What is DOM

Sunday, January 29, 2017 12:22 AM

From lynda-javascript-enhancing-the-dom by Ray Villalobos

Enhancing the DOM

What is the DOM?

Document Object Model

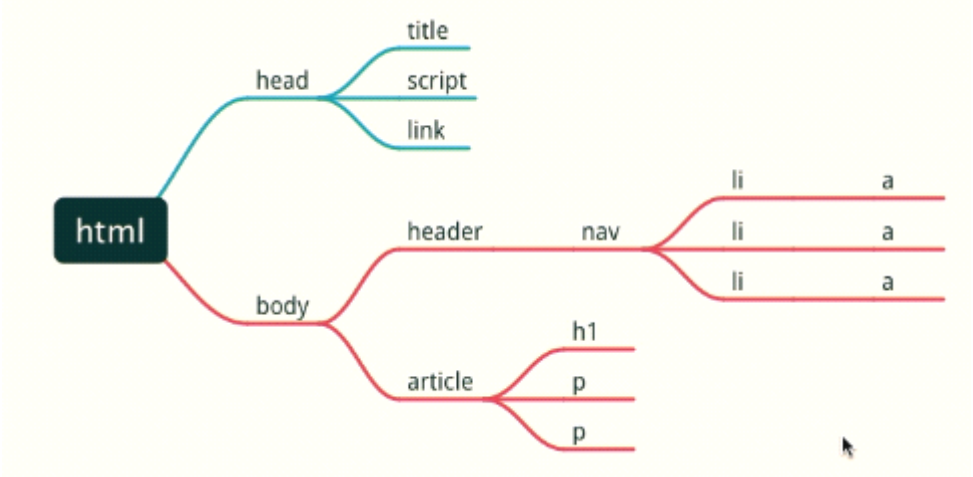
Describe relationships in HTML

- DOM is used to describe the structure of HTML document and relation btw them.
- There is two types of relation btw elements
  - Siblings
  - Parent-child

Browsers interpret and organize HTML as a DOM

API for CSS and JavaScript

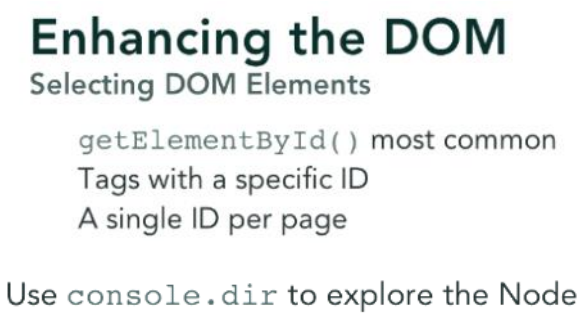
- Languages like CSS and JS uses DOM to target and modify an HTML element.



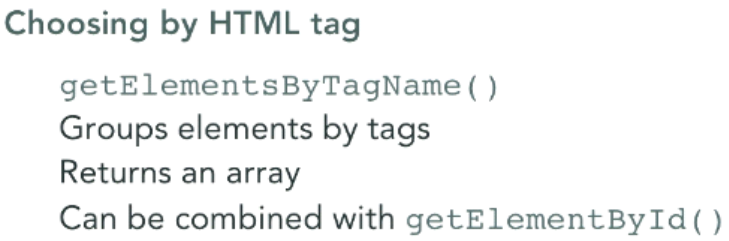
FOR Developer Tools see Video just just after this.
FOR Communicating with Console see Video

Using getElementByXXX()

Monday, January 30, 2017 12:04 PM



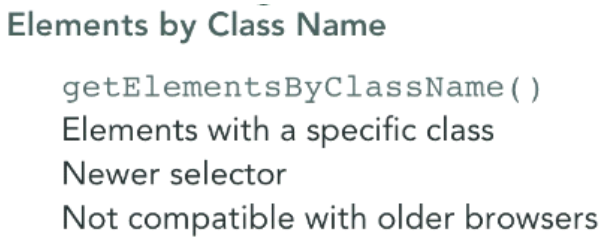
- We can use console.dir() which will show us all property of an element, like
  - **firstChild -** using which we can access first child of a node.
  - **childNodes** - will give all childrens of a node.
  - **parentNode** - will give parent of a node(can only be 1 so Node not Nodes)



- We can combine getElementById() and getElementByTagName() to filter more .

```
document.getElementsByTagName('li')
[▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>,
▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>]
document.getElementById('featuredartists').getElementsByTagName('li')
[▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>, ▶<li>…</li>]
v
```

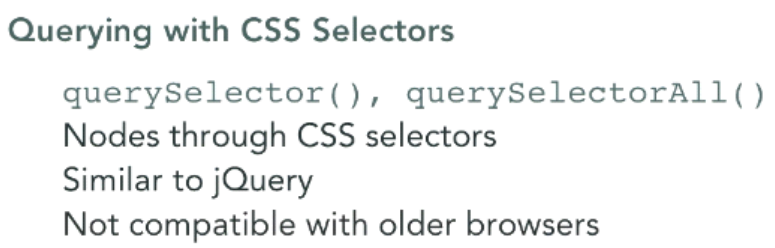
- We can use getElementByClassName() but it is not implemented in older version of IE.
- To know if we can use a function or not visit **caniuse.com**



