FUNDAMENTAL

陈磊

2018 年 3 月 2 日

目录

1 HTML

1.1 HTML5 新标签

标签	描述
`<article>`	定义文章。
`<aside>`	定义页面内容之外的内容。
`<audio>`	定义声音内容。
`<bdi>`	定义文本的文本方向，使其脱离其周围文本的方向设置。
`<canvas>`	定义图形。
`<command>`	定义命令按钮。
`<datalist>`	定义下拉列表。
`<details>`	定义元素的细节。
`<dialog>`	定义对话框或窗口。
`<embed>`	定义外部交互内容或插件。
`<figcaption>`	定义 figure 元素的标题。
`<figure>`	定义媒介内容的分组，以及它们的标题。
`<footer>`	定义 section 或 page 的页脚。

标签	描述
`<header>`	定义 section 或 page 的页眉。
`<keygen>`	定义生成密钥。
`<mark>`	定义有记号的文本。
`<meter>`	定义预定义范围内的度量。
`<nav>`	定义导航链接。
`<output>`	定义输出的一些类型。
`<progress>`	定义任何类型的任务的进度。
`<rp>`	定义若浏览器不支持 ruby 元素显示的内容。
`<rt>`	定义 ruby 注释的解释。
`<ruby>`	定义 ruby 注释。
`<section>`	定义 section。
`<source>`	定义媒介源。
`<summary>`	为 `<details>` 元素定义可见的标题。
`<time>`	定义日期/时间。
`<video>`	定义视频。
`<wbr>`	定义视频。

reference: w3school, HTML 参考手册

1.2 内联元素和块级元素

1. 块级元素

```
<address> <caption> <dd> <div> <dl> <dt> <fieldset>
<form> <h1> <h2> <h3> <h4> <h5> <h6> <hr> <legend>
<li> <noframes> <noscript> <ol> <ul> <p> <pre>
<table> <tbody> <td> <tfoot> <th> <thead> <tr>
```

2. 内联元素

```
<a> <abbr> <acronym> <b> <bdo> <big> <br> <cite> <code>
<dfn> <em> <i> <img> <input> <kbd> <label> <q> <samp> <select>
<small> <span> <strong> <sub> <sup> <textarea> <tt> <var>
```

3. 可变元素：根据上下文确定是块级元素还是内联元素

```
<applet> <button> <del> <iframe> <ins> <map> <object> <script>
```

reference: A1ter, 块级元素和行内元素的区别, segmentfault

1.3 defer 和 async

```
<script async src="script.js"></script>
<script defer src="script.js"></script>
```

1. defer 和 async 在网络读取（下载）这块儿是一样的，都是异步的（相较于 HTML 解析）
2. 它俩的差别在于脚本下载完之后何时执行，显然 defer 是最接近我们对于应用脚本加载和执行的要求的
3. 关于 defer，此图未尽之处在于它是按照加载顺序执行脚本的，这一点要善加利用
4. async 则是一个乱序执行的主，反正对它来说脚本的加载和执行是紧紧挨着的，所以不管你声明的顺序如何，只要它加载完了就会立刻执行
5. async 对于应用脚本的用处不大，因为它完全不考虑依赖（哪怕是最低级的顺序执行），不过它对于那些可以不依赖任何脚本或不被任何脚本依赖的脚本来说却是非常合适的，最典型的例子：Google Analytics

reference: nightire, defer 和 async 的区别, segmentfault

1.4 img 属性 alt 和 title

- alt：当图片无法显示时，将显示 alt 指定的文字。
- title：鼠标滑过时，会显示 title 指定文本。

1.5 Web 存储

- localStorage - 没有时间限制的数据存储
- sessionStorage - 针对一个 session 的数据存储

  sessionStorage, in practice, is best used for temporary data storage.

(Cookie 4kb, userData 64kb, Flash 100kb, SQLite, HTML5 localStorage 5MB)

reference: Web Storage 初探, Web 前端实现本地存储, Web Storage API

2 CSS

定位, 盒模型, 浮动, 媒体查询 (自适应布局).

2.1 定位

CSS 有三种基本的定位机制：普通流、浮动和绝对定位。任何元素都可以定位，不过绝对或固定元素会生成一个块级框，而不论该元素本身是什么类型。

说明

1. 默认值: static
2. 继承性: no
3. 版本: CSS2
4. JavaScript 语法: `object.style.position="absolute"`

值	描述
absolute	生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。元素的位置通过 “left”, “top”, “right” 以及 “bottom” 属性进行规定。
fixed	生成绝对定位的元素，相对于浏览器窗口进行定位。元素的位置通过 “left”, “top”, “right” 以及 “bottom” 属性进行规定。
relative	生成相对定位的元素，相对于其正常位置进行定位。因此，“left:20” 会向元素的 LEFT 位置添加 20 像素。
static	默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。
inherit	规定应该从父元素继承 position 属性的值。

- 如果一个标签的位置是绝对的，它又不在其他任何设定了 absolute、relative 或 fixed 定位的标签里面，那它就是相对于浏览器窗口进行定位。
- 如果一个标签处在另一个设定了 absolute、relative 或 fixed 定位的标签里面，那它就是相对于另一个元素的边沿进行定位。

reference: `http://www.w3school.com.cn/cssref/pr_class_position.asp`，CSS 实战手册（第 2 版）

2.2 盒模型

IE 盒模型和标准存在区别。

2.3 优先级

优先级**从高到低**，分三个层次描述。

1. 作者/用户/浏览器样式。作者样式指网页本身的样式，或者开发者编写的样式；用户样式指浏览网页的用户自己添加的样式表 (通过浏览器设置)；浏览器样式指浏览器提供的默认样式。
    1. 标有 `!important` 的用户样式。
    2. 标有 `!important` 的作者样式。
    3. 作者样式。
    4. 用户样式。
    5. 浏览器样式。
2. 作者样式：内部样式（internal）、内联样式（inline）、外部样式（external）。内部样式 style 标签中声明的样式；内联样式指元素属性 style 中的样式；外部样式指通过 link 链接的外部文件中样式。
    1. 内联样式（行内样式）
    2. 内部样式
    3. 外部样式

