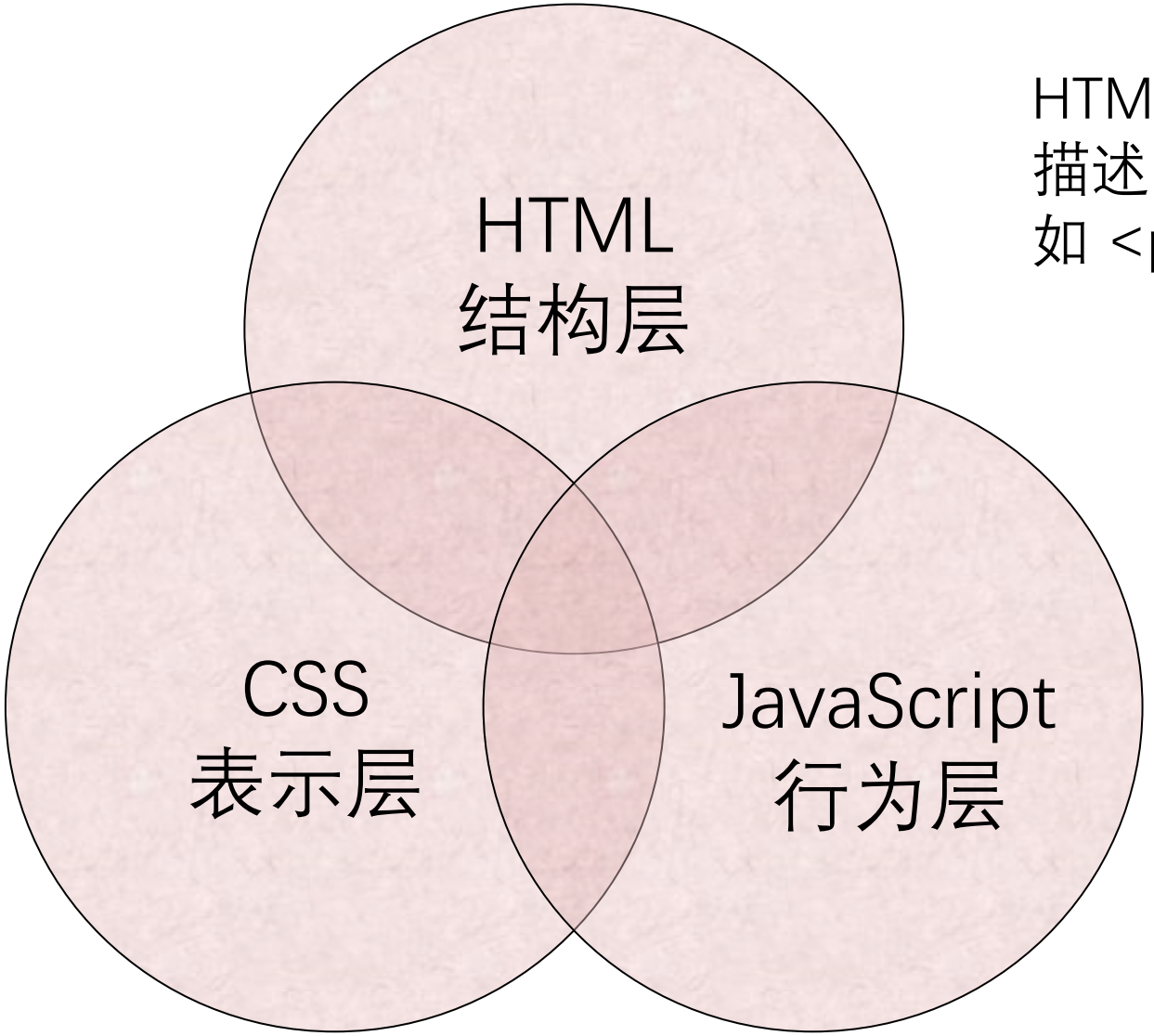
JavaScript DOM

内容提要

- 1 三位一体的网页：HTML/CSS/JavaScript
- 2 DOM 和节点类型
- 3 获取元素
- 4 元素属性操作：动态修改元素属性
- 5 DOM 遍历
- 6 元素操作：动态创建标记
- 7 CSS 属性操作
- 8 事件

