symbol

引用数据类型：对象(Object)、数组(Array)、函数(Function)。

注：Symbol 是 ES6 引入了一种新的原始数据类型，表示独一无二的值。

Object，Object 本质上是由一组无序的名值对组成的

介绍 js 有哪些内置对象？

Object 是 JavaScript 中所有对象的父对象

数据封装类对象：Object、Array、Boolean、Number 和 String

其他对象：Function、Arguments、Math、Date、RegExp、Error

写 JavaScript 的基本规范？

1.不要在同一行声明多个变量。

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。

作用域和闭包？

简单的说，**作用域**是针对变量的，比如我们创建一个函数 a1，函数里面又包了一个子函数 a2。此时就存在三个作用域：

全局作用域----a1 作用域-----a2 作用域：即全局作用域包含了 a1 的作用域，a2 的作用域包含了 a1 的作用域。

当 a1 在查找变量时会先从自身的作用域去查找，找不到再到上一级 a2 的作用域查找，如果还没找到就到全局作用域去查找，这样就形成了一个作用域链。

理解**闭包（closure）**首先要理解，js 垃圾回收机制，也即当一个函数被执行完后，其作用域会被收回，如果形成了闭包，执行完后其作用域就不会被收回。

如果某个函数被他的父函数之外的一个变量引用，就会形成闭包。

闭包的作用，就是保存自己私有的变量，通过提供的接口（方法）给外部使用，但外部不能直接访问该变量。

闭包是指有权访问另一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一个函数内创建另一个函数，通过另一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

原型、原型链、特点

原型：

每个函数都有一个 prototype 属性，指向一个原型对象【空 Object 对象】

每一个实例对象都会从原型对象中继承属性

实例的隐式原型(_proto_保存的地址)等于构造函数的显示原型(prototype 保存的地址)

原型链：

Javascript 是面向对象的，每个实例对象都有一个_proto_属性，该属性指向它原型对象，这

个实例对象的构造函数有一个原型属性 prototype，与实例的_proto_属性指向同一个对象。当一个对象在查找一个属性时，自身没有根据_proto_向它的原型进行查找，如果都没有，则向它的原型的原型继续查找，直到查到 Object.prototype._proto_为 null，这样也就形成了原型链。

每个对象都会在其内部初始化一个属性，就是 prototype(原型)，当我们访问一个对象的属性时，如果这个对象内部不存在这个属性，那么他就会去 prototype 里找这个属性，这个 prototype 又会有自己的 prototype，于是就这样一直找下去，也就是我们平时所说的原型链的概念。

关系：instance.constructor.prototype = instance.__proto__

特点：

JavaScript 对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变。

谈谈 This 对象的理解。

this 总是指向函数的直接调用者（而非间接调用者）；

如果有 new 关键字，this 指向 new 出来的那个对象；

在事件中，this 指向触发这个事件的对象，特殊的是，IE 中的 attachEvent 中的 this 总是指向全局对象 Window；

null，undefined 的区别？

null 表示一个对象是“没有值”的值，也就是值为“空”；

undefined 表示一个变量声明了没有初始化(赋值)；

undefined 不是一个有效的 JSON，而 null 是；

undefined 的类型(typeof)是 undefined；

null 的类型(typeof)是 object；

Javascript 将未赋值的变量默认值设为 undefined；

Javascript 从来不会将变量设为 null。它是用来让程序员表明某个用 var 声明的变量时没有值

的。

typeof undefined

//"undefined"

undefined :是一个表示"无"的原始值或者说表示"缺少值"，就是此处应该有一个值，但是还没有定义。当尝试读取时会返回 undefined；

例如变量被声明了，但没有赋值时，就等于 undefined

typeof null

//"object"

null : 是一个对象(空对象, 没有任何属性和方法)；

例如作为函数的参数，表示该函数的参数不是对象；

注意：

在验证 null 时，一定要使用 === ，因为 == 无法分别 null 和 undefined

null == undefined // true

null === undefined // false

DOM 操作——怎样添加、移除、移动、复制、创建和查找节点?