外部样式和内部样式在优先级相同的情况下，后定义的会覆盖先定义的。

3. CSS 选择器
    1. ID 选择器：`#idname`
    2. 伪类：`:hover`
    3. 属性选择器：`input[type="text"]`
    4. 类选择器：`.classname`

5. 元素 (类型) 选择器（包括伪元素）：`input`, `:after`
6. 通用选择器：`*`

reference: Cameron Moll, 精通 CSS（第 2 版），优先级, mdn, INLINE VS INTERNAL VS EXTERNAL CSS

2.4 link 和 @import

- `@import` 和 link 混用时，可能会出现不同时下载的情况。
- 考虑两者混合使用的浏览器实现的不一样（下载次序）。

reference: [Steve Souders, don't use @import](http://www.stevesouders.com/blog/2009/04/09/dont-use-import/), 外部引用 CSS 中 link 与 @import 的区别

2.5 浮动和文档流

2.6 两栏等高布局

父元素设置 `overflow:hidden;`，子元素设置 `padding-bottom:10000px;` ↪ `margin-bottom:-10000px;`。`overflow:hidden;` 截除超出的高度，`margin-bottom` ↪ `:-10000px;` 抵消较高内容超出的 `padding`。

```
<div id="fa">
  <div class="col">
    <p>1231321321</p>
    <p>1313213</p>
    <p>1313213</p>
    <p>1313213</p>
    <p>1313213</p>
    <p>1313213</p>
  </div>
  <div class="col">45645456456</div>
</div>
```

```
#fa {
  width: 800px;
  margin: 0 auto;
  background-color: #1524e5;
  overflow: hidden;
}

.col {
```

```
  float: left;
  width: 50%;
  padding-bottom: 10000px;
  margin-bottom: -10000px;
}

.col:first-child {
  background-color: #34ef34;
}
.col:last-child {
  background-color: #ef34ef;
}
```

refernce: CSS/两栏并列等高布局

2.7 动画

动画的实现方式可以分为几种：CSS3 原生实现；JavaScript 实现 (操作 CSS、canavs、svg)；动画文件 (gif, flash)。

CSS3 的实现可以通过 @keyframes 和 animation 完成，@keyframes 定义动画，animation 调用动画。

```
/* 定义 */
@keyframes changebackcolor{
  from {background: red;}
  to {background: yellow;}
}
/* 调用 */
div{
  animation: changebackcolor 5s;
}
```

reference: `http://www.w3school.com.cn/css3/css3_animation.asp`

2.8 垂直居中和水平居中

2.8.1 块元素垂直居中 (transform)

此方案存在兼容性问题 (Firefox 43 不支持, Chrome 46 支持)

```
width: 250px;
height: 250px;
```

```
position: relative;
top: 50%;
transform: translateY(-50%);
```

2.8.2 块元素垂直居中 (绝对定位)

```
width: 250px;
height: 250px;
position: absolute;
top: 50%;
margin-top: -125px;
```

2.8.3 内联元素垂直居中

```
/* 方案一: 块元素内容会居中, 需要设置高度 */
display: table-cell;
vertical-align: middle;

/* 方案二: 一般单行文本的上下居中 */
line-height: 50px;
```

2.8.4 块元素的水平居中

```
width: 100px;
margin-left: auto;
margin-right: auto;
```

3 JavaScript

3.1 JavaScript 组成

一个完整的 JavaScript 实现是由以下 3 个不同部分组成的：

- 核心（ECMAScipt）
- 文档对象模型（DOM）
- 浏览器对象模型（BOM）
- JavaScript 实现
- JavaScript 学习总结（三）BOM 和 DOM 详解

3.2 类型、值和变量

3.2.1 数据类型

- 基本数据类型：String 字符串；Number 数字；Boolean 布尔。
- 复合数据类型：Object 对象；Array 数组。
- 特殊数据类型：Null 空对象；Undefined 未定义。
- 数据类型 (JavaScript), msdn

3.2.2 null, NaN, undefined

JavaScript 中有 6 个值为 “假”: `false`, `null`, `undefined`, `0`, `''`(空字符串), `NaN`. 其中 `NaN` 是 JavaScript 中唯一不等于自身的值, 即 `NaN == NaN` 为 `false`.

```
console.log( false == null ) // false
console.log( false == undefined ) // false
console.log( false == 0 )    // true
console.log( false == '' )   // true
console.log( false == NaN )  // false

console.log( null == undefined ) // true
console.log( null == 0 )    // false
console.log( null == '' )   // false
console.log( null == NaN )  // false

console.log( undefined == 0 ) // false
console.log( undefined == '' ) // false
console.log( undefined == NaN ) // false

console.log( 0 == '' )      // true
console.log( 0 == NaN )     // false
```

对于 `===` 以上全为 `false`。对于 `==`，以下几组为 `true`: `null` 和 `undefined`; `false`、`0`、`''`。

- JavaScript 中奇葩的假值
- 阮一峰, undefined 与 null 的区别, 2014

3.3 函数

3.3.1 常用函数

- `string.slice(start,end)` 复制 string 中的一部分。

- `string.indexOf(searchString, position)` 在 string 中查找 searchString。如果被找到，返回第一个匹配字符的位置，否则返回 -1。可选参数 position 可设置从 string 的某个指定的位置开始查找。
- `object.hasOwnProperty(name)`