1 三位一体的网页



HTML 标记语言：
描述页面的结构。
如 Hello

CSS：
描述页面内容
该如何呈现。
如设置文本段的
字体、字号。

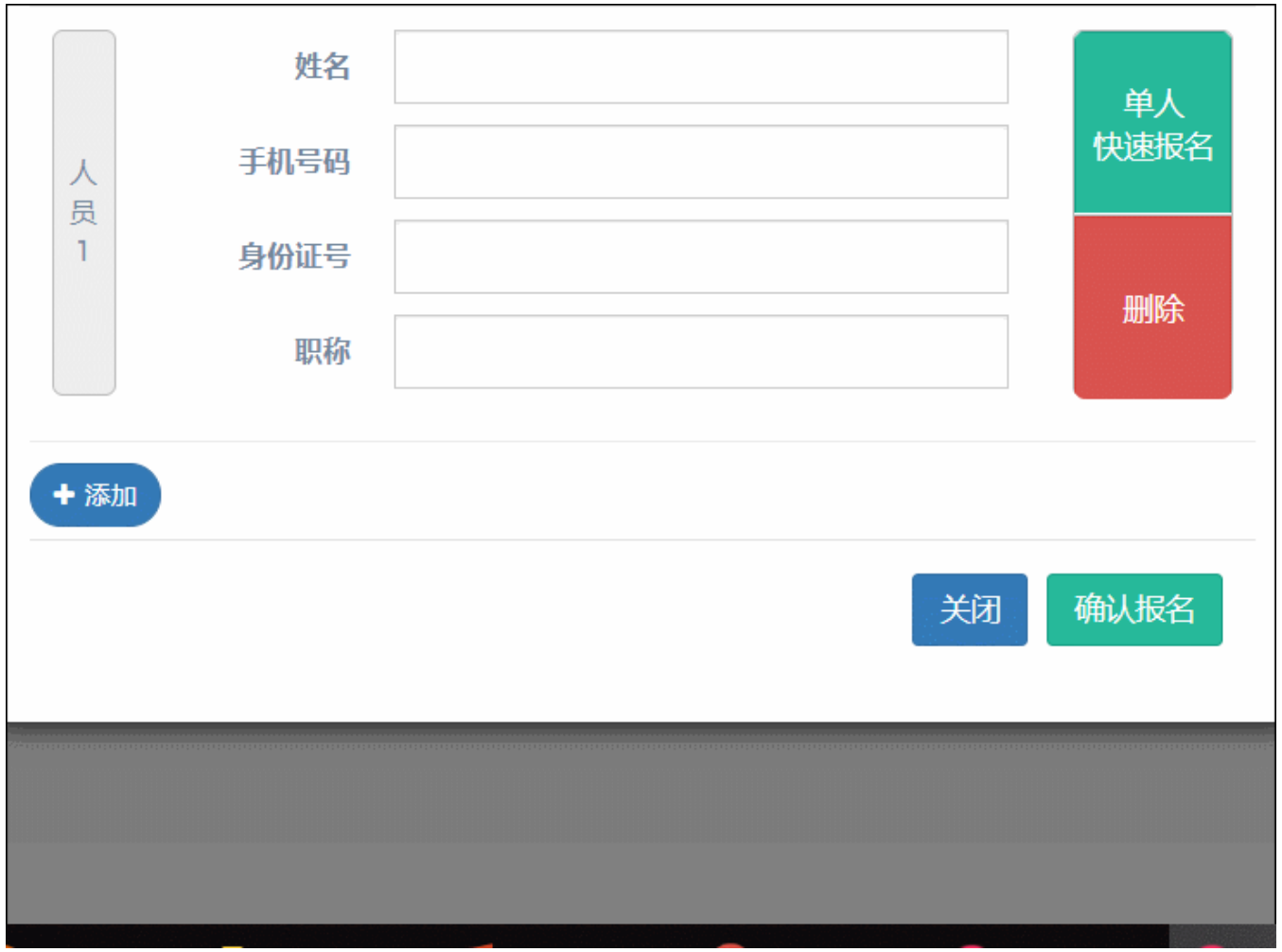
JavaScript：
负责内容应该
如何响应。
如点击文本段时，
显示 alert 对话框

DOM（Document Object Model，文档对象模型）

- DOM 是针对 HTML/XML 文档，提供方法允许开发人员添加、删除和修改页面中的某一部分
- 文档（Document）
  - 创建网页并把它加载到浏览器中，网页文档被转换为一个 document 文档对象。
- 对象（Object）
  - 将元素抽象为对象，HTML中的每个元素都是一个对象，作为document对象的属性。
- 模型（Model）
  - 体现映射关系，DOM 将一份文档表示为一棵树

DOM树的操作

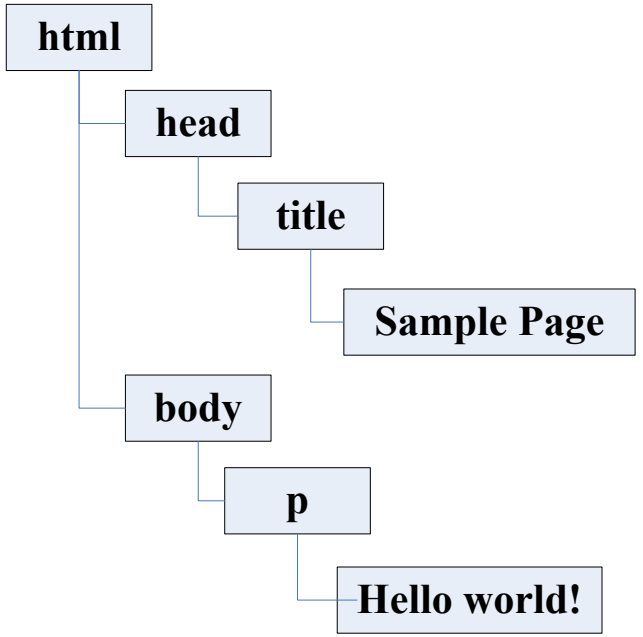
- **使用 DOM 提供的方法操作 DOM 树，添加、修改或删除页面元素和元素的属性。**



