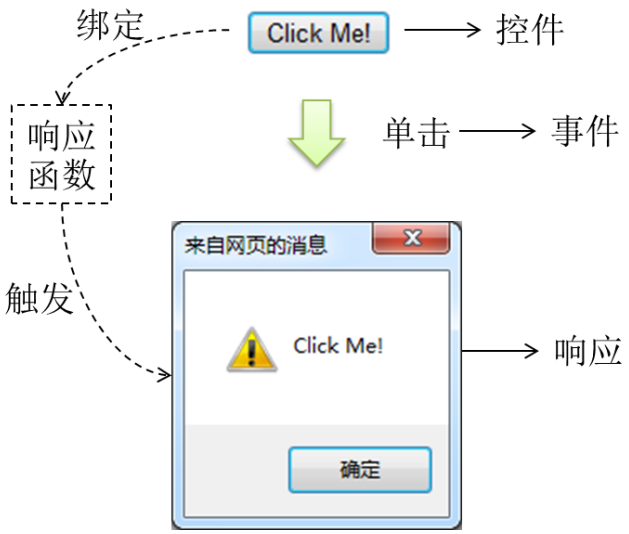
JavaScript 学习笔记

1 JavaScript 简介：JavaScript 是一种基于对象和事件驱动并具有相对安全性的客户端脚本语言，是一种动态、弱类型、基于原型的语言，内置支持类。它的解释器被称为 JavaScript 引擎，为浏览器的一部份。同时也是一种广泛用于客户端 Web 开发的脚本语言。最初由网景公司（Netscape）的布兰登 • 艾克（Brendan Eich）设计，1997 年，在 ECMA（欧洲计算机制造商协会）的协调下，由 Netscape、Sun、微软、Borland 组成的工作组确定统一标准：ECMA-262。JavaScript 是 Sun 公司的注册商标，和 Java 没有直接关系。

2 JavaScript HelloWorld

2.1 点击按钮弹出文本值



2.2 HTML 代码：<button>Click Me!</button>

2.3 JavaScript 代码

```
<script type="text/javascript">
    //借助 window.onload 事件，在文档加载之后，进行相关操作
    window.onload = function(){
        //获取按钮对应的元素节点对象
        var btn = document.getElementsByTagName("button")[0];
        //绑定单击响应函数
        btn.onclick = function(){
            //打印文本值
            alert(this.firstChild.nodeValue);
        }
    }
</script>
```

2.4 由 Hello World 引出的四个问题

2.4.1 JavaScript 基本语法和 Java 有什么区别？

2.4.2 为什么要使用 window.onload？

2.4.3 在我单击按钮后如何让程序执行我希望的操作？

2.4.4 HTML 代码中的 button 标签在 JavaScript 代码中是怎么表示的？如何获取？

3 JavaScript 基本语法

3.1 认识 script 标签

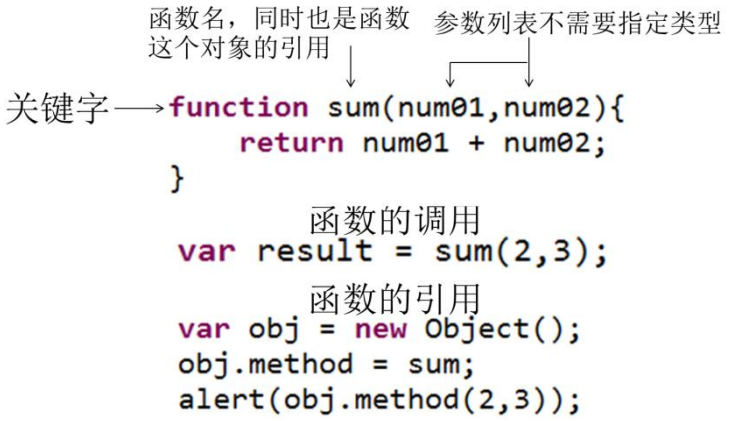
3.1.1 <script> 标签用于定义客户端脚本，比如 JavaScript
3.1.2 在语法上， <script> 标签可以写在<html></html>标签内的任何位置
3.1.3 type 属性是必需的，用来规定脚本的 MIME 类型。对于 JavaScript，其 MIME 类型是 "text/javascript"。
3.1.4 language 属性：不赞成使用
3.1.5 在文档中直接插入 JavaScript 代码

```
<script type="text/javascript">
    alert("Hello World!");
</script>
```