Query CSS Selector

Monday, January 30, 2017 12:31 PM

- This lets us to select query using CSS like notation.(as CSS targets an element)



- For selecting first element we can use querySelector()

Example - 1 this will select first <article> tags.

```
> document.querySelector('article')
  ▶<article id="abouttheevent">…</article>
> |
```

- For selecting all elements we can use querySelectorAll()

```
document.querySelectorAll('article')
[▶<article id="abouttheevent">…</article>, ▶<article id="featuredartists">…</article>, ▶<article id="thevenue">…</article>,
▶<article id="schedule">…</article>]
```

Selecting class

- We can select using class as we do in CSS

```
> document.querySelectorAll('.artist')
  [▶<li class="artist group">…</li>, ▶<li class="artist group">…</li>, ▶<li class="artist group">…</li>,
  ▶<li class="artist group">…</li>, ▶<li class="artist group">…</li>, ▶<li class="artist group">…</li>,
> |
```

Selecting Type

- Same as CSS we can select type in a tag , following example will select
  - Input then filter only those who have type=checkbox.
- This can be done in CSS also.

```
> document.querySelectorAll('input[type=checkbox]')
  []
```

Selecting Descendent

- This will select all <li> nodes which are descendent of id #artistlist

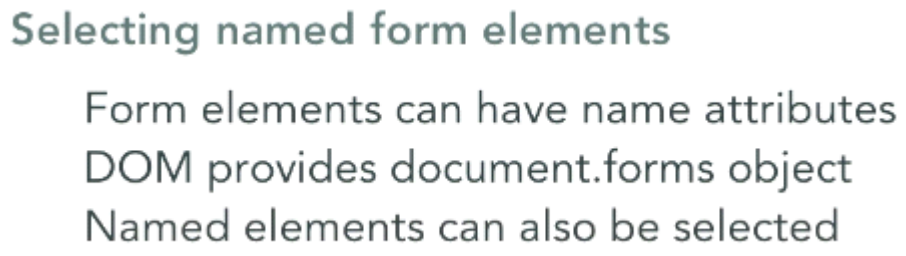
```
> document.querySelectorAll('#artistlist li')
```

Selecting Child

- This will select all <li> nodes which are child of id #artistlist

Named form selector

Monday, January 30, 2017 2:38 PM



- Document.forms can be used to get all forms as an array on a page.

```
> document.forms
▶ [form#register]
>
```

- So to get an individual form we use index

```
> document.forms[0]
▶ <form id="register" name="register" action="#">…</form>
>
```

- If a form have a name we can go to it using its name

```
> document.register
▶ <form id="register_form_id" name="register" action="#">…</form>
```

- Name selector can able to select also element inside a form, If 2 or more field have same name then it will return array.

here we have 'companyname' named text field

```
<input type="text" name="companyname" id="companyname">

document.register.companyname
  <input type="text" name="companyname" id="companyname">
```

changing value of forms

- We can change the value of an form by assigning value to is using 'value' attribute.
- Let we are assigning 'facebook' to the text field

```
<input type="text" name="companyname" id="companyname">
```

Using

```
document.register.companyname.value="facebook"
```

- This will populate field 'Company name' to 'facebook'

getElementsByName()

- We can use method getElementsByName() to select all elements having some name.

```
> document.getElementsByName('companyname')
< ▶ [input#companyname]
```

Assigning Value to Radio Button and checkbox

- We can check a radiobutton using 'checked' property
- First select that 'input' field and then assign.

```
document.getElementsByName('subscribe')[1].checked = "checked"
```

- Similarly we can do with checkbox

```
document.querySelectorAll("[type='checkbox']")[0].checked='checked'
"checked"
document.querySelectorAll("[type='checkbox']")[1].checked='checked'
"checked"
```

- This will select form field as below

Assigning Value to Dropdown list

- We can assign value to dropdown list, but the value should be from its options.

```
> document.register.reference.value='facebook'
< "facebook"
```

- Here 'reference' is name of that form field

- We can also use 'selectedIndex' which will option number

```
> document.register.reference.selectedIndex
< 2
```

NOTE- 0 based index but see as 1 based as at 0 we have 'choose'

Node Properties

Monday, January 30, 2017 3:26 PM

Understanding Node Properties

`node.nodeType` - numerical value of a node
`node.nodeName` - the name of the node
`node.atributes` - array of node attributes
`node.nodeValue` - element inside a node

NODETYPE -

```
var myNode = document.querySelectorAll('nav li a')[4]
undefined
myNode
<a href="register.html">register</a>
myNode.nodeType
1
```

- Here it returns the node type , there are many node types , they are following.