3.3.2 Array

改变原数组

- `array.pop()` 移除最后一个元素。
- `array.push(item...)` 将一个或多个参数附加到数组的尾部。
- `array.reverse()` 反转 array 元素的顺序。
- `array.sort(camprefn)` 数组排序。
- `array.shift()` 移除数组中的第一个元素并返回该元素。
- `array.splice(start, deleteCount, item)` 从 array 中移除一个或多个元素并用新的 item 替换他们。

不改变原数组

- `array.concat(item...)` 产生一个新数组，它包含一份 array 的前复制，并把一个或多个参数 item 附加在其后面。
- `array.join(separator)` join 方法把一个 array 构成一个字符串。它先把 array 中的每个元素构造成一个字符串，接着用一个 separator 分隔符把它们连接在一起。
- `array.map(item => )` 数组映射, 依次处理数组中的每一项.
- `array.filter(item => )` 数组过滤, 保存返回为真的项.
- forEach(callback(currentValue[, index[, array]]){}), 循环.
- some, 是否有某个元素满足条件 (回掉返回 true).
- every, 是否全部元素满足条件.

3.3.3 Date

```
1 var t = new Date();
2 var tt = [
3    t.getFullYear(), '年', // 不是 getYear()
4    t.getMonth() + 1, '月',
5    t.getDate(), '日', ' ',
6    t.getHours(), '时',
7    t.getMinutes(), '分',
```

```
    t.getSeconds(), '秒'
].join('');
console.log(tt); // 2015年10月30日 22时6分21秒
```

注意 `getMonth()` 和 `getDay()` 都是从 0 开始的，需要加 1。

- Date, mdn

3.3.4 typeof 和 instanceof

1. typeof: `typeof operand`
   typeof 操作符返回一个字符串, 表示未经求值的操作数 (unevaluated operand) 的类型。typeof 只有一个实际应用场景，就是用来检测一个对象是否已经定义或者是否已经赋值。而这个应用却不是来检查对象的类型。除非为了检测一个变量是否已经定义，我们应尽量避免使用 typeof 操作符。

   ```
   typeof foo == 'undefined' // 若 foo 未定义, 返回 true
   ```

类型	结构
Undefined	“undefined”
Null	“object”
布尔值	“boolean”
数值	“number”
字符串	“string”
Symbol (ECMAScript 6 新增)	“symbol”
宿主对象 (JS 环境提供的，比如浏览器)	Implementation-dependent
函数对象 (implements [[Call]] in ECMA-262 terms)	“function”
任何其他对象	“object”

2. instanceof: `object instanceof constructor`
   instanceof 运算符可以用来判断某个构造函数的 prototype 属性是否存在另外一个要检测对象的原型链上。(instanceof 运算符用来检测 constructor.prototype 是否存在于参数 object 的原型链上。)

   ```
   function C(){} // 定义一个构造函数
   function D(){} // 定义另一个构造函数

   var o = new C();
   ```

```
o instanceof C; // true,因为:Object.getPrototypeOf(o) === C.
    ↪ prototype
o instanceof D; // false,因为D.prototype不在o的原型链上
o instanceof Object; // true,因为Object.prototype.isPrototypeOf(o)返
    ↪ 回true
C.prototype instanceof Object // true,同上

C.prototype = {};
var o2 = new C();
o2 instanceof C; // true
o instanceof C; // false,C.prototype指向了一个空对象,这个空对象不在o
    ↪ 的原型链上.

D.prototype = new C();
var o3 = new D();
o3 instanceof D; // true
o3 instanceof C; // true
```

- typeof 和 instanceOf 的区别
- instanceof
- typeof

3.3.5 setInterval 和 setTimeout

- `setTimeout` 只执行一次
- `setInterval` 连续执行多次

3.3.6 call 和 apply

通过 call 和 apply, 可以实现自定义函数调用的上下文.

1. call: `fun.call(thisArg, arg1, arg2, ...)`
   call() 方法在使用一个指定的 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.

   thisArg 在 fun 函数运行时指定的 this 值。需要注意的是，指定的 this 值并不一定是该函数执行时真正的 this 值，如果这个函数处于非严格模式下，则指定为 null 和 undefined 的 this 值会自动指向全局对象 (浏览器中就是 window 对象)，同时值为原始值 (数字，字符串，布尔值) 的 this 会指向该原始值的自动包装对象。

   arg1, arg2, ... 指定的参数列表。

```
/* 将函数的参数 arguments 转换为数组 */
function listFirst(){
    // this 指向 arguments
    // 下面的语句相当于 arguments.slice(0)，但由于 arguments 不是数
        ↪ 组，不能直接调用 slice 方法
    var arr = Array.prototype.slice.call(arguments, 0);
    for (var i=0; i < arr.length; i++){
        console.log(arr[i]);
    }
}

listFirst(1,2,3); // 调用，输出 1,2,3
```

2. apply: `fun.apply(thisArg, [argsArray])`
   apply() 方法在指定 this 值和参数（参数以数组或类数组对象的形式存在）的情况下调用某个函数。

   thisArg 在 fun 函数运行时指定的 this 值。需要注意的是，指定的 this 值并不一定是该函数执行时真正的 this 值，如果这个函数处于非严格模式下，则指定为 null 或 undefined 时会自动指向全局对象（浏览器中就是 window 对象），同时值为原始值（数字，字符串，布尔值）的 this 会指向该原始值的自动包装对象。

   argsArray 一个数组或者类数组对象，其中的数组元素将作为单独的参数传给 fun 函数。如果该参数的值为 null 或 undefined，则表示不需要传入任何参数。从 ECMAScript 5 开始可以使用类数组对象。浏览器兼容性请参阅本文底部内容。

   在调用一个存在的函数时，你可以为其指定一个 this 对象，无需此参数时第一个参数可用 null（比如对于 add）。this 指当前对象，也就是正在调用这个函数的对象。使用 apply，你可以只写一次这个方法然后在另一个对象中继承它，而不用在新对象中重复写该方法。

```
/* 例一：将函数的参数 arguments 转换为数组 */
function listFirst(){
    // this 指向 arguments
    var arr = Array.prototype.slice.apply(arguments, [0]);
    for (var i=0; i < arr.length; i++){
        console.log(arr[i]);
    }
}

