- DOM (Document Object Model)
- ▼ DOM Selector
  - ▼ Read
    - 1. document
    - 2. document.getElementById('id')
    - 3. document.getElementsByTagName('element')
    - 4. document.getElementsByClassName('class')
    - 5. document.getElementsByName('name')
    - 6. document.querySelector('query')
    - 7. document.querySelectorAll()
- ▼ Type of Nodes & Node Value
  - 1. Element Node - 1
  - 2. Attribute Node ** - 2
  - 3. Text Node - 3
  - 4. Comment - 8
  - 5. Document - 9
  - 6. DocumentFragment - 11
- ▼ Node Properties
  - 1. nodeName
  - 2. nodeValue
  - 3. nodeType
  - 4. attributes
- ▼ Node Method
  - 1. Node.hasChildNodes()
- ▼ Node Tree Traversal
  - 1. parentNode

- 2. childNodes
- 3. firstChild
- 4. lastChild
- 5. nextSibling
- 6. previousSibling

▼ DOM Tree Traversal
- 1. parentElement
- 2. children**
- 3. Node.childElementCount === Node.children.length
- 4. firstElementChild
- 5. lastElementChild
- 6. nextElementSibling
- 7. previousElementSibling

▼ Inheritance
- Prototype Chain of DOM
- Basic Operation

▼ DOM Manipulation
- ▼ Add / Create
  - 1. document.createElement(); ***
  - 2. document.createTextNode();
  - 3. document.createComment();
  - 4. document.createDocumentFragment();
- ▼ Insert
  - 1. parentNode.appendChild() ***
  - 2. parentNode.insertBefore(a, b) ***
- ▼ Delete
  - 1. parentNode.removeChild()

    - 2. child.remove() - ES5 ***
- ▼ Replace
    - 1. parentNode.replaceChild(new, origin)
- ▼ Element Node Properties
    - 1. innerHTML ***
    - 2. innerText
    - 3. textContent
- ▼ Element Node Method
    - 1. ele.setAttribute('prop', 'value')
    - 2. ele.getAttribute('prop')
    - 3. ele.className

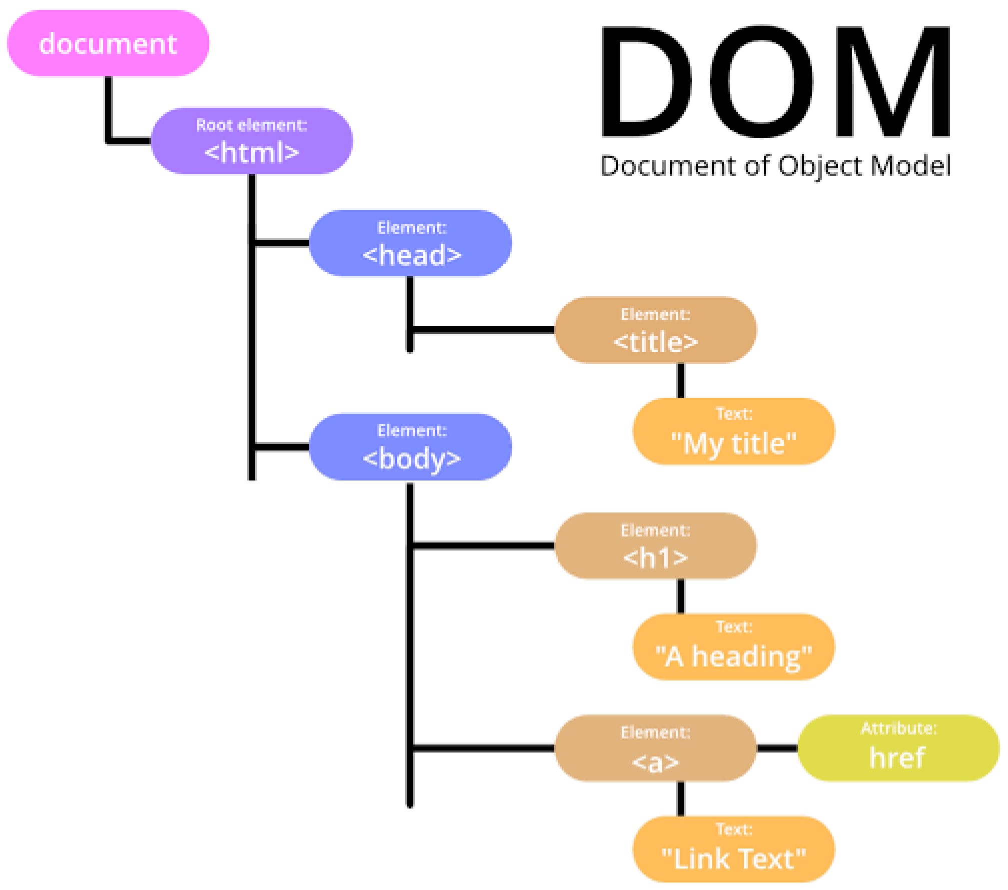
DOM (Document Object Model)

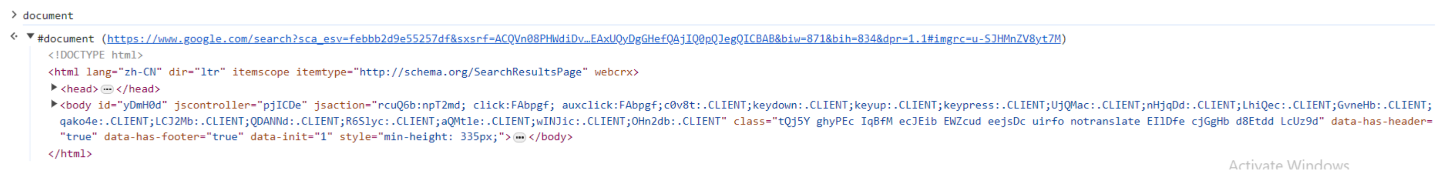
The data representation of the objects that comprise the structure and content of a document on the web.

DOM is a programming interface for web documents, allowing us to manipulate HTML and XML with various built-in DOM methods.

Document object represents the entire document.

