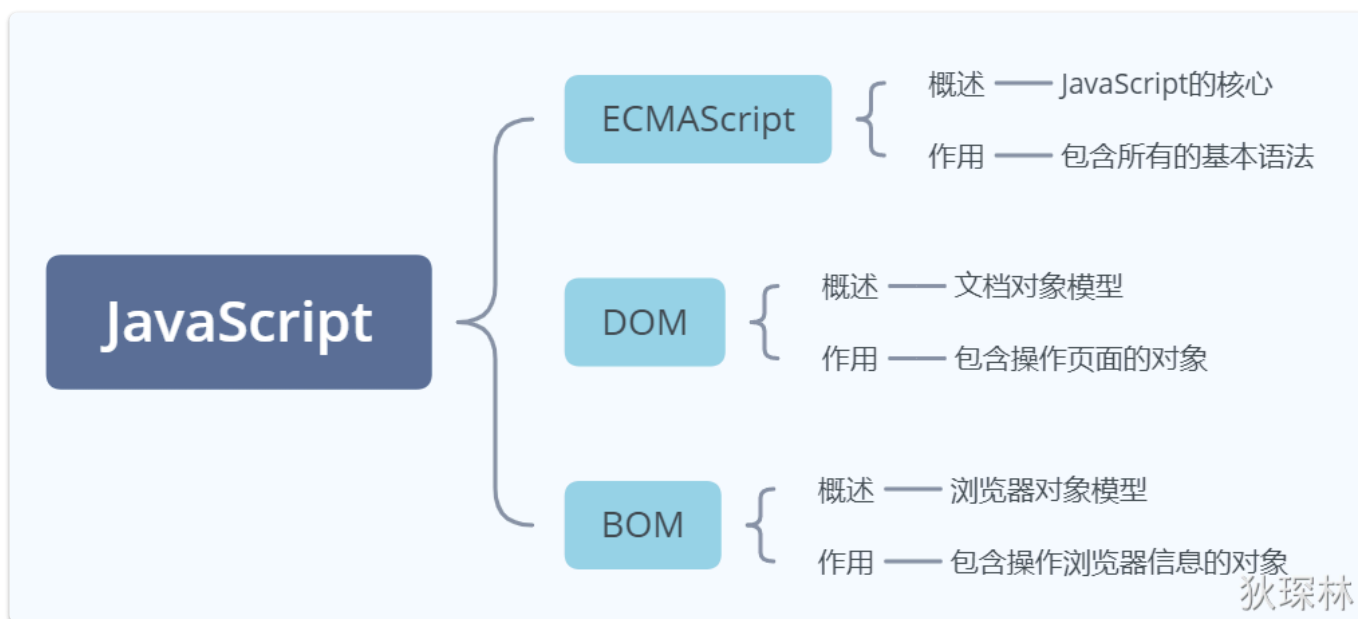


一、认识DOM

DOM：全称Document Object Model，翻译过来叫做“文档对象模型”。该内容是JavaScript的组成部分之一。

在JavaScript的世界里中一切皆对象，包括.html文档，为了更好的与html文档进行交互，抽象出来文档对象模型，即。

真正的JavaScript是由三部分组成，ECMAScript、DOM、BOM，如下图所示。



1.1 DOM对象概述

DOM是W3C-万维网联盟的标准，它定义了访问HTML和XML文档的标准。W3C文档对象模型（DOM）是中立与平台和语言的接口，它允许程序和脚本动态的访问和更新文档的内容、结构和样式的。通过DOM，可以访问所有的HTML元素，连同它们所包含的文本和属性。可以对其中的内容进行修改和删除，同时也可以创建新的元素。