NodeTypes - Named Constants

NodeType	Named Constant
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

- *We will mostly use 3 top (EAT)*

NODE NAME -

- Node name is tag name so for <p> it will be 'p'

```
x=document.querySelectorAll('nav li a')[4]
  <a href="register.html">register</a>
x
  <a href="register.html">register</a>
x.nodeType
1
x.nodeName
"A"
```

- 'A' means it was an anchor tag <a> or <A>

NODE ATTRIBUTE -

- This list all attributes of a list in array

```
> x.attributes
< ▼NamedNodeMap
    ▶0: href
     length: 1
    ▶__proto__: NamedNodeMap
```

NODE VALUE -

- We can not set value to text-node itself beause, if a node is text-node it doesn't means
  It is text and we can assign value to it.
  For assigning value we have to 'nodeValue' property and assign value to it.
- First we have to see property of a node using console.dir(), in which we see firstChild of that node is 'text'

- This will not work as 'firstChild' is a text-node but not text

```
> myNode.firstChild = "registration"
  "registration"
```

- This will work

```
> myNode.firstChild.nodeValue = 'registration'
  "registration"
>
```

Node Traversal

Monday, January 30, 2017 3:57 PM

Moving up and down

`parentNode` - Goes up a level

`childNodes` - Array of children

`firstChild/lastChild` - First/Last Element

`previousSibbling/nextSibbling`
Elements with same parent

* note we have childNodes because there can be more than 1 children

- We have 'parentNode' because there are only 1 possible parent.

Example 1 -

```
myNode.parentNode
▶ <ul class="group">…</ul>
myNode.parentNode.childNodes
[▶ #text , ▶ <li>…</li>, ▶ #text , ▶ <li>…</li>, ▶ #text , ▶ <li>…</li>, ▶ #text
▶ <li>…</li>, ▶ #text , ▶ <li>…</li>, ▶ #text ]
myNode.parentNode.firstChild
▶ #text
myNode.parentNode.lastChild
▶ #text

myNode.parentNode.firstChild.nextSibling
▶ <li>…</li>
```

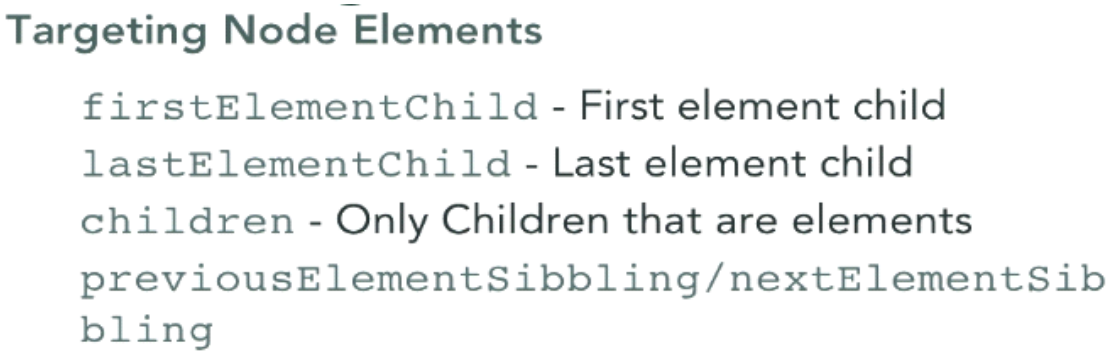
Example 2 - here we get first Element child instead of text

```
> myNode.parentNode.firstChild
  ▶ #text
> myNode.parentNode.firstElementChild
  ▶ <li>…</li>
```

Targeting Element nodes

Monday, January 30, 2017 4:04 PM

- Sometimes we may not wanted to get all children of a node as there may be text node , comment node etc.
- If wanted to target to Element Node only we can use some other property.
- Only Insert 'Element' in previous property to get new properties.



Example -

```
> myNode
  ▶ <li>…</li>
> myNode.parentNode
  ▶ <ul class="group">…</ul>
> myNode.parentNode.childNodes
  [▶ #text , ▶ <li>…</li>, ▶ #text , ▶ <li>…</li>, ▶ #text , ▶ <li>…</li>,
  ▶ <li>…</li>, ▶ #text , ▶ <li>…</li>, ▶ #text ]
> myNode.parentNode.children
  [▶ <li>…</li>, ▶ <li>…</li>, ▶ <li>…</li>, ▶ <li>…</li>, ▶ <li>…</li>, ▶
>
```

- Here we get rid of lot of text nodes (may be due to
)

DOM Quick Refrence

Monday, January 30, 2017 3:41 PM

DOM refrence

Monday, January 30, 2017 3:41 PM

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

Changing HTML attribute

Thursday, February 2, 2017 12:18 AM

Changing HTML attributes

- We can easily access attribute of a node using dot(.) notation, so if x is node we can access 'value' of x using => '**x.value**' (here 'value' is attribute of x)

Dot notation provides easy access

OPERATIONS

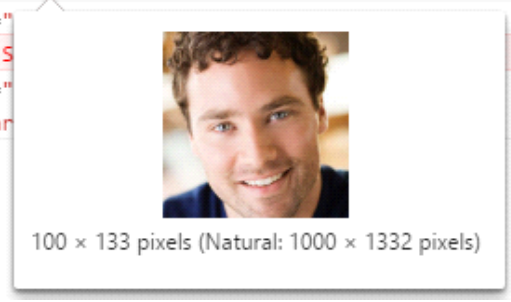
Read and write properties
Add attributes that don't exist
Be careful of reserved words

- If attribute is not present then it will be added, if it is present then it will be modified.
- We have to be careful about some words which means something to JS as well, like 'class' this can't be handled like this and we have to use different method.