（1）创建新节点

createDocumentFragment() //创建一个 DOM 片段

createElement() //创建一个具体的元素

createTextNode() //创建一个文本节点

（2）添加、移除、替换、插入

appendChild()

removeChild()

replaceChild()

insertBefore() //在已有的子节点前插入一个新的子节点

（3）查找

getElementsByTagName() //通过标签名称

getElementsByName() //通过元素的 Name 属性的值(IE 容错能力较强，会得到一个数组，其中包括 id 等于 name 值的)

getElementById() //通过元素 Id，唯一性

列举字符串操作的方法？

CharCodeAt 方法返回一个整数，代表指定位置字符的 Unicode 编码；

CharAt 方法返回指定索引位置处的字符。如果超出有效范围的索引值返回空字符串；

Slice 方法返回字符串的片段；

Substring 方法返回位于 String 对象中指定位置的子字符串。

Substr 方法返回一个从指定位置开始的指定长度的子字符串

IndexOf 方法返回 String 对象内第一次出现子字符串位置。如果没找到子字符串，则返回-1

LastIndexOf 方法返回 String 对象中字符串最后出现的位置。如果没匹配到子字符串，则返回-1

Search 方法返回与正则表达式查找内容匹配的第一个字符串的位置

Concat 方法返回字符串值，该值包含了两个或多个提供的字符串的连接

Split 将一个字符串分割为子字符串，然后将结果作为字符串数组返回

谈谈你对递归的认识？

递归：程序调用自身的编程技巧称为递归，自己调用自己。

```
1 function factorial(num) {
2 return ( num <= 1 ) ? 1 :  num * factorial(num - 1);
3 }
4 console.log(factorial(8))
```

cookie，sessionStorage 和 localStorage 的区别

cookie 兼容所有浏览器，html5 提供的 storage 存储方式。

Document.Cookie

Window.localstorage

Window.sessionstorage

- 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。

- 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。

- 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一直保存，因此用作持久数据；

  cookie 只在设置的 cookie 过期时间之前一直有效，即使窗口或浏览器关闭。

- 作用域不同，sessionStorage 不在不同的浏览器窗口中共享，即使是同一个页面；

  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

网络：post 和 get

Post 与 get 的区别，什么时候用 post，什么时候用 get？

区别：

Get 存储内容小，不能超过 2kb，有限；文件上传只能用 post

Get 不安全，显示在地址栏

Get 效率高，因为 post 请求需要加密和解密过程，get 不需要。

在做数据查询时，建议用 get 方式；而在做数据添加、修改或删除时，建议用 post 方式；在提交一些不紧要信息时，使用 get，效率高。

前端如何优化网站性能？

减少 HTTP 请求数量

在浏览器与服务器进行通信时，主要是通过 HTTP 进行通信。浏览器与服务器需要经过三次握手，每次握手需要花费大量时间，而不同浏览器对资源文件并发请求数量有限（不同浏览器允许并发数），一旦 HTTP 请求数量达到一定数量，资源请求就存在等待状态，这是很致命的，因此减少 HTTP 的请求数量可以很大程度上对网站性能进行优化。

CSS Sprites:国内俗称 CSS 精灵，这是将多张图片合并成一张图片达到减少 HTTP 请求的一种解决方案，可以通过 CSS background 属性来访问图片内容。这种方案同时还可以减少图片总字节数。

合并 CSS 和 JS 文件：现在前端有很多工程化打包工具，如：grunt、gulp、webpack 等。为了减少 HTTP 请求数量，可以通过这些工具再发布前将多个 CSS 或者 多个 JS 合并成一个文件。

采用 lazyLoad：俗称懒加载，可以控制网页上的内容在一开始无需加载，不需要发请求，等到用户操作真正需要的时候立即加载出内容。这样就控制了网页资源一次性请求数量。