W3C将DOM分为3个不同的部分：

1.核心DOM：针对任何结构化文档的标准模型；

2.XML DOM部分：针对XML文档的标准模型；

3.HTML DOM部分：针对HTML文档的标准模型；

XML DOM：定义了所有XML元素的对象和属性及访问它们的方法；

HTML DOM：定义了所有HTML元素的对象和属性及访问它们的方法；

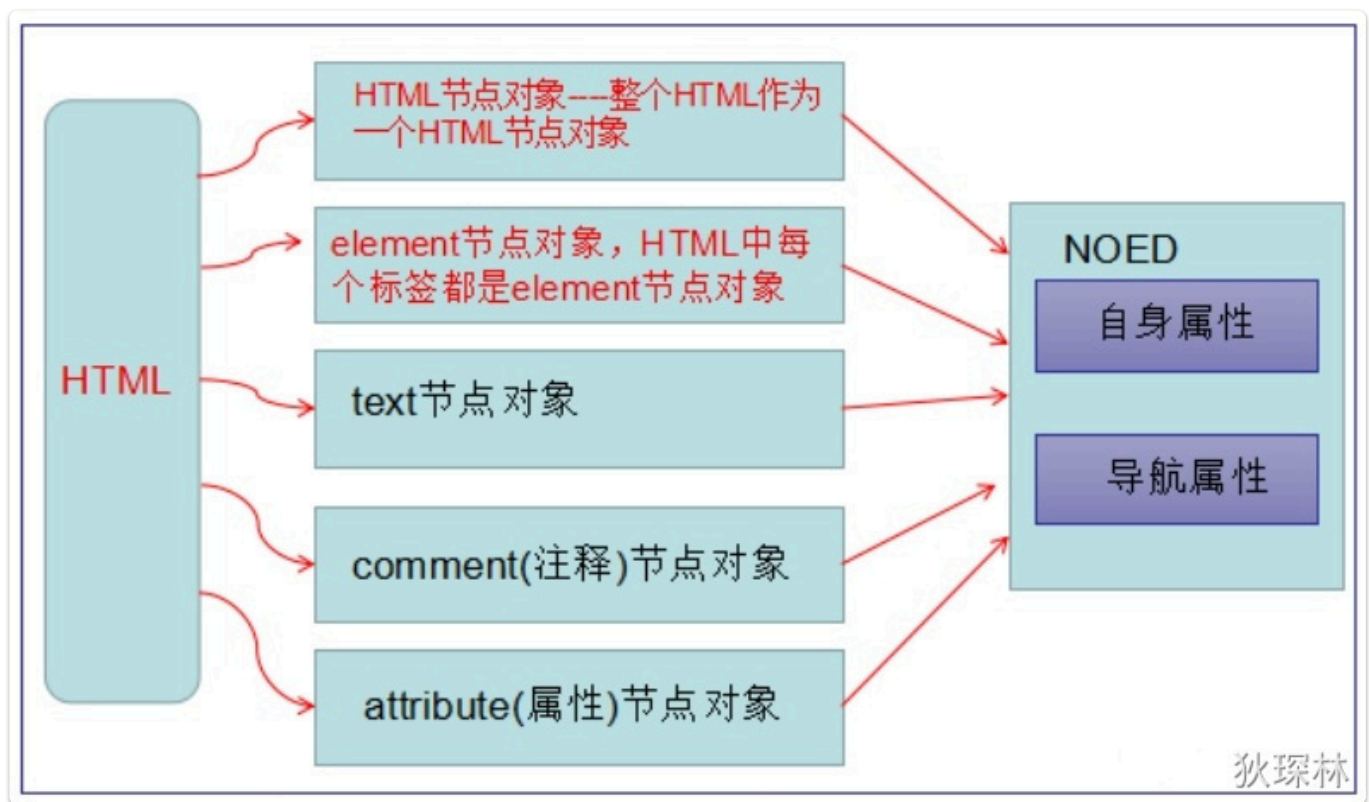
HTML DOM独立于平台和编程语言。它可被任何编程语言诸如Java、JavaScript和VBScript使用。

1.2 DOM节点分类

根据W3C的HTML DOM标准，HTML文档中的所有内容都是节点（NODE），分为5种节点类型：

- 1.整个文档是一个文档节点(Document对象),每个html页面整体;
- 2.每个HTML元素是元素节点(Element对象)，页面中的每个标签;
- 3.HTML元素内的文本是文本节点(Text对象);
- 4.每个HTML中属性是属性节点(Attribute对象);
- 5.注释是注释的节点(Comment对象);

如下图所示：



1.3 DOM节点树

当网页被加载时，浏览器会根据页面结构，创建页面的文档对象模型（Document Object Model）。

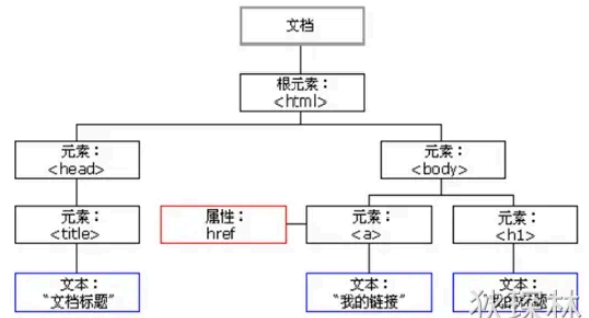
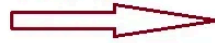
HTML 会被构造为DOM对象的树：

```

<html>
  <head>
    <title>文档标题</title>
  </head>
  <body>
    <h1>我的标题</h1>
    <a href="#">我的链接</a>
  </body>
</html>