USING UNRESERVED ATTRIBUTES

Example-
Step 1 - access that node using querySelector[this will give only the first matched) ,
and we get a <img> node.

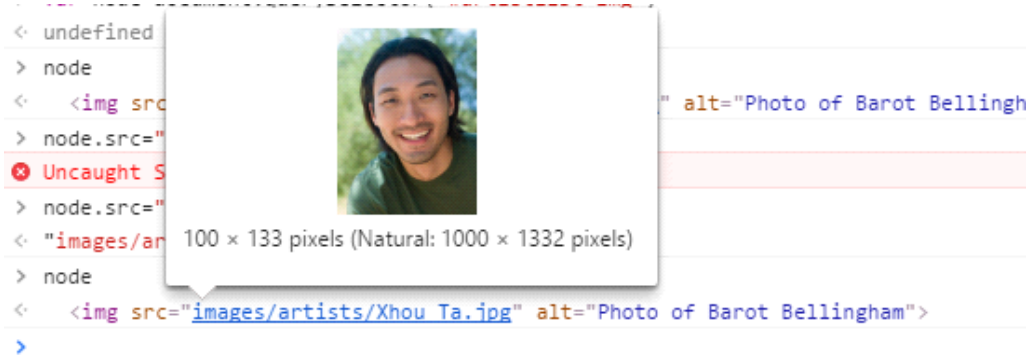
```
> var node=document.querySelector("#artistlist img")
< undefined
> node
<   <img src="images/artists/Barot_Bellingham_tn.jpg" alt="Photo of Barot Bellingham">
> node.src="
Uncaught S
> node.src="
< "images/ar
>
```



Step 2- now we can change the content of image 'src' node, using following command.

```
> node.src="images/artists/Xhou_Ta.jpg"
< "images/artists/Xhou_Ta.jpg"
```

SO we get final output <img> tag having following image.



USING RESERVED ATTRIBUTES

- Using dot(.) operation we can't change words which have special meaning for JS like,
    - Class,for etc.

Adding or modifying class

- We have to use 'className' property to add/modify class
- If it is not present then it will be added, if it is present then it will be modified.

```
> node
<- <img src="images/artists/Xhou_Ta.jpg" alt="Photo of Barot Bellingham">
> node.class="newClass"
<- "newClass"
> node
<- <img src="images/artists/Xhou_Ta.jpg" alt="Photo of Barot Bellingham">
> /* so above method do not worked out, we have to use 'classNa   me' for adding class */
<- undefined
> node.className="newClass"
<- "newClass"
> node
<- <img src="images/artists/Xhou_Ta.jpg" alt="Photo of Barot Bellingham" class="newClass">
> node.className="modifiedClass"
<- "modifiedClass"
> node
<- <img src="images/artists/Xhou_Ta.jpg" alt="Photo of Barot Bellingham" class="modifiedClass">
>
```

Adding or modifying 'for' attribute

- 'For' can't be accessed using dot(.) operation because it is used for loop in javascript.
- We have to use 'htmlFor' property for this purpose.

```
> var node=document.querySelector("label");
<- undefined
> node
<-   <label for="myname">Name</label>
> /* now we get the first node having <label> */
<- undefined
> node.for="newfor"
<- "newfor"
> node
<-   <label for="myname">Name</label>
> node.htmlFor="newfor"
<- "newfor"
> node
```

```
> var node=document.querySelector("label");
<- undefined
> node
<-   <label for="myname">Name</label>
> /* now we get the first node having <label> */
<- undefined
> node.for="newfor"
<- "newfor"
> node
<-   <label for="myname">Name</label>
> node.htmlFor="newfor"
<- "newfor"
> node
<-   <label for="newfor">Name</label>
>
```

PROS

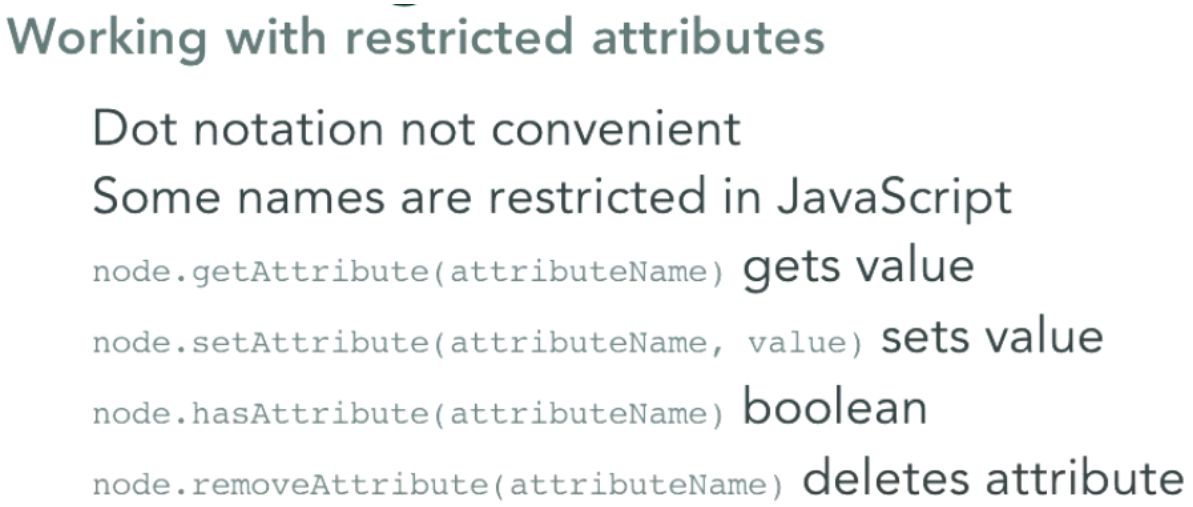
- Although the dot(.) notation is the fastest way of accesing attribute.