DOM 树

- DOM 将一份文档表示为一棵树，每个元素都被表示为一个节点。

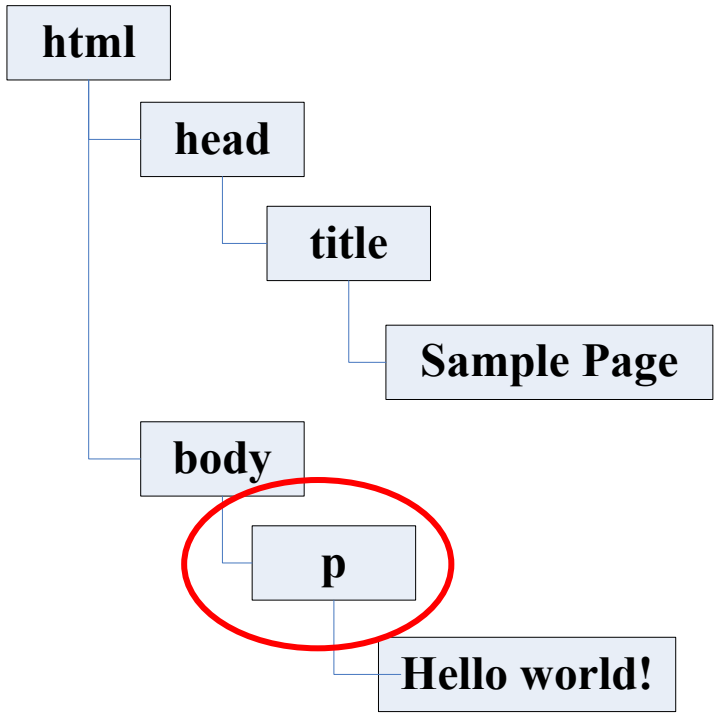
```
<html>
  <head>
    <title>Sample Page</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```



节点类型

- 文档中的每一段标记都能通过树中的一个节点表示，如文档节点（Document）、注释节点等。HTML 中，常用元素节点、属性节点和文本节点操作文档。

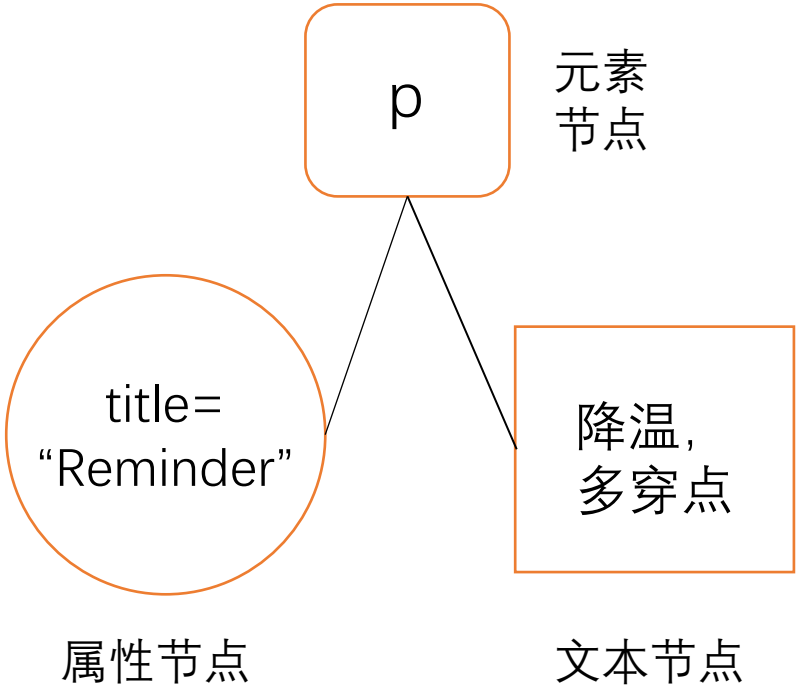
  - 元素节点
    - DOM 的原子。每个标签如 <p>
<a> 被独立地表示为一类节点。
    - 元素节点的名字即为标签名。
    - 元素节点可以包含其他的元素节点。



属性节点

- 属性节点用来对元素做出更具体的描述。
- 标签的一个属性被表示为一个属性节点。
- 属性总是被写在 HTML 的开标签中，所以属性节点总是被包含在元素节点中。

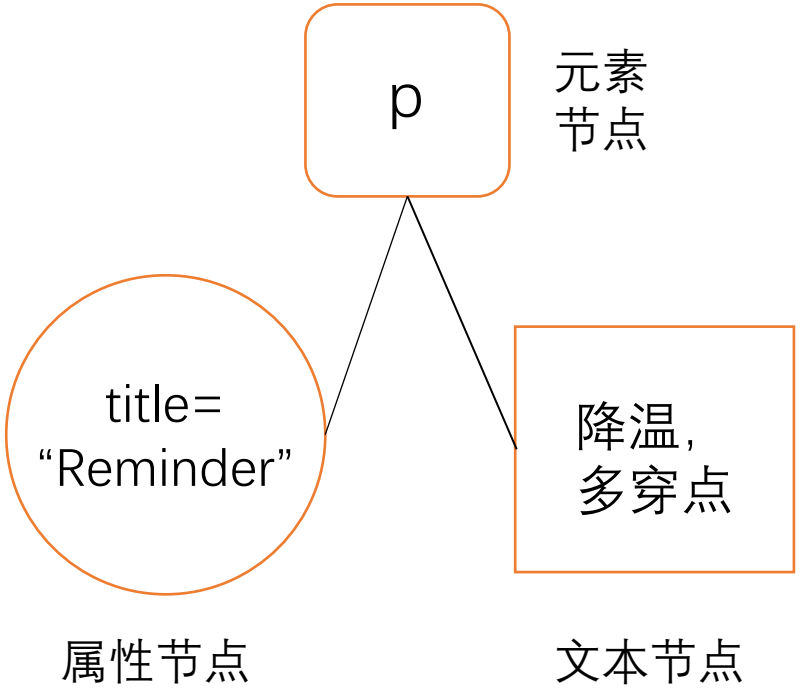
```
<p title="Reminder">降温，多穿点</p>
```



文本节点

- 描述 HTML 开闭标签中的文本内容。
- 文本节点总是被包含在元素节点内部，元素节点不一定包含文本节点。