HTML element is only the root element, which is the most important part of the document, but not the document itself.





```
> document
< ▼#document (https://www.google.com/search?sca_esv=febbb2d9e55257df&sxsrf=ACQVn08PHWdiDv…EAxUQyDgGHefQAjIQ0pQJegQICBAB&biw=871&bih=834&dpr=1.1#imgrc=u-SJHMnZV8yt7M)
    <!DOCTYPE html>
    <html lang="zh-CN" dir="ltr" itemscope itemtype="http://schema.org/SearchResultsPage" webcrx>
     ▶<head> ⋯ </head>
     ▶<body id="yDmH0d" jscontroller="pjICDe" jsaction="rcuQ6b:npT2md; click:FAbpgf; auxclick:FAbpgf;c0v8t:.CLIENT;keydown:.CLIENT;keyup:.CLIENT;keypress:.CLIENT;UjQMac:.CLIENT;nHjqDd:.CLIENT;LhiQec:.CLIENT;GvneHb:.CLIENT;
      qako4e:.CLIENT;LCJ2Mb:.CLIENT;QDANNd:.CLIENT;R6Slyc:.CLIENT;aQMtle:.CLIENT;wINJic:.CLIENT;OHn2db:.CLIENT" class="tQj5Y ghyPEc IqBfM ecJEib EWZcud eejsDc uirfo notranslate EIlDfe cjGgHb d8Etdd LcUz9d" data-has-header=
      "true" data-has-footer="true" data-init="1" style="min-height: 335px;"> ⋯ </body>
    </html>
```

Activate Windows

```
<!---->
<document>
  <html>
  </html>
</document>
```

Note: XML has been replaced by JSON, no longer used.

DOM cannot directly manipulate CSS, but we can use it to indirectly add inline styles to HTML elments.

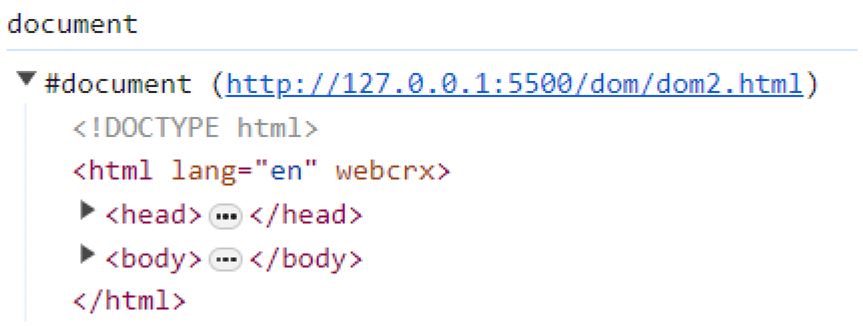
```
var div = document.getElementsByTagName('div')[0];
div.style.width = '100px';
div.style.height = '100px';
div.style.backgroundColor = 'red';
```

Any collection generated from DOM is an array-like object, thus not able to use array method.

DOM Selector

Read

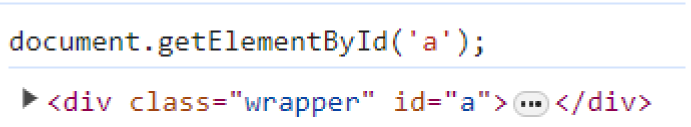
1. document



```
document
▼ #document (http://127.0.0.1:5500/dom/dom2.html)
    <!DOCTYPE html>
    <html lang="en" webcrx>
    ▶ <head>…</head>
    ▶ <body>…</body>
    </html>
```

2. document.getElementById('id')

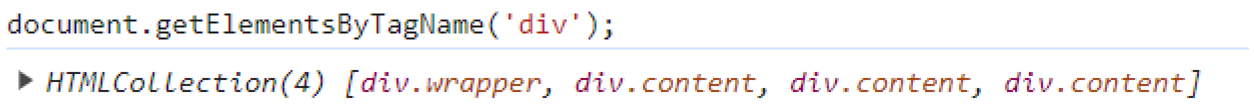
1. returns an Element object representing the element whose id property matches the specified string.
   only one element is returned with given id.
2. IE ver.8 and below, id is case-insensitive.
3. IE ver.8 and below, select an element with name attribute having matching string is possible, if id attribute is not present.
4. do not rely on id selector too much as connect to backend program might change it.
5. use only when necessary, e.g., each section has an id.



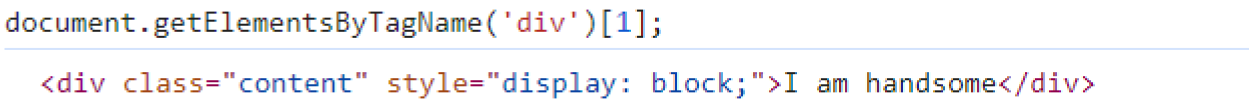
```
document.getElementById('a');
▶ <div class="wrapper" id="a">…</div>
```

3. document.getElementsByTagName('element')

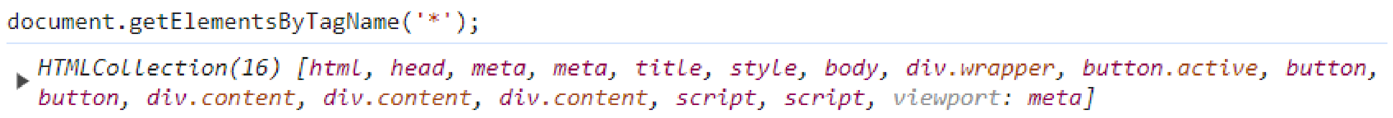
1. returns a live *HTMLCollection - an array-like object* of elements with the given tag name.
2. select specific element based on index.
3. select all elements with asterisk(*)
4. very good compatibility in every browser - mainstream



```
document.getElementsByTagName('div');
▶ HTMLCollection(4) [div.wrapper, div.content, div.content, div.content]
```



```
document.getElementsByTagName('div')[1];
  <div class="content" style="display: block;">I am handsome</div>
```



