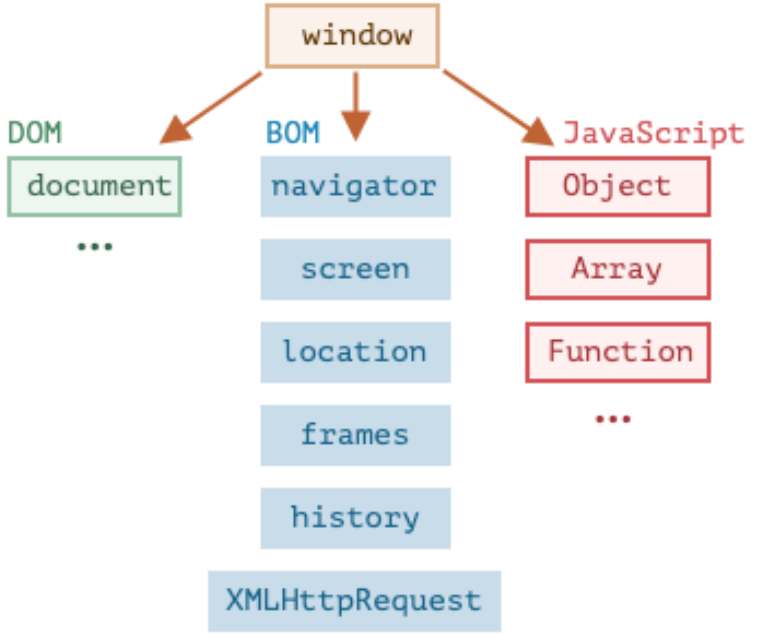
# JavaScript的DOM操作(一)

王红元 coderwhy

# 目录
# content

# 认识DOM和BOM

- **前面我们花了很多时间学习JavaScript的基本语法，但是这些基本语法，但是这些语法好像和做网页没有什么关系，和前面学习的HTML、CSS也没有什么关系呢?**
  - 这是因为我们前面学习的部分属于ECMAScript，也就是JavaScript本身的语法部分；
  - 除了语法部分之外，我们还需要学习浏览器提供给我们开发者的DOM、BOM相关的API才能对页面、浏览器进行操作；
- **前面我们学习了一个window的全局对象，window上事实上就包含了这些内容：**
  - 我们已经学习了JavaScript语法部分的Object、Array、Date等；
  - 另外还有DOM、BOM部分；
- **DOM：文档对象模型（Document Object Model）**
  - 简称 DOM，将页面所有的内容表示为可以修改的对象；
- **BOM：浏览器对象模型（Browser Object Model）**
  - 简称 BOM，由浏览器提供的用于处理文档（document）之外的所有内容的其他对象；
  - 比如navigator、location、history等对象；



# 深入理解DOM

- **浏览器会对我们编写的HTML、CSS进行渲染，同时它又要考虑我们可能会通过JavaScript来对其进行操作：**
  - 于是浏览器将我们编写在HTML中的每一个元素（Element）都抽象成了一个个对象；
  - 所有这些对象都可以通过JavaScript来对其进行访问，那么我们就可以通过JavaScript来操作页面；
  - 所以，我们将这个抽象过程称之为 文档对象模型（Document Object Model）；
- **整个文档被抽象到 document 对象中：**
  - 比如document.documentElement对应的是html元素；
  - 比如document.body对应的是body元素；
  - 比如document.head对应的是head元素；
- **下面的一行代码可以让整个页面变成红色：**