```
<p title="Reminder">降温，多穿点</p>
```



获取元素

获取元素节点。对于 HTML 文件，要对页面的元素进行操作，首先要先进行获取。

HTML元素选择方法		
	方法名称	功能说明
1	getElementById()	通过 id 名称来选择元素，每个元素有唯一的 id.
2	getElementsByTagName()	返回数组。通过标签的名称选择元素节点，可能有多个，选择的可以是个数组。
3	getElementsByClassName()	返回数组。通过 class 属性选择元素节点。
4	getElementsByName()	通过 name 属性获得元素节点。
5	document.title 和 document.body	获取 title 和 body 节点。

NodeList 对象

- 是一种类数组对象，用户保存一组有序的节点，可以通过索引访问这些节点。
- 具有 length 属性，表示 NodeList 对象中的元素长度。
- 具有**动态性**。是给予 DOM 结构动态执行查询的结果，DOM 结构的变化能够自动反应在 NodeList 对象中。

getElementById(*String*)

按元素 id 属性获取节点，返回该元素节点对象。

```
<h1>What to buy</h1>
<p title="reminder">Don't forget to buy this stuff</p>
<ul id="purchases">
  <li>A tin of beans</li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
```

Demo 4.1

```
<script>
var purchases = document.getElementById("purchases");
alert(typeof purchases);          // object
</script>
```

getElementsByClassName(*String*)

按元素 class 属性获取节点，返回搜索到的元素节点数组。

```
<h1>What to buy</h1>
<p title="reminder">Don't forget to buy this stuff</p>
<ul id="purchases">
  <li>A tin of beans</li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
```

```
<script>
var sales = document.getElementsByClassName("sale");
for (var i = 0; i < sales.length; i++) {
  alert(sales[i].textContent);          // Cheese Milk
}
</script>
```

Demo 4.1

getElementsByTagName(*String*)

- 按元素标签获取节点，返回搜索到的元素节点数组。

```
<h1>What to buy</h1>
<p title="reminder">Don't forget to buy this stuff</p>
<ul id="purchases">
  <li>A tin of beans</li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
```

```
<script>
var items = document.getElementsByTagName("li");
for (var i = 0; i < items.length; i++) {
  alert(items[i].textContent);    // A tin of beans Cheese Milk
}
</script>
```

Demo 4.1

对元素属性进行添加、修改删除或查询

- 如 `title/name` 的设置，或表单 `disabled` 属性的添加或删除等操作。

HTML 属性操作方法		
	方法名称	功能说明
1	getAttribute()	获取元素的某个属性值 obj.getAttribute(“attr”)
2	setAttribute()	设置元素的某个属性值。 obj.setAttribute(“attr”,值)
3	removeAttribute()	删除元素的某个属性值 obj.removeAttribute(“attr”)
4	hasAttribute()	判断元素是否包含某个属性 obj.has Attribute(“attr”)
5	getAttribute()	获取元素的某个属性值 obj.getAttribute(“attr”)

object.hasAttribute(*String*)，object.getAttribute(*String*)

object.hasAttribute(*String*)

返回 true/false，判断元素是否拥有某属性。

object.getAttribute(*String*)

返回元素属性值。

Demo 4.2

```
<h1>What to buy</h1>
<p title="reminder">Don't forget to buy this stuff</p>
<ul id="purchases">
  <li>A tin of beans</li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
<script>
var items = document.getElementsByTagName("li");
for (var i = 0; i < items.length; i++) {
  if (items[i].hasAttribute("class")) {
    alert(items[i].getAttribute("class")); // sale, sale important
  }
}
</script>
```

object.setAttribute(*attr:String, value:String*)

- 设置元素的 attr 属性为 value.

