布兰登·艾奇（Brendan Eich，1961年～），JavaScript 的发明人，2005年至 2014年期间，在Mozilla 公司担任首席技术长（Chief Technology Officer）。出任Mozilla的 CEO 十天就被迫辞职。

目标

- [ ] 了解 js 组成以及特点
- [ ] 掌握 js 的编写位置
- [ ] 掌握 js 的基本语法(和 Java 一样)
- [ ] 掌握 js 变量定义
- [ ] 掌握 js 常用数据类型
- [ ] 掌握 js 的函数定义和调用

- [ ] 了解 js 的面向对象内容
- [ ] 掌握 js 中数组的定义和使用

1_JavaScript入门

1.1_JavaScript 基本介绍

简介

JavaScript（简称"JS"） 是一种具有函数优先的轻量级，解释型或即时编译型的编程语言。虽然它是作为开发Web 页面的脚本语言而出名的，但是它也被用到了很多非浏览器环境中，JavaScript 基于原型编程、多范式的动态脚本语言，并且支持面向对象、命令式和声明式（如函数式编程）风格。

JavaScript 在1995年由 Netscape 公司的布兰登·艾奇，在网景导航者浏览器上首次设计实现而成。因为 Netscape与Sun合作，Netscape 管理层希望它外观看起来像 Java，因此取名为 JavaScript。但实际上它的语法风格与 Self 及Scheme 较为接近。

JavaScript 的标准是 ECMAScript 。截至 2012 年，所有浏览器都完整的支持 ECMAScript 5.1，旧版本的浏览器至少支持 ECMAScript 3 标准。2015 年 6 月17日，ECMA国际组织发布了 ECMAScript 的第六版，该版本正式名称为 ECMAScript 2015，但通常被称为 ECMAScript 6 或者 ES6。

组成部分

JavaScript 的内容，包含以下三部分：

1. **ECMAScript（核心）**： JavaScript 语言基础(规定了 JavaScript 脚本的核心语法，如 数据类型、关键字、保留字、运算符、对象和语句等，它不属于任何浏览器；
2. **DOM（文档对象模型）**： 规定了访问 HTML 和 XML 的接口（提供了访问 HTML 文档（如 body、form、div、textarea 等）的途径以及操作方法）；Node：Document、Element、Attr、Text

3. BOM（浏览器对象模型）：提供了独立于内容而在浏览器窗口之间进行交互的对象和方法(提供了访问某些功能（如浏览器窗口大小、版本信息、浏览历史记录等）。

JavaScript 的特点

1. 是一种解释性脚本语言（代码不进行预编译）；
2. 主要用来向 HTML（标准通用标记语言下的一个应用）页面添加交互行为；
3. 可以直接嵌入 HTML 页面，但写成单独的 js 文件有利于结构和行为的分离。

1.2_JS 代码编写的位置

html 和 JS 都是直接交给浏览器去执行和渲染, 每个浏览器中都有对应的 JS 引擎

1. **直接编写在 HTML 的 script 标签中**

   script 标签可以存放在 html 标签中的任何位置，但是推荐写在 head 标签里面。

   注意: HTML 中js 的所有代码都是顺序执行,出错则不再往下执行.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>JS可编写的位置</title>
<script type="text/javascript">
    alert("Hello JavaScript!");
</script>
</head>
<body>
</body>
</html>
```

2. **编写在 JS 文件中**

   单独编写一个文件 (*.js) 来存放 Javascript 代码，然后在需要使用的 html 页面直接引入该 js 文件。

   好处：可以使 js 文件和 HTML 文件相分离、一个 js 文件可被多个 HTML 文件使用、维护起来也更方便等等。

index.js: 编写JS代码

```
//这里只能编写JS代码
alert("Hello JavaScript");
```

index.html: 引入 JS 文件

```
<!--使用 script 标签的 src 属性加载指定的 js文件-->
<script type="text/javascript" src="index.js"></script>
```

注意: 方式 2 不能跟方式 1 杂糅使用;

方式 2 中 script 标签是双标签,不能弄成单标签。

下面是错误演示