3.1.6 连接外部 JavaScript 文件
3.1.6.1 <script>标签的 src 属性：规定外部脚本文件的 URL。
3.1.6.2 纯 JavaScript 代码可以保存到“.js”文件中，在“.js”文件中，写 JavaScript 代码和在 script 标签中完全一样
3.1.6.3 <script type="text/javascript" src="script/jquery-1.7.2.js"></script>
3.1.6.4 注意：加载外部 CSS 样式表使用的是 link 标签，和加载外部 JS 文件不同！

```
<link rel="stylesheet" type="text/css" href="theme.css" />
```

3.2 JavaScript 数据类型
3.2.1 字符串：JavaScript 中没有“字符型”数据，单个字符也被当成字符串处理
3.2.2 数字型：JavaScript 不区分整型和浮点型，所有数字都是以浮点型来表示的
3.2.3 布尔型
true：true、非零的数字、非空的字符串
false：false、数字 0、空字符串、undefined、null
3.2.4 函数：在 JavaScript 中，函数也是一个对象，可作为一个值赋给变量，函数名就是这个对象的引用。
3.2.5 Null：代表“空”。Null 是数据类型，它只有一个值：null
不代表数字型的 0，不代表字符串类型的””空字符串
不是一个有效的数字、字符串、对象、数组和函数，什么数据类型都不是
JavaScript 区分大小写，Null、NULL 都不等同于 null
typeof(null)返回 Object（为了向下兼容）
3.2.6 undefined：表示：未定义
定义了一个变量但未赋值
使用了一个未定义的变量
使用了一个不存在的对象的属性
3.3 JavaScript 严格区分大小写
3.4 JavaScript 标识符命名规范和 Java 完全一样
3.5 变量
3.5.1 使用 var 关键字声明
3.5.2 JavaScript 是弱类型语言，声明变量时，不需要指定类型。 变量在使用中也可以存储各种类型的数据
3.6 函数



4 JavaScript 代码写在什么位置

4.1 HTML 标签内

4.1.1 <button onclick="alert('hello')">ClickMe</button>

4.1.2 js 和 html 强耦合，不利于代码的维护。例如：给 10 个 button 按钮绑定同样的单击响应函数。如果要修改函数名则必须在 10 个 button 标签处都修改，容易造成遗漏或不一致

4.2 <head> 标签内

4.2.1 这个位置更符合习惯，但有严重问题： 在 body 节点之前执行的代码无法直接获取 body 内的节点。 原因：此时 html 文档树还没有加载完成， 更准确的说就是——内存中的 DOM 结构还不完整，不包括未加载的 DOM 节点，所以相关节点 JavaScript 程序获取不到

4.2.2 浏览器加载原理分析

4.2.2.1 浏览器按照从上到下的顺序下载 HTML 文档，一边下载，一边加载到内存， 不是全部下载后再全部加载到内存。 另外，DOM 树结构的绘制优先于关联元素的加载，比如图片。