```
<h1>What to buy</h1>
<p title="reminder">Don't forget to buy this stuff</p>
<ul id="purchases">
  <li>A tin of beans</li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
<script>
var items = document.getElementsByTagName("li");
for (var i = 0; i < items.length; i++) {
   items[i].setAttribute("title", items[i].textContent);
  alert(items[i].getAttribute("title"));
}
</script>
```

Demo 4.2

object.removeAttribute(*attr:String*)

- 删除元素的 attr 属性。

```
<h1>What to buy</h1>
<button id="button" disabled="disabled">按钮</button>

<script>
var button = document.getElementById("button");
button.removeAttribute("disabled");
</script>
```

Demo 4.2

查找指定元素的父元素、子元素或兄弟元素

并非所有元素具有 id 等标记。需要遍历 DOM 树，根据元素节点关系进行查询。

节点属性		
	类别	节点属性
1	查找父元素（parent）	obj.parentNode
2	查找子元素（child）	obj.childNodes/firstChild/lastChild/ children/firstElementChild/lastElementChild
3	查找兄弟元素（sibling）	obj.previouElementSibling/nextElementSibling

object.parentNode

返回当前节点的父节点和其对应的文本节点。

Demo 4.3

```
<button class="btn" disabled="disabled">上一步</button>
<ul id="purchases">
  <li>Beans<span id="tin">/tin</span></li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
<button class="btn" disabled="disabled">确认</button>
<script>
tinSpan = document.getElementById("tin");
beans = tinSpan.parentNode;
alert(beans.textContent);     // Beans/tin
</script>
```

5 DOM 遍历

object.children

返回当前节点的所有子节点与对应的文本节点。

Demo 4.3

```
<button class="btn" disabled="disabled">上一步</button>
<ul id="purchases">
  <li>Beans<span id="tin">/tin</span></li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
<button class="btn" disabled="disabled">确认</button>
<script>
purchases = document.getElementById("purchases");
items = purchases.children;
for (var i = 0; i < items.length; i++) {
  alert(items[i].textContent);
}
</script>
```

object.firstChild, object.firstElementChild

object.firstChild

返回当前节点的首个子节点（无文本节点）。

object.firstElementChild

返回当前节点的首个子节点与文本节点。

```
<button class="btn" disabled="disabled">上一步</button>
<ul id="purchases">
  <li>Beans<span id="tin">/tin</span></li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
<button class="btn" disabled="disabled">确认</button>
<script>
purchases = document.getElementById("purchases");
alert("firstChild: " + purchases.firstChild.textContent);                    // “”
alert("firstElementChild: " + purchases.firstElementChild.textContent);      // Beans/tin
</script>
```

Demo 4.3

object.lastChild, object.lastElementChild

object.lastChild

返回当前节点的末子节点（无文本节点）。

object.lastElementChild

返回当前节点的末子节点与文本节点。

object.previousSibling, object.previousElementSibling

object.previousSibling

返回当前节点的前一个兄弟节点（无文本节点）。

object.previousElementSibling

返回当前节点的前一个兄弟节点与文本节点。

