JavaScript - DOM

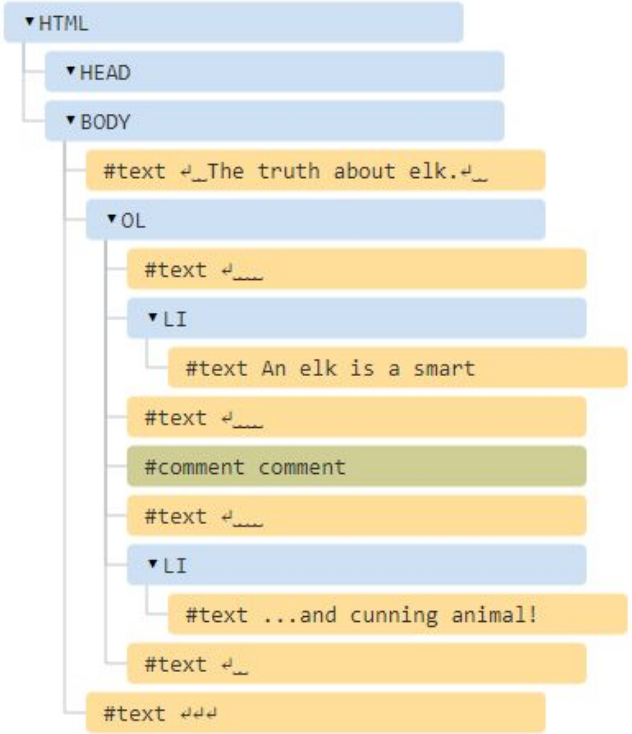
Document Object Model - DOM

- The HTML DOM defines a standard way to access and **manipulate** HTML documents
- Represents all page content as objects that can be modified
- All HTML elements, along with their containing text and attributes, can be **accessed** through the DOM
- The contents can be **modified** or **deleted** and new elements can be created
- JavaScript can query or modify the HTML document
- The HTML DOM is platform and language independent