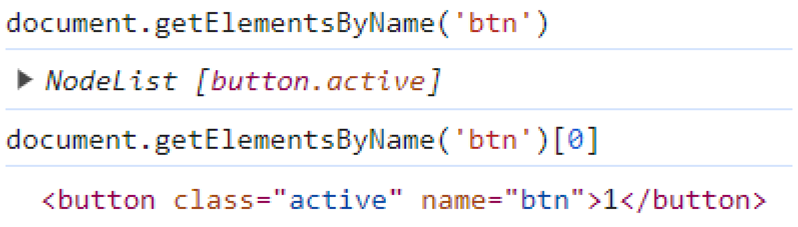
```
document.getElementsByTagName('*');
▶ HTMLCollection(16) [html, head, meta, meta, title, style, body, div.wrapper, button.active, button,
  button, div.content, div.content, div.content, script, script, viewport: meta]
```

4. document.getElementsByClassName('class')

1. returns an array-like object of all child elements which have all of the given class name(s).
2. IE ver.8 and below not support.
3.

5. document.getElementsByName('name')

1. returns a NodeList Collection of elements with a given name attribute in the document.
2. only specific elements works, e.g., form, form elements: e.g., input, img, iframe where name attribute has meaning.
3. in previous browser ver, elements like div did not support selection by name, but in current browser, it is already supported.

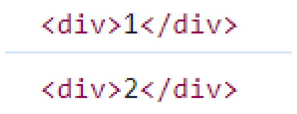


```
document.getElementsByName('btn')
▶ NodeList [button.active]
document.getElementsByName('btn')[0]
  <button class="active" name="btn">1</button>
```

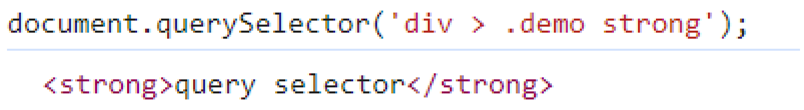
6. document.querySelector('query')

1. returns the first Element within the document that matches the specified selector, or group of selectors. If no matches are found, null is returned.
2. query is similar to how we write CSS selector.
3. IE ver.7 and below not support.
4. manipulate element (styles) is allowed.
5. weakness: not live (a copy): changing (e.g., deleting) doesn't reflect on nodelist, but the element on display is changed.

```
<body>
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <script>
    var div1 = document.querySelector('div');
    div1.remove();
    console.log(div1); // changes doesn't reflect.
    console.log(document.getElementsByTagName('div')[0]); // changes does reflect. index 0 is n
  </script>
</body>
```



```
<div>1</div>
<div>2</div>
```

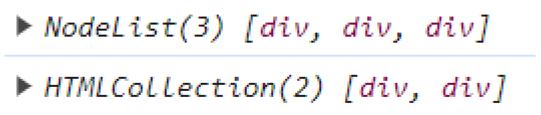


```
document.querySelector('div > .demo strong');
  <strong>query selector</strong>
```

7. document.querySelectorAll()

1. returns a static (not live) NodeList representing a list of the document's elements that match the specified group of selectors.
2. IE ver.7 and below not support.
3. manipulate element (style) is allowed.
4. weakness: not live (a copy): changing (e.g., deleting) doesn't reflect on nodelist, but the element on display is changed.

```
<body>
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <script>
    var static = document.querySelectorAll('div');
    var live = document.getElementsByTagName('div');
    static[0].remove(); // remove the first div
    console.log(static); // the nodelist is not updated.
    console.log(live); // live HTMLcollection reflect the change accordingly
  </script>
</body>
```



```
▶ NodeList(3) [div, div, div]
▶ HTMLCollection(2) [div, div]
```

Type of Nodes & Node Value

1. Element Node - 1

2. Attribute Node ** - 2

Attr objects inherit the Node interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree.

3. Text Node - 3

4. Comment - 8

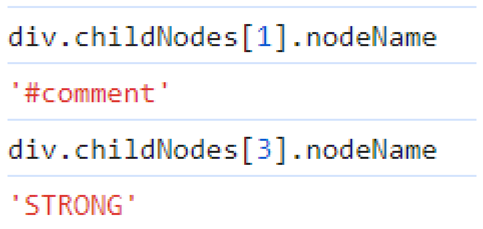
5. Document - 9

6. DocumentFragment - 11

Node Properties

1. nodeName

- returns the name of the current node as a string.
- changing nodeName is not allowed, it is read-only.



```
div.childNodes[1].nodeName
'#comment'
div.childNodes[3].nodeName
'STRONG'
```

2. nodeValue

- returns or sets the value of the current node.
- text, comment, CDATA nodes, return the content.
- else, return null.
- nodes with null nodeValues (e.g., DOM elements) are not modifiable.

Node	Value of nodeValue
`CDATASection`	Content of the CDATA section
`Comment`	Content of the comment
`Document`	`null`
`DocumentFragment`	`null`
`DocumentType`	`null`
`Element`	`null`
`NamedNodeMap`	`null`
`ProcessingInstruction`	Entire content excluding the target
`Text`	Content of the text node

Note: When `nodeValue` is defined to be `null`, setting it has no effect.

3. nodeType

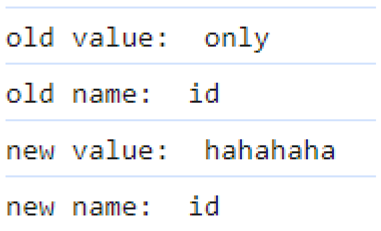
- return integers representing node type to distinguishes different kind of nodes from each other.

```
var div = document.getElementsByTagName('div')[0];

function retElementChild(node) {
  var nodeList = node.childNodes;
  var len = nodeList.length; // save efficiency -method chaining every time is not efficient.
  var temp = {
    length: 0,
    push: Array.prototype.push,
    splice: Array.prototype.splice,
  };
  for (let i = 0; i < len; i ++) {
    if (nodeList[i].nodeType === 1) {
      temp.push(nodeList[i]);
    }

  }
  return Array.from(temp);
}
console.log(retElementChild(div));
```