```
<button class="btn" disabled="disabled">上一步</button>
<ul id="purchases">
  <li>Beans<span id="tin">/tin</span></li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>
<button class="btn" disabled="disabled">确认</button>
<script>
var itemBean = document.getElementsByClassName("sale")[0].previousElementSibling;
alert(itemBean.textContent);            // Beans/tin
</script>
```

Demo 4.3

object.nextSibling, object.nextElementSibling

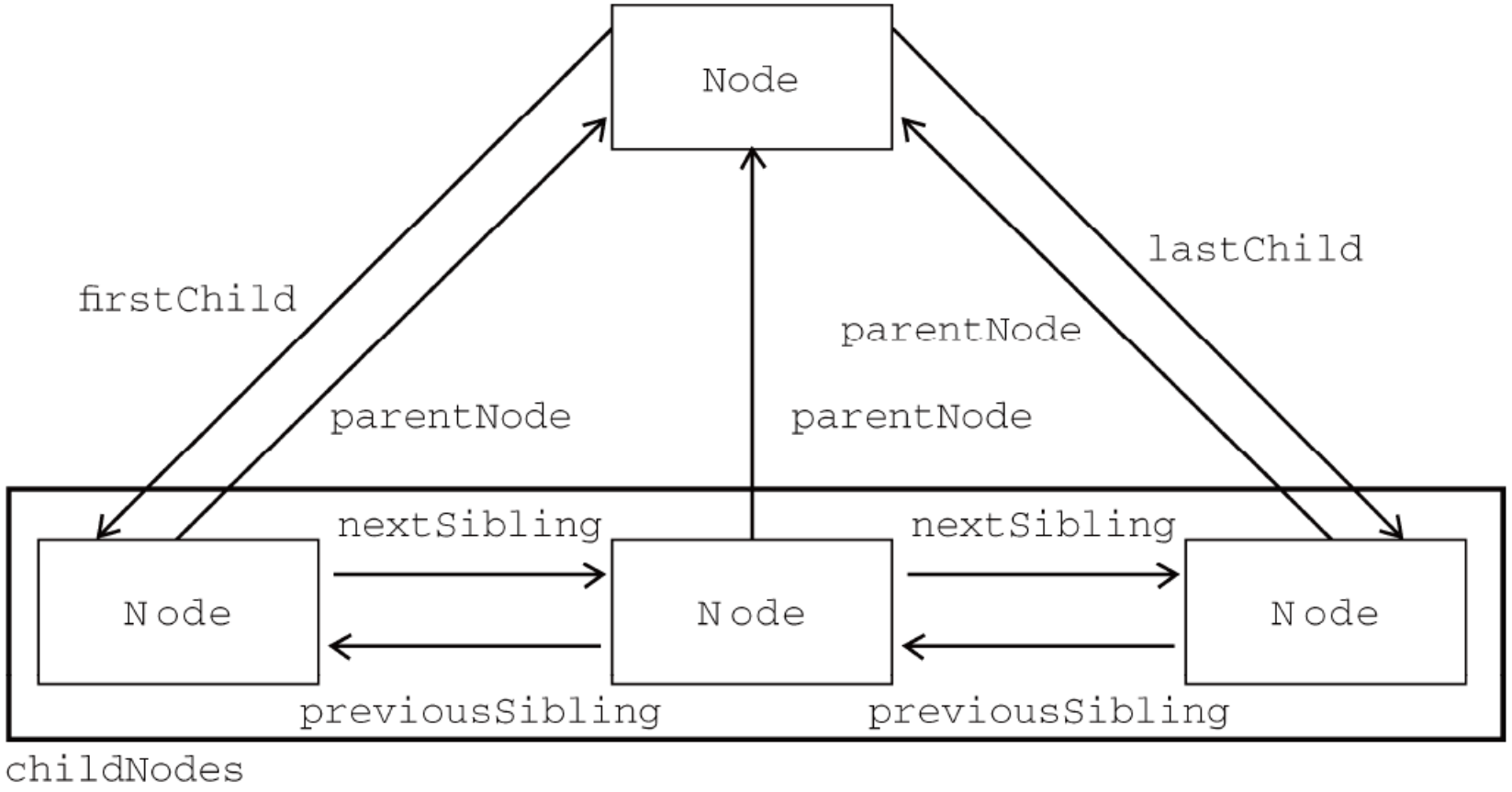
object.nextSibling

返回当前节点的后一个兄弟节点（无文本节点）。

object.nextElementSibling

返回当前节点的后一个兄弟节点与文本节点。

总表


Node
firstChild
lastChild
parentNode
parentNode
parentNode
nextSibling
nextSibling
Node
Node
Node
previousSibling
previousSibling
childNodes

使用 DOM 改变网页的结构和内容

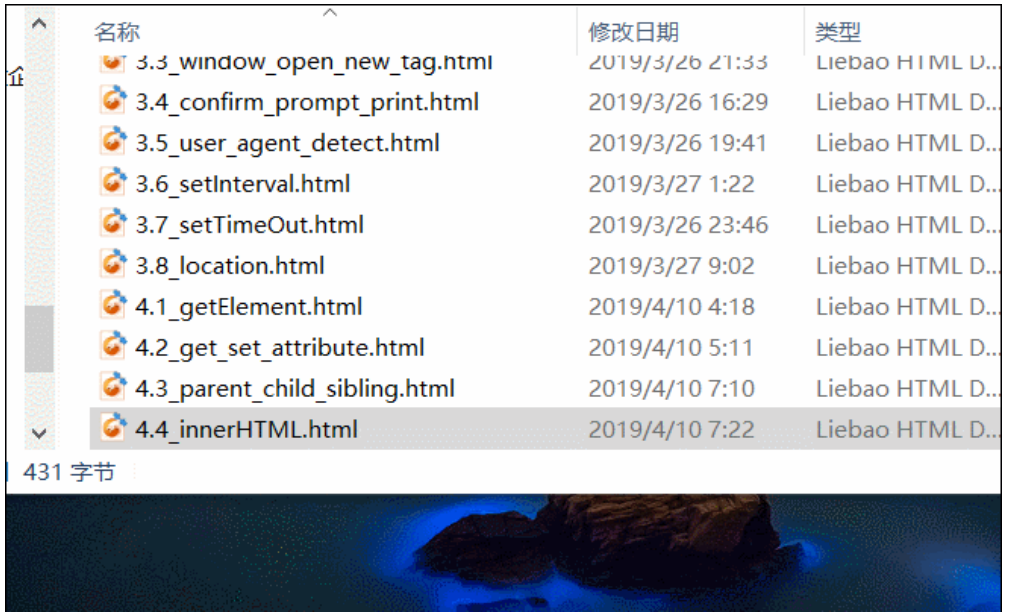
在 DOM 树中动态插入、修改和删除节点。

object.innerHTML 属性

用来读写给定元素内的 HTML 内容。

Demo 4.4

```
<h1 id="title">What to buy</h1>
<script>
title = document.getElementById("title");
alert(title.innerHTML);
setTimeout(function() {
  title.innerHTML = "天真冷!";
}, 2000);
</script>
```



createElement(*nodeName*)

创建新的元素节点对象。

createTextNode(*text*)

创建新的文本节点对象。

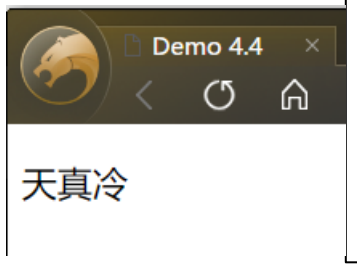
parent.appendChild(*child*)

将 child 节点作为**末子节点**插入 parent 节点中