```

浏览器加载



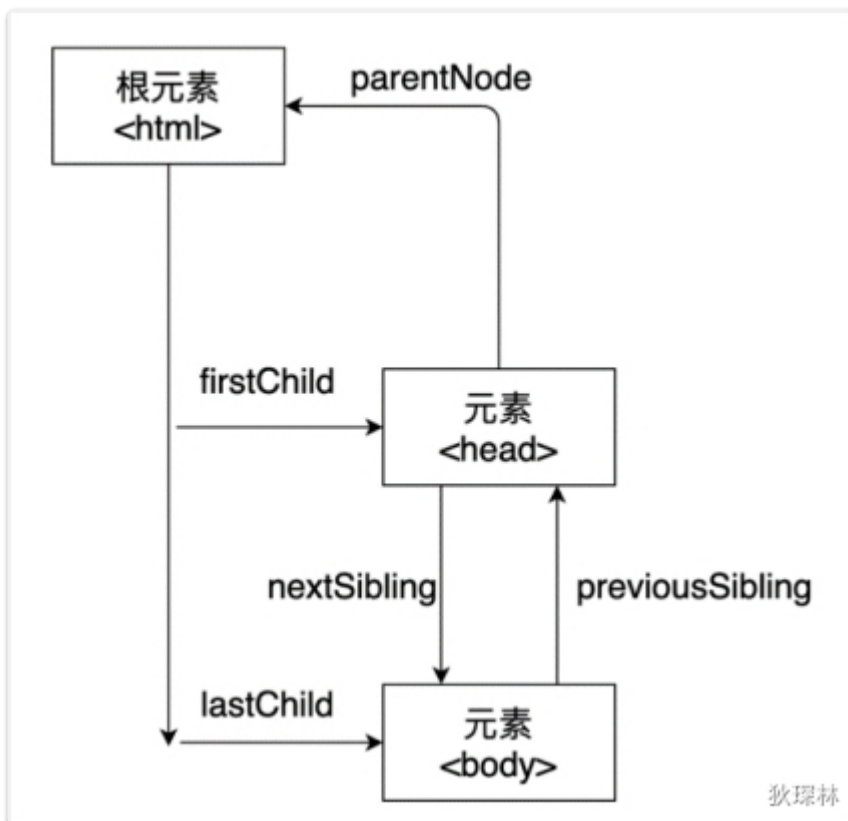
1.4 节点关系

节点树中的节点彼此拥有层级关系。

父(parent)，子(child)和同胞(sibling)等术语用于描述这些关系。父节点拥有子节点。同级的子节点被称为同胞（兄弟或姐妹）。

- 1.在节点树中，顶端节点被称为根（root）。
- 2.每个节点都有父节点、除了根（它没有父节点）。
- 3.一个节点可拥有任意数量的孩子。
- 4.同胞是拥有相同父节点的节点。

下面的图片展示了节点树的一部分，以及节点之间的关系：



二、DOM操作-查找节点

2.1 Document概述

当浏览器载入 HTML 文档时，整个 HTML 就会成为一个 **Document 对象**。
Document 对象是 HTML 文档的根节点。
Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

2.2 查找节点-直接查找

因为Document代表的是整个HTML文档，所以文档中的所有内容都包含在Document中。

如果想获取(访问)到某个HTML文档的节点，则需要通过Document的方法来完成。

访问HTML文档的节点主要有两种方式，一种是直接访问指定节点，另外一种是根据节点的层次关系访问节点。

直接访问指定节点就是一个查找方法就能找到相关的元素。在JavaScript中，有多种方法选取元素。这类方法有：

方法名	描述
document.getElementById(id)	返回匹配指定id属性的元素节点
document.getElementsByName(name)	用于选择拥有name属性的HTML元素
document.getElementsByClassName(className)	返回包括了所有class名字符合指定条件的元素
document.getElementsByTagName(tagName)	返回所有指定HTML标签的元素
document.querySelector(selector)	接受一个CSS选择器作为参数，返回第一个匹配该选择器的元素节点
document.querySelectorAll(selectors)	接受一个CSS选择器作为参数，返回所有匹配该选择器的元素节点

以下面文档为例：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>DOM查找节点</title>
  </head>
  <body>
    <div id="container">
      <div class="box">
        <div class="box-item">
          <p>这是段落1</p>
        </div>
      </div>
    </div>
  </body>
</html>
```

```

        </div>
        <div class="box-item">
            <p>这是段落2</p>
            <ul id="list">
                <li class="list-item"><a
href="#">百度</a></li>
                <li class="list-item"><a
href="#">腾讯</a></li>
                <li class="list-item"><a
href="#">阿里巴巴</a></li>
            </ul>
        </div>
        <input name="cityname" type="radio"
value="北京"> 北京
        <input name="cityname" type="radio"
value="上海"> 上海
    </div>
</div>
</body>
</html>

```

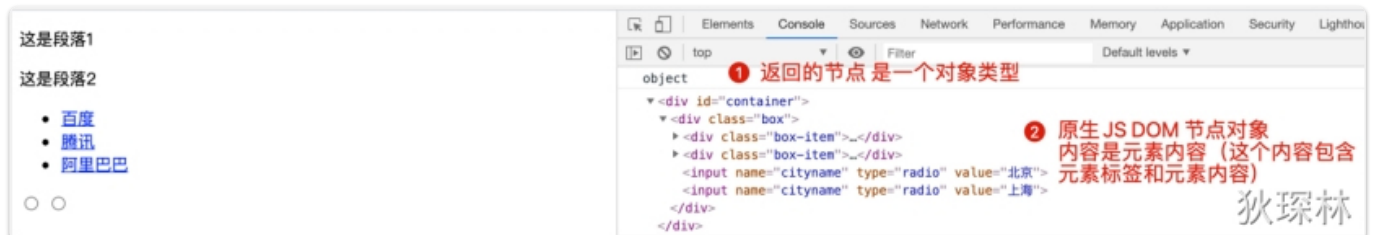
① 用指定的id属性获取