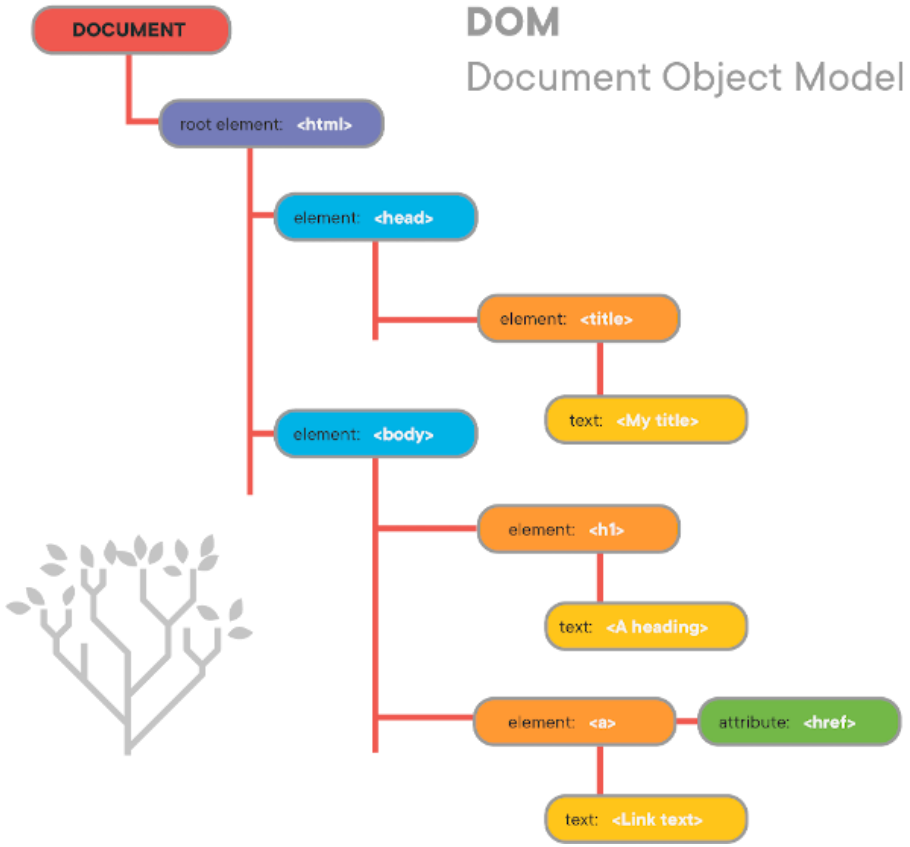
```
document.body.style.backgroundColor = "red"
```

- **所以我们学习DOM，就是在学习如何通过JavaScript对文档进行操作的；**

# DOM Tree的理解

- **一个页面不只是有html、head、body元素，也包括很多的子元素：**
  - 在html结构中，最终会形成一个树结构；
  - 在抽象成DOM对象的时候，它们也会形成一个树结构，我们称之为DOM Tree；