4. attributes

- returns a live collection of all attribute nodes registered to the specified node.
- access attribute name by attributes[i].name: we cannot change attribute name.
- access attribute value by attributes[i].value: we can change value.

```
div.attributes;
div.attributes[i];
div.attributes[i].value;
div.attributes[i].name;
div.attributes[i].value = 123; // we can change the value
div.attributes[i].name = 123; // changing name is not allowed

// Example
console.log('old value: ', div.attributes[0].value);
console.log('old name: ', div.attributes[0].name);
div.attributes[0].value = 'hahahaha';
div.attributes[0].name = 'hahahaha';
console.log('new value: ', div.attributes[0].value);
console.log('new name: ', div.attributes[0].name);
```



```
old value:  only
old name:  id
new value:  hahahaha
new name:  id
```

Node Method

1. Node.hasChildNodes()

- returns a boolean value indicating whether the given Node has child nodes or not.

```
<!--Example 1-->
<div></div>

<!--Example 2-->
<div> </div>

<!--Example 3-->
<div>
  <!--comment-->
</div>
<script>
// Example 1
console.log(div.hasChildNodes()); // false

// Example 2
console.log(div.hasChildNodes()); // true - when there is a space between opening and closing ta

// Example 3
console.log(div.hasChildNodes()); // true
</script>
```

Node Tree Traversal

based on relationship of elements.

compatible for all browsers.

1. parentNode

- returns the parent of the specified node in the DOM tree.
- #document is the top of the node and can't have parentNode (null)

```
<body>
  <div>
    <strong></strong>
    <span></span>
    <em></em>
  </div>
  <script>
    var strong = document.getElementsByTagName('strong')[0];
    console.log(strong.parentNode); // <div>...</div>
    console.log(strong.parentNode.parentNode); // <body>...</body>
    console.log(strong.parentNode.parentNode.parentNode.parentNode); // #document (the top node
    console.log(strong.parentNode.parentNode.parentNode.parentNode.parentNode); // null
  </script>
</body>
```

2. childNodes

- returns a live NodeList of child nodes of the given element where the first child node is assigned index 0.

```
<body>
  <div>
    <!--This is a comments-->
    <strong>
      <span>
        1
      </span>
    </strong>
    <span></span>
    <em></em>
  </div>
  <script>
    var div = document.getElementsByTagName('div')[0];
    console.log(div.childNodes); // NodeList(9) [text, comment, text, strong, text, span, text,
    console.log(div.childNodes.length); // 9 - includes text nodes - empty spaces. Note: remove
  </script>
</body>
```

3. firstChild

- returns the node's first child in the tree, or null if the node has no children.

4. lastChild

- returns the last child of the node, or null if there are no child nodes.

5. nextSibling

- returns the node immediately following the specified one in their parent's childNodes, or returns null if the specified node is the last child in the parent element.

6. previousSibling

- returns the node immediately preceding the specified one in its parent's childNodes list, or null if the specified node is the first in that list.

DOM Tree Traversal

Note:

1. Convenient but low compatibility
2. Only `children` is supported in all browser.

1. parentElement

- returns the DOM node's parent Element, or null if the node either has no parent, or its parent isn't a DOM Element.
- IE ver. 9 and below not support.

2. children**

- similar to Node.childNodes, but includes only element nodes.

3. Node.childElementCount === Node.children.length

- returns the number of child elements of this element.
- IE ver. 9 and below not support.

4. firstElementChild

- returns an element's first child, or null if there are no child elements.
- IE ver. 9 and below not support.

5. lastElementChild

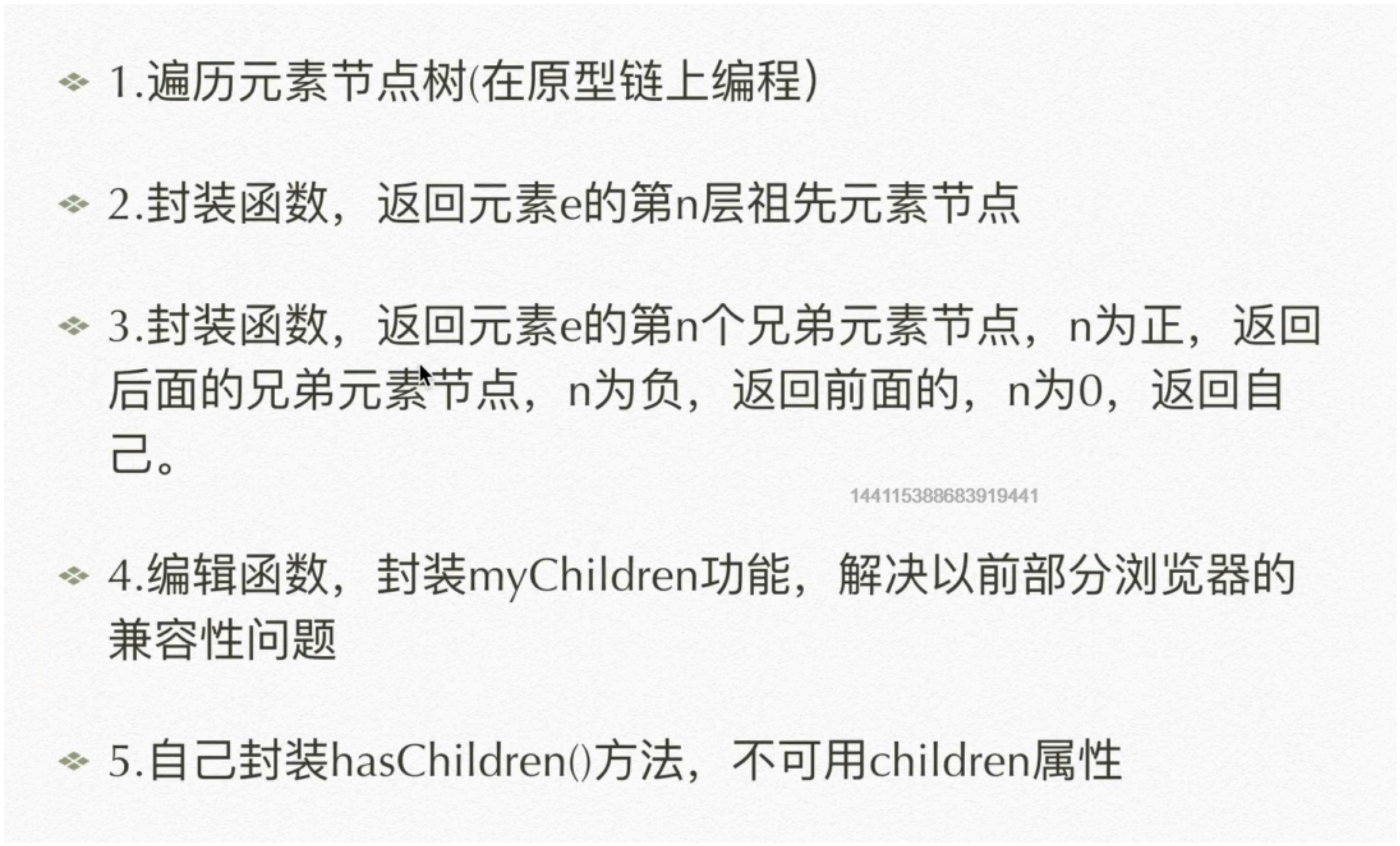
- returns an element's last child Element, or null if there are no child elements.
- IE ver. 9 and below not support.