```

<script type="text/javascript">
    // 1. 通过 id 选择器的名字找到指定元素节点对象
    var get_container = document.getElementById("container");
    // 返回的内容是 元素节点对象
    console.log(typeof get_container);
    console.log(get_container);
</script>

```

执行显示：



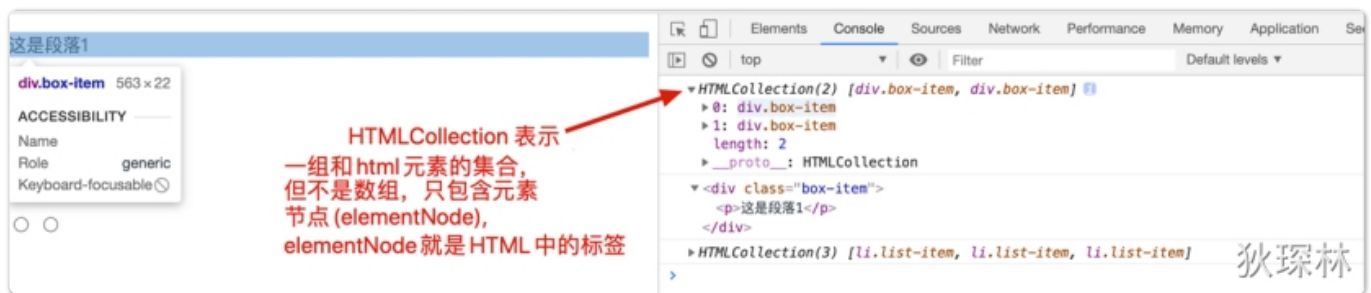
② 用指定的class属性获取

```

<script type="text/javascript">
    // 2. 通过 class 选择器的名字找到当前html文档中所有指定class 名的元素节点
    var get_box_items = document.getElementsByClassName("box-item");
    // 返回值是一个数组
    console.log(get_box_items);
    // 通过取下标的方式可以取出对应的元素
    console.log(get_box_items[0]);
    var get_list_items = document.getElementsByClassName("list-item");
    console.log(get_list_items);
</script>

```

执行结果：



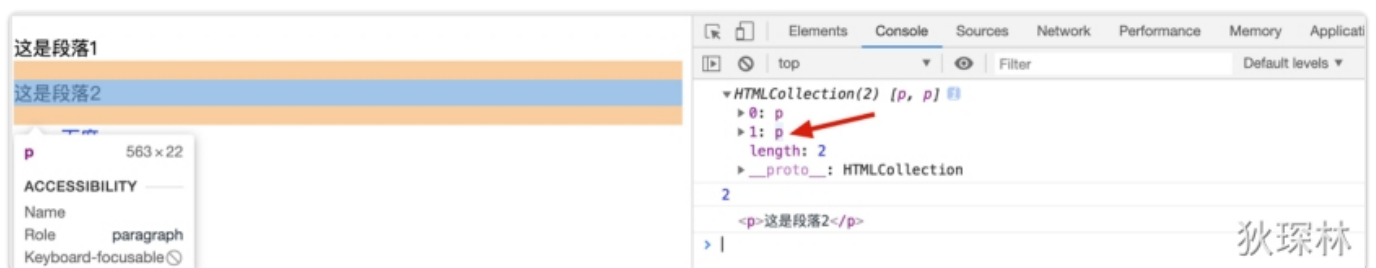
③ 通过标签名字获取

```

<script type="text/javascript">
    // 3. 通过 元素名称找到的所有匹配的元素节点
    var get_els_by_tag = document.getElementsByTagName("p");
    // 返回的是数组
    console.log(get_els_by_tag);
    // 获取数组长度
    console.log(get_els_by_tag.length);
    // 通过取下标的方式可以取出对应的元素
    console.log(get_els_by_tag[1]);
</script>

```

执行结果：



④ 指定的name属性获取

```
<script type="text/javascript">
  // 4. 通过 属性 name 获取所有匹配的元素节点
  var get_els_by_name = document.getElementsByName("cityname");
  console.log(get_els_by_name);
</script>
```

执行显示:

这是段落1

这是段落2

- 百度
- 腾讯
- 阿里巴巴

☒ 北京 ☐ 上海

NodeList 表示这是一个节点集合 (既包含元素, 也包含节点)

狄琛林

⑤ 匹配指定的CSS选择器

```
<script type="text/javascript">
  // 5. 通过 选择器名称查找匹配的元素节点
  var get_el_css_id_selector = document.querySelector("#list");
  console.log(get_el_css_id_selector);
  // 如果是匹配到许多, 也只会返回找到的第一个元素节点
  var get_el_css_class_selector = document.querySelector(".list-item");
  console.log(get_el_css_class_selector);
</script>
```

执行显示:

这是段落1

这是段落2

- 百度
- 腾讯
- 阿里巴巴

☐ 北京 ☐ 上海

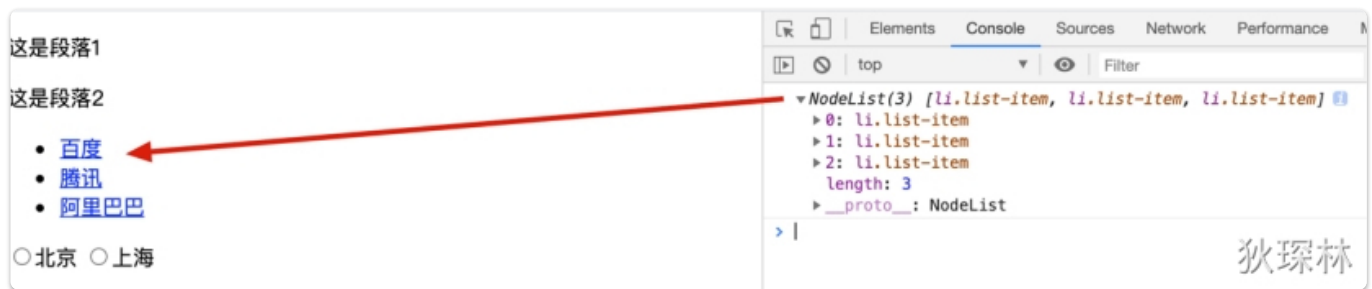
匹配指定的CSS选择器

狄琛林

⑥ 通过选择器名称获取所有匹配的元素节点