控制资源文件加载优先级

浏览器在加载 HTML 内容时，是将 HTML 内容从上至下依次解析，解析到 link 或者 script 标签就会加载 href 或者 src 对应链接内容，为了第一时间展示页面给用户，就需要将 CSS 提前加载，不要受 JS 加载影响。

一般情况下都是 CSS 在头部，JS 在底部。

利用浏览器缓存

浏览器缓存是将网络资源存储在本地，等待下次请求该资源时，如果资源已经存在就不需要到服务器重新请求该资源，直接在本地读取该资源。

减少重排（Reflow）

基本原理：重排是 DOM 的变化影响到了元素的几何属性（宽和高），浏览器会重新计算元素的几何属性，会使渲染树中受到影响的部分失效，浏览器会验证 DOM 树上的所有其它结点的 visibility 属性，这也是 Reflow 低效的原因。如果 Reflow 的过于频繁，CPU 使用率就会急剧上升。

减少 Reflow，如果需要在 DOM 操作时添加样式，尽量使用 增加 class 属性，而不是通过

style 操作样式。

减少 DOM 操作

图标使用 IconFont 替换

CDN 是啥？

- CDN 的全称：是 Content Delivery Network，即内容分发网络，加速的意思，那么网站 CDN 服务就是网站加速服务。
- CDN 加速原理：CDN 加速将网站的内容缓存在网络边缘(离用户接入网络最近的地方)，然后在用户访问网站内容时，通过调度系统将用户的请求路由或者引导到离用户接入网路最近或者访问效果最佳的缓存服务器上，有该缓存服务器为用户提供内容服务；相对直接访问源站，这种方式缩短了用户和内容之间的网络距离，从而达到加速的效果。
- CDN 的特点：
  - ✓ 本地加速　提高了企业站点(尤其含有大量图片和静态页面站点)的访问速度，并大大提高了以上性质站点的稳定性
  - ✓ 镜像服务　消除了不同运营商之间互联的瓶颈造成的影响，实现了跨运营商的网络加速，保证不同网络中的用户都能得到良好的访问质量。
  - ✓ 远程加速　远程访问用户根据 DNS 负载均衡技术　智能自动选择 Cache 服务器，选择最快的 Cache 服务器，加快远程访问的速度
  - ✓ 带宽优化　自动生成服务器的远程 Mirror(镜像)cache 服务器，远程用户访问时从 cache 服务器上读取数据，减少远程访问的带宽、分担网络流量、减轻原站点 WEB 服务器负载等功能。
  - ✓ 集群抗攻击 广泛分布的 CDN 节点加上节点之间的智能冗余机制，可以有效地预防黑客入侵以及降低各种 D.D.o.S 攻击对网站的影响，同时保证较好的服务质量。

对跨域的理解和解决办法

跨域简单的理解即为当前服务器目录下的前端需求要请求非本服务器(本域名)下的后台服务就称为跨域请求。跨域请求我们使用普通的 ajax 请求是无法进行的同源策略，一般来说位于 server1.example.com 的网页无法与不是 server2.example.com 的服务器沟通，或者说如

果在 server1.example.com 下想获取 server2.example.com 的话就得用跨域请求。

跨域的解决方法及解决：通过 script 标签请求，或者通过 jQuery 跨域

Ajax 的原理

Ajax 的原理简单来说通过 XmlHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后用 javascript 来操作 DOM 而更新页面。

XMLHttpRequest 是 ajax 的核心机制，它是在 IE5 中首先引入的，是一种支持异步请求的技术。简单的说，也就是 javascript 可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

jQuery 库中的$()是什么

$()函数用于将任何对象包裹成 jQuery 对象，接着你就被允许调用定义在 jQuery 对象上的多个不同方法。你甚至可以将一个选择器字符串传入$()函数，它会返回一个包含所有匹配的 DOM 元素数组的 jQuery 对象。