6. nextElementSibling

- returns the element immediately following the specified one in its parent's children list, or null if the specified element is the last one in the list.

7. previousElementSibling

- returns the Element immediately prior to the specified one in its parent's children list, or null if the specified element is the first one in the list.
- IE ver. 9 and below not support.

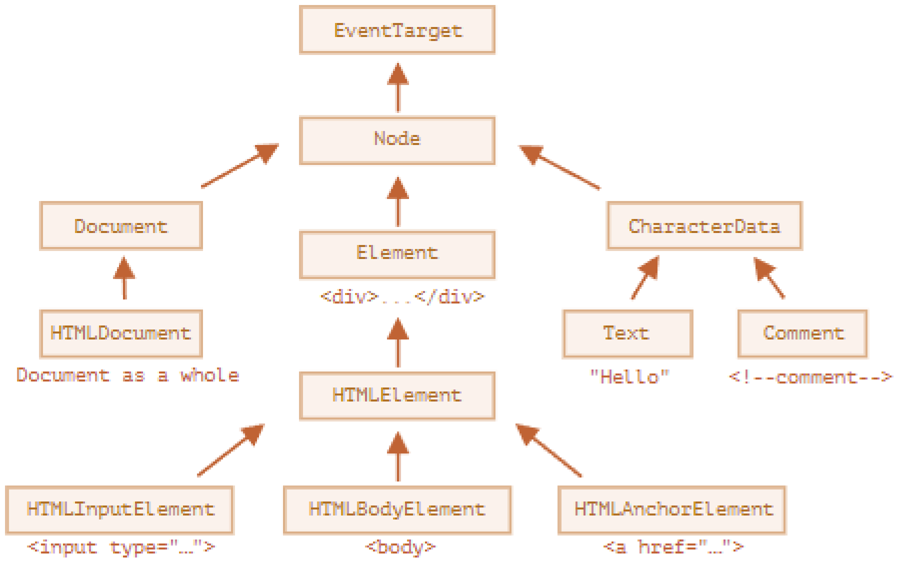


- 1.遍历元素节点树(在原型链上编程)
- 2.封装函数，返回元素e的第n层祖先元素节点
- 3.封装函数，返回元素e的第n个兄弟元素节点，n为正，返回后面的兄弟元素节点，n为负，返回前面的，n为0，返回自己。

1441153886839194441

- 4.编辑函数，封装myChildren功能，解决以前部分浏览器的兼容性问题
- 5.自己封装hasChildren()方法，不可用children属性

Inheritance



Document Object Model (DOM API) is represented as a hierarchical tree-like structure in web browser, each individual parts represents as nodes.

Each nodes could correspond to the specific constructor functions and relate to each other via prototype chain.

```
<script>
  HTMLBodyElement.prototype.abc = 'abc';
  var body = document.getElementsByTagName('body')[0];
  var head = document.getElementsByTagName('head')[0];
</script>
```

```
body.abc
'abc'
head.abc
undefined
```

Prototype Chain of DOM

1. Document and document is not same.

- Document: constructor function, its prototype is inherited by HTMLDocument.prototype, and eventually document.
- document object is created by HTMLDocument.
- `HTMLDocument.prototype.__proto__ == Document.prototype` is true.