```
<script type="text/javascript">
    // 6. 通过 选择器名称获取所有匹配的元素节点
    var get_els_by_class_selector = document.querySelectorAll(".list-
item");
    console.log(get_els_by_class_selector);
</script>
```

执行显示：



2.3 查找节点-层级查找

通过上述的查找方式，我们可以快速的定位到某个节点元素，但如果想对查询到的元素进行再次的过滤的话，可以考虑层级查找。

需要注意的是，我们获取到的节点元素都是Node，所谓的层级查找，就是通过Node的属性来操作。

① Node常见属性

属性名	描述
parentNode	该节点的父节点， <code>**parentElement**</code> 获取父元素。
childNodes	该节点的所有子节点。 <code>**children**</code> 获取所有子元素。
firstChild	该节点的第一个子节点。 <code>**firstElementChild**</code> 获取第一个子元素。
lastChild	该节点的最后一个子节点。 <code>**lastElementChild**</code> 获取最后一个子元素。
nextSibling	该节点的后一个兄弟节点。 <code>**nextElementSibling**</code> 获取后一个兄弟元素。
previousSibling	该节点的前一个兄弟节点。 <code>**previousElementSibling**</code> 获取前一个兄弟元素。

② 案例演示

页面结构

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>文档结构和遍历</title>
  </head>
  <body>
    <div id="wrapper">
      <div class="test" id="first">
        <span>First text</span>
      </div>
      <!-- here is the comment -->
      <div class="test" id="second">Second text</div>
      <div class="test" id="third">Third text</div>
      <div class="test" id="four">Four text</div>
      <div class="test" id="five">Five text</div>
      <div class="test" id="six">Six text</div>
    </div>
  </body>
</html>
```

Node节点遍历

```
//0.必须提前找到一个元素，然后以这个元素为起点。进行一层层查找。
let first = document.getElementById("first");
console.log(first);
console.log("-----");

//1.parentElement:获取父元素。
console.log(first.parentElement);

//2.children:获取所有子元素。
console.log(first.children);

//3.firstElementChild:获取第一个子元素。
console.log(first.firstElementChild);

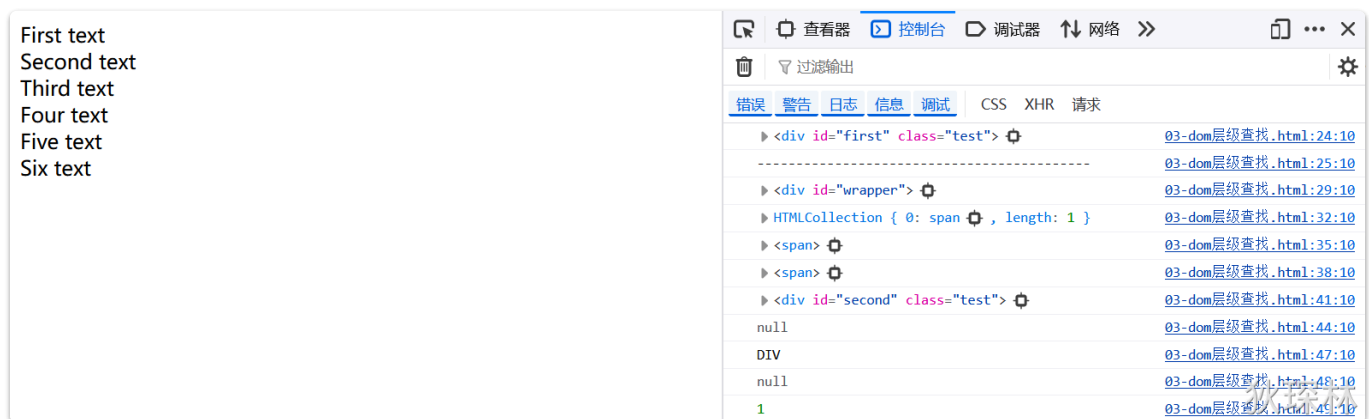
//4.lastElementChild:获取最后一个子元素。
console.log(first.lastElementChild);
```

```
//5.nextElementSibling:获取后一个兄弟元素。
console.log(first.nextElementSibling);

//6.previousElementSibling:获取前一个兄弟元素。
console.log(first.previousElementSibling);

//7.获取节点相关属性
console.log(first.nodeName);    //获取标签名
console.log(first.nodeValue);  //获取标签内容
console.log(first.nodeType);    //获取标签类型(1标签, 3文本, 8注释, 9文档,
11文档声明)
```

执行显示



三、DOM操作-操作内容

在获取元素之后，可以根据开发功能的需要，获取或修改制定元素的内容。元素的内容包括含普通字符串和带有元素标签的字符串，所以，Element对象提供了4种属性进行操作：