CONS -

- Problem 1 - with this type of solution is that there are many reserved words and it is not possible to remember property for each type, so we will use another method in next Page.
- Problem 2 - using this kind of method it is not possible to delete an attribute.

Working with Restricted Attributes

Thursday, February 2, 2017 12:55 AM



- These functions are better to use without worrying about if a word is a reserved attributes or not.

<attribute_value> getAttribute(<attribute_name>)

- This function returns the **value** of attribute if it is present , **null** otherwise.

```
node=document.querySelector('#register label')
  <label for="newfor">Name</label>
node.htmlFor
"newfor"
node.getAttribute('for');
"newfor"
```

- Both 'htmlFor' and 'getAttribute()' returns same result.

setAttribute(<attribute_name>,<attribute_value>)

- Return nothing

```
> node.setAttribute('for','hiFor')
< undefined
> node
<   <label for="hiFor">Name</label>
>
```

Boolean hasAttribute(<attribute_name>)

- Return value true/false according to presence and absence of attribute for a node.

```
> node
<-   <label for="hiFor">Name</label>
> node.hasAttribute('for')
<- true
> node.hasAttribute('value')
<- false
>
```

removeAttribute(<attribute_name>)

- Remove attribute If it has

```
> node
<-   <label for="hiFor">Name</label>
> node.removeAttribute('for')
<- undefined
```

- If that attribute is not there then do nothing.

Detecting and creating Custom attributes

1:18 AM

Detecting data attributes

- If we give attributes that are not present or valid at that point Browser just ignore these attributes. suppose we added 'coolness' attribute, then browser just ignore it.
- We can check validation of our html document at W3C validator.
- So if we want to add our own attribute we have to use certain protocol.

Users can type anything as an attribute
Browsers ignore them, but it's not valid HTML

Adding our Own Attribute

- We can create our data-attribute using syntax
  data-<attribute_name>

Create your own attributes using data
`data-coolness` valid attribute

- So, if we make attribute using above syntax then it will considered to be valid attribute using browser.
- These types of attributes are used a lot using Jquery etc.

Accessing our Own Attribute

- For accessing we have to use 'dataset' property of node and then we can use 'name' to access it(we do not need 'data-' here)

Example - we have a node having user made attribute **'task'** written as **'data-task'.**

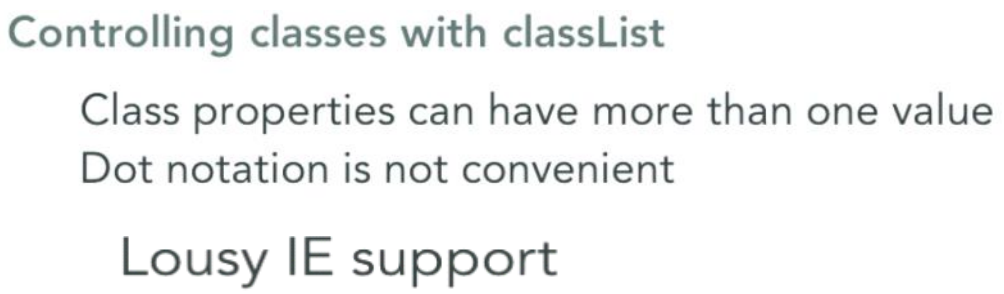
```
> myNode[1]
  <img src="images/artists/Constance_Smith_tn.jpg" alt="Photo of Constance Smith" data-task="speaker">
```

- Now to change the value of attribute 'task' to 'presenter' we can use following command.

```
> myNode[1].dataset.task
  "speaker"
> myNode[1].dataset.task = "presenter"
  "presenter"
> myNode[1]
  <img src="images/artists/Constance_Smith_tn.jpg" alt="Photo of Constance Smith" data-task="presenter">
```

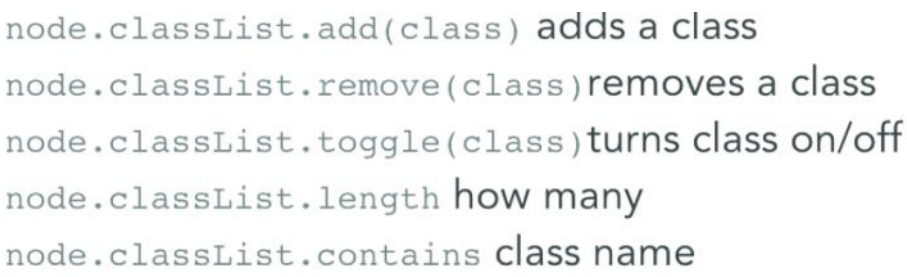
Adding more than one class

Thursday, February 2, 2017 2:04 AM



- A node can have more than 1 classes so 'class' attribute is not sufficient there for each node have a property 'classList' which stores the list of all classes a node have.