```

```
listFirst(1,2,3); // 调用, 输出 1,2,3

/* 例二: push 一个数组 */
var arr1=new Array("1","2","3");
var arr2=new Array("4","5","6");
Array.prototype.push.apply(arr1,arr2);
```

- Function.prototype.apply()
- apply 和 call 的用法

3.3.7 new/构造函数

构造函数只是一些使用 new 操作符时被调用的普通函数。使用 new 来调用函数，或者说发生构造函数调用时，会自动执行下面的操作。

1. 创建（或者说构造）一个全新的对象。
2. 这个新对象会被执行 [[prototype]] 链接。
3. 这个对象会绑定到函数调用的 this。
4. 如果函数没有返回其他对象，那么 new 表达式中的函数调用会自动返回这个新对象。

3.3.8 函数调用模式

除了声明时定义的形式参数, 每个函数还接收两个附加的参数: this 和 arguments. 在 JavaScript 中一共有 4 种调用模式: 方法调用模式、函数调用模式、构造器调用模式和 apply 调用模式. 函数调用的模式不同, 对应的 this 值也会不同。

1. 方法调用模式
   当一个函数被保存为对象的一个属性时，我们称它为一个方法。当一个方法被调用时，this 被绑定到该对象。

```
// print 作为 obj 属性被保存, 当 print 被调用时, this 指向 obj
var obj = {
    value: 'I am a string.',
    print: function (){
        console.log(this.value);
    }
}

obj.print(); // 调用
```

2. 函数调用模式
   当一个函数并非一个对象的属性时, 那么它就是被当做一个函数来调用。以此模式调用函数时，this 被绑定到全局对象。

```
var func = function(){
    console.log('Hello here.');
}

func(); // 调用
```

3. 构造器调用模式
   如果在一个函数前面带上 new 来调用，那么背地里将会创建一个连接到该函数的 prototype 成员的新对象，同时 this 会绑定到那个新对象上。

```
var Quo = function(string){
    this.status = string;
}
Quo.prototype.get_status = function(){
    return this.status;
}

var myQuo = new Quo('new Quo.') // 调用
myQuo.get_status(); // "new Quo."
```

4. apply 调用模式
   apply 方法让我们构建一个参数数组传递给调用函数。它允许我们选择 this 的值。apply 方法接收两个参数，第 1 个是要绑定给 this 的值，第 2 个就是一个参数数组.

- 《JavaScript 语言精粹 (修订版)》，第 4 章函数

3.3.9 this

this 在运行时绑定，它的上下文取决于函数调用时的各种条件。this 的绑定和函数声明的位置没有任何关系，只取决于函数的调用方式。

判断 this 的优先级，可以按照下面的顺序进行判断：

1. 函数是否在 new 中调用 (new 绑定)？如果是的话 this 绑定的是新创建的对象。

```
var bar = new foo(); // 绑定 bar
```

2. 函数是否通过 call、apply (显示绑定) 或者硬绑定调用？如果是的话，this 绑定的是指定的对象。

```
var bar = foo.call(obj2); // 绑定 obj2
```

3. 函数是否在某个上下文对象中调用 (隐式调用)？如果是的话，this 绑定的是那个上下文对象。(调用时是否被某个对象拥有或包含，对象属性引用链中只有最顶层或者最后一层会影响调用位置 `obj1.obj2.foo()` ↪ `; // 绑定 obj2`)

```
var bar = obj1.foo(); // 绑定 obj1
```

4. 如果都不是的话，使用默认绑定。如果在严格模式下，就绑定到 undefined，否则绑定到全局对象 (window)。

```
var bar = foo(); // 绑定 undefined 或 window
```

可总结为:

1. 作为方法调用, 上下文为方法的拥有者.
2. 作为全局函数调用, 上下文为 window.
3. 作为构造函数调用, 上下文为新创建的实例对象.
4. call 和 apply 可以自定义上下文.
5. 箭头函数的上下文为其所在作用域的上下文.

- 你不知道的 JavaScript（上卷）
- JavaScript 忍者秘籍, p52

3.3.10 闭包

当函数可以记住并访问所在的词法作用域时，就产生了闭包，即使函数是在当前词法作用域之外执行[1]。(闭包是发生在定义时的。)

```
// foo() 定义的中括号内就是 bar 的词法作用域
function foo() {
  var a = 2;
  function bar() {
    console.log( a );
  }
  return bar;
}
var baz = foo();
baz(); // 2, 这就是闭包, 用到变量 a
```

[1] 你不知道的 JavaScript（上卷）

```
var fn;
function foo() {
  var a = 2;
  function baz() {
    console.log( a ); // 2
  }
  fn = baz; // 将 baz 赋值给全局变量
  // 调用 fn 相当于执行的 baz, 而其词法作用域在 foo() 定义函数内.
}

function bar(fn) {
  fn(); // 这就是闭包, 用到变量 a
}

foo();
bar(); // 2
```

3.4 继承方法

- 原型链继承
- 构造继承
- 实例继承
- 拷贝继承
- 继承与原型链, mdn, RayChase, JavaScript 实现继承的几种方式

3.5 正则表达式

参数

- `i`, 忽略大小写.
- `g`, 全局匹配, 找到所有匹配，而不是在第一个匹配后停止.

```
// 替换所有 a 字符, 不区分大小写
"aAopa".replace(/a/gi, '')
```

特殊字符

- `\d` 任意一个数字，等价于 `[0-9]`
- `\D` 任意一个非数字，等价于 `[^0-9]`
- `\w` 任意一个字母、数字或下划线字符，等价于 `[a-zA-Z_]`
- `\W` 任意一个非字母、数字和下划线字符，等价于 `[^a-zA-Z_]`