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My Title</title>
</head>
<body>
  <h1>A Heading</h1>
  <a href="#">Link Text</a>
</body>
</html>
```



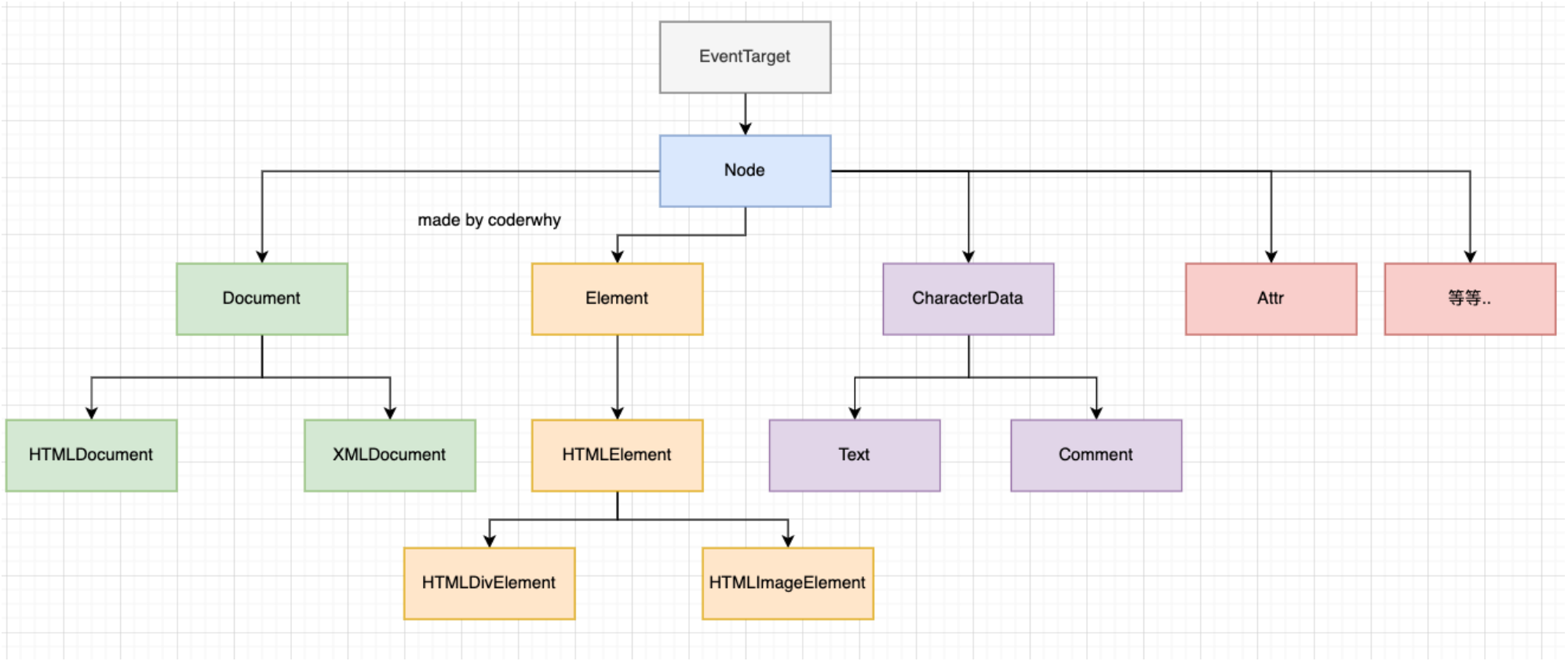
# DOM的学习顺序

- **DOM相关的API非常多，我们会通过如下顺序来学习：**
- 1.DOM元素之间的关系
- 2.获取DOM元素
- 3.DOM节点的type、tag、content
- 4.DOM节点的attributes、properies
- 5.DOM节点的创建、插入、克隆、删除
- 6.DOM节点的样式、类
- 7.DOM元素/window的大小、滚动、坐标

- **整体会按照这个顺序来学习，也会额外补充其他的知识。**

# DOM的继承关系图

- DOM相当于是JavaScript和HTML、CSS之间的桥梁
  - 通过浏览器提供给我们的DOM API，我们可以对元素以及其中的内容做任何事情；
- 类型之间有如下的继承关系：



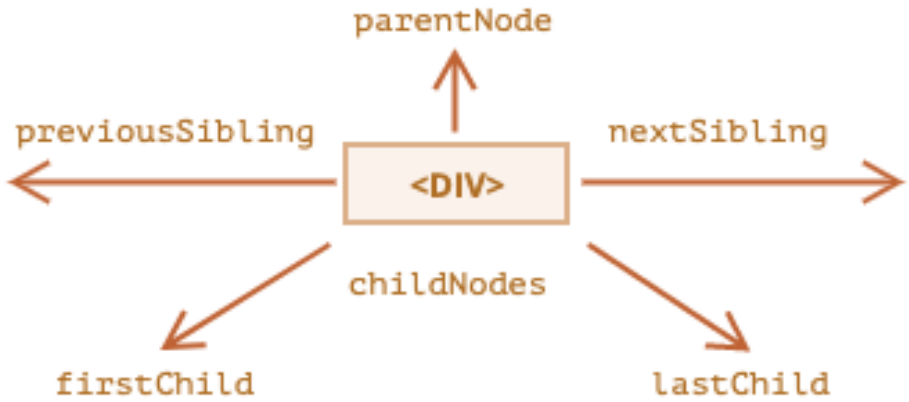
# document对象

- **Document节点表示的整个载入的网页，它的实例是全局的document对象：**
  - 对DOM的所有操作都是从 document 对象开始的；
  - 它是DOM的 入口点，可以从document开始去访问任何节点元素；
- **对于最顶层的html、head、body元素，我们可以直接在document对象中获取到：**
  - html元素：<html> = document.documentElement
  - body元素：<body> = document.body
  - head元素：<head> = document.head
  - 文档声明：<!DOCTYPE html> = document.doctype