The DOM Tree and Nodes

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>
The truth about elk.
<ol>
   <li>An elk is a smart</li>
   <!-- comment -->
   <li>...and cunning animal!</li>
</ol>
</body>
</html>
```

- Every tree node is an object.



DOM hierarchy

- Rooted at window.document
- Follows HTML document structure
  - window.document.head
  - window.document.body
- Every HTML tag is an object
- Nested tags are "children" of the tag that contains them
- The text inside a tag is an object as well

Nodes

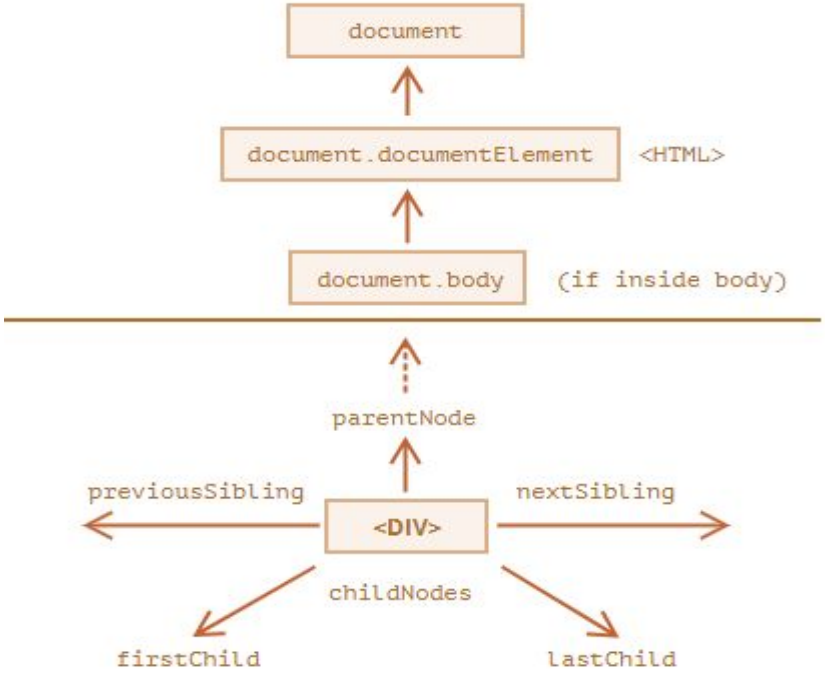
- All items in the DOM are defined as nodes.
- There are many types of nodes
- The three main ones that we work with most often:
  - Element nodes
  - Text nodes, contains only a string. It may not have children and is always a leaf of the tree
  - Comment nodes

Nodes

- To get the “type” of a DOM node:
  - elem.nodeType → old way
    - elem.nodeType == 1 for element nodes
    - elem.nodeType == 3 for text nodes
    - elem.nodeType == 9 for the document object
  - instanceof → new way

document

- The document object is the main “entry point” to the page.



children, childNodes, firstChild and lastChild

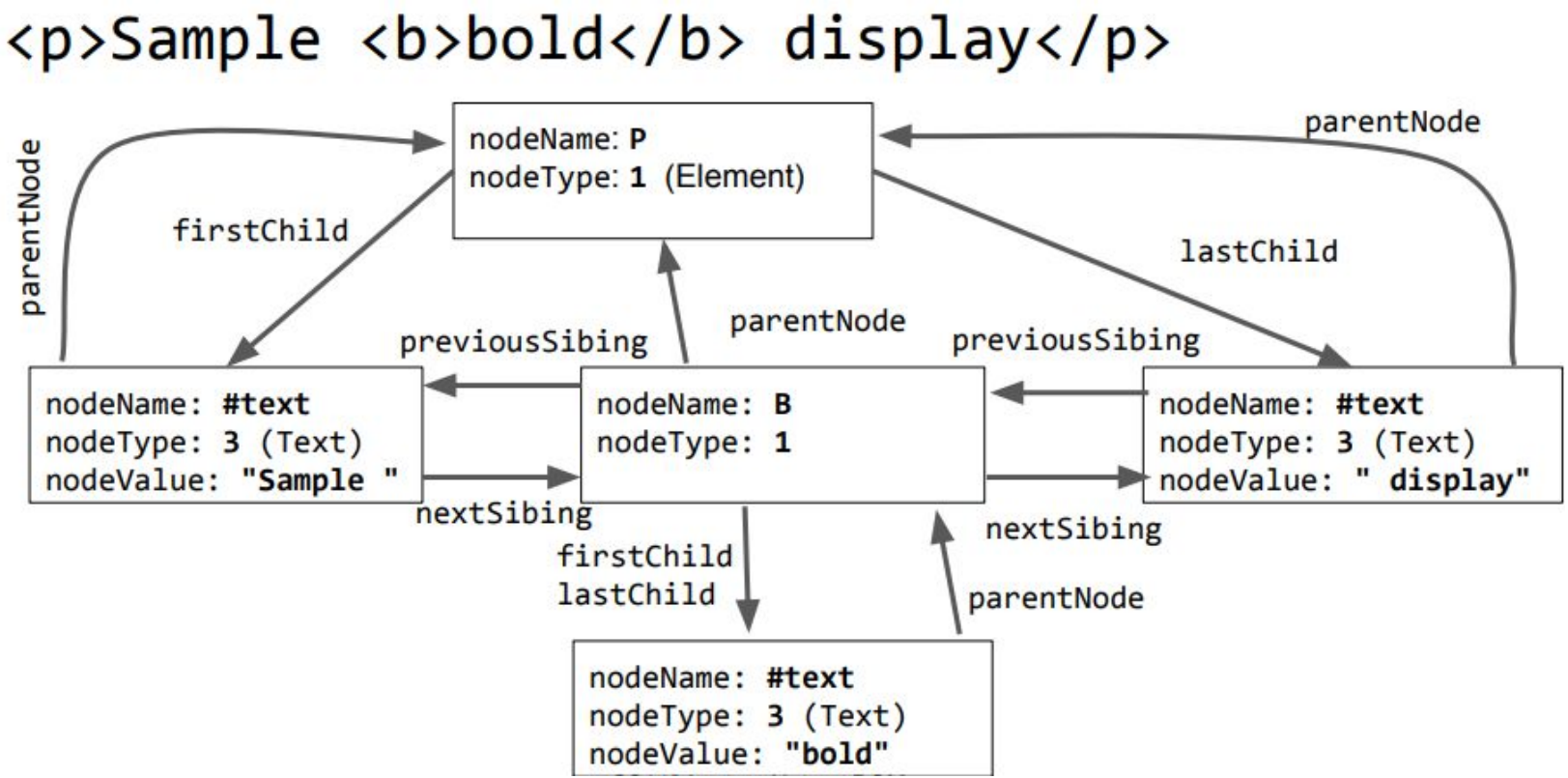
- childNodes looks like an array