属性名	描述
innerHTML	设置或获取位于对象起始和结束标签内的HTML
outerHTML	设置或获取对象及其内容的HTML形式
innerText	设置或获取位于对象起始和结束标签内的文本
outerText	设置(包括标签)或获取(不包括标签)对象的文本

3.1 演示案例

① 页面内容

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>文档内容</title>
  </head>
  <body>
    <div id="box">
      这是div中的文字
      <span>这是span中的文字</span>
    </div>
    <div id="show" style="border: 1px solid red;"></div>
  </body>
</html>
```

② 测试代码

```
<script type="text/javascript">
  var get_box = document.querySelector("#box");
  var temp_str = "";
  temp_str += "innerHTML:" + get_box.innerHTML + "<br />" + "\r\n";
  temp_str += "outerHTML:" + get_box.outerHTML + "<br />" + "\r\n";
  temp_str += "innerText:" + get_box.innerText + "<br />" + "\r\n";
  temp_str += "outerText:" + get_box.outerText + "<br />" + "\r\n";

  var show = document.querySelector("#show");
  show.innerText = temp_str;
</script>
```

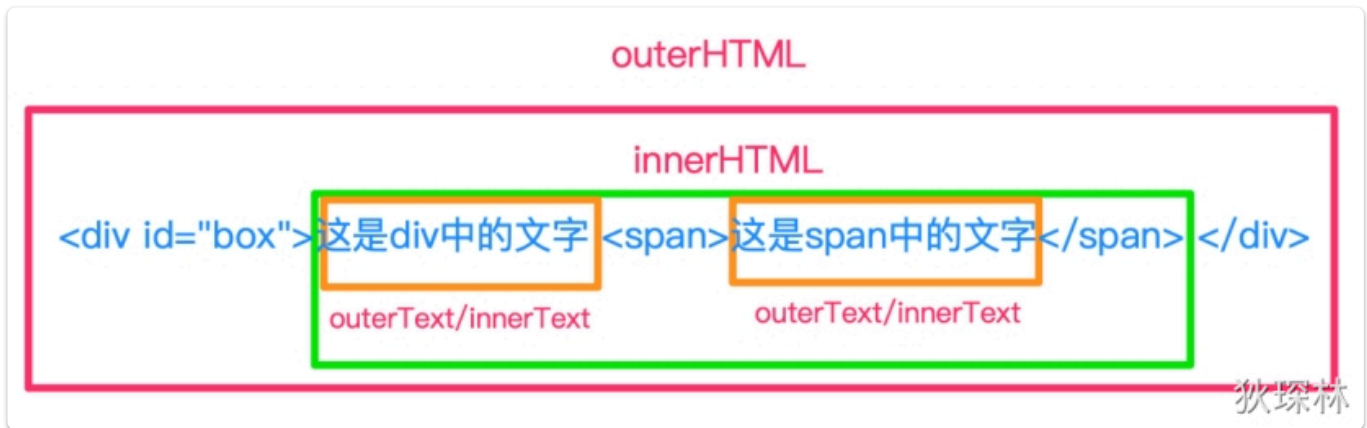
③ 显示结果

这是div中的文字 这是span中的文字

```
innerHTML:
这是div中的文字
<span>这是span中的文字</span>
<br />
outerHTML:<div id="box">
这是div中的文字
<span>这是span中的文字</span>
</div><br />
innerText:这是div中的文字 这是span中的文字<br />
outerText:这是div中的文字 这是span中的文字<br />
```

3.2 案例总结

1. `innerHTML` 获取的是当前对象(标签) **内部** 的所有内容, 包含html标签和文本。
2. `outerHTML` 获取的是当前对象(标签) **本身+ 内部** 的所有内容, 包含html标签和文本。
3. `innerText` 和 `outerText` 获取的都是当前对象(标签) **内部** 的所有文本, 不包含html



3.3 相关补充

上述四个属性, 既可以用于获取, 也可以用于设置。但需要注意的是:

- `innerText`: 内容以字符串的方式出现。
- `innerHTML`: 内容以HTML格式进行解析。

```
// 设置元素内容以字符串的方式
show.innerText = temp_str;

// 设置元素内容以HTML格式进行解析
show.innerHTML = temp_str;
```

显示效果:

这是div中的文字这是span中的文字

innerHTML:这是div中的文字这是span中的文字

outerHTML:

这是div中的文字这是span中的文字

innerText:这是div中的文字这是span中的文字

outerText:这是div中的文字这是span中的文字

狄琛林

四、DOM操作-操作属性

DOM会把HTML文档映射为一棵DOM树, HTML文档中的每个标签都是DOM树上的一个节点, 而DOM树上的每个节点其实就是一个JS对象。所以, 如果想操作HTML文档中的

某个标签，则找到这个标签在DOM数上对应的元素节点，直接操作这个JS对象即可。操作这个JS对象的属性，就可以看作是在操作标签的属性。

4.1 获取标签属性

① 案例代码

```
<input type="text" name="city-name" id="city" value="北京" />
```

```
// 1. 通过 name 属性找到标签
var get_input = document.querySelector("[name=city-name]");