Handling classList



- Toggle will add if not present , if present it will remove.
  it will return 'true' if it has added, 'false if it has remove.

```
> node
<-   <label>Name</label>
> node.classList.add("class1");
<- undefined
> node
<-   <label class="class1">Name</label>
> node.classList.add("class2");
<- undefined
> node
<-   <label class="class1 class2">Name</label>
> node.classList
<- ▶ ["class1", "class2"]
> node.classList.contains("class2");
<- true
> node.classList.contains("class4");
<- false
>
```

```
> node.classList.remove("class1");
< undefined
> node
<   <label class="class2">Name</label>
>

> node.classList.length;
< 1
> node.classList.toggle("class1");
< true
> node
<   <label class="class2 class1">Name</label>
>

> node
<   <label class="class2 class1">Name</label>
> node.classList.toggle("class2");
< false
> node;
<   <label class="class1">Name</label>
```

- If we apply removeAttribute() to multiple 'class' then it will remove whole list

```
> node
<   <label class="class1 class2">Name</label>
> node.removeAttribute("class");
< undefined
> node
<   <label>Name</label>
```

Targeting all attributes

Thursday, February 2, 2017 2:24 AM

- We can get a list of all property using 'attributes' property of node

Targeting the attributes property

`node.attributes` returns a node list

Accessed in a variety of ways

By numeric index
By named index
Using dot notation

```
> node=document.querySelector("#artistlist img")
<-   <img src="images/artists/Barot_Bellingham_tn.jpg" alt="Photo of Barot
     Bellingham">
> node.attributes
<- ▶ NamedNodeMap {0: src, 1: alt, length: 2}
> /* each attribute can be accesed in 3 ways */
<- undefined
> node.attributes[0]
<-   src="images/artists/Barot_Bellingham_tn.jpg"
> node.attributes['src']
<-   src="images/artists/Barot_Bellingham_tn.jpg"
> node.attributes.src
<-   src="images/artists/Barot_Bellingham_tn.jpg"
> |
```

LIMITATION -

- The limitation to this is that the list returned using 'attributes' property is a readonly list and we can only access the value and can't change it.

```
node
  <img src="images/artists/Barot_Bellingham_tn.jpg" alt="Photo
  of Barot Bellingham" data-coolness="100%" class>
node.attributes
▶ NamedNodeMap {0: src, 1: alt, 2: data-coolness, 3: class,
  length: 4}
node.attributes.src="new.jpg"
"new.jpg"
node
  <img src="images/artists/Barot_Bellingham_tn.jpg" alt="Photo
  of Barot Bellingham" data-coolness="100%" class>
```

- The value does not changed because 'attributes' is a readonly list, so for changing use set attributes.

```
> node.setAttribute('src',"new.jpg")
< undefined
```

HTML Modifiers

Thursday, February 2, 2017 2:36 AM

Using text content modifiers

`node.innerHTML` changes text as HTML

`node.outerHTML` includes element's tags

`node.insertAdjacentHTML(insertionPoint, htmlText)`

- Text-modifiers allows us to modify the text for a tag.
- innerHTML means text between tags
  ex- **<a>** --- some --- text ---- ****
- outerHTML (comes in newer browsers), which includes everything including tags.
  if we wanted to change full structure we can reassign it with some other tag.

```
> node
<-   <img src="images/artists/Barot_Bellingham_tn.jpg" alt="Photo of Barot
     Bellingham">
> node.outerHTML
<- "<img src="images/artists/Barot_Bellingham_tn.jpg" alt="Photo of Barot
   Bellingham">"
> node.innerHTML
<- ""
>
```

Element.insertAdjacentHTML()

Summary

`insertAdjacentHTML()` parses the specified text as HTML or XML and inserts the resulting nodes into the DOM tree at a specified position. It does not reparse the element it is being used on and thus it does not corrupt the existing elements inside the element. This avoiding the extra step of serialization make it much faster than direct `innerHTML` manipulation.

Syntax

```
1 | element.insertAdjacentHTML(position, text);
```

`position` is the position relative to the `element`, and must be one of the following strings:

`'beforebegin'`
Before the `element` itself.

`'afterbegin'`
Just inside the `element`, before its first child.

`'beforeend'`
Just inside the `element`, after its last child.

`'afterend'`
After the `element` itself.

Visualization of position names

```
<!-- beforebegin -->
<p>
  <!-- afterbegin -->
  foo
  <!-- beforeend -->
</p>
<!-- afterend -->
```

- To learn - begin-2 then end-2 before-after consecutive.(bb->ab->be->ae)

Example

```
1 // <div id="one">one</div>
2 var d1 = document.getElementById('one');
3 d1.insertAdjacentHTML('afterend', '<div id="two">two</div>');
4
5 // At this point, the new structure is:
6 // <div id="one">one</div><div id="two">two</div>
```

NOTE- we can use innerText etc. to get only text, see in next page.