```
elem.childNodes[0] === elem.firstChild;
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild;
```

- for..of can be used to iterate over it

```
for (let node of document.body.childNodes) {
   alert(node); // shows all nodes from the collection
}
```

DOM node properties and methods



children, childNodes, firstChild and lastChild

- For all nodes:
  - parentNode
  - childNodes
  - firstChild
  - lastChild
  - previousSibling
  - nextSibling
- For elements nodes:
  - parentElement
  - children
  - firstElementChild
  - lastElementChild
  - previousElementSibling
  - nextElementSibling

Siblings and parent

- Siblings are nodes that are children of the same parent.

```
// parent of <body> is <html>
alert( document.body.parentNode === document.documentElement ); // true

// after <head> goes <body>
alert( document.head.nextSibling ); // HTMLBodyElement

// before <body> goes <head>
alert( document.body.previousSibling ); // HTMLHeadElement
```

Searching: getElementBy*, querySelector*

- `document.getElementById(id) or id`
  - it returns the element with the id attribute
  - the id must be unique
- `document.getElementsByTagName(tag)`
  - looks for elements with the given tag and returns the collection of them
- `document.getElementsByClassName(css)`
  - returns elements that have the given CSS class
- `elem.querySelectorAll(css)`
  - returns all elements inside elem matching the given CSS selector
- `elem.querySelector(css)`
  - returns the first element for the given CSS selector

Searching: getElementBy*, querySelector*

- The most used are querySelector and querySelectorAll
- getElement(s)By* can be sporadically helpful or found in the **old** scripts

Method	Searches by...	Can call on an element?	Live?
`querySelector`	CSS-selector	✓	-
`querySelectorAll`	CSS-selector	✓	-
`getElementById`	`id`	-	-
`getElementsByName`	`name`	-	✓
`getElementsByTagName`	tag or `'*'`	✓	✓
`getElementsByClassName`	class	✓	✓

innerHTML

- The innerHTML property allows to get the HTML inside the element as a string and modify it
- The innerHTML property is only valid for **element nodes**

```
<p>A paragraph</p>

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

<script>

    alert(document.getElementById("container").innerHTML); // A div

    document.getElementById("container").innerHTML = 'The new BODY!<br>Changed!';

                                                    // The new BODY!<br>Changed!

</script>
```

textContent

- The textContent provides access to the text inside the **element**: only text, minus all <tags>

```
<div id="news">
    <h1>Headline!</h1>
    <p>Martians attack people!</p>
</div>
<script>
    // Headline! Martians attack people!
    alert(news.textContent);
</script>
```

nodeValue and data

- The nodeValue property allows to get the string inside **text or comment nodes** as a string and modify it

```
<div id="elem">This foobar</div>

<script>
      alert( document.getElementById("elem").firstChild.nodeValue);
                                                    // This foobar

       alert( document.getElementById("elem").firstChild.data);
                                                    // This foobar