JavaScript 的 window.onload 事件和 jQuery 的 ready 函数有何不同

JavaScript 的 window.onload 事件和 jQuery 的 ready 函数之间的主要区别是，前者除了要等待 DOM 被创建还要等到包括大型图片、音频、视频在内的所有外部资源都完全加载。如果加载图片和媒体内容花费了大量时间，用户就会感受到定义在 window.onload 事件上的代码在执行时有明显的延迟。

另一方面，jQuery 的 ready()函数只需对 DOM 树的等待，而无需对图像或外部资源加载的等待，从而执行起来更快。使用 jQuery $(document).ready()的另一个优势是你可以在网页里多次使用它，浏览器会按它们在 HTML 页面里出现的顺序执行它们，相反对于 onload 技术而言，只能在单一函数里使用。鉴于这个好处，用 jQuery 的 ready()函数比用 JavaScript 的 window.onload 事件要更好些。

Vue 部分

谈谈你对 MVVM 开发模式的理解

MVVM 分为 Model、View、ViewModel 三者。

Model 代表数据模型，数据和业务逻辑都在 Model 层中定义

View 代表 UI 视图，负责数据的展示；

ViewModel 负责监听 Model 中数据的改变并且控制试图的更新，处理用户交互操作；

Model 和 View 并无直接关联，而是通过 ViewModel 来进行联系的，Model 和 ViewModel 之间有着双向数据绑定的联系。因此当 Model 中的数据改变时会触发 View 层的刷新，View 中由于用户交互操作而改变的数据也会在 Model 中同步。

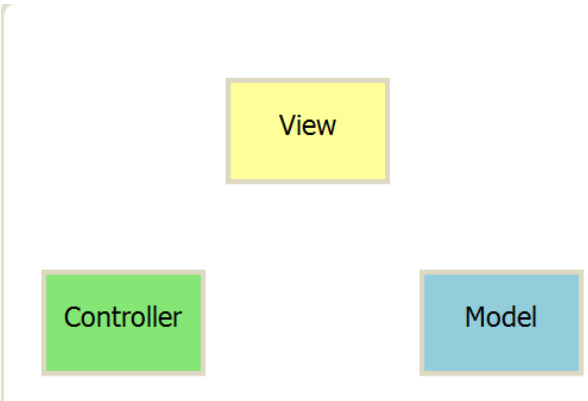
这种模式实现了 Model 和 View 的数据自动同步，因此开发者只需要专著对数据的维护操作即可，而不需要自己操作 DOM。