```
<script type="text/javascript" src="index.js">
    alert("script标签不能同时引入js文件和编写其他js代码");
</script>
```

1.3_JS 的基本语法

1. JavaScript 中的标识符

   变量，常量，函数，语句块也有名字，我们统统称之为标识符。标识符可以由任意顺序的大小写字母、数字、下划线(_)和美元符号($)组成，标识符不能以数字开头，不能是 JavaScript 中的保留字或关键字。

合法的标识符举例：indentifier、username、user_name、_userName、$username；
非法的标识符举例：int、8.3、Hello World。

2. JavaScript 严格区分大小写
   username 和 userName 是两个完全不同的符号。
3. JavaScript 程序代码的格式
   每条功能执行语句的最后必须用分号；结束

   每个词之间用空格、制表符、换行符或大括号、小括号这样的分隔符隔开 。

   语句块使用{}来表示。
4. JavaScript 程序的注释
   `/*...*/`中可以嵌套“//”注释，但不能嵌套“ `/*...*/`”。 、 `/**..文档注释.*/`
   和 Java 对比起来学习,相同的直接过掉,重点记忆理解不同点
5. JS 中的关键字和保留字

break	do	instanceof	typeof
case	else	new	var
catch	finally	return	void
continue	for	switch	while
debugger*	function	this	with
default	if	throw	delete
in	try		

abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchronized
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	

1.4_变量

1. **作用**
   系统为之设置一个标识，程序可以用变量名来指向具体的对象内存，并通过变量名来获得对应的对象。

2. **声明变量**：**JS 是弱类型的语言**,未严格区分数据类型.

   使用 var 关键字，例如：var username; 若变量没有初始化，默认是 undefined。

   也可以在声明变量的同时为其赋值, 例如 ： var username = "小狼";

   单独对变量赋值（注意类型）,例如：username =18;

   变量的类型由值来决定,值是什么类型,则变量就为什么类型.

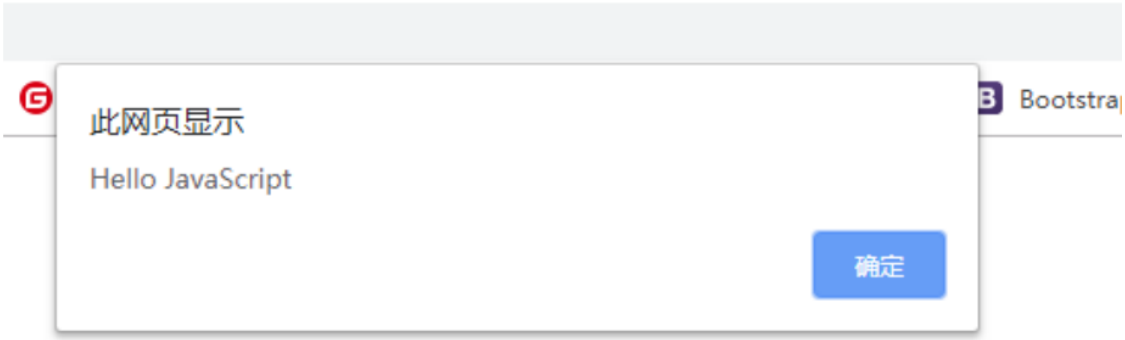
   注意:

   1 不事先声明变量而直接使用会报错;

   提示：JavaScript 定义变量无需指定类型，任何类型都使用 var 声明，可以将 var 理解为 Java 中的 Object ,可以接收任何类型的值.

3. **打印变量**

   方式 1： alert(变量名); 直接在浏览器中弹出对话框 (需点击确定,麻烦,不利于语法实践操作和测试)



   方式 2(推荐): console.log(变量名); （info、**log**、debug、warn、error）: 在浏览器控制台打印内容

所有的方法都是在控制台打印内容,只是显示效果有所差异,这里咱们统一使用 log 来显示即可,等价于 Java 中的: System.out.println();

console.log(); 可类似 Java 的可变参数,一次传入多个数据, console.log(username,age);

1.5_数据类型

概述

JavaScript 中变量似乎很简单，因为它声明变量只需要一个 “var” 就可以，不像其他编程语言严格区分了数据类型（int/double/char/boolean...）。这样做也是有好处的，变量可以被赋予任何类型的值，同样也可以给这个变量重新赋予不同类型的值。并不是“一定终身”。

可是，JavaScript 并没有避开数据类型，只是在声明时统一使用无类型（untyped）的“var”关键字而已，它的数据类型是根据所赋值的类型来确定的。

简单类型: String，Number，Boolean，Null，undefined

对象类型: Object, Array, Function

Object 对象自身用处不大，不过在了解其他类之前，还是应该了解它。因为 ECMAScript 中的 Object 对象与 Java 中的 java.lang.object 相似，ECMAScript 中的所有对象都由这个对象继承而来，Object 对象中的所有属性和方法都会出现在其他对象中，所以理解了 Object 对象，就可以更好地理解其他对象。

常用的数据类型

1. Number 数值
    1. 整数常量,如: 100
    2. 实数常量, 如: 12.32、193.98、 5E7、4e5等。
    3. 特殊数值(了解)：NaN、Infinity (除数为零)，所对应的判断函数 isNaN()、isFinite()
2. Boolean 布尔

    两种值: true 和 false。
3. String 字符串

    值可以使用单引号或者双引号括起来。注意 js 中没有 char 类型，所以 'a' 也是一个字符串。
4. 其他常量

    null 常量、undefined 常量 （定义未赋值）。

typeof 运算符：在 js 中获取变量的数据类型

```
var msg ='hello';
console.log(typeof msg );//string
msg = 19;
console.log(typeof msg);
msg=18;
console.log(typeof msg );//number
```

注意: 1 在 js 中只有" **var 变量名 = 值** ", 不存在 " **变量类型 变量名 = 值** "

```
// 正例
var age = 10;
// 反例
Number age = 10;
```

2 变量名不能使用name

name 是自带的一个成员变量,类型无法修改,所以结果一直为 string

1.5_运算符

JS 中的大部分运算符都和 Java 中的一样,这里只说几个不一样的运算符

1. **比较运算符**
   = 和 == 以及 === 之间的区别:
   = 赋值运算符：用于把一个常量/变量的值赋值给另外一个变量
   == 比较运算符：用于比较两个数据的值是否相等，不会去判断类型
   console.log("18" == 18);
   === 比较运算符：先判断数据类型是否相等，然后再去判断值是否相等
   console.log("18" === 18);

2. **逻辑运算符**

   在逻辑运算中 0、""、false、NaN、undefined、null 表示为 false，其他类型数据都表示 true。
   a && b 将 a, b 先转换为 Boolean 类型，在执行逻辑与，若 a 为 false，则返回 a，否则就返回 b；

a || b 将 a, b 先转换为 Boolean 类型，再执行逻辑或，若 a 为 true，则返回 a，否则就返回 b。

&& 和 || 运算符的区别

&& 操作：从左往右依次判断，返回第一个为 false 的值，否则返回最后一个为 true 的值；

&& 找 false ,找到则返回对应的值,直到最后一个如果没有找到,则返回最后一值.

|| 操作：从左往右依次判断，返回第一个为 true 的值，否则返回最后一个为 false 的值。

|| 找 true ,找到则返回对应的值,直到最后一个如果没有找到,返回最后一个值.

完成下面的练习:

```
console.log(true && true);
console.log(1 && true);
console.log(1 && 2);
console.log("A" && 2);
console.log("" && 2);
console.log(null && "B");
console.log("A" && "B");
console.log(1 && 2 && 3);
console.log(1 && null && 3);
console.log("" && null && 0);
```

```
    console.log(true || true);
    console.log(1 || true);
    console.log(1 || 2);
    console.log("A" || 2);
    console.log("" || 2);
    console.log(null || "B");
    console.log("A" || "B");
    console.log(1 || 2 || 3);
    console.log(1 || null || 3);
    console.log("" || null || 0);
```

经典应用场景：

```
    function func(x){
            x = x || 0;
            console.info(x);  //如果x有为true的值,则返回x的值,否则返回0
    }
```

2_函数

2.1_函数概述

作用和 Java 中的方法一样,都是为了实现代码的复用

将脚本编写为函数，避免页面载入时就执行该脚本。

写在函数里面的 js 代码，只需要定义一次，就可以多次调用，提高代码的复用性。

2.2_函数的定义

1. 普通函数语法:

```
// Java 中方法的定义
/*
public String doWork(String name){
    return name;
}
*/

// JS中不用指定函数的返回值类型(无论怎样都有返回,JS是弱类型语言)
function 函数名([参数名称1, 参数名称2, ..., 参数名称N]){
    //程序代码
    [return 值;]
}
```

参数和返回值都是可选的, 由自己的实际需求决定,
如果函数没有返回值，就默认返回 undefined.

案例:

```
// 普通函数定义,有参有返回
function doWork1(String name){
    return name;
}

// 普通函数定义,无参无返回
function doWork2(){
    console.log(name);
}
```

2. 定义匿名函数语法:

```
var 变量名 = function([参数名称1, 参数名称2, ..., 参数名称N]){
        //程序代码
    [return 值;]
}
```

案例:

```
var add = function(x, y){
        return x+y;
}
```

一般定义一个函数时，我们会给函数起一个固定的名称，我们称之为函数名。

匿名函数是指没有固定的函数名称，通常情况是定义一个变量接收该匿名函数。

实际开发中用的非常之多，可以把匿名函数看成一个普通的值来理解。

3. 定义箭头函数

箭头函数其实类似匿名函数，把匿名函数的 function 去除，在参数右边加 `=>` 即可,简洁，但可读性差。

```
// 无参单行函数体的箭头函数
() => console.log("箭头函数");
// 等价如下
/*
function (){
    console.log("箭头函数";
}
*/
// 无参多行函数体的箭头函数
() => {
    console.log("函数体1");
    console.log("函数体2");
};
// 一个形参数的箭头函数
username => console.log(username);
// 多个参数多行函数体
(username,age) => {
    console.log(username);
    console.log(age);
}
```

注意：箭头函数没有自己的 this,arguments,super 或 new.target

2.3_函数的调用

函数定义好之后不会立即执行, 需要执行到调用函数的代码时才会执行

语法:

```
var 变量 = 函数名(实参1,实参2,实参3,...);
```

方法如果有返回值, 那么变量接收到的结果则是实际返回的值

方法如果没有返回值,那么变量的值为 undefined, 此时通常不用去接收.

```
// 普通函数定义,有参有返回
function doWork1(String name){
    return name;
}
// 普通函数定义,无参无返回
function doWork2(){
    console.log(name);
}

// 匿名函数定义，把没有名称的函数赋值给一个变量（当变量的值）
var add = function(x, y){
       return x+y;
}

// 多个参数多行函数体
var fun1 = (username,age) => {
    console.log(username);
    console.log(age);
}

// 普通函数调用
var ret1 = doWork1("小狼");
var ret2 = doWork2("小码");
console.log(ret1);      // 小狼
console.log(ret2);      // 小码 -->  undefined

// 匿名函数调用
add(1,3); // 直接把变量作为函数名去调用匿名行数
```

```
// 把匿名函数作为变量的值传递给另一个函数（printAddResult），在另一个函数中来调用
function printAddResult(add){
    console.log(add(3,5));
}
// 调用printAddResult 时带入匿名函数add
printAddResult(add);    // 传入已定义的函数

// 调用 printAddResult 时重新定义函数
printAddResult(function (x,y){
    return x + y;
});

// 箭头函数的定义
fun1("小丽",18);
// 可以将箭头函数作为匿名函数使用
printAddResult((x,y) =>{
    return x-y;
});
```

注意: 调用函数时, 实参和定义函数的形参可以不一致

```
add();          // add 函数被调用，但是参数都为undefined
add(1,3,5,6);   // 只会只有前两个，后面的自动忽略
```

查看函数被调用时带了哪些参数：arguments

