Collections in JavaScript

Introduction

Groups of data in different forms are one of the fundamental data structures in most programming languages. In many cases, **groups of data expressed through different data types** are referred to as **Collections**.

In this guide - we'll take a look at *Collections in JavaScript* and when to use which type of collection. The three main **Collection Groups** we'll be taking a look at are:

- **Indexed Collections**
- **Keyed Collections**
- **DOM Collections**

Indexed Collections

An *indexed collection* is a collection of data which is *listed by their index*. JavaScript collection indices are *0-based*, which means they start at 0, not 1 and go up to n-1, n being the number of objects in the collection. JavaScript has two kinds of indexed collections Arrays and Typed Arrays.

Array object

An Array object in JavaScript is an ordered list, whose elements can be accessed using indices. There are multiple ways of creating an Array object in JavaScript, and there's not much difference under the hood:

Let myArray1 = [x1, x2, xN];

Let myArray2 = new Array(x1, x2, xN);

Let myArray3 = Array(x1, x2, xN);

Arrays in JavaScript are not type-based, which means that you don't have to define the type of the array beforehand and you don't have to add only homogenous elements:

Let my Array = ["one", 1, "two", 2];

Adding Elements to an Array

We've seen how to create an Array, be it empty or non-empty. Now let's see how to add new elements to it. Since we are working with *indexed collections* we will be operating with indexes.

As we already had created an Array of 4 empty elements, let's work with that. To add an element, all we have to do is *access the element through its index* and *assign a value* to it:

Let myArray1 = new Array (4)

myArray1 [0] = "one"

myArray1 [1] = "two"

myArray1 [2] = "three"

Typed Array Object

Array objects are perfect for working with any data type in JavaScript, since it can store different types of elements in one array and has powerful methods to manipulate those elements.

However, when there is need to work with raw binary data - that's when TypedArray objects come into play. Raw data is processed when manipulating audio and video for example.

Architecture of a Typed Array Object

JavaScript typed arrays are divided into ***buffers*** and ***views***. A *buffer* is an object only storing a chunk of data, with no methods to access or manipulate that data. In order to achieve that, you must use a *view* - which provides a *context*, a data type that turns data into a Typed Array.

A *buffer* is implemented through an Array Buffer object. It is used to represent a fixed-length binary data buffer. To represent this buffer, we have to create a view - Data View - which represents that buffer in a chosen format. There are various types of views, representing the most common numeric types:

- Int8Array - value range [-128, 127]
- UInt8Array - value range [0, 255], u stands for *unsigned*
- Int16Array - value range [-32768, 32767]
- UInt16Array - value range [0, 65535]
- Float32Array - value range [1.2E-38, 3.4E38]

Keyed Collections

A Keyed Collection is a collection of data represented in the key-value notation. Elements' values are accessed and manipulated through their respective keys.

In JavaScript, there are two types of keyed collections: Map and Set.

Both Maps and Sets in JavaScript can have a single value attributed to a single key, though you could hack it by attributing a List as a value, containing multiple elements. It's still worth noting that the List itself is the value - not its constituent elements.

Additionally, keys have to be unique.

The Map Object

A Map object in JavaScript is a standard map containing key-value pairs. To create a new Map object, we simply call the constructor:

Let my Map = new Map ();

Adding an Element to a Map

An empty map won't do us much good. Let's add some elements to it via the set () method, which accepts a key name which has to be a String, and a value which can be of any type:

myMap.set ("one", 1);

myMap.set ("two", 2);

The Set Object

A Set object in JavaScript is a collection of values only. These values are unique, which means there are no duplicates allowed and trying to add a duplicate element simply won't add anything.

We can also test this since printing sets prints their elements in insertion order, and adding a duplicate element at the start and end will only result in the first one being present.

Creating a Set is as simple as calling its constructor:

Let mySet = new Set ();

Adding an Element to a Set

To add a new element to a set, we use the add (value) method.

HTML DOM Collections

This type of collection is front-end web development related.

When working on a web page, we can access all the elements on the page thanks to the *DOM tree*. So, when accessing multiple elements at once, they are being returned as an **HTMLCollection** - an array-like collection of HTML elements.

If we have a web page containing multiple <p> tags, we can retrieve them with document.getElementsByTagName ("p") - which returns a collection of all the <p> elements on the page:

Let myHTMLCollection = document.getElementsByTagName ("p");

console.log (myHTMLCollection [1]);

We can now recognize that an HTMLCollection is an "indexed" collection, since we're accessing an element from it using an index value. It's not a *true* indexed JavaScript collection since it's not an array, because it doesn't have the array methods, but index accessing is available.

An HTMLCollection has the length property, which returns its size.