MVC，MVP 和 MVVM 的图示

http://www.ruanyifeng.com/blog/2015/02/mvcmvp_mvvm.html

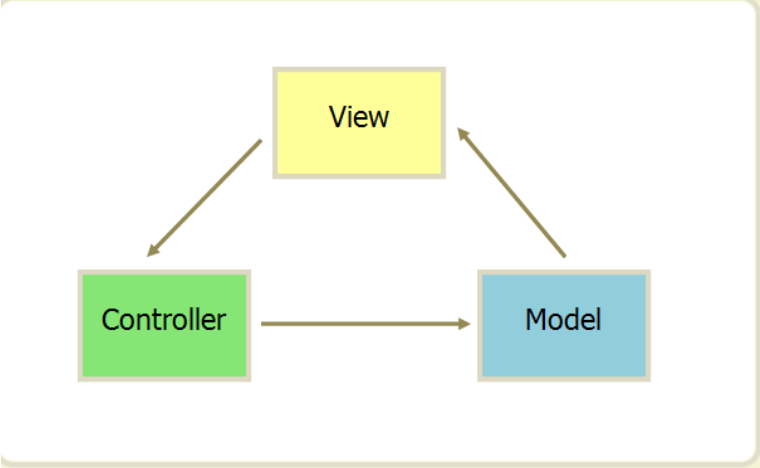
MVC

MVC 模式的意思是，软件可以分成三个部分。



- 视图（View）：用户界面。
- 控制器（Controller）：业务逻辑
- 模型（Model）：数据保存

各部分之间的通信方式如下。



1．View 传送指令到 Controller

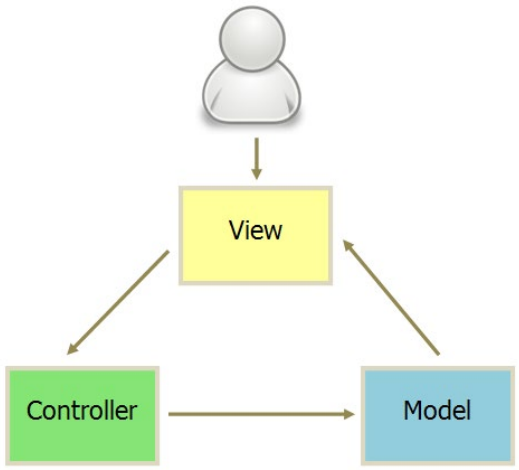
2. Controller 完成业务逻辑后，要求 Model 改变状态

3. Model 将新的数据发送到 View，用户得到反馈

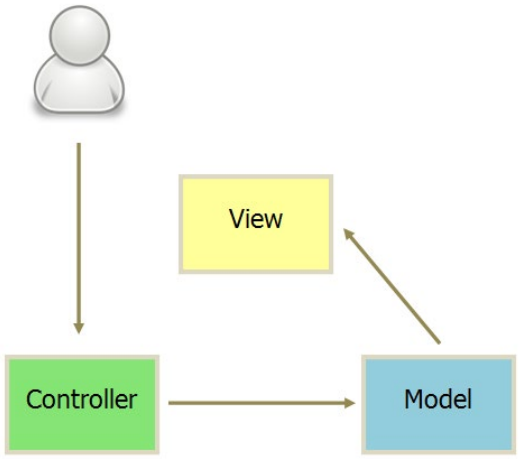
二、互动模式

接受用户指令时，MVC 可以分成两种方式。

一种是通过 View 接受指令，传递给 Controller。

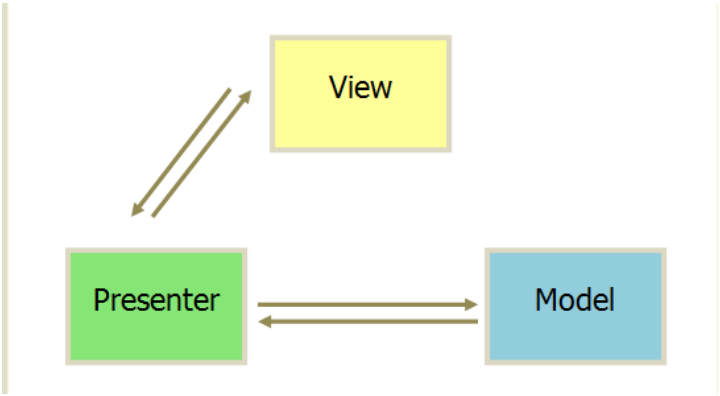


另一种是直接通过 controller 接受指令。



四、MVP

MVP 模式将 Controller 改名为 Presenter，同时改变了通信方向



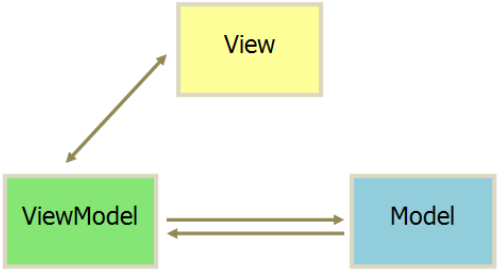
1 各部分之间的通信，都是双向的。

2. View 与 Model 不发生联系，都通过 Presenter 传递。

3. View 非常薄，不部署任何业务逻辑，称为"被动视图"（Passive View），即没有任何主动性，而 Presenter 非常厚，所有逻辑都部署在那里。

五、MVVM

MVVM 模式将 Presenter 改名为 ViewModel，基本上与 MVP 模式完全一致。



唯一的区别是，它采用双向绑定（data-binding）：View 的变动，自动反映在 ViewModel，反之亦然。Angular 和 Ember 都采用这种模式。