```
function doWork3(){
    console.log(arguments); // 每个函数都内置了一个arguments属性，用于存放调用时带入的实际参数值
}

doWork3(1,2,3);.
doWork3();
doWork3("小狼","小码",18);
```

注意： 箭头函数没有 arguments；

3_面向对象

JavaScript 提供了一个构造函数（Constructor）模式。 所谓"构造函数"，其实就是一个普通函数，但是内部使用了 this 变量。对构造函数使用 new 运算符，就能生成实例，并且 this 变量会绑定在实例对象上。加 new 执行的函数构造内部变化：自动生成一个对象，this 指向这个新创建的对象，函数自动返回这个新创建的对象

构造函数(类)定义：

```
//如何使用JS来定义一个类(构造函数)
function Person(name, age){
    //添加属性
    this.name = name;
    this.age = age;
    //添加方法
    this.sleep = function(){
        console.log("困了,睡一觉");
    }
}
```

创建对象和操作成员：

```
// 使用构造函数来创建对象
var p = new Person("小狼", 10);
console.log(p);
// 访问对象中的成员
console.log(p.name);
console.log(p.age);
// 访问对象中的方法
p.sleep();

// JS中,可以单独为某一个对象添加成员
p.xxx = "ooo";
console.log(p);
```

```
var p2 = new Person("xxx",10);
console.log(p2);
```

构造函数注意事项:

此时 Person 称之为构造函数，也可以称之类，构造函数就是类 。

p1, p2 均为 Person 的实例对象。

Person 构造函数中 this 指向 Person 实例对象, 即 new Person( ) 出来的对象。

构造函数首字母大写，这是规范，官方都遵循这一个规范，如 Number() Array()。

创建对象的快捷方式：字面式创建对象

```
var obj = {user_name:"小狼",age:"18"};
console.log(obj.username);
console.log(obj.age);
```

注意：属性名如果有特殊字符需要加引号括起来。

内置对象(了解)

参见 W3C javascript.chm ---> ECMAScript 对象类型。

1. Object
    - constructor

      对创建对象的函数的引用（指针）。对于 Object 对象，该指针指向原始的 Object() 函数。
    - Prototype

      对该对象的对象原型的引用。对于所有的对象，它默认返回 Object 对象的一个实例。

Object 对象还具有几个方法：

- hasOwnProperty(property)

  判断对象是否有某个特定的属性。必须用字符串指定该属性。（例如，o.hasOwnProperty("name")）

- IsPrototypeOf(object)

  判断该对象是否为另一个对象的原型。

- PropertyIsEnumerable

  判断给定的属性是否可以用 for...in 语句进行枚举。

- ToString()

  返回对象的原始字符串表示。对于 Object 对象，ECMA-262 没有定义这个值，所以不同的 ECMAScript 实现具有不同的值。

2. Date

   - Date() 返回当前日期和事件
   - getFullYear()　获取Date对象中四位数字的年份
   - getMonth()　获取Date对象中的月份(0~11)
   - getDate()　获取Date对象中的天(1~31)
   - getHours()　获取Date对象中的小时
   - getMinutes()　获取Date对象中的分钟
   - getSeconds()　获取Date对象中的秒

```
var d = new Date();
var time = d.getFullYear() + "-" + (d.getMonth()+1) +
"-" + d.getDate() + " " + d.getHours() + ":" +
d.getMinutes() + ":" + d.getSeconds();
```

3. String

   属性:

   - length 字符个数

   方法

   - charAt(index) 返回指定位置的字符
   - concat(string1,string2,string3,....)　拼接字符串
   - fromCharCode(num)　可接受一个指定的 Unicode 值，然后返回一个字符串

- subString(start,stop) 提取字符串中两个指定的索引号之间的字符。

4_数组

4.1_数组的定义方式

- 创建数组对象
  var arr = new Array();
- 类似数组中的静态初始化
  var arr2 = new Array("西施","王昭君","貂蝉","杨贵妃");
  var arr2 = new Array(1, 2, 3, 4);
- 类似数组的动态初始化
  var arr3 = new Array(4);//这里的4是数组的长度,而不是元素
- 简写

  var arr4 = ["西施", "王昭君", "貂蝉", "杨贵妃"];

注意:
对于 js 中的数组不会出现数组越界的异常，也不是定长的。

4.2_数组中的常用属性和方法

1. 属性
   length 属性，获取数组长度。
2. 方法
   concat(array1, array2, ..., arrayX)：连接两个或更多的数组，并返回结果。
   join(separator)：把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。
   reverse()：颠倒数组中元素的顺序（该方法会改变原来的数组，而不会创建新的数组）。
   slice(start[, end])：从某个已有的数组返回选定的元素（返回的是一个新数组）。

pop()：删除并返回数组的最后一个元素。
shift()：删除并返回数组的第一个元素。
push(newelement1, newelement2, ..., newelementX)：向数组的末尾添加一个或更多元素，并返回新的长度。
unshift(newelement1, newelement2, ..., newelementX)：向数组的开头添加一个或更多元素，并返回新的长度。
splice(index, howmany, element1, ..., elementX)：用于插入、删除或替换数组的元素。

4.3_数组的遍历方式

1. for
   最简单的一种循环遍历方法，也是使用频率最高的一种

```
var arr = ["A","B","C","D"];
for(var i = 0; i< arr.length; i++){
    console.log(arr[i]);
}
```

2. forEach

   数组里的元素个数有几个，该方法里的回调函数就会执行几次
   第一个参数是数组里的元素

   第二个参数为数组里元素的索引

   第三个参数则是遍历的数组本身
   数组自带的遍历方法，虽然使用频率略高，但是性能仍然比普通循环略低

```
var arr = ["A","B","C","D"];
arr.forEach(function(item,index,array){
    console.log(item);
});
```

   注意: 函数中的 arguments 存了调用函数时带入的实际数据,可通过自带属性获取带入的数据.

3. map

遍历数组中的每个元素,将回调函数中的返回值存到一个新的数组中,并返回