Text Modifiers

Thursday, February 2, 2017 2:58 AM

Using text content modifiers

`node.innerText` just the text of a node

- We can get just the text of an element using 'innerText' ,
  Here note that node is like
  <p> <img ……. > ……..some text…….. </P>
  That is why <img> tag is coming in inner html

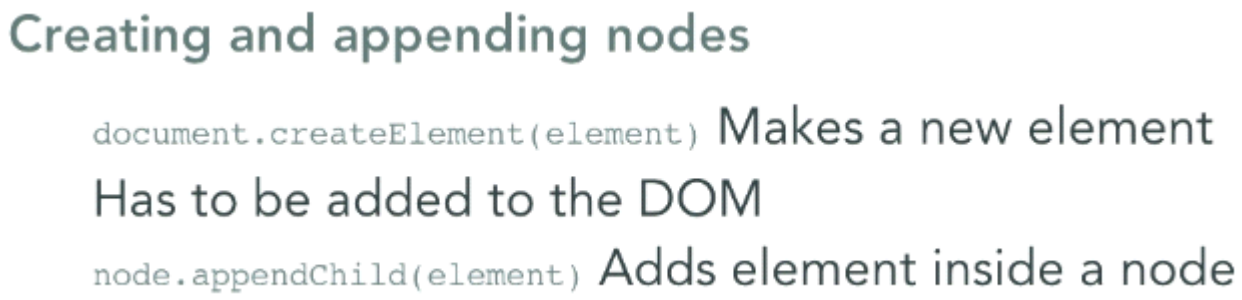
```
> node.innerHTML
<- "<img src="images/artists/Jonathan_Ferrar_tn.jpg" alt="Photo of Jonathan
   Ferrar">Labeled as “The Artist to Watch in 2012” by the London Review,
   Johnathan has already sold one of the highest priced commissions paid to
   an art student, ever on record. The piece, entitled Gratitude Resort, a
   work in oil and mixed media, was sold for $750,000."
> node.innerText
<- "Labeled as “The Artist to Watch in 2012” by the London Review,
   Johnathan has already sold one of the highest priced commissions paid to
   an art student, ever on record. The piece, entitled Gratitude Resort, a
   work in oil and mixed media, was sold for $750,000."
>
```

- 'innerText' will contain anything except anchors.
- We can set new 'innerText' also.

```
> node.innerText="new inner text"
<- "new inner text"
>
```

Creating and Appending Nodes

Monday, February 6, 2017 2:52 AM



- There are two steps
  - Create node
  - Add it to the dom (using append() or insertBefore())

Creating Element

- We just need to pass the tag name and the element will be created.

```
> n=document.createElement("p")
<-   <p></p>
> n=document.createElement('img')
<-   <img>
>
```

- Now we can insert the attributes using either dot(.) notation or using setAttribute() method.

```
> n.src="image.jpg"
<- "image.jpg"

> n
<-   <img src="image.jpg">
>
```

Appending Element

- Find element who's child this new node gonna be, and then append the new node as child to it.

```
> var par=document.querySelectorAll(".artistlist li")
<- undefined
> par
<- ▶ [li, li, li, li, li, li, li, li, li]
> par[6]
<-   <li></li>
> /* now we have to add <img> tag to this node */
```

```
> n
<-   <img src="image.jpg">
> par[6].appendChild(n)
<-   <img src="image.jpg">
```

Controlling Node Insertion

Monday, February 6, 2017 3:07 AM

Controlling node insertions

`appendChild()` lacks precision

Need to insert a node anywhere in the node list

Use `insertBefore()` for surgical insertions

Definition and Usage

The insertBefore() method inserts a node as a child, right before an existing child, which you specify.

Syntax-

```
parentNode.insertBefore(newChild,existingChild)
```

- This will insert a new node before existing child in parent node.
- If wanted to append at last use appendChild() method

Example 1 - we have insert a new <p> between these two <p>s.

Step (1) - create a new <P> for adding between two <p>s.

```
> node
<- ▼<article id="thevenue">
     <h2>The Venue</h2>
     <h3>Hotel Contempo</h3>
   ▶<p>…</p>
   ▶<p>…</p>
   </article>
> newp=document.createElement("p")
<-   <p></p>

> newp.innerHTML="some text"
<- "some text"
> newp
<-   <p>some text</p>
```

- We can also add a textNode() a children to 'newp'.