- `\s` 任意一个空白字符，包括换页符、换行符、回车符、制表符和垂直制表符，等价于 `[\f\n\r\t\v]`
- `\S` 任意一个非空白符，等价于 `[^\f\n\r\t\v]`
- . 换行和回车以外的任意一个字符，等价于 `[^\n\r]`

次数匹配

- `?` 最多一次 (零次或一次)
- `+` 至少一次
- `*` 任意次
- `{n}` 只能出现 n 次
- `{n,m}` 至少 n 次，最多 m 次

reference

- mozilla, RegExp
- 正则总结：JavaScript 中的正则表达式

3.6 ES5

新功能包括：原生 JSON 对象、继承的方法、高级属性的定义以及引入严格模式。

- 梦禅, ECMAScript 各版本简介及特性, segmentfault

3.7 ES6

模块，类，块级作用域，Promise，生成器...

3.7.1 箭头函数

1. 优点: 箭头函数可省略 `function`, 单行的情况下可省略 `return`, 单个参数的情况括号也可省略.

```
function func1(a) {
  return a + 1;
}

var func1 = a => a + 1;
```

2. 优点: 箭头函数不会创建自己的上下文 (context), 意味着函数内 `this` 和箭头函数所定义的上下文一样.

```
var _this = this;
axios.get(api).then(function(res){
  _this.model = res.data;
});

// 无需 _this 或者 bind 传递 this
axios.get(api).then(res => {
  this.model = res.data;
})
```

3. 技巧: 返回对象, 需要在对象外层加括号 (对象大括号会被解析为函数体).

```
// 错误
var func1 = a => {a: a};
console.log(func1(a));

// 正确
var func1 = a => ({a: a});
console.log(func1(a));
```

4. 技巧: 通过 || 添加日志, 而不用单独一行输出日志.

```
a => {
  console.log(a);
  return {
    a: a
  }
}

a => console.log(a) || ({
  a: a
});
```

- Arrow Functions in JavaScript

3.8 异步请求

3.8.1 Ajax (Asynchronous Javascript And XML)

Ajax 实现

1. 原生 Ajax 实现: GET

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET","test1.txt",true);
xmlhttp.send();
```

2. 原生 Ajax 实现: POST

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function(){
  if (xmlhttp.readyState==4 && xmlhttp.status==200){
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
  }
}
xmlhttp.open("POST","ajax_test.asp",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-
    ↪ urlencoded");
xmlhttp.send("fname=Bill&lname=Gates");
```

- 原生 JS 与 jQuery 对 AJAX 的实现
- AJAX - 向服务器发送请求

Ajax 的 5 个状态

新建对象 -> 建立连接 -> 接收响应原始数据 -> 解析原始数据 -> 响应就绪 (待后续处理)

1. 0: 请求未初始化
   此阶段确认 XMLHttpRequest 对象是否创建，并为调用 open() 方法进行未初始化作好准备。值为 0 表示对象已经存在，否则浏览器会报错——对象不存在。
2. 1: 服务器连接已建立
   此阶段对 XMLHttpRequest 对象进行初始化，即调用 open() 方法，根据参数 (method,url,true) 完成对象状态的设置。并调用 send() 方法开始向服务端发送请求。值为 1 表示正在向服务端发送请求。
3. 2: 请求已接收
   此阶段接收服务器端的响应数据。但获得的还只是服务端响应的原始数据，并不能直接在客户端使用。值为 2 表示已经接收完全部响应数据。并为下一阶段对数据解析作好准备。
4. 3: 请求处理中
   此阶段解析接收到的服务器端响应数据。即根据服务器端响应头部返回的 MIME 类型把数据转换成能通过 responseBody、responseText 或 responseXML 属性存取的格式，为在客户端调用作好准备。状态 3 表示正在解析数据。

5. 4: 请求已完成，且响应已就绪
   此阶段确认全部数据都已经解析为客户端可用的格式，解析已经完成。
   值为 4 表示数据解析完毕，可以通过 XMLHttpRequest 对象的相应属性取得数据。

- Panda, Ajax readyState 的五种状态, LOFTER

3.8.2 Fetch

fetch 和 Ajax (`jQuery.ajax()`) 的区别

- The Promise returned from `fetch()` **won't reject on HTTP error status** even if the response is an HTTP 404 or 500. Instead, it will resolve normally (with ok status set to false), and it will only reject on network failure or if anything prevented the request from completing.
- By default, `fetch` **won't send or receive any cookies** from the server, resulting in unauthenticated requests if the site relies on maintaining a user session (to send cookies, the credentials init option must be set).

```
fetch(api, config)
.then(response => response.json())
.then(data => {
  // handle
})
.catch(err => console.error(err));
```

- Using Fetch, MDN web docs

3.9 异步

1. 回调, 适应于异步处理较少的情况

   ```
   get(url, callback);
   ```

2. Promise, 链式方式写异步处理

   ```
   promise(somthing).then(res => handle)
   ```

3. async 和 await, 同步的方式写异步, 可适用于异步处理之间存在依赖的情况.

```
const promiseFunc = async () => {
  const result1 = await getSomething();
  const result2 = await dependOnResult1(result1);
  return result2;
};
promiseFunc().then(result => {
  console.log(result);
})
```

3.9.1 async 和 await

async, await

扩展

1. 理解 JavaScript 的 async/await, 边城, 2016/11/19

3.10 文档对象 DOM

3.10.1 浏览器事件冒泡和捕获

事件分为三个阶段:

- 捕获阶段
- 目标阶段
- 冒泡阶段

TODO:IE 和 w3c 标准的区别;阻止事件传播 (捕获,冒泡) `e.stopPropagation();` 阻止事件默认行为 `e.preventDefault()`。

```
function registerEventHandler(node, event, handler) {
  if (typeof node.addEventListener == "function")
    node.addEventListener(event, handler, false);
  else
    node.attachEvent("on" + event, handler);
}

registerEventHandler(button, "click", function(){print("Click (2)")
    ↪ ;});
```

- 本期节目, 浏览器事件模型中捕获阶段、目标阶段、冒泡阶段实例详解, segmentfault, 2015.8

3.11 浏览器对象 BOM

3.11.1 弹框

- `prompt(text,defaultText)` 提示用户输入的对话框。`text` 对话框中显示的纯文本，`defaultText` 默认的输入文本。返回值为输入文本。
- `alert(message)` 警告框。`message` 对话框中要实现的纯文本。
- `confirm(message)` 显示一个带有指定消息和 OK 及取消按钮的对话框。`message` 对话框中显示的纯文本。点击确认返回 true，点击取消返回 false。
- `http://www.w3school.com.cn/jsref/met_win_prompt.asp`

3.11.2 localStorage

> 存储在浏览器中的数据，如 localStorage 和 IndexedDB，以源进行分割。每个源都拥有自己单独的存储空间，一个源中的 Javascript 脚本不能对属于其它源的数据进行读写操作。

localStorage 在当前源下设置的值，只能在当前源下查看。

```
localStorage.setItem('key',value) // 设置值，或 localStorage.key =
    ↪ value
localStorage.getItem('key')    // 获取值，或 localStorage.key
localStorage.removeItem('key') // 删除值
localStorage.clear()           // 清空 localStorage
```

- JavaScript 的同源策略, MDN
- localStorage, MDN
- 杜若, localStorage 介绍

3.12 jQuery

3.12.1 jQuery 知识结构

- jQuery 基础: 选择器
- jQuery 效果: hide, show, toggle, fadeIn, fadeOut, animate
- jQuery 操作 HTML: text, html, val, attr, append, after, prepend, before, remove, empty
- jQuery 遍历 Dom: parent, parents, children, find

3.12.2 jQuery 和 Dom 对象相互转换

1. jQuery 对象转换成 Dom 对象: 下标和 get 方法

```
/* 方法一: 下标 */
var $v = $("#v") ; //jQuery对象
var v = $v[0];  //DOM对象

/* 方法二: get */
var $v = $("#v"); //jQuery对象
var v = $v.get(0); //DOM对象
```

2. Dom 对象转 jQuery 对象: 通过 `$()` 把 Dom 对象包装起来实现

```
var v = document.getElementById("v"); //DOM对象
var $v = $(v); //jQuery对象
```

- jQuery 对象与 dom 对象的区别与相互转换

3.12.3 jQuery 命名冲突解决

1. 使用 `jQuery.noConflict()`

```
jQuery.noConflict();
// 之后使用 jQuery 调用, jQuery("#id").methodname()
```

2. 自定义别名

```
var $j = jQuery.noConflict();
// $j("#id").methodname();
```

3. 传入参数，继续使用 `$`

```
jQuery.noConflict();
(function($){
    $("#id").methodname();
})(jQuery)
```

- 邦彦, 谈谈 jQuery 中的防冲突（noConflict）机制, TaoBaoUED
- RascallySnake, Jquery 的 $ 命名冲突, cnblogs
- TerryChen, Jquery 命名冲突解决的五种方案, cnblogs

3.13 其他

1. switch 以 `===` 匹配。
2. 函数会首先被提升，然后才是变量。

4 浏览器

4.1 浏览器兼容性

4.1.1 IE6/IE7/IE8 支持 html5 新标签

创建标签

```
document.createElement('section'); // 其他标签一样处理
```

使用 JS 方案

1. html5shiv:`https://github.com/aFarkas/html5shiv/` (用的也是 createElement())
2. modernizr: `https://github.com/modernizr/modernizr/` (这个功能要多一点，还兼顾 CSS3，更多)

4.1.2 IE 兼容性测试

Modern.IE 提供的虚拟机测试。下载 virtualbox (免费) 或者 VMware (收费)，然后导入对应的下载包 (不同系统 IE 版本不同)。如果是 Windows 10, 虚拟机可启用 Hyper-V.

4.1.3 提示不支持 JavaScript

noscript 元素用来定义在脚本未被执行时的替代内容（文本）。此标签可被用于可识别 `<script>` 标签但无法支持其中的脚本的浏览器。

```
<noscript>Your browser does not support JavaScript!</noscript>
```

reference: HTML noscript 标签, w3school

4.1.4 浏览器 hack

以 IE 为例，展示几个 CSS hack 方法，更多的见参考链接。

IE6

```
.selector { _property: value; }
.selector { -property: value; }
```

IE <= 7 (! $ & * ( )= % + @ , . / ` [ ] # ~ ? : < > |)

```
.selector { !property: value; }
.selector { $property: value; }
.selector { &property: value; }
.selector { *property: value; }
.selector { )property: value; }
.selector { =property: value; }
.selector { %property: value; }
.selector { +property: value; }
.selector { @property: value; }
.selector { ,property: value; }
.selector { .property: value; }
.selector { /property: value; }
.selector { `property: value; }
.selector { ]property: value; }
.selector { #property: value; }
.selector { ~property: value; }
.selector { ?property: value; }
.selector { :property: value; }
.selector { |property: value; }
```

IE 6-8

```
.selector { property: value\9; }
.selector { property/*\**/: value\9; }
```

reference: `http://browserhacks.com/`

4.1.5 CSS3 前缀

虽然目前 CSS3 得到广泛支持，但各个浏览器厂商对标准实现并不完全一样，可以对 CSS3 使用前缀。建议把特殊的 CSS 语句放在前面，一般的语句放置在后面。可以使用 Autoprefixer 自动添加前缀。