```
var arr = ["A","B","C","D"];
var newArr = arr.map(function(item,index,array){
    return item+index;
});
//newArr中的数据:["A1","B2","C3","D4"]
```

4、for-in

可以遍历数组（获取数组的索引）；也可以遍历对象（获取对象的属性名）。

```
//如果是遍历数组,i对应的是数组的索引
var arr = ["A","B","C","D"];
for(var i in arr){
    console.log(i);       //索引
    console.log(arr[i]); //元素
}

//如果是遍历对象,name对应的是属性名
var obj = new Object();
// 给 obj 对象增加属性
obj.username = "小狼";
obj.age = 18;

for(var name in obj){
    console.log(name);    //属性名
    console.log(obj[name]);
}
```

小结

js 代码编写的位置 : html script 标签 , 把 js 代码抽取到 js 文件中

变量的定义和使用 "" true function(){}

运算符

```
1. = == ===
```

2. && ||

函数的定义和调用(**重中之重**)

普通函数定义

匿名函数的定义

函数调用： 函数名(实际参数);

理解到匿名函数的使用(难点)

构造函数/创建对象/对象的访问

数组的定义(简写方式)

数组的遍历(普通)

需要理解的是 forEach / map

5_DOM

5.1 DOM的概述

DOM 是 Document Object Model 文档对象模型的缩写。根据 W3C DOM 规范，DOM 是一种与浏览器，平台，语言无关的接口，可动态地修改 XML 和 HTML。

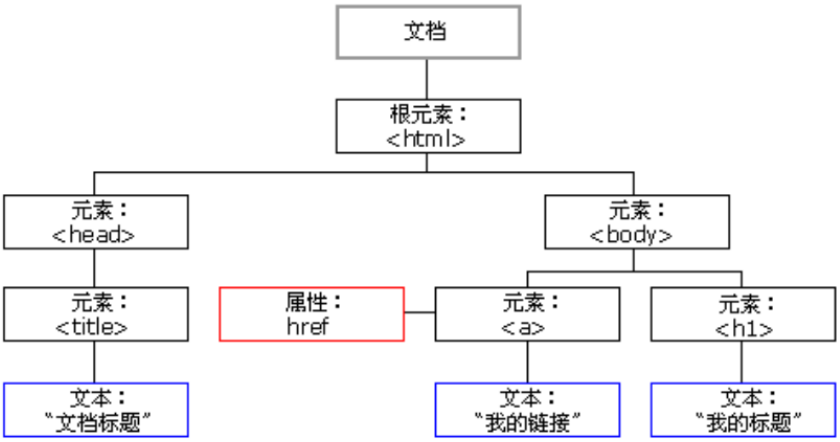
D：文档 – HTML文档 或 XML 文档
O：对象 – document 对象的属性和方法
M：模型

今天我们讲的是 HTML DOM。

2、HTML DOM

DOM 是将 HTML 文档表达为树结构， 定义了访问和操作 HTML 文档的标准方法；
DOM 树：节点（node）的层次。文档节点（document）、元素节点、属性节点、文本节点；

DOM 把一个文档表示为一棵家谱树（父，子，兄弟）。



5.2 获取元素

两个核心点:

1 什么时候可以去获取元素？ 只有元素被加载到 document 中才可以获取

2 如何去获取元素?

元素获取方式:

1. 通过元素 Id
   getElementById，返回拥有指定 id 的第一个元素，如果不存在则返回 null
2. 通过标签名字
   getElementsByTagName，返回一个包括所有给定标签名称的元素集合，如果没有匹配的元素，返回一个空集。
3. 通过 class 名字
   getElementsByClassName，返回一个包含所有指定class名称的元素集合，可以在任意元素上调用该方法。

```
    <script type="text/javascript">
        // 获取元素对象，首先得获取到文档对象 document
        // 当html 被加载时就会自动创建一个document对象
        // 每加载一个元素就存到documen
        // 文档加载完成去执行函数
        window.onload = function () {
            // 通过id 去获取 div元素对象，id 在html 元素中必须值唯一
            var divEL = document.getElementById("div1");
            console.log(divEL);

            // 2 通过标签的名称去获取元素对象,得到是一个集合,
            var divEles = document.getElementsByTagName("div");
            // 通过 [index]获取某个元素对象
            console.log(divEles[0]);

            // 通过 class 来获取元素对象,class 的值是可以重复的
            var clzEle = document.getElementsByClassName("div3");
            console.log(clzEle);
        }
    </script>
</head>
<body>
    <div id="div1" class="div3">div1</div>
    <div id="div2" class="div3">div2</div>
</body>
```


5.3 元素节点的属性操作

- **操作标准属性**

  获取属性值

  元素对象[“属性名”]

  元素对象.属性名

  元素对象.getAttribute("属性名")

  设置属性值:

  元素对象[”属性名”] = 值

  元素对象.属性名 = 值

  元素对象.setAttribute("属性名", 值)

- **操作自定义属性**

  获取属性值:

  元素对象.getAttribute("属性名")

  设置属性值:

  元素对象.setAttribute("属性名", 值)

```
<script type="text/javascript">
    window.onload = function(){
        var div1 = document.getElementById("div1");
        //访问属性
        //标准属性 元素对象.属性名
        console.log(div1.id);
        //如果属性名中包含一些特殊字符(.  -)
        console.log(div1["xxx.ooo"]);
        console.log(div1.getAttribute("id"));
        console.log(div1.getAttribute("class"));
        //class属性,如果使用.去访问的时候,名称className
        console.log(div1.className);