生命周期

- 创建前后 beforeCreate/created

在 beforeCreate 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 实例的数据对象有了，el 还没有。

- 载入前后 beforeMount/mounted

在 beforeMount 阶段，vue 实例的$el 和 data 都初始化了，但还是挂载之前未虚拟的 DOM 节点，data 尚未替换。

在 mounted 阶段，vue 实例挂载完成，data 成功渲染。

- 更新前后 beforeUpdate/updated

当 data 变化时，会触发 beforeUpdate 和 updated 方法。这两个不常用，不推荐使用。

- 销毁前后 beforeDestory/destoryed

beforeDestory 是在 vue 实例销毁前触发，一般在这里要通过 removeEventListener 解除手动绑定的事件。实例销毁后，触发的 destroyed。

理解 Vue 中的 Render 渲染函数

VUE 一般使用 template 来创建 HTML，然后在有的时候，我们需要使用 javascript 来创建 html，这时候我们需要使用 render 函数。

render 函数 return 一个 createElement 组件中的子元素存储在组件实列中 $slots.default 中。

return createElement('h1', this.title); createElement 返回的是包含的信息会告诉 VUE 页面上需要渲染什么样的节点及其子节点。我们称这样的节点为虚拟 DOM，可以简写为 VNode。

createElement 参数

一个HTML标签字符串，组件选项对象，或者一个返回值类型为String/Object的函数。该参数是 必须的

子节点

子节点，可选，String 或 Array

```
Vue.component('anchored-heading', {
  render: function (createElement) {
    return createElement(
      'h' + this.level,   // 标签名称
      this.$slots.default // 由子节点构成的数组
    )
  },
  props: {
    level: {
      type: Number,
      required: true
    }
  }
})
```

Vue 有哪些指令

插值属性的：Mustache、v-once、v-html、v-text、v-pre、v-clock

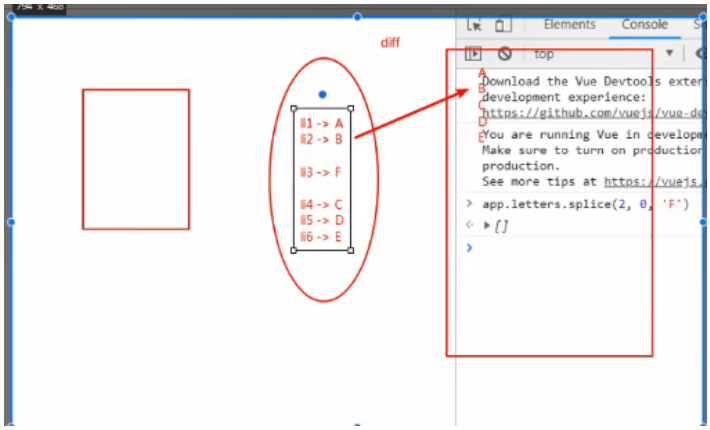
动态绑定指令:　v-bind

事件监听指令：v-on

条件判断指令：v-if、v-elseif 、v-else

循环遍历指令：v-for

v-for 使用时，为什么要给对应元素添加一个:key 属性（海康）



列表在显示到真实 DOM 前，会先显示在虚拟 DOM 上

即：为了更好的复用

v-if 和 v-show 有什么区别？

v-show 仅仅控制元素的显示方式，将 display 属性在 block 和 none 来回切换；

而 v-if 会控制这个 DOM 节点的存在与否。

当我们需要经常切换某个元素的显示/隐藏时，使用 v-show 会更加节省性能上的消费；

当只需要一次显示或隐藏时，使用 v-if 更加合理。

v-show 是 CSS 切换，v-if 是完整的销毁和重新创建。

频繁切换时用 v-show，运行较少改变时用 v-if

动态绑定 class 的方法

- 对象方法 `v-bind:class="{'orange': isRipe, 'green': isNotRipe}"`
- 数组方法`v-bind:class="[class1, class2]"`
- 行内 `v-bind:style="{color: color, fontSize: fontSize+'px' }"`

