1/What is append () in JavaScript?

The Element.append() method inserts a set of Node objects or string objects after the last child of the Element . String objects are inserted as equivalent Text nodes.

How to append an element in an array in JavaScript ?

1. using JavaScript push() Method.
2. using JavaScript unshift() Method.
3. using JavaScript splice() Method.
4. using JavaScript concat() Method.
5. using Javascript spread operator.

Element: append() method

The **Element.append()** method inserts a set of Node objects or string objects after the last child of the Element. String objects are inserted as equivalent Text nodes.

Differences from Node.appendChild():

- Element.append() allows you to also append string objects, whereas Node.appendChild() only accepts Node objects.
- Element.append() has no return value, whereas Node.appendChild() returns the appended Node object.
- Element.append() can append several nodes and strings, whereas Node.appendChild() can only append one node.

- **Syntax**
- JSCopy to Clipboard
- append(param1)
- append(param1, param2)
- append(param1, param2, /* …, */ paramN)

Examples

1/Appending an element

```
 let div = document.createElement("div");
let p = document.createElement("p");
div.append(p);

console.log(div.childNodes); // NodeList [ <p> ]
```

2/Appending text

```
let div = document.createElement("div");
div.append("Some text");

console.log(div.textContent); // "Some text"
```

3/Appending an element and text

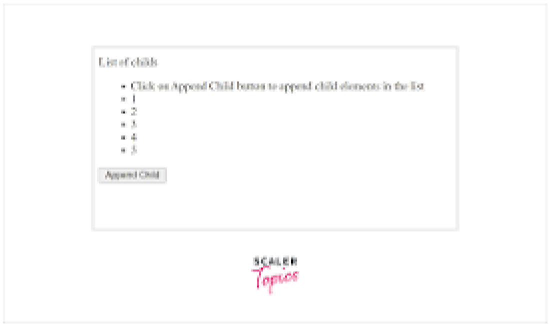
```
let div = document.createElement("div");

let p = document.createElement("p");

div.append("Some text", p);



console.log(div.childNodes); // NodeList [ #text "Some text", <p> ]
```

The append method is unscopable

The append() method is not scoped into the with statement. See Symbol.unscopables for more information.

2/HTML DOM Element appendChild():

What is append child in JavaScript?


List of childs

- Click on Append Child button to append child elements in the list
- 1
- 2
- 3
- 4
- 5

Meaning the appendChild() method is used to add a new child element at the end of a parent element, along with this, the method can also be used to move an existing child element within the document from one parent element to another.

Examples

Append an item to a list:

```
const node = document.createElement("li");
const textnode = document.createTextNode("Water");
node.appendChild(textnode);
document.getElementById("myList").appendChild(node);
```

Before:

- Coffee
- Tea

After:

- Coffee
- Tea
- Water

3/HTMLCollection

The **HTMLCollection** interface represents a generic collection (array-like object similar to arguments) of elements (in document order) and offers methods and properties for selecting from the list.

An HTMLCollection in the HTML DOM is live; it is automatically updated when the underlying document is changed. For this reason it is a good idea to make a copy (e.g., using Array.from) to iterate over if adding, moving, or removing nodes.

Instance properties

HTMLCollection.length Read only

Returns the number of items in the collection.

Instance methods

HTMLCollection.item()

Returns the specific node at the given zero-based index into the list. Returns null if the index is out of range.

An alternative to accessing collection[i] (which instead returns undefined when i is out-of-bounds). This is mostly useful for non-JavaScript DOM implementations.

HTMLCollection.namedItem()

Returns the specific node whose ID or, as a fallback, name matches the string specified by name. Matching by name is only done as a last resort, only in HTML, and only if the referenced element supports the name attribute. Returns null if no node exists by the given name.

An alternative to accessing collection[name] (which instead returns undefined when name does not exist). This is mostly useful for non-JavaScript DOM implementations.

Usage in JavaScript

HTMLCollection also exposes its members directly as properties by both name and index. HTML IDs may contain : and . as valid characters, which would necessitate using bracket notation for property access. Currently HTMLCollections does not recognize purely numeric IDs, which would cause conflict with the array-style access, though HTML5 does permit these.

For example:

```
let elem1, elem2;


// document.forms is an HTMLCollection
```

```
elem1 = document.forms[0];

elem2 = document.forms.item(0);


alert(elem1 === elem2); // shows: "true"


elem1 = document.forms.myForm;

elem2 = document.forms.namedItem("myForm");


alert(elem1 === elem2); // shows: "true"


elem1 = document.forms["named.item.with.periods"];
```

4/What is a NodeList?

A NodeList is a collection of document nodes (element nodes, attribute nodes, and text nodes). HTMLCollection items can be accessed by their name, id, or index number. NodeList items can only be accessed by their index number

What is the use of NodeList?

A NodeList object is a list (collection) of nodes extracted from a document. A NodeList object is almost the same as an HTMLCollection object. Some (older) browsers return a NodeList object instead of an HTMLCollection for methods like getElementsByClassName().

For example:

```
const parent = document.getElementById("parent");

let childNodes = parent.childNodes;

console.log(childNodes.length); // let's assume "2"

parent.appendChild(document.createElement("div"));

console.log(childNodes.length); // outputs "3"
```

Static NodeLists

In other cases, the NodeList is *static*, where any changes in the DOM do not affect the content of the collection. The ubiquitous document.querySelectorAll() method returns a *static* NodeList.

It's good to keep this distinction in mind when you choose how to iterate over the items in the NodeList, and whether you should cache the list's length.

Instance properties

NodeList.length Read only

The number of nodes in the NodeList.

Instance methods

NodeList.item()

Returns an item in the list by its index, or null if the index is out-of-bounds.

An alternative to accessing nodeList[i] (which instead returns undefined when i is out-of-bounds). This is mostly useful for non-JavaScript DOM implementations.

NodeList.entries()

Returns an iterator, allowing code to go through all key/value pairs contained in the collection. (In this case, the keys are integers starting from 0 and the values are nodes.)

NodeList.forEach()

Executes a provided function once per NodeList element, passing the element as an argument to the function.

NodeList.keys()

Returns an iterator, allowing code to go through all the keys of the key/value pairs contained in the collection. (In this case, the keys are integers starting from 0.)

NodeList.values()

Returns an iterator allowing code to go through all values (nodes) of the key/value pairs contained in the collection.

Example

It's possible to loop over the items in a NodeList using a for loop:

JSCopy to Clipboard

```
for (let i = 0; i < myNodeList.length; i++) {
  let item = myNodeList[i];
}
```