        //操作元素的样式中的属性
        console.log(div1.style["background-color"]);
        console.log(div1.style.backgroundColor);

        //为属性设值
        div1.id = "div2";
        div1["id"] = "div3";
        div1.setAttribute("id","div4");
        // 自定义属性只能使用setAttribute 来给属性设置值
        div1.setAttribute("xxx","ooo");


        //自定义属性
        console.log(div1.xxx);
        console.log(div1["xxx"]);
        console.log(div1.getAttribute("xxx"));

        //属性名和属性值相同的属性,此时使用JS获取到的属性值是
true/false
        //checked="checked"
        //selected="seleted"
        //readonly="readonly"
        var cb = document.getElementById("cb");
        console.log(cb.checked);
    }
</script>
```

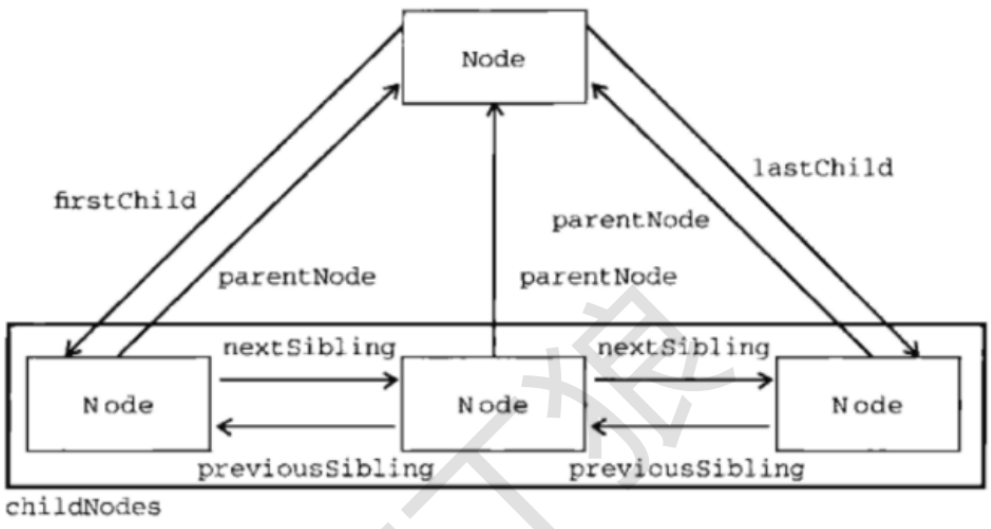
```
<body>
    <!-- 元素 -->
    <div id="div1" class="div" style="background-color:
yellow;">div1</div>

    <input type="checkbox" id="cb" checked="checked"/>
</body>
```

5.4 Node 对象的属性和方法

常用属性

属性名	描述
firstChild	指向在子节点列表中的第一个节点
lastChild	指向在子节点列表中的最后一个节点
childNodes	所有子节点的列表
previousSibling	指向前一个兄弟节点,如果这个节点就是第一个,那么该值为null
nextSibling	指向后一个兄弟节点,如果这个节点就是最后一个,那么该值为null
parentNode	父节点



常用方法

方法名	描述
hasChildNodes()	是否包含子节点
appendChild(node)	将节点添加到子节点列表的末尾
removeChild(node)	从子节点中删除node
replaceChild(newNode,oldNode)	替换子节点
insertBefore(newNode,refNode)	在一个子节点前插入一个新的子节点,在refNode元素前插入newNode

元素节点的增删改

使用原节点中的方法,可以完成对元素的增删改操作

完成 [资料/练习/DOM练习/index.html] 页面中的练习


```
 <script type="text/javascript">
        function fun1() {
            // 获取 span1 元素
            var span1El =
document.getElementById("span1");
            // 获取div1 元素
            var div1El = document.getElementById("div1");
            // 把 span1元素加入到div1 元素中
            div1El.appendChild(span1El);
```

```
        }
        function fun2() {
            // 1 获取所有的span
            var spanEls =
document.getElementsByTagName("span");
            console.log(spanEls);
            // 2 获取div2 元素
            var div2El = document.getElementById("div2");
            // 3 把获取到的span 加到div2中
            for(var i = 0;i<spanEls.length;i++){
                div2El.appendChild(spanEls[0]);
            }
        }
        function fun3() {
             var div3El = document.getElementById("div3");
             // 直接把元素作为文本加入到另一个元素中,这种方式会覆盖
之前的文本
             div3El.innerHTML = div3El.innerHTML + "
<span>newSpan</span>";
             // 可以使用方法去创建一个新的元素对象,然后在追加到
div3中
        }
    </script>
</head>
<body>
<span id="span1">span1</span>
<span id="span2">span2</span>
<span id="span3">span3</span>
```

6_事件处理

事件驱动编程

所谓事件驱动,简单地说就是你怎么触碰按钮(即产生什么事件),电脑执行什么操作(即调用什么函数).当然事件不仅限于用户的操作. 当对象处于某种状态时，可以发出一个消息通知，然后对这个消息感兴趣的程序就可以执行。

在生活中: 风和日丽的周末,你在大街上看到一个美女,很美,在她脸上轻轻抚摸了一下,美女给了你一巴掌,很痛.

事件驱动编程的核心对象

1. 事件源

   谁发出事件通知，发出消息；也就是事件主体（通常指元素和标签）；

2. 事件名称

   发出什么样的通知的名称，比如鼠标到我头上了，我被别人点了一下；

3. 事件响应函数

   谁对这个这个事件感兴趣，当这个事件发生时要执行什么样的操作；

4. 事件对象

   一般来说，当事件发生时，会产生一个描述该事件的具体对象，包括具体的参数等一起发给响应函数，好让他们通过事件对象来了解事件更加详细的信息。