// 2. 获取该元素的属性
console.log(get_input.attributes);
```

② 执行效果



4.2 操作标签属性

① 相关概述

在DOM中，通过`element.attributes`获取到属性，称为属性property，property是DOM元素在JS中作为对象拥有的属性节点。也就是说，JS对象可以直接通过`**.**`来操作这些属性。

在DOM中，还有一种是属性节点(Attribute节点)，而这些节点无法通过`**.**`来操作，要操作特性节点，必须在DOM元素上通过`getAttribute("属性名")`、`setAttribute("属性名", 属性值)`、`hasAttribute("属性名")`、`removeAttribute("属性名")`等方法来实现，这些称为特性节点。

② 补充说明

Attribute：特性，其实是标签语言中的概念。用于表述标签的附加信息。如：*type*、*value*均是元素的特性

Property：属性，JavaScript对象的概念，用于描述JavaScript对象的成员。即：JavaScript对象的属性。

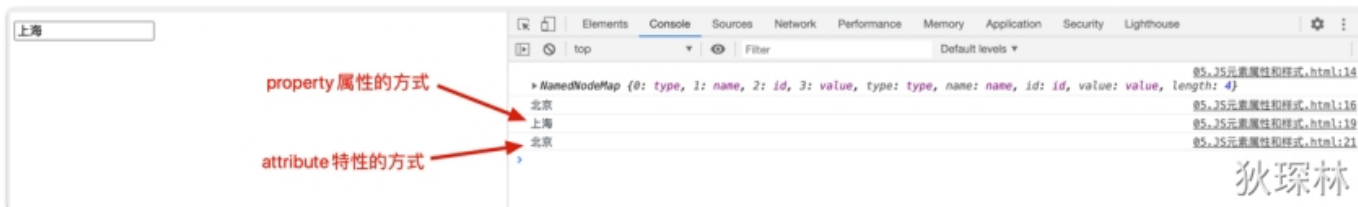
③ 注意事项

1. 多数情况下，*Attribute*和*Property*名称相同，且效果相当。
2. *Property*属性的操作更关注内存中的对象，*Attribute*特性更关注标签本身。
3. 通过*Property*属性的方式修改的值，*Attribute*特性获取到的可能还是原来的值。

```
// 1. 获取 input 的 value
console.log(get_input.value); // 北京

// 2. 修改 value的值
get_input.value = "上海";
console.log(get_input.value);

// 3. 使用特性的方式来获取 指定的属性
console.log(get_input.getAttribute("value"));
```



五、DOM操作-操作节点

在网页开发中，如果想要动态地改变网页内容，可以对现有节点属性、节点、节点样式进行操作，修改其内容。对于可变数据的内容的加载显示，就需要根据数据的内容进行动态操作HTML元素，这个操作包括增、删、改、查。增就是新增元素标签，删就是删除元素标签。

5.1 创建和插入节点

使用JavaScript操作DOM有很多方法可以创建或增加一个新节点，主要方法如下表：

方法	描述
document.createElement(tagName)	通过指定名称创建一个元素（创建一个节点）
element.appendChild(new_element)	向节点的子节点列表的末尾添加新的子节点
element.insertBefore(newNode,oldNode)	在oldNode前边添加加入newNode

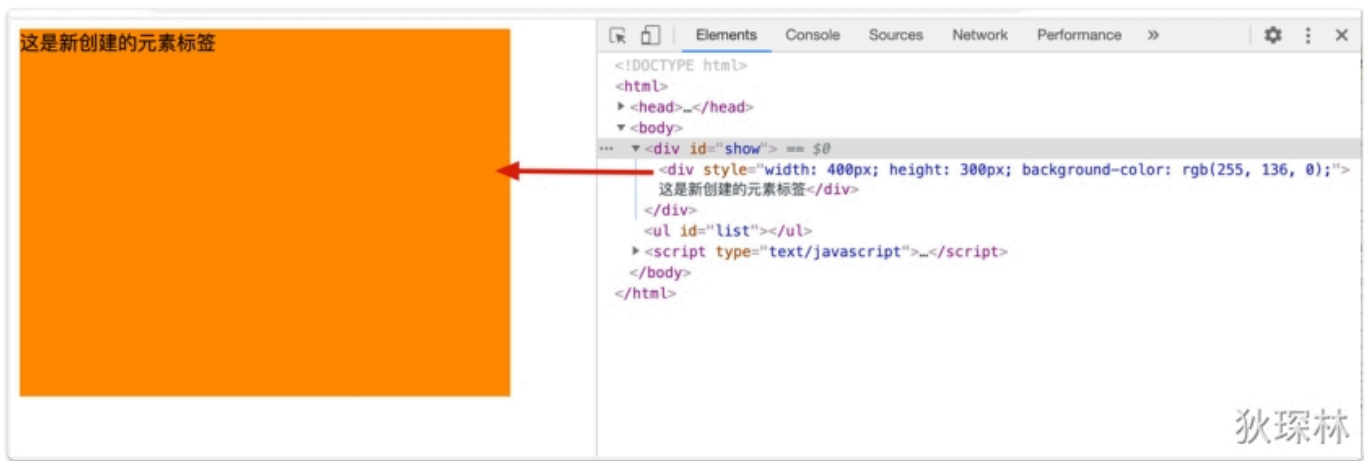
① 演示-页面代码

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>创建和插入节点</title>
  </head>
  <body>
    <div id="show"></div>
    <ul id="list"></ul>
  </body>
</html>
```