4.2.2.2 通常来说，浏览器执行 JavaScript 脚本有两大特性：

①载入后马上执行。

②执行时会阻塞页面后续内容（包括页面的渲染、其它资源的下载）。

4.2.2.3 浏览器加载 HTML 文档时，如果遇到<script>标签就会停止后面元素的加载， 先对 JavaScript 脚本文件进行解析和执行。

4.3 <body>标签后面

4.3.1 能获取到 body 内的元素节点

4.3.2 把代码写在这个位置严重不符合习惯

4.4 window.onload

4.4.1 window.onload 事件是在当前文档完全加载之后被触发，此时 HTML 文档树已经加载完成，可以获取到当前文档中的任何节点

4.4.2 最终的加载执行顺序

1st. 浏览器加载 HTML 文档

2nd. 遇到<script>标签停止加载后面的 HTML 元素，开始解析执行 JavaScript 代码

3rd. 将封装了相关操作的响应函数绑定到 window.onload 事件上

4th. 加载 HTML 元素，绘制 DOM 结构

5th. HTML 文档加载完成，触发 window.onload 事件

6th. 开始执行 window.onload 事件响应函数中的 JavaScript 代码

5 事件的监听和响应

5.1 事件的监听和响应是用户和应用程序进行交互的方式

5.2 为控件绑定事件响应函数

5.2.1 从文档对象模型中获取控件对象

5.2.2 声明一个事件响应函数

5.2.3 将事件响应函数的引用赋值给控件对象的 onclick 属性

```
window.onload = function(){
    //从文档对象模型中获取控件对象
    var btn = document.getElementsByTagName("button")[0];
    //将事件响应函数的引用赋值给控件对象的onclick属性
    btn.onclick = function(){
        var text = this.firstChild.nodeValue;
        alert(text);
    }
}
```

5.3 取消控件默认行为

5.3.1 超链接的跳转功能：return false;

5.3.2 提交按钮的提交功能：return false;

5.4 事件冒泡

5.4.1 概念：用户操作的动作要传递给当前控件，响应函数执行完之后，还要继续传递给它的父元素，并一直向上传递，直到顶层

5.4.2 产生的原因：监听区域的重合

5.4.3 取消的方法：return false;