```
<!--
    p元素是事件源
    click是事件名称
-->
<p style="color: red" id="p1"
onclick="shout(event);">Hello world!</p>
<script>
    //响应函数
    function shout(e){//响应函数，监听函数
      alert(e.clientX);//e事件对象
    }
</script>
```

事件类型(常用)

鼠标事件

属性	描述
onclick	当用户点击某个对象时调用的事件句柄
ondblclick	当用户双击某个对象时调用的事件句柄
onmousedown	鼠标按钮被按下
onmouseleave	当鼠标指针移出元素时触发
onmousemove	鼠标被移动
onmouseover	鼠标移到某元素之上
onmouseout	鼠标从某元素移开
onmouseup	鼠标按键被松开

键盘事件

属性	描述
onkeydown	某个键盘按键被按下
onkeypress	某个键盘按键被按下并松开
onkeyup	某个键盘按键被松开

表单事件 form

属性	描述
onblur	元素失去焦点时触发
onchange	该事件在表单元素的内容改变时触发
onfocus	元素获取焦点时触发
onsubmit	表单提交时触发

事件绑定的方式

元素本身是不自带事件的, 如果需要给元素添加功能的时候, 需要先给其绑定上对应的事件, 然后用户触发元素对应的事件时,执行之前绑定好的响应函数

给元素绑定事件有三种方式

在元素上使用onXxx属性绑定

xxx事件名称/类型.

```
<input type="button" value="点我啊" onclick="alert('点我干啥?');"/>
```

该方式特别简单,但是缺点是 HTML 和 JS 交错在一起,维护性差.

通过元素对象的事件属性绑定

```
<!--在该元素被加载完的时候没有绑定事件-->
<input type="button" value="点我啊" id="btn"/>
```

```
//使用JS代码为元素绑定事件
var btn = document.getElementById("btn");
btn.onclick = function(){
    alert("点我干啥");
}
```

这种方式,使得 HTML 和 JS 完全分离开来,便于后期维护, 推荐使用.

注意: 使用这种方式为元素绑定事件, 一定是要在元素被加载后,然后再执行上面的JS代码, 这样才能绑定成功. 试想,一个人都还没出生, 你怎么能让他做事呢?

所以,绑定事件的 JS 代码编写的位置很重要,通常有两种方式

1. 在元素后面编写js代码

```
<input type="button" value="点我啊" id="btn"/>
<script>
    //在input元素被加载后再根据id获取这个元素,可行
    var btn = document.getElementById("btn");
    btn.onclick = function(){
        alert("点我干啥");
    }
</script>
```

2. 使用文档加载事件, 在整个html文档加载完成之后再获取元素,绑定事件

```
window.onload = function(){
    //这个函数中的代码会在整个文档加载完成之后再执行
    //此时获取元素,可行
    var btn = document.getElementById("btn");
    btn.onclick = function(){
        alert("点我干啥");
    }
}
```

案例二：美女图片切换

```
function fun1() {
    // 获取 img 标签对象
    var imgEl = document.getElementById("image");
    // 修改src属性值
    imgEl.src = "b.png";
}

function fun2(){
    // 获取 img 标签对象
    var imgEl = document.getElementById("image");
    // 修改src属性值
    imgEl.src = "a.png";
}

window.onload = function () {
    // 使用js 来给事件源绑定事件
```

```
    document.getElementById("image2").onmouseover =
function () {
        this.src = "b.png";
    }

    document.getElementById("image2").onmouseout =
function () {
        this.src = "a.png";
    }
}
```

```
<img id="image" src="a.png" onmouseover="fun1()"
onmouseout="fun2()">

<img id="image2" src="a.png">
```

前面两种方式都存在一个小问题,不能为元素绑定多次相同的事件,因为是通过为事件属性赋值响应函数的方式来实现的,后面的赋值会将前面的覆盖掉.

如果想要为元素绑定多次相同的事件,此时需要使用方式三, 当然,在实际开发中,这种需求很少, 所以我们了解即可

通过元素对象的方法绑定事件

这种方式存在浏览器兼容的问题, IE和非IE使用的方法不一样

IE: attachEvent

[Object].attachEvent(“name_of_event_handler”, fnHandler);

name_of_event_handler: 事件名称前必须加"on", 多次添加监听后,触发顺序: **先添加的后执行**

fnHandler: 事件响应函数

[Object].detachEvent(“name_of_event_handler”, fnHandler);

fnHandler : 移除时,传入的"事件响应函数",必须和添加时,传入的是同一个(通过相同标识符引用的那一个函数)

匿名函数,每次创建的都不同

```
window.onload = function(){
    var btn = document.getElementById("btn");
    // 绑定事件
    btn.attachEvent("onclick", function(){
        alert("点我干啥?");
    });
    btn.attachEvent("onclick", function(){
        alert("又来,想si啊!");
    });
}
```

非IE浏览器和IE9及以上: addEventListener

[Object].addEventListener(“name_of_event”, fnHandler);

name_of_event: 直接使用事件(操作)名称,没有on, 多次添加监听后,触发顺序: **先添加的先执行**

fnHandler: 事件响应函数

[Object].removeEventListener(“name_of_event”, fnHandler);

fnHandler : 移除时,传入的"事件响应函数",必须和添加时,传入的是同一个(通过相同标识符引用的那一个函数)

匿名函数,每次创建的都不同