- `-moz-` Firefox,
- `-webkit-` Safari, Chrome
- `-o-` Opera (不过, Opera 和 Chrome 现在都采用 Blink 内核)
- `-ms-` Internet Explorer

4.1.6 浏览器/设备检测

通过 js 检测浏览器方法包括：条件语句检测, 依靠浏览器各自的特性检测；通过 `userAgent` 检测。

IE 条件语句 可以添加条件语句检测浏览器，针对性的编写样式。

```
<!--[if IE 8]>
<body class="ie8">
<![endif]-->
<!--[if !IE]>
<body class="notie">
<![endif]-->
```

reference: About conditional comments, msdn

userAgent **userAgent 示例**

```
Chrome 60
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
    ↪ , like Gecko) Chrome/60.0.3112.113 Safari/537.36

360 安全浏览器/极速模式
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
    ↪ like Gecko) Chrome/55.0.2883.87 Safari/537.36 QIHU 360SE

360 安全浏览器/IE 兼容模式
Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0
    ↪ E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR
    ↪ 3.5.30729; rv:11.0) like Gecko

IE 11
Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0
    ↪ E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR
    ↪ 3.5.30729; rv:11.0) like Gecko

Edge 40
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
    ↪ , like Gecko) Chrome/52.0.2743.116 Safari/537.36 Edge
    ↪ /15.15063
```

userAgent 判别浏览器

可直接查看此插件的 js 代码:jquery-browser-plugin。实测了 IE、Firefox、Opera、Chrome，输出结果都是对的。

```
<html>
<head>
    <meta charset="UTF-8">
```

```
    <title>Document</title>
    <script src="jquery.browser.js"></script>
</head>
<body>
<script>
    console.log("jQBrowser.webkit: " + jQBrowser.webkit);
    console.log("jQBrowser.mozilla: " + jQBrowser.mozilla);
    console.log("jQBrowser.msie: " + jQBrowser.msie);
    console.log("jQBrowser.version: " + jQBrowser.version);
</script>
</body>
</html>
```

userAgent 判别设备

通过检测设备, 可以针对不同设备提供不同功能, 并处理不同设备的兼容需求.

```
// 检测 IOS
ios = /iphone|ipod|ipad/i.test(window.navigator.userAgent);
// 检测 Android
android = /android/i.test(window.navigator.userAgent);
```

通过特性检测判断 reference: 露兜博客, 检测访客浏览器的 2 种方法, 2009

4.1.7 IE 图标字体未加载降级处理 (IE icon)

IE 下图标字体未加载, 导致图标无法显示, `element-ui` 就存在这个情况. 可以判断浏览器是否是 IE, 若果是则加载覆盖的样式, 用文本或其他字符代替图标. 例如通过下面的方式, 修复表格中操作图标无法展示.

```
.el-table .el-icon-delete:before {
  content: '删除';
}
```

```
const ua = window.navigator.userAgent.toLowerCase()
// 不包含 Edge
let ifIE = ua.indexOf('msie') > -1 || ua.indexOf('trident') > -1
if (ifIE) {
  // 依赖 loadjs
  loadjs('/static/css/iefix.css')
}
```

4.2 本地缓存

- 详说 Cookie, LocalStorage 与 SessionStorage

5 HTTP

- http 中 get 和 post；
- HTTP 格式: 方案://服务器位置/路径, `<scheme>://<user>:<password>@`
  ↪ `<host>:<port>/<path>;<params>?<query>#<frag>`

5.1 同源

> 如果两个页面拥有相同的**协议**（protocol），**端口**（如果指定），和**主机**，那么这两个页面就属于同一个源（origin）。
>
> 来自 about:blank，javascript: 和 data:URLs 中的内容，继承了将其载入的文档所指定的源，因为它们的 URL 本身未指定任何关于自身源的信息。

注意 IE 的区别（比如：IE 未将端口号加入到同源策略的组成部分之中）。

reference: JavaScript 的同源策略, MDN

5.2 从输入 URL 到页面加载完成的过程

1. 输入地址
2. 浏览器查找域名的 IP 地址。这一步包括 DNS 具体的查找过程，包括：浏览器缓存-> 系统缓存-> 路由器缓存...
3. 浏览器向 web 服务器发送一个 HTTP 请求
4. 服务器的永久重定向响应（从 http://example.com 到 http://www.example.com）
5. 浏览器跟踪重定向地址
6. 服务器处理请求
7. 服务器返回一个 HTTP 响应
8. 浏览器显示 HTML
9. 浏览器发送请求获取嵌入在 HTML 中的资源（如图片、音频、视频、CSS、JS 等等）
10. 浏览器发送异步请求

reference: 从输入 URL 到页面加载完成的过程中都发生了什么事情？, segmentfault

5.3 状态码

整体范围	已定义范围	分类
100 ~ 199	100 ~ 101	信息提示
200 ~ 299	200 ~ 206	成功
300 ~ 399	300 ~ 305	重定向
400 ~ 499	400 ~ 415	客户端错误
500 ~ 599	500 ~ 505	服务器错误

状态码	原因短语	含义
100	Continue	说明收到了请求的初始部分, 轻客户端继续. 发送了这个状态码之后，服务器在收到请求之后必须进行响应。
101	Switching Protocols	说明服务器正在根据客户端的指定，将协议切换成 Update 首部所列的协议
200	OK	请求没问题，实体的主体部分包含了所请求的资源
201	Created	用于创建服务器对象的请求 (比如, PUT)。响应的实体主体部分中应该包含各种引用了已创建的资源 URL, Location 首部包含的则是最具体的引用。

