Walking the DOM

Tag	Node	Description