计算属性和 watch 的区别

计算属性是自动监听依赖值得变化，从而动态返回内容，监听是一个过程，在监听的值变化时，可以触发一个回调，并做一些事情。

所以区别来源于用法，只是需要动态值，那就用计算属性；需要指导值的改变后执行业务逻辑，才用 watch，用反或混用虽然可行，但都是不正确的用法。

区分：

computed 是一个对象时，他有哪些选项？

有 get 和 set 两个选项

Computed 和 methods 有什么区别？

methods 是一个方法，它可以接受参数，而 computed 不能，computed 是可以缓存的，methods 不会。

Computed 是否能依赖其他组件的数据？

Computed 可以依赖其他 computed，甚至是其他组件的 data

Watch 是一个对象时，他有哪些选项？

Watch 的配置

Handler

Deep 是否深度

Immeditate 是否立即执行

总结

当有一些数据需要随着另外一些数据变化时，建议使用 computed。

当有一个通用的响应数据变化的时候，要执行一些业务逻辑或异步操作的时候建议使用 watcher

组件中 data 为什么是函数

为什么组件中的 data 必须是一个函数，然后 return 一个对象，而 new Vue 实例里，data 可以直接是一个对象？

因为组件是**用来复用的**，JS 里对象是引用关系，这样作用域没有隔离，而 new Vue 的实例，是不会被复用的，因此不存在引用对象的问题。

自定义组件的语法糖 v-model 是怎样实现的

根据官方文档介绍，v-model本质上就是语法糖，即利用v-model绑定数据后，其实就是既绑定了数据，又添加了一个input事件监听，如下：

自定义事件也可以用于创建支持 `v-model` 的自定义输入组件。记住：

```HTML
<input v-model="searchText">
```

等价于：

```HTML
<input
  v-bind:value="searchText"
  v-on:input="searchText = $event.target.value"
>
```

当用在组件上时，`v-model` 则会这样：

```HTML
<custom-input
  v-bind:value="searchText"
  v-on:input="searchText = $event"
></custom-input>
```

怎样理解单向数据流

这个概念出现在组件通信。父组件是通过 prop 把数据传递到子组件的，但是这个 prop 只能由父组件修改，子组件不能修改，否则会报错。子组件想修改时，只能通过$emit 派发一个自定义事件，父组件接收到后，由父组件修改。

一般来讲，对于子组件想要更改父组件状态的场景，可有两种方案：

在子组件的 data 中拷贝一份 prop，data 是可以修改的，但 prop 不能：

```
export default {
  props: {
    value: String
  },
  data () {
    return {
      currentValue: this.value
    }
  }
}
```

如果是对 prop 值的转换，可以使用计算属性：

```
export default {
  props: ['size'],
  computed: {
    normalizedSize: function () {
      return this.size.trim().toLowerCase();
    }
  }
}
```