5.5 焦点

5.5.1 一个控件被激活时就获得了焦点，也可以反过来说：一个控件获得焦点时被激活

5.5.2 获得焦点事件：onfocus

5.5.3 失去焦点事件：onblur

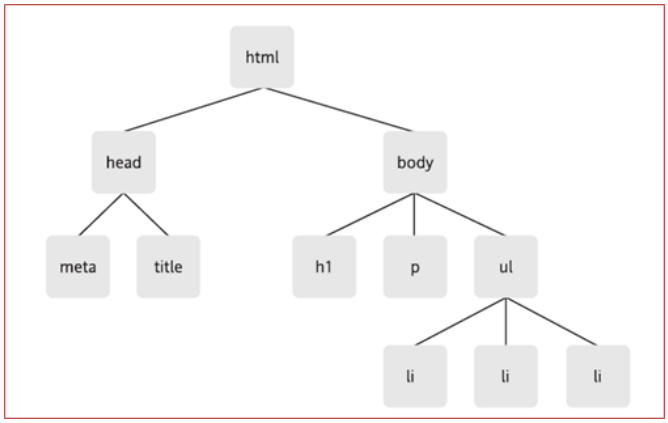
6 DOM

6.1 Document Object Model：文档对象模型 定义了访问和处理 HTML 文档的标准方法。是 W3C 国际组织制定的统一标准，在很多计算机语言中都有不同实现如 C#、PHP、Java、Ruby、perl、python 等

6.2 DOM 树

6.2.1 上下为父子关系

6.2.2 左右为兄弟关系



6.3 节点

6.3.1 节点的概念来源于网络理论，代表网络中的一个连接点。网络是由节点构成的集合。 相应的，我们可以说 HTML 文档是由 DOM 节点构成的集合

6.3.2 节点的分类

6.3.2.1 元素节点：构成 HTML 文档最基本的 HTML 元素，对应 HTML 文档中的 HTML 标签

6.3.2.2 属性节点：对应元素的属性

6.3.2.3 文本节点：对应 HTML 标签内的文本内容

7 节点的属性

7.1.1 nodeName: 代表当前节点的名字，只读属性。如果给定节点是一个文本节点，nodeName 属性将返回内容为 #text 的字符串

7.1.2 nodeType：返回一个整数，这个数值代表着给定节点的类型. 只读属性. 1 -- 元素节点, 2 -- 属性节点, 3 -- 文本节点

7.1.3 nodeValue：返回给定节点的当前值(字符串). 可读写的属性

7.1.3.1 元素节点, 返回值是 null

7.1.3.2 属性节点: 返回值是这个属性的值

7.1.3.3 文本节点: 返回值是这个文本节点的内容

	nodeName	nodeType	nodeValue
元素节点	标签名	1	null
属性节点	属性名	2	★属性值
文本节点	#text	3	★★文本内容

8 获取元素节点

8.1 根据 id 属性值获取：调用 document 对象的 getElementById(元素的 id 属性)，返回的是当前元素节点的引用

8.2 根据标签名，获取元素节点

8.2.1 调用 document 对象的 getElementsByTagName(标签名)，返回的是标签名对应的元素节点的数组

8.2.2 也可以通过一个元素节点来调用 getElementsByTagName(标签名)方法

8.3 根据 name 属性值获取一组元素节点

8.3.1 通过调用 document 对象的 getElementsByName(元素的 name 属性值) 方法，返回的是一个元素的数组

8.3.2 如果 HTML 标签本身没有 name 属性，这时通过 getElementsByName() 方法获取元素节点

IE 获取不到

严格遵守 W3C 标准的浏览器可以获取到

8.4 操作属性

8.4.1 推荐方法

8.4.1.1 读：element.attrName

8.4.1.2 写：element.attrName=”new value”;

8.4.2 经典方法

8.4.2.1 获取属性节点对象：element.getAttributeNode(属性名) 返回一个属性节点对象的引用

8.4.2.2 读取值：attrNode.nodeValue

8.4.2.3 修改值：attrNode.nodeValue = " new value"

9 获取元素节点的子节点

9.1 查看元素节点是否存在子节点:hasChildNodes()。只有元素节点才可能有子节点

9.2 通过 childNodes 属性获取全部子节点。IE 和 W3C 的标准有区别：IE 忽略空格（空行），W3C 标准会把空格也当作子节点

9.3 通过 firstChild 属性获取元素节点的第一个子节点。该方法经常被用来获取文本节点如：元素节点对象.firstChild。如果要获取文本值：元素节点对象.firstChild.nodeValue

9.4 调用父节点的 getElementsByTagName()，更有针对性

10 节点的属性

10.1 属性节点的 nodeValue 是它的属性值

10.2 文本节点的 nodeValue 是它的文本值

10.3 获取文本节点的文本值的方法：文本节点的父节点.firstChild.nodeValue

11 创建节点

11.1 注意：新创建的节点，包括文本节点和元素节点，都不会自动的加入到当前的文档中

11.2 创建文本节点 document.createTextNode(文本值) 方法，返回一个文本节点的引用

```
var textNode = document.createTextNode("广州");
```

11.3 创建元素节点 document.createElement(标签名) 方法，返回一个新的元素节点的引用

```
var liEle = document.createElement("li");
```

11.4 将新创建的文本节点，添加到元素节点当中 元素节点.appendChild(文本节点)

```
liEle.appendChild(textNode);
alert(liEle.firstChild.nodeValue);
var cityEle = document.getElementById("city");
```

11.5 将新创建的元素节点，添加到文档中 指定的元素节点.appendChild(新创建的元素节点)

```
cityEle.appendChild(liEle);
```

12 替换节点

12.1 调用 父节点.replaceChild(新节点,目标节点) 方法，实现节点的替换

13 获取元素节点的父节点：子节点.parentNode

14 复制节点：原元素节点.cloneNode(true) 参数为布尔值：true 包括子节点都复制，false 只复制当前节点（默认）

15 插入节点

15.1 将新创建的元素节点，添加到文档中指定元素节点前面

15.2 父节点.insertBefore(新节点,目标节点);

15.3 自定义 inserAfter()函数

```
function insertAfter(newNode,targetNode){
        //将一个新节点，添加到目标节点的后面
        var parentNode = targetNode.parentNode;
```

```
            //判断目标节点是否为最后一个子节点
            if(targetNode == parentNode.lastChild){
                //如果是，调父节点的 appendChild
                parentNode.appendChild(newNode);
            }else{
                //如果不是，获取它后面的兄弟节点 targetNode.nextSibling
                var sibling = targetNode.nextSibling;
                //调用父节点的 insertBefore 方法，以 sibling 为新的目标节点
                parentNode.insertBefore(newNode,sibling);
            }
        }
```

16 删除节点 元素节点.removeChild(要删除的子节点)

17 innerHTML 属性可读可写。读：读出指定元素节点内的 HTML 代码，写：将 HTML 代码写入指定元素节点的开始和结束标签内

18 nextSibling:元素节点.nextSibling 返回元素节点的下一个兄弟元素

19 previousSibling：元素节点.previousSibling 返回元素节点的上一个兄弟元素