```
window.onload = function(){
    var btn = document.getElementById("btn");
    // 绑定事件
    btn.addEventListener("click", function(){
        alert("点我干啥?");
    });
    btn.addEventListener("click", function(){
        alert("又来,想si啊!");
    });
}
```

综合练习

使用提供好的 html 页面(4_js\资料\练习\DOM练习),完成以下练习

练习的目的在于熟悉以下点

1. html元素
    1. 复选框
    2. 下拉框
    3. 表单元素(文本框)
2. dom
    1. 获取元素的方法
3. 绑定事件
4. 复选框的 checked 属性
5. 下拉框的 selected 属性
6. 文本框的 value 属性
7. 数组的遍历
8. 添加子元素
9. 创建新元素
10. 为元素设置属性
11. 删除子元素

练习一参考代码

```
function checkAll(checked) {
    // 将兴趣爱好的复选框全部选中
    // 根据元素的name属性获取元素
    var hobbys = document.getElementsByName("hobby");
    console.log(hobbys);

    // 操作每个复选框的checked属性的值
    for (var i = 0; i < hobbys.length; i++) {
        hobbys[i].checked = checked;
    }
}

function checkUnAll() {
    // 将兴趣爱好的复选框全部选中
    // 根据元素的name属性获取元素
```

```
    var hobbys = document.getElementsByName("hobby");
    console.log(hobbys);

    // 操作每个复选框的checked属性的值
    for (var i = 0; i < hobbys.length; i++) {
        /*if (hobbys[i].checked) {
            hobbys[i].checked = false;
        }else{
            hobbys[i].checked = true;
        }*/
        hobbys[i].checked = !hobbys[i].checked;
    }
}

//如果事件源的复选的是选中,那么让下面的三个都选中,反之都取消
function checkChange(cb){
    var hobbys = document.getElementsByName("hobby");

    // 操作每个复选框的checked属性的值
    for (var i = 0; i < hobbys.length; i++) {
        hobbys[i].checked = cb.checked;
    }
}
```

练习二参考代码

```
function sel1AddAllTosel2(srcSelect, distSelect){
    //将左下拉框中的所有option元素追加到右下拉框中
    //获取select1, select1中的所有option, select2
    var srcSelect = document.getElementById(srcSelect);
    var distSelect = document.getElementById(distSelect);
    //select1:从select1中获取的元素
    var options =
srcSelect.getElementsByTagName("option");
    //appendChild
    /*for (var i = 0; i < options.length; i++) {
        console.log(options.length);
        select2.appendChild(options[i]);
        i--;
    }*/
```

```
    for (var i = 0; i < options.length; ) {
        console.log(options.length);
        distSelect.appendChild(options[i]);
    }
}

function sel1AddTosel2(srcSelect, distSelect){
    //将选中的option进行移动
    //获取select1, select1中的所有option, select2
    var srcSelect = document.getElementById(srcSelect);
    var distSelect = document.getElementById(distSelect);
    //select1:从select1中获取的元素
    var options =
srcSelect.getElementsByTagName("option");
    //appendChild
    for (var i = 0; i < options.length; i++) {
        //如果option元素被选中,那么他的selected属性值为true
        if(options[i].selected){
            distSelect.appendChild(options[i]);
            i--;
        }
    }
}
```

练习三参考代码

```
//添加用户
//每次点击添加的时候,在tbody中添加下面的元素
/*
<tr>
    <td>乔峰</td>
    <td>qiao@163.com</td>
    <td>18212345678</td>
    <td><a href='#'>删除</a></td>
</tr>
*/

function add(){
    //创建tr,四个td元素
    var tr = document.createElement("tr");
```

```
    //获取表单元素中的数据
    var usernameTd = document.createElement("td");
    var emailTd = document.createElement("td");
    var telTd = document.createElement("td");
    var deleteTd = document.createElement("td");
    //将获取到的数据设置为td的文本
    //获取文本框中的数据,使用input元素的value属性
    var username =
document.getElementById("username").value;
    var email = document.getElementById("email").value;
    var tel = document.getElementById("tel").value;

    usernameTd.innerHTML = username;
    emailTd.innerHTML = email;
    telTd.innerHTML = tel;
    deleteTd.innerHTML="<a href='#'
onclick='deleteUser(this)'>删除</a>";
    //将td追加到tr中
    tr.appendChild(usernameTd);
    tr.appendChild(emailTd);
    tr.appendChild(telTd);
    tr.appendChild(deleteTd);
    //将tr追加到tbody中
    document.getElementById("userTbody").appendChild(tr);
}
/*
<tr>
    <td>乔峰</td>
    <td>qiao@163.com</td>
    <td>18212345678</td>
    <td><a href='#'>删除</a></td>
</tr>
*/
function deleteUser(aEle){
    //删除 tbody 中的 tr 元素
    //根据 a 元素找到所在行的 tr 元素
    var tr = aEle.parentNode.parentNode;
    tr.parentNode.removeChild(tr);
}

window.onload = function(){
```

```
    //为删除所有按钮绑定事件
    document.getElementById("btn_removeAll").onclick =
function(){
        document.getElementById("userTbody").innerHTML="";
    }
}
```

三个练习题, 每个练习完成至少两遍