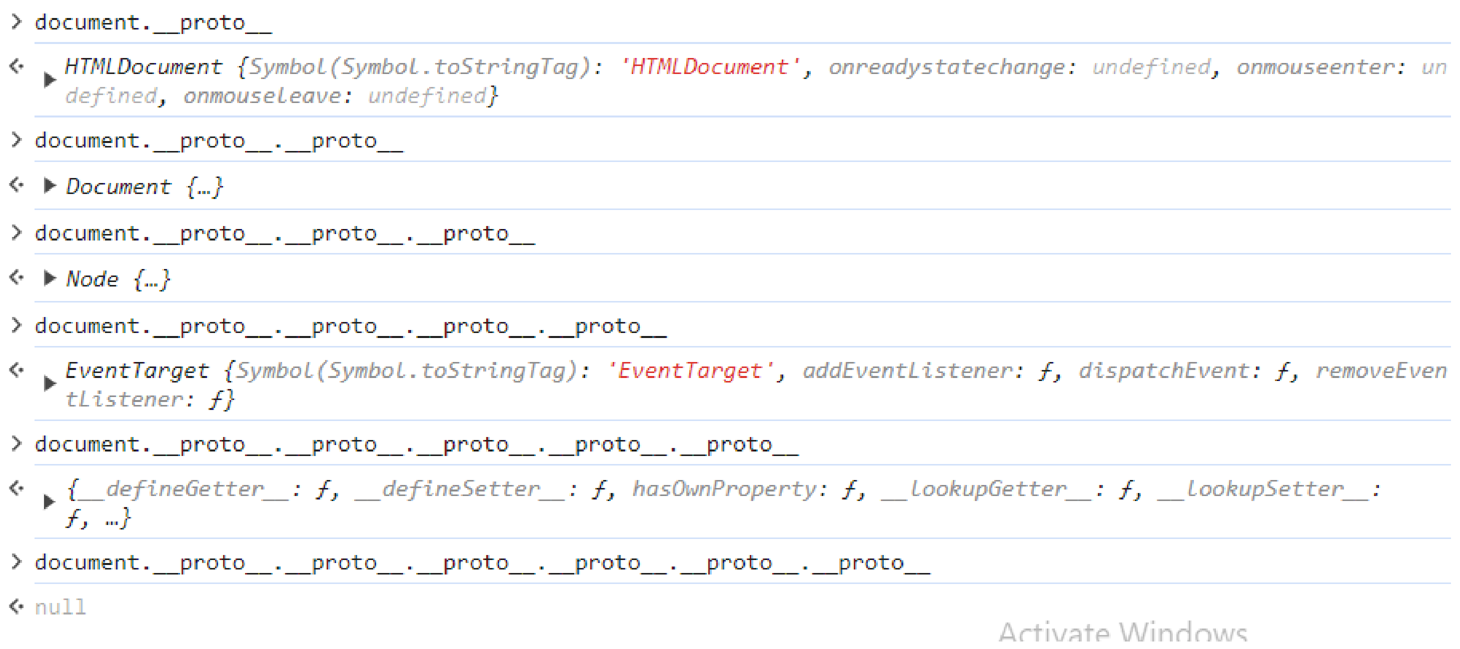
```
HTMLDocument.prototype = {
  __proto__: Document.prototype
}
```

- `HTMLDocument.prototype === document.__proto__` is true.

2. thus, the inheritance is as follow:
   `document (instance) --> HTMLDocument.prototype --> Document.prototype`
3. demonstrate the prototype chain of DOM, in which Object.prototype is the end of chain before null.

- thus, DOM elements can access properties and methods from Object.prototype.

```
document.body.toString(); // '[object HTMLBodyElement]'
```



```
> document.__proto__
<· ▶ HTMLDocument {Symbol(Symbol.toStringTag): 'HTMLDocument', onreadystatechange: undefined, onmouseenter: undefined, onmouseleave: undefined}
> document.__proto__.__proto__
<· ▶ Document {…}
> document.__proto__.__proto__.__proto__
<· ▶ Node {…}
> document.__proto__.__proto__.__proto__.__proto__
<· ▶ EventTarget {Symbol(Symbol.toStringTag): 'EventTarget', addEventListener: ƒ, dispatchEvent: ƒ, removeEventListener: ƒ}
> document.__proto__.__proto__.__proto__.__proto__.__proto__
<· ▶ {__defineGetter__: ƒ, __defineSetter__: ƒ, hasOwnProperty: ƒ, __lookupGetter__: ƒ, __lookupSetter__: ƒ, …}
> document.__proto__.__proto__.__proto__.__proto__.__proto__.__proto__
<· null
```

Activate Windows

Basic Operation

1. `getElementById` is defined on Document.prototype.

- HTMLDocument & XMLDocument can use.
- Element nodes are inaccessible.

2. `getElementsByName` is defined on HTMLDocument.prototype.

- Element nodes are inaccessible.

3. `getElementByTagName` is defined on both Document.prototype and Element.prototype.

- thus, both html document and element are accessible.

```
var div = document.getElementsByTagName('div')[0]; // select from document
var span = div.getElementsByTagName('span')[0]; // select from div element
```

4. HTMLDocument.prototype has defined some commonly-used properties.

- we don't need to select them again.
- body: ...
- head: ...

```
// predefined properties that ease our life.
document.body; // <body>...</body>
document.head; // <head>...</head>
```

5. Document.prototype has defined `documentElement` property, representing root element `<html>` of the document.

```
document.documentElement; // <html>...</html>
```

6. `getElementsByClassName` , `querySelectorAll` , `querySelector` are defined on both Document.prototype and Element.prototype.

DOM Manipulation

Add / Create

create nodes by Js, only effective when we insert into HTML.

1. document.createElement(); ***

```
var div = document.createElement('div'); // create element node
document.body.appendChild(div);
```

2. document.createTextNode();

```
var text = document.createTextNode('text'); // create text node
```

3. document.createComment();

```
var comment = document.createComment('This is a comment'); // create comment node
```

4. document.createDocumentFragment();

Insert

1. parentNode.appendChild() ***

every element have this method.

similar to `array.push` , adding node to the end of the list of children of a specific parent node.

if the element doesn't exist, it insert the element into another element.

if the element already exist, appendChild cut and move it to the element we append to.

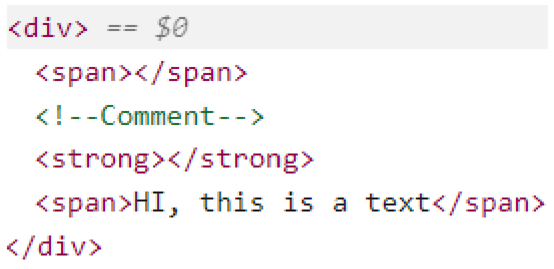
```
var divElem = document.getElementsByTagName('div')[0];
var span = document.createElement('span');
var text = document.createTextNode('HI, this is a text');
var comment = document.createComment('This is a comment');

divElem.appendChild(comment);
divElem.appendChild(span);
divElem.appendChild(text);
```

2. parentNode.insertBefore(a, b) ***

inserts a node before a reference node as a child of a specified parent node.

insert a before b.

```
var divElem = document.getElementsByTagName('div')[0];
var strong = document.createElement('strong');
div.insertBefore(strong, span);
var span = document.createElement('span');
var text = document.createTextNode('HI, this is a text');
span.appendChild(text);
divElem.appendChild(span);
divElem.insertBefore(strong, span);
```



```
<div> == $0
  <span></span>
  <!--Comment-->
  <strong></strong>
  <span>HI, this is a text</span>
</div>
```