```
<div id="container"></div>
```

例：将一段文本插入 container 元素：
步骤一：创建一个新的 <p> 元素；
步骤二：创建一个文本节点对象；
步骤三：将文本节点对象插入 <p> 中；
步骤四：将 <p> 插入 container 中。

Demo 4.5




```
<script>
container =
document.getElementById("container");
p = document.createElement("p");
text = document.createTextNode("天真冷");
p.appendChild(text);
container.appendChild(p);
</script>
```

parent.InsertBefore(*newElement*, *targetElement*)

将新元素作为子节点插入到父元素的某个子元素之前。

parent：目标元素的父元素；

newElement：想插入的元素；

targetElement：想把新元素插入到哪个元素之前。

Demo 4.6

```
<div id="container"><span id="question_mark">? </span></div>
<script>
container = document.getElementById("container");
p = document.createElement("p");
text = document.createTextNode("天真冷");
p.appendChild(text);
questionMark = document.getElementById("question_mark");
container.insertBefore(p, questionMark);
</script>
```

removeChild, replaceChild

- **parent.removeChild(*element*)**

删除 *parent* 下的 element 子元素。

- **parent.replaceChild(*newElement*, *oldElement*)**

用 newElement 替代 parent 下的 oldElement 子元素。

```
<ul id="purchases">
  <li>Beans<span id="tin">/tin</span></li>
  <li class="sale">Cheese</li>
  <li class="sale important">Milk</li>
</ul>

<script>
purchases = document.getElementById("purchases");
/* removeChild */
beans = document.getElementsByTagName("li")[0];
purchases.removeChild(beans);

/* replaceChild */
cheese = document.getElementsByTagName("li")[0];
milk = document.getElementsByTagName("li")[1];
purchases.replaceChild(milk, cheese);
</script>
```

NodeList 对象的动态性
（11 页）

element.style

➢通过 element.style 属性查看和修改 CSS 样式

每个 CSS 样式是 style 对象的属性，每个样式使用**驼峰标记法**表示。

如：font-size -> fontSize; font-color -> fontColor

只能查看和修改定义在 HTML 标签中 style 属性确定的样式。

Demo 4.8

```
<p id="example" style="color: indianred; text-align: center; font: 12px 'Arial', sans-serif;">天真冷</p>

<script>
example = document.getElementByld("example");
alert(example.style.textAlign);     // center
alert(example.style.fontSize);      // 12px
example.style.fontSize = "24px";
</script>
```

8 事件

- 事件处理函数
- 事件分类和应用
- 事件监听器
- event 对象

JavaScript 与 HTML 之间的交互通过事件实现

- 事件：文档或浏览器窗口中发生的一些特定的交互瞬间，如点击鼠标、按下键盘、提交表单、勾选复选框等
- 事件分类
  - 鼠标事件
  - 键盘事件
  - 表单事件
  - 编辑事件
  - 页面事件

调用方式

- 事件处理函数 方式 → 使用方便
  - 在 script 标签中调用事件
    如 obj.onclick = function(){ }
  - 在元素中调用事件
    如 <input type="button" onclick="alertFunction()" value="提交">
- 事件监听器 方式 → 可以为一个元素添加多次相同事件
  - addEventListener()
  - removeEventListener()

```
<p>天真冷(点击)</p>

<script>
  example = document.getElementById("example");
  example.style.cursor = "pointer";
  example.onclick = function (event) {
    example.innerHTML = '是的';
  }
</script>
```

Demo 4.9

天真冷(点击)

事件处理函数

- 在特定时间发生时调用特定的 JS 代码。
  - 在给元素添加事件处理函数后，一旦事件发生，相应的 JS 代码得到执行。被调用的 JS 代码可以返回值，返回值将被传递给事件处理函数。
  - 事件处理函数名为 on + 事件名，如 onclick/onkeyup
  - 返回值为 true，继续原来元素执行时间处理函数后的动作；返回 false，取消执行。
  - 对象与事件解绑：设置 object.event = null;

例：点击链接后不进行跳转

```
<a href="bjfu.edu.cn" id="href" onclick="return false;">跳转链接</a>
```

（1）鼠标事件

click	鼠标单击事件
mouseover	鼠标移入事件
mouseout	鼠标移出事件
mousedown	鼠标按下事件
mouseup	鼠标松开事件
mousemove	鼠标移动事件

（2）键盘事件

keydown	按下键盘事件
keyup	松开键盘事件

场景：可通过键盘时间验证表单，松开瞬间验证输入合法性并反馈错误提示。

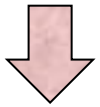
（2）键盘事件例：即时验证输入框的手机号码合法性

1. 选择事件?

- 需要在按下按钮并抬起后读取输入数据，利用 RegExp 正则表达式方法检查合法性。
故使用 onkeyup 事件处理函数。

2. 选择反馈效果?

- 验证失败输入框变红、在输入框上方显示提示文字等。



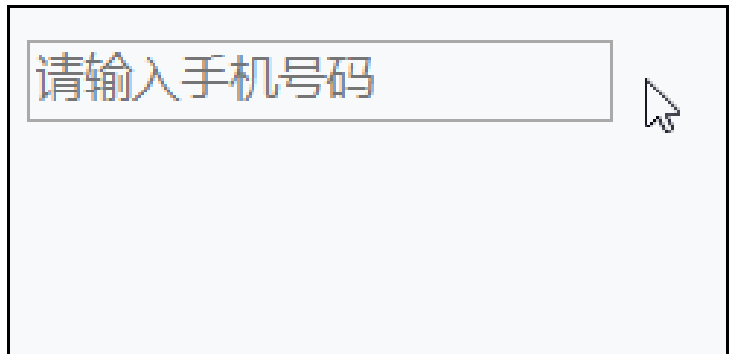
3. 如何实现反馈效果?

提前写好错误 CSS 样式代码，错误即修改 className 的值，添加或删除该样式。

提前预留 等标签，错误添加文本元素等。

键盘事件例：即时验证输入框的手机号码合法性

(2) 键盘事件例：即时验证输入框的手机号码合法性。只使输入框变红



CSS 样式定义