</script>
```

HTML attributes

- Attributes are accessible by using the following methods:
  - elem.hasAttribute(name) – checks for existence
  - elem.getAttribute(name) – gets the value
  - elem.setAttribute(name, value) – sets the value
  - elem.removeAttribute(name) – removes the attribute

- Their name is case-insensitive
- Their values are always strings

HTML attributes - DOM objects

	Properties	Attributes
Type	Any value, standard properties have types described in the spec	A string
Name	Name is case-sensitive	Name is not case-sensitive

data-* attributes, dataset

- There exist data-* attributes
- They are non-standard attributes used to pass custom data from HTML to JavaScript or to “mark” HTML-elements for JavaScript
- All attributes starting with “data-” are reserved for programmers’ use.
- They are available in the dataset property.
- Multiword attributes like data-order-state become camel-cased: dataset.orderState.

How to Get and Use HTML5 Data Attributes in JavaScript

Creating an element

- `document.createElement(tag);`

  creates an element with the given tag

- `document.createTextNode(value);`

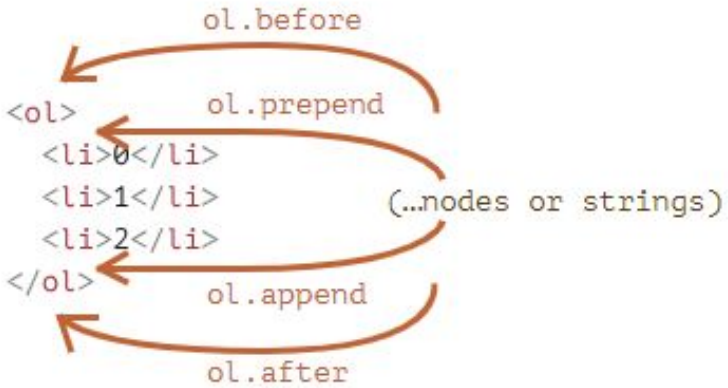
  creates a text node (rarely used)

- `elem.cloneNode(deep);`

  clones the element, if deep==true then with all descendants.

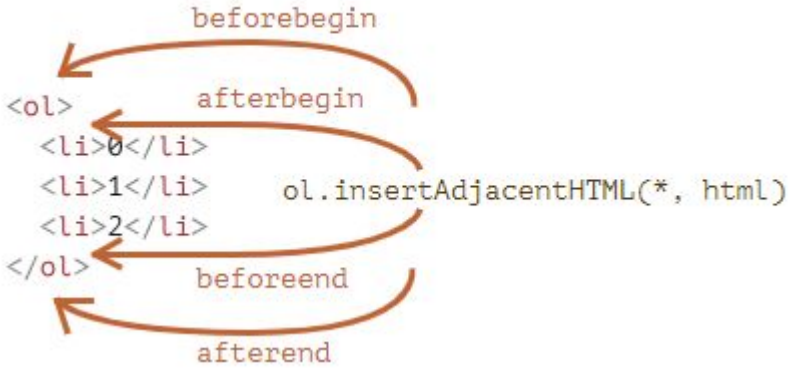
Insertion methods

- `node.append(...nodes or strings)`
  - insert into node, at the end
- `node.prepend(...nodes or strings)`
  - insert into node, at the beginning
- `node.before(...nodes or strings)`
  - insert right before node
- `node.after(...nodes or strings)`
  - insert right after node
- `node.replaceWith(...nodes or strings)`
  - replace node
- `node.remove()`
  - remove the node



Insertion methods

- `node.insertAdjacentHTML(where, html)`
  - insert into node, at the end
- Where to insert relative to elem
  - `beforebegin`
    insert `html` immediately before `elem`
  - `afterbegin`
    insert `html` into `elem`, at the beginning
  - `beforeend`
    insert `html` into `elem`, at the end
  - `afterend`
    insert `html` immediately after `elem`



table

- Certain types of DOM elements may provide additional properties
- The <table> element supports specific properties to their type, i.e:
  - `table.rows`
  - `table.tBodies`
  - `tr.cells`
  - `td.cellIndex`
  - …

hidden

- The “hidden” attribute and the DOM property specifies whether the element is visible or not

```
<div hidden>With the attribute "hidden"</div>

<div id="elem">JavaScript assigned the property "hidden"</div>

<script>

    elem.hidden = true;

</script>
```

Styles and classes

- There are generally two ways to style an element:
  - Create a class in CSS and add it:
  - Write properties directly into style:
- CSS is more flexible and easier to support

Styles and classes

- Classes can be operated using **className** or using **classList**:
  - `elem.classList.add("class");`

    adds the class
  - `elem.classList.remove("class");`

    adds/removes the class.
  - `elem.classList.toggle("class");`

    adds the class if it doesn't exist, otherwise removes it
  - `elem.classList.contains("class");`

    checks for the given class, returns true/false.