② 演示-JS操作

```
<script type="text/javascript">
  // 1. 创建节点
  var get_div = document.querySelector("div#show");
  // 1.1 创建一个 div 节点对象
  var new_div = document.createElement("div");
  // 1.2 初始化节点元素
  new_div.style.width = "400px";
  new_div.style.height = "300px";
  new_div.style.backgroundColor = "#FF8800";
  new_div.innerText = "这是新创建的元素标签";
  // 1.3 添加到指定元素节点中
  get_div.appendChild(new_div);
</script>
```

③ 演示-执行结果



④ 演示-循环添加多个

```
// 2. 一次添加多个节点
var get_el = document.querySelector("ul#list");
// 2.1 通过for 循环
for (let i = 0; i < 10; i++) {
  // 2.2 创建节点元素
  let li_el = document.createElement("li");
  // 2.3 初始化节点元素
  li_el.innerHTML = "这是第" + i + "li标签";
  li_el.style.font = "28px/1.5 #FF8800 tahoma,arial,'Hiragino Sans GB', '\5b8b\4f53',sans-serif";
  // 2.4 添加到指定元素节点中
  get_el.appendChild(li_el);
}
```

⑤ 演示-最终效果



5.2 删除节点

删除节点的dom方法：

方法	描述
<code>element.removeChild(node)</code>	父节点删除指定子节点

① 演示-页面结构：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>删除节点</title>
  </head>
  <body>
    <ul id="list">
      <li><span>第1个li</span><a href="#" onclick="handleClick(this)">删除</a></li>
      <li><span>第2个li</span><a href="#" onclick="handleClick(this)">删除</a></li>
      <li><span>第3个li</span><a href="#" onclick="handleClick(this)">删除</a></li>
      <li><span>第4个li</span><a href="#" onclick="handleClick(this)">删除</a></li>
      <li><span>第5个li</span><a href="#" onclick="handleClick(this)">删除</a></li>
    </ul>
  </body>
</html>
```

② 演示-Dom操作

```
<script>
  function handleClick(element) {
    // 这里的element是触发事件的按钮元素
    console.log(element); // 输出: <button onclick="handleClick(this)">点击我</button>

    // 获取链接所在的父元素li
```

```

    let li = element.parentElement;

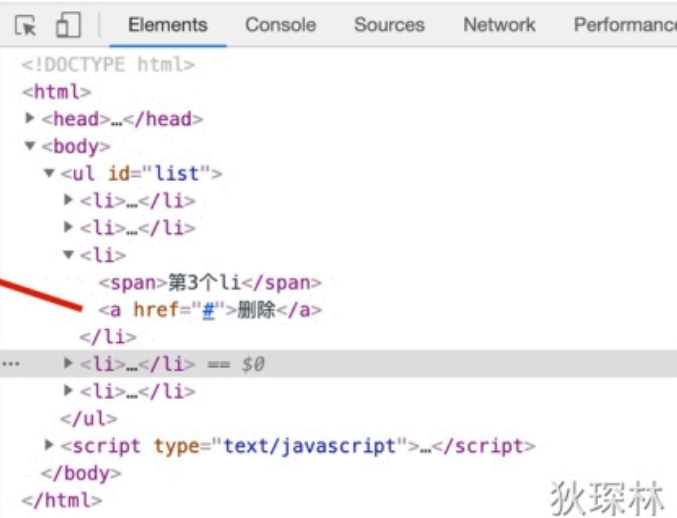
    // 获取链接所在的祖父元素ul
    let ul = element.parentElement.parentElement;

    // 执行删除：祖父删除父亲
    ul.removeChild(li);
  }
</script>

```

③ 演示-点击删除

- 第1个li删除
- 第2个li删除
- 第3个li删除
- 第4个li删除
- 第5个li删除

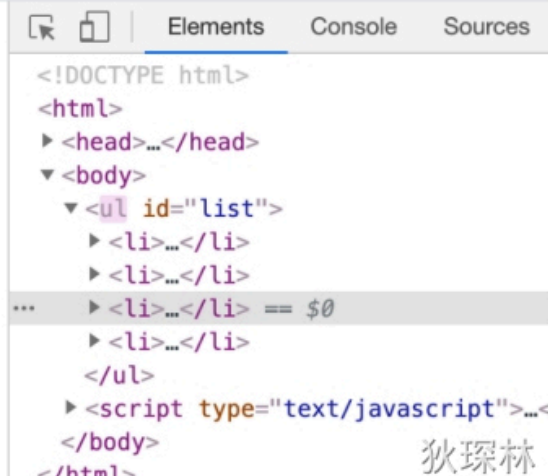


The screenshot shows the browser's 'Elements' panel. The DOM tree displays a <ul id='list'> containing three elements. The third is expanded, showing a 第3个li and a 删除 link. A red arrow originates from the '第3个li删除' text in the list on the left and points to the third element in the DOM tree.

狄琛林

④ 演示-最终效果

- 第1个li删除
- 第2个li删除
- 第4个li删除
- 第5个li删除



The screenshot shows the browser's 'Elements' panel after the third list item has been removed. The DOM tree displays a <ul id='list'> containing only two elements. The third element is no longer present, and the fourth (which originally contained the delete link) is now the third element in the list.

狄琛林