```
console.log(document.doctype)
console.log(document.documentElement)
console.log(document.head)
console.log(document.body)
```

# 节点（Node）之间的导航（navigator）

- 如果我们获取到**一个节点（Node）**后，可以根据**这个节点去获取其他的节点**，我们称之为**节点之间的导航**。
- 节点之间存在如下的关系：
  - 父节点：parentNode
  - 前兄弟节点：previousSibling
  - 后兄弟节点：nextSibling
  - 子节点：childNodes
  - 第一个子节点：firstChild
  - 第二个子节点：lastChild

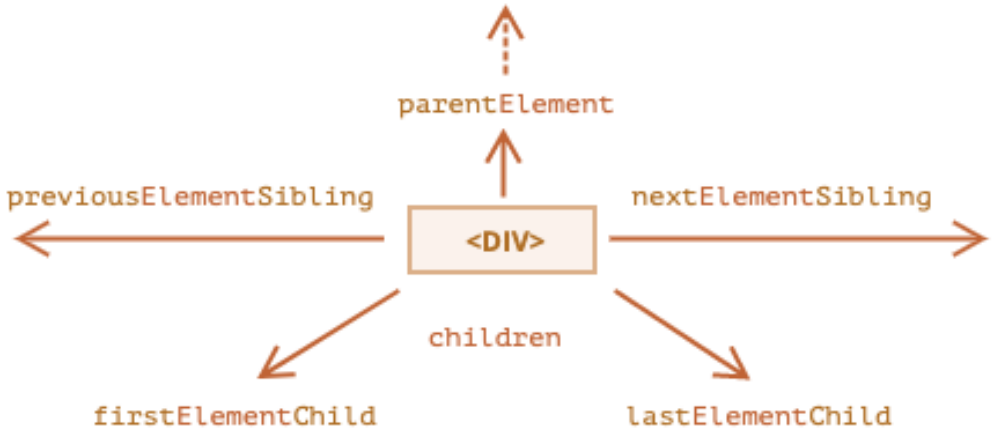



- 尝试获取下面结构的节点：

```
<div class="box">
  <!-- 我是注释 -->
  <h1 class="title">我是标题</h1>
  <div class="container">我是div元素</div>
  <div class="desc">我是一个段落</div>
  我是文本
</div>
```

# 元素（Element）之间的导航（navigator）

- 如果我们获取到一个**元素（Element）**后，可以根据**这个元素去获取其他的元素**，我们称之为**元素之间的导航**。
- 节点之间存在如下的关系：
  - 父元素：parentElement
  - 前兄弟节点：previousElementSibling
  - 后兄弟节点：nextElementSibling
  - 子节点：children
  - 第一个子节点：firstElementChild
  - 第二个子节点：lastElementChild




- 尝试获取下面结构的元素：