组件间的通信

1.父子 props/event parent/parent/children ref provide/inject

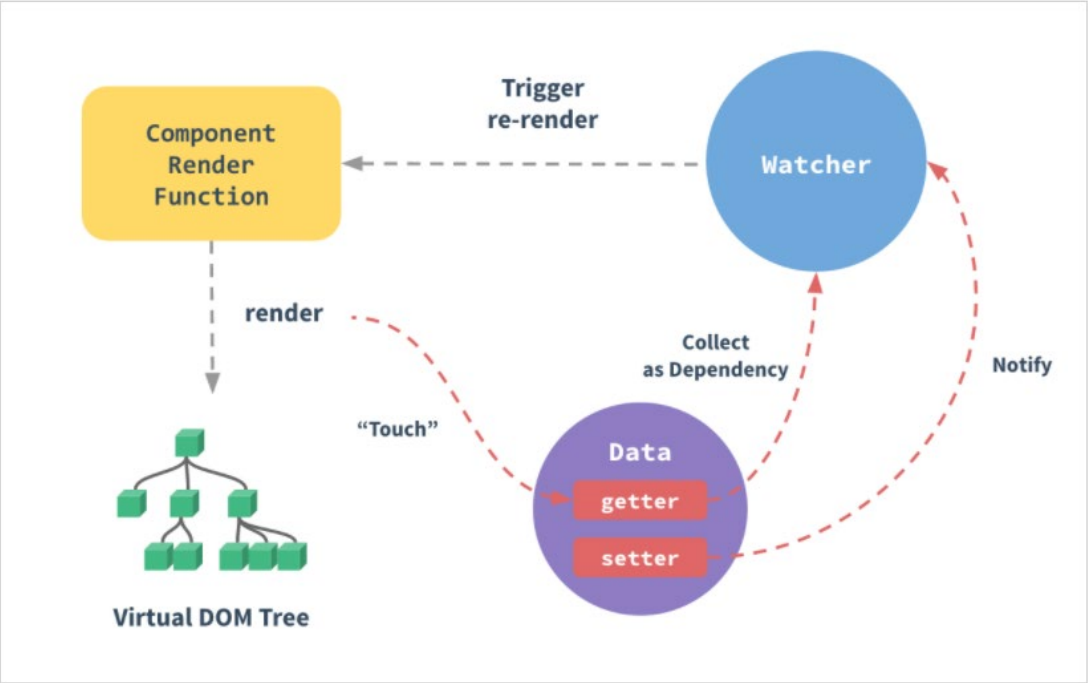
2.兄弟 bus vuex

3.跨级 bus vuex provide.inject

简述 Vue 的响应式原理

当一个 Vue 实例创建时，Vue 会遍历 data 选项的属性，用 Object.defineProperty 将他们转化为 getter/setter 并且在内部追踪相关依赖，在属性被访问和修改时通知变化。

每个组件实例都有相应的 watcher 程序实例，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 setter 被调用时，会通知 watcher 重新计算，从而致使它关联的组件得以更新。



slot 插槽

单个插槽

当组件模板只有一个没有属性的插槽时，父组件传入整个内容片段将插入到插槽所在的 DOM 位置，并替换掉插槽标签本身。

最初在<slot>标签中的任何内容都被视为备用内容。备用内容在子组件的作用域内编译，并且只有在宿主元素为空，且没有要插入的内容时才显示备用内容。

命名插槽

Slot 元素可以用一个特殊的特性 name 来进一步配置如何分发内容。多个插槽可以有不同的名字。

这样可以将父组件模板中 slot 位置和子组件 slot 元素产生关联，便于插槽内容对应传递

作用域插槽 scoped slots

可以访问组件内部数据的可复用插槽(reusable slot)

在父级中，具有特殊特性 slot-scope 的<template>元素必须存在，表示它是作用域插槽的模板。Slot-scope 的值将被用作一个临时变量名，此变量接收从子组件传递过来的 prop 对象。

Vue 中如何在组件内部实现一个双向数据绑定？

假设有一个输入框组件，用户输入时，同步父组件页面中的数据

具体思路：父组件通过 props 传值给子组件，子组件通过 $emit 来通知父组件修改相应的 props 值，具体实现如下

网页从输入网址到渲染完成经历了哪些过程？

大致可以分为如下 7 步：

- 输入网址；
- 发送到 DNS 服务器，并获取域名对应的 web 服务器对应的 ip 地址；
- 与 web 服务器建立 TCP 连接；
- 浏览器向 web 服务器发送 http 请求；
- web 服务器响应请求，并返回指定 url 的数据（或错误信息，或重定向的新的 url 地址）；
- 浏览器下载 web 服务器返回的数据及解析 html 源文件；
- 生成 DOM 树，解析 css 和 js，渲染页面，直至显示完成；