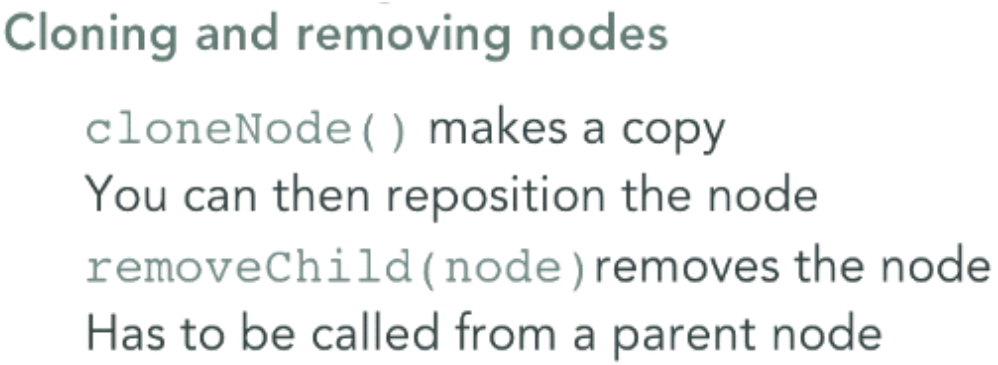
Step(2) - now add <p>.

```
> node
<- ▼<article id="thevenue">
     <h3>Hotel Contempo</h3>
   ▶<p>…</p>
   ▶<p>…</p>
   </article>
> newp
<-   <p>some text</p>
> node.childNodes
<- ▶[text, text, h3, text, p, text, p, text]
```

```
> node.insertBefore(newp,node.childNodes[6])
<-   <p>some text</p>
> node
<- ▼<article id="thevenue">
     <h3>Hotel Contempo</h3>
   ▶<p>…</p>
     <p>some text</p>
   ▶<p>…</p>
   </article>
```

Cloning and removing nodes

Monday, February 6, 2017 4:22 AM



Cloning Node

Syntax

```
node.cloneNode(deep)
```

Parameter	Type	Description
deep	Boolean	Optional. Specifies whether all descendants of the node should be cloned. • true - Clone the node, its attributes, *and* its descendants • false - Default. Clone only the node and its attributes

Definition and Usage

The cloneNode() method creates a copy of a node, and returns the clone.

The cloneNode() method clones all attributes and their values.

- Clone node will make exact copy of a node.

```
> node
<- ▶<article id="thevenue">…</article>
> copy_of_node=node.cloneNode()
<-   <article id="thevenue"></article>
> copy_of_node
<-   <article id="thevenue"></article>
>
```

- Now we can place this new node anywhere we wanted.

NOTE - we have created copy, but id does not clone recursively, if we wanted to clone also childrens, then we have to pass **'deep'** function argument as true.

```
> node
<- ▼<article id="thevenue">
     <h3>Hotel Contempo</h3>
   ▶<p>…</p>
     <p>some text</p>
   ▶<p>…</p>
   </article>
> full_copy=node.cloneNode(true)
<- ▼<article id="thevenue">
     <h3>Hotel Contempo</h3>
   ▶<p>…</p>
     <p>some text</p>
   ▶<p>…</p>
   </article>
```

Removing Node

The **`Node.removeChild()`** method removes a child node from the DOM. Returns removed node.

Syntax

```
var oldChild = node.removeChild(child);
OR
element.removeChild(child);
```

- `child` is the child node to be removed from the DOM.
- `node` is the parent node of `child`.
- `oldChild` holds a reference to the removed child node. `oldChild === child`.

The removed child node still exists in memory, but is no longer part of the DOM. With the first syntax-form shown, you may reuse the removed node later in your code, via the `oldChild` object reference. In the second syntax-form however, there is no `oldChild` reference kept, so assuming your code has not kept any other reference to the node elsewhere, it will immediately become unusable and irretrievable, and will usually be automatically deleted from memory after a short time.

```
> node
<- ▼<article id="thevenue">
     <h3>Hotel Contempo</h3>
   ▶<p>…</p>
     <p>some text</p>
   ▶<p>…</p>
   </article>
> /* now we wanted to remove paragraph having 'some text' */
```

```
> dir(node)
  ▶ article#thevenue
< undefined
> /* we found that is is childNode[6] of node */
< undefined
> node.removeChild(node.childNodes[6])
<   <p>some text</p>
> node
< ▼<article id="thevenue">
    <h3>Hotel Contempo</h3>
   ▶<p>…</p>
   ▶<p>…</p>
   </article>
```

Replacing a Node

Monday, February 6, 2017 4:31 AM

`replaceChild()` replaces a node
You must call it from the parent node
Saves you the step of having to delete the original

The **`Node.replaceChild()`** method replaces one child node of the specified node with another.

Syntax

```
replacedNode = parentNode.replaceChild(newChild, oldChild);
```

- `newChild` is the new node to replace `oldChild`. If it already exists in the DOM, it is first removed.
- `oldChild` is the existing child to be replaced.
- `replacedNode` is the replaced node. This is the same node as `oldChild`.

```
> node
<- ▼<article id="thevenue">
     <h3>Hotel Contempo</h3>
   ▶<p>…</p>
   ▶<p>…</p>
   </article>
> /* let replace h3 with h2 */
<- undefined
> newh2=document.createElement('h2');
<-   <h2></h2>
> newh2.innerHTML="I am new H2"
<- "I am new H2"
> newh2
<-   <h2>I am new H2</h2>
> dir(node)
  ▶article#thevenue
<- undefined
> /* h2 is child node index 2 */
```

```
> node.replaceChild(newh2,node.childNodes[2])
    <h3>Hotel Contempo</h3>
> node
▼<article id="thevenue">
    <h2>I am new H2</h2>
  ▶<p>…</p>
  ▶<p>…</p>
  </article>
```