```
<div class="box">
  <!-- 我是注释 -->
  <h1 class="title">我是标题</h1>
  <div class="container">我是div元素</div>
  <div class="desc">我是一个段落</div>
  我是文本
</div>
```

# 表格（table）元素的导航（navigator）

- **<table> 元素支持 (除了上面给出的，之外) 以下这些属性：**
  - table.rows — <tr> 元素的集合；
  - table.caption/tHead/tFoot — 引用元素 <caption>，<thead>，<tfoot>；
  - table.tBodies — <tbody> 元素的集合；
- **<thead>，<tfoot>，<tbody> 元素提供了 rows 属性：**
  - tbody.rows — 表格内部 <tr> 元素的集合；
- **<tr>：**
  - tr.cells — 在给定 <tr> 中的 <td> 和 <th> 单元格的集合；
  - tr.sectionRowIndex — 给定的 <tr> 在封闭的 <thead>/<tbody>/<tfoot> 中的位置（索引）；
  - tr.rowIndex — 在整个表格中 <tr> 的编号（包括表格的所有行）；
- **<td> 和 <th>：**
  - td.cellIndex — 在封闭的 <tr> 中单元格的编号。

# 获取元素的方法

- **当元素彼此靠近或者相邻时，DOM 导航属性（navigation property）非常有用。**
  - 但是，在实际开发中，我们希望可以任意的获取到某一个元素应该如何操作呢?
- **DOM为我们提供了获取元素的方法:**

| 方法名 | 搜索方式 | 可以在元素上调用? | 实时的? |
| --- | --- | --- | --- |
| querySelector | CSS-selector | ✓ | - |
| querySelectorAll | CSS-selector | ✓ | - |
| getElementById | id | - | - |
| getElementsByName | name | - | ✓ |
| getElementsByTagName | tag or '*' | ✓ | ✓ |
| getElementsByClassName | class | ✓ | ✓ |

- **开发中如何选择呢?**
  - 目前最常用的是querySelector和querySelectAll;
  - getElementById偶尔也会使用或者在适配一些低版本浏览器时;

# 节点的属性 - nodeType

- **目前，我们已经可以获取到节点了，接下来我们来看一下节点中有哪些常见的属性：**
  - 当然，不同的节点类型有可能有不同的属性；
  - 这里我们主要讨论节点共有的属性；
- **nodeType属性：**
  - nodeType 属性提供了一种获取节点类型的方法；
  - 它有一个数值型值（numeric value）；
- **常见的节点类型有如下：**

| 常量 | 值 | 描述 |
| --- | --- | --- |
| Node.ELEMENT_NODE | 1 | 一个 元素 节点，例如 <p> 和 。 |
| Node.TEXT_NODE | 3 | Element 或者 Attr 中实际的 文字 |
| Node.COMMENT_NODE | 8 | 一个 Comment 节点。 |
| Node.DOCUMENT_NODE | 9 | 一个 Document 节点。 |
| Node.DOCUMENT_TYPE_NODE | 10 | 描述文档类型的 DocumentType 节点。例如 <!DOCTYPE html> 就是用于 HTML5 的。 |

- 其他类型可以查看MDN文档： https://developer.mozilla.org/zh-CN/docs/Web/API/Node/nodeType

# 节点的属性 – nodeName、tagName

- nodeName：获取node节点的名字；
- tagName：获取元素的标签名词；

```
var textNode = document.body.firstChild
var itemNode = document.body.childNodes[3]
console.log(textNode.nodeName)
console.log(itemNode.nodeName)
```

- tagName 和 nodeName 之间有什么不同呢?
  - tagName 属性仅适用于 Element 节点；
  - nodeName 是为任意 Node 定义的：
    - ✓ 对于元素，它的意义与 tagName 相同，所以使用哪一个都是可以的；
    - ✓ 对于其他节点类型（text，comment 等），它拥有一个对应节点类型的字符串；

# 节点的属性 - innerHTML、textContent

- **innerHTML 属性**
  - 将元素中的 HTML 获取为字符串形式；
  - 设置元素中的内容；
- **outerHTML 属性**
  - 包含了元素的完整 HTML
  - innerHTML 加上元素本身一样；
- **textContent 属性**
  - 仅仅获取元素中的文本内容；

- **innerHTML和textContent的区别：**
  - 使用 innerHTML，我们将其“作为 HTML”插入，带有所有 HTML 标签。
  - 使用 textContent，我们将其“作为文本”插入，所有符号（symbol）均按字面意义处理。

# 节点的属性 - nodeValue

- nodeValue/data
  - 用于获取非元素节点的文本内容

```
coderwhy
<!-- 注释内容 -->

<script>
  var text = document.body.firstChild
  console.log(text.nodeValue)

  var comment = text.nextSibling
  console.log(comment.nodeValue)
</script>
```

# 节点的其他属性

- **hidden属性：也是一个全局属性，可以用于设置元素隐藏。**

```
<div class="box">哈哈哈哈</div>

<script>
  var box = document.querySelector(".box")
  box.hidden = true
</script>
```

- **DOM 元素还有其他属性：**
  - value
    - ✓ <input>，<select> 和 <textarea>（HTMLInputElement，HTMLSelectElement……）的 value。
  - href
    - ✓ <a href="..."> （HTMLAnchorElement）的 href。
  - id
    - ✓ 所有元素（HTMLElement）的 “id” 特性（attribute）的值。
- **class和style我们会在后续专门讲解的。**