Delete

1. parentNode.removeChild()

removes a child node from the DOM and returns the removed node.
the removed child is still exist in memory, and can be reused later.

```
divElem.removeChild(span);

// removeChild return removed part, which we can store for later use.
var a = divElem.removeChild(span);
```

2. child.remove() - ES5 ***

removes the element from the DOM.
literally remove and no return anything.
when we no longer need an element

```
divElem.remove();
```

Replace

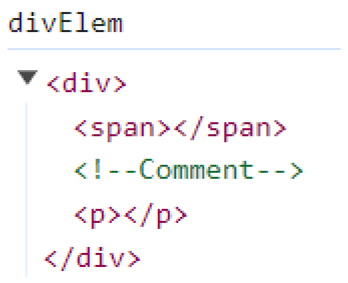
1. parentNode.replaceChild(new, origin)

replaces a child node within the given (parent) node.

the replaced part still exist, store it in a variable for future use.

```
var p = document.createElement('p');
var replaced = divElem.replaceChild(p, strong);
```

```
<div>
  <span></span>
  <!--Comment-->
  <strong></strong>
</div>
```



```
divElem
▼<div>
   <span></span>
   <!--Comment-->
   <p></p>
 </div>
```

Element Node Properties

1. innerHTML ***

- gets or sets the HTML or XML markup contained within the element.
- use to set html element, and adding styles.

```
var div = document.createElement('div');
document.body.appendChild(div);
console.log(div.innerHTML); // ''
div.innerHTML = '123';
div.innerHTML += '456';
div.innerHTML = '<span style="background-color: green;">123</span>'
```

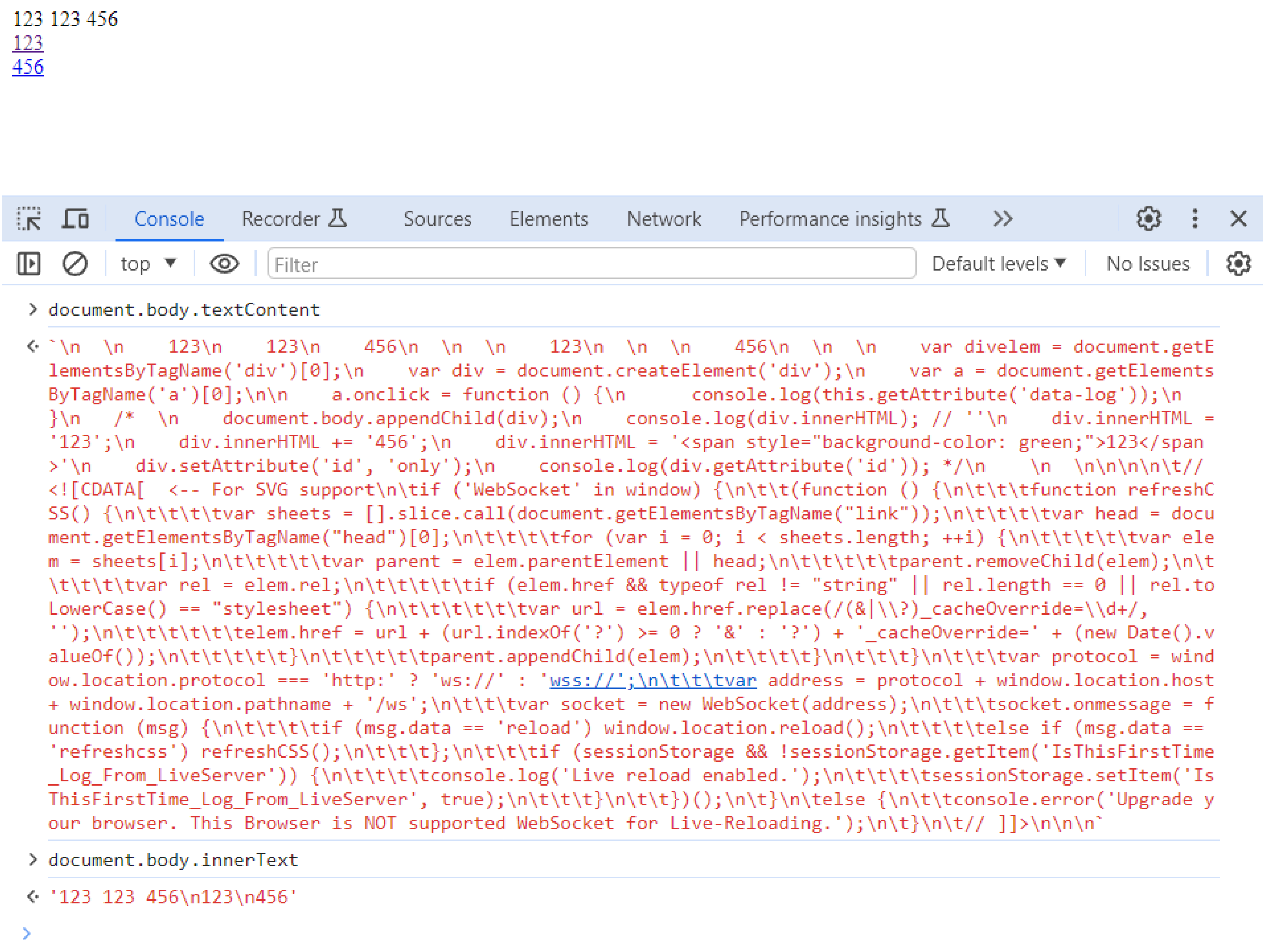
2. innerText

- gets or sets inner text - give you text node.
- old Firefox not support.
- cause reflow - computational expensive***
- setting innerText might remove all of the node's children and replace with a single text node based on given string value.