```
<style>
.error_bg {
    border: 1px solid indianred !important;
}
</style>
```

8 事件

HTML5 新特性，手机点击直接打开数字键盘

（2）键盘事件例：即时验证输入框的手机号码合法性。只使输入框变红。

Demo 4.10

```
<body>
<input type="mobile" id="mobile-input" placeholder="请输入手机号码">
<script>
input = document.getElementById("mobile-input");
var mobilePattern = /1[2-9]\d{9}/;  // 定义正则表达式
input.onkeyup = function(event) {   // 在用户输入后
  value = input.value;          // 读取 input 的输入值
  input.className = input.className.replace(/error_bg/, '');
      // 先删除 class 列表中的 .error_bg，防止多次添加 class
  if (!mobilePattern.test(value)) { // 正则表达式检查
    input.className += "error_bg";  // 如果错误，为 input 添加 class
  }
}
</script>
</body>
```

（3）表单事件

blur	当前字段失去焦点
focus	当前字段获得焦点
change	对于 <input><textarea> 元素，在失去焦点且 value 值改变时触发；<select> 元素在选项改变时触发
select	选中文字

8.3 表单事件

```
<input type="text" id="mobile-input">

<script>
input = document.getElementById("mobile-input");
input.onblur = function () {
  input.setAttribute("placeholder", "失去焦点");
}
input.onfocus = function () {
  input.setAttribute("placeholder", "获得焦点");
}
input.onchange = function () {
  input.style.borderColor = "indianred";
}
input.onselect = function (event) {
  alert("你选择的文字: " + window.getSelection().toString());
}
</script>
```

获得选中文字

Demo 4.11

8.4 编辑事件

copy	复制
selectstart	选中内容
contextmenu	鼠标右击

事件监听器

- 具备给同一个元素添加多个相同事件的能力。

  如：为同一个按钮添加三次 onclick 事件，事件处理函数方式只执行最后一次定义的函数代码。

- 使用 addEventListener() 将元素与事件绑定，

  使用 removeEventListener() 解除绑定。

object.addEventListener

- **object.addEventListener(*event, function, false*)**
  - *object:* 一个 DOM 对象;
  - *event:* 字符串表示的事件名，如 "click"/"keyup";
  - *function:* 函数，常用匿名函数;
  - *false:* 冒泡阶段调用，常用 false.

```
<button type="button" id="btn">多次点击</button>
<script>
button = document.getElementById("btn");
button.addEventListener("click", function() {
  alert("第一次");
}, false);
button.addEventListener("click", function() {
  alert("第二次");
}, false);
</script>
```

Demo 4.12

点击一次按钮，弹出两次 alert

object.removeEventListener

- **object.removeEventListener(*event*, *function*, *false*)**
  - *object:* 一个 DOM 对象;
  - *event:* 字符串表示的事件名，如 "click"/"keyup";
  - *function:* 函数;
  - *false:* 冒泡阶段调用，常用 false.

event 对象

event 对象

特定事件发生时，发生事件的信息被作为参数传入事件处理函数。

- event 的常用属性

属性	说明
type	事件类型
keyCode	键码值
shiftKey	是否按下 Shift 键
ctrlKey	是否按下 Ctrl 键
altKey	是否按下 Alt 键

```
<p>天真冷(点击)</p>

<script>
  example = document.getElementById("example");
  example.onclick = function (event) {
    example.innerHTML = '是的';
    alert(event.type);   // click
  }
</script>
```

event.keycode 属性

- **event.keycode 属性：返回键盘输入的键码值**

按键	键码
W	87
S	83
A	65
D	68
←	37
↑	38
→	39
↓	40

event.keycode

event 对象例：显示 WASD/↑←↓→ 对应关系

```
var error = document.getElementById("error");
document.onkeydown = function (e) {
  if (e.shiftKey || e.ctrlKey || e.altKey) {
    alert("请不要按 Shfit/Ctrl/Alt 键");
  }
}
window.addEventListener("keydown", function(e) {
  if (e.keyCode == 37 || e.keyCode == 65) {
     error.innerHTML = "上";
  } else if (e.keyCode == 38 || e.keyCode == 87) {
     error.innerHTML = "左";
  } else if (e.keyCode == 39 || e.keyCode == 68) {
     error.innerHTML = "右";
  } else if (e.keyCode == 40 || e.keyCode == 83) {
     error.innerHTML = "下";
  }
});
```

Demo 4.13