状态码	原因短语	含义
202	Accepted	请求已被接受，但服务器还未对其执行任何动作。不能保证服务器会完成这个请求; 这只是意味着接收请求时, 他看起来是有效的，服务器应该在实体的主体中包含对请求状态的描述，或许还应该有队请求完成时间的估计
203	Non-Authoritative Information	实体首部包含的信息不是来自于源服务器，二是来自资源的一份副本。如果中间节点上有一份资源副本，但无法或者没有对它所发送的与资源有关的元星系进行验证，就会出现这种情况。
204	No Content	响应报文中包含若干首部和一个状态行，但没有实体的主体部分。主要用于在浏览器不转为显示新文档的情况下，对其进行更新
205	Reset Content	另一个主要用于浏览器的代码。负责告知浏览器清除当前页面中的所有 HTML 表单元素
206	Partial Content	成功执行了一个部分或 Range 请求。

状态码	原因短语	含义
300	Multiple Choices	客户端请求一个实际指向多个资源的 URL 时会返回这个状态码，比如服务器上有某个 HTML 文档的英语和法语版本。
301	Moved Permanently	在请求的 URL 已被移除时使用。响应的 Location 首部中应该包含资源现在所处的 URL
302	Found	与 301 状态码类似; 但是，客户端应该使用 Location 首部给出的 URL 来临时定位资源。将来的请求仍使用老的 URL
303	See Other	告知客户端应该用另外一个 URL 来获取资源。新的 URL 位于响应报文的 Location 首部。其主要目的是允许 POST 请求的响应将客户端丁香到某个资源上去

状态码	原因短语	含义
304	Not Modified	客户端可以通过所包含的请求首部，使其请求变成有条件的。如果客户端发起了一个 GET 请求，而最近资源未被修改的话，就可以用这个状态码来说明资源未被修改。带有这个状态码的响应不应该包含实体的主体部分。
305	Use Proxy	用来说明必须通过一个代理来访问资源; 代理的位置有 location 首部给出。很重要的一点是，客户端是相对某一个特定的资源来解析这条响应的，不能假定所有请求，甚至所有持有所请求资源的服务器的请求都通过这个代理进行。如果客户端错误的让代理介入了某条请求，可能会引发破坏性的行为，而且会造成安全漏洞。
306	(未使用)	当前未使用

状态码	原因短语	含义
307	Temporary Redirect	与 301 状态码类似; 但客户端应该使用 Location 首部给出 URL 来临时定位资源。将来的请求应该使用老的 URL
400	Bad Request	用于告知客户端它发送了一个错误的请求
401	Unauthorized	与适当的首部一起返回，在这些首部中请求客户端在获取对资源的访问权之前，对自己进行认证。
402	Payment Required	现在这个状态码还未使用，但已经被保留，一做未来之用
403	Forbidden	用于说明请求被服务器拒绝了。如果服务器想说明为什么拒绝请求，可以包含实体的主体部分来对原因进行描述。但这个状态码通常是在服务器不想说明拒绝原因的使用使用的。
404	Not Found	用于说明服务器无法找到所请求的 url。通常会包含一个实体一般客户端应用程序显示给用户看。

状态码	原因短语	含义
405	Mehtod Not Allowed	发起的请求中带有所请求的 url 不支持的方法时，使用此状态码。应该在响应中包含 Allow 首部，已告知客户端对所请求的的资源可以使用哪些方法。
406	Not Acceptable	客户端可以指定参数来说明他们愿意接收什么类型的实体。服务器没有与客户端可以接受的 url 相匹配的资源时，使用此代码。通常服务器会包含一些首部，以便客户端弄清楚为什么请求无法得到满足。
407	Proxy Authentication Required	与 401 类似，但用于要求对资源进行认证的代理服务器。
408	Request Timeout	如果客户端完成请求所化的时间太长，服务器可以回送此状态码，并关闭连接。超时时常随服务器的设置不同而不同，但通常对所有的合法请求来说，都是够长的。

状态码	原因短语	含义
409	Confict	用于说明请求可能在资源上引发一些冲突。服务器担心会引发一些冲突时，可以发送此状态码。响应中应该包含描述冲突的主体。
410	Gone	与 404 类似，只是服务器曾经拥有过此资源。主要用于 Web 站点的维护，这样服务器的管理者就可以在资源被移除的情况下通知客户端了。
411	Length Required	服务器要求在请求报文中包含 Content-Length 首部时使用。
412	Precondition Failed	客户端发起了条件请求，且其中一个条件失败了的时候使用。
413	Request Entity Too Large	客户端所发送请求中的请求 url 比服务器能够或者希望处理的要大时，使用此状态码。
414	Request Uri Too Long	客户端所发送请求中的请求 url 比服务器能够或者希望处理的要长时，使用此状态码。
415	Unsupported Media Type	服务器无法理解或无法支持客户端所发实体的内容类型时，使用此状态码。

状态码	原因短语	含义
416	Requested Range Not Satisfiable	请求报文所请求的是指定资源的某个范围，而此范围无效或无法满足时，使用此状态码
417	Expectation Failed	请求的 Expect 请求首部包含了一个期望，但服务器无法满足此期望时，使用此状态码
500	Internal Server Error	服务器遇到了一个妨碍它为请求提供服务的错误时，使用此状态码
501	Not Implemented	客户端发起的请求超出服务器的能力范围时比如使用了服务器不支持的请求方法)，使用此状态码
502	Bad Gateway	作为代理或网关使用的服务器从请求响应链的链路上收到了一条伪响应 (比如，它无法连接到其父网关) 时，使用此状态码
503	Service Unavailable	用来说明服务器现在无法为请求提供服务，当将来可以。如果服务器知道什么时候资源会变为可使用的，可以在响应中包含一个 Retry-After 首部。

状态码	原因短语	含义
504	Gateway Timeout	与状态码 408 类似，只是这里的响应来自一个网关或代理，他们在等待另一个服务器对其请求进行相应是超时了
505	http Version Not Supported	服务器收到的请求使用了它无法或不愿支持的协议版本时，使用此状态码。有些服务器应用程序会选择不支持协议的早期版本