3. textContent

- get the content of all elements, including `<script>` and `<style>` elements.
- previous version IE not support.
- setting innerText might remove all of the node's children and replace with a single text node based on given string value.

```
<!--Difference between innerText and textContent-->
<div>
    123
    <span>123</span>
    456
  </div>

  <script>
    // be careful replace text using innerText and textContent
    div.innerText = 123; // it replaces all the content in div.

  </script>
```



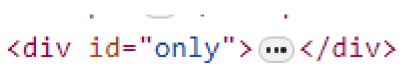
```
123 123 456
123
456
```

```
> document.body.textContent
<· `\n  \n    123\n    123\n    456\n  \n  \n    123\n  \n  \n    456\n  \n  \n    var divelem = document.getElementsByTagName('div')[0];\n    var div = document.createElement('div');\n    var a = document.getElementsByTagName('a')[0];\n\n    a.onclick = function () {\n      console.log(this.getAttribute('data-log'));\n    }\n   /*  \n    document.body.appendChild(div);\n    console.log(div.innerHTML); // ''\n    div.innerHTML = '123';\n    div.innerHTML += '456';\n    div.innerHTML = '<span style="background-color: green;">123</span>'\n    div.setAttribute('id', 'only');\n    console.log(div.getAttribute('id')); */\n    \n  \n\n\n\n\t// <![CDATA[  <-- For SVG support\n\tif ('WebSocket' in window) {\n\t\t(function () {\n\t\t\tfunction refreshCSS() {\n\t\t\t\tvar sheets = [].slice.call(document.getElementsByTagName("link"));\n\t\t\t\tvar head = document.getElementsByTagName("head")[0];\n\t\t\t\tfor (var i = 0; i < sheets.length; ++i) {\n\t\t\t\t\tvar elem = sheets[i];\n\t\t\t\t\tvar parent = elem.parentElement || head;\n\t\t\t\t\tparent.removeChild(elem);\n\t\t\t\t\tvar rel = elem.rel;\n\t\t\t\t\tif (elem.href && typeof rel != "string" || rel.length == 0 || rel.toLowerCase() == "stylesheet") {\n\t\t\t\t\t\tvar url = elem.href.replace(/(&|\\?)_cacheOverride=\\d+/, '');\n\t\t\t\t\t\telem.href = url + (url.indexOf('?') >= 0 ? '&' : '?') + '_cacheOverride=' + (new Date().valueOf());\n\t\t\t\t\t}\n\t\t\t\t\tparent.appendChild(elem);\n\t\t\t\t}\n\t\t\t}\n\t\t\tvar protocol = window.location.protocol === 'http:' ? 'ws://' : 'wss://';\n\t\t\tvar address = protocol + window.location.host + window.location.pathname + '/ws';\n\t\t\tvar socket = new WebSocket(address);\n\t\t\tsocket.onmessage = function (msg) {\n\t\t\t\tif (msg.data == 'reload') window.location.reload();\n\t\t\t\telse if (msg.data == 'refreshcss') refreshCSS();\n\t\t\t};\n\t\t\tif (sessionStorage && !sessionStorage.getItem('IsThisFirstTime_Log_From_LiveServer')) {\n\t\t\t\tconsole.log('Live reload enabled.');\n\t\t\t\tsessionStorage.setItem('IsThisFirstTime_Log_From_LiveServer', true);\n\t\t\t}\n\t\t})();\n\t}\n\telse {\n\t\tconsole.error('Upgrade your browser. This Browser is NOT supported WebSocket for Live-Reloading.');\n\t}\n\t// ]]>\n\n\n`
> document.body.innerText
<· '123 123 456\n123\n456'
>
```

Element Node Method

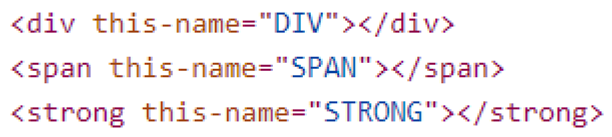
1. ele.setAttribute('prop', 'value')

```
div.setAttribute('id', 'only');
```



```
<div id="only">…</div>
```

```
<!--Example Setting Attr-->
<div></div>
<i></i>
<strong></strong>

<script>
  var all = document.getElementsByTagName('*');
  for (var i = 0; i < all.length; i ++) {
    all[i].setAttribute('this-name', all[i].nodeName);
  }
</script>
```

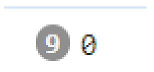


```
<div this-name="DIV"></div>
<span this-name="SPAN"></span>
<strong this-name="STRONG"></strong>
```

2. ele.getAttribute('prop')

returns the value of a specified attribute on the element.

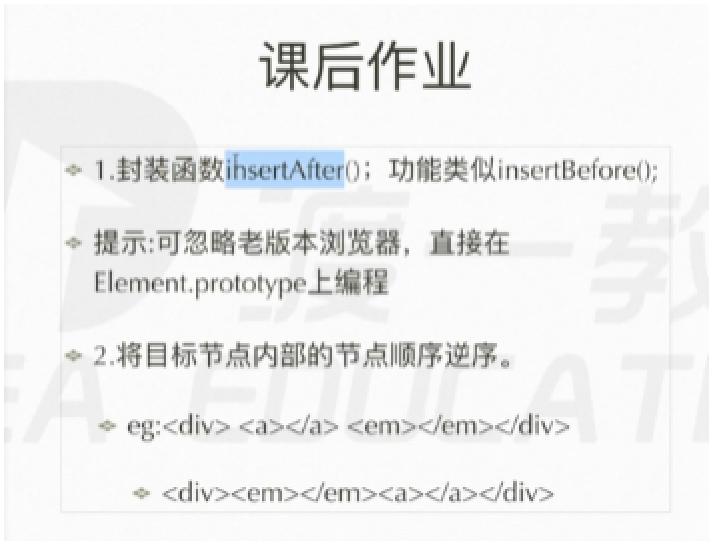
```
div.getAttribute('id'); // only

var attr = div.getAttribute('id');

// Example 2 - work with data-attribute
var div = document.createElement('div');
var a = document.getElementsByTagName('a')[0];

a.onclick = function () {
  console.log(this.getAttribute('data-log'));
}
```



```
9 0
```

3. ele.className

gets and sets the value of the class attribute of the specified element.



课后作业

- 1.封装函数ihsertAfter()；功能类似insertBefore();
- 提示:可忽略老版本浏览器，直接在Element.prototype上编程
- 2.将目标节点内部的节点顺序逆序。
  - eg:<div> <a></a> <em></em></div>
    - <div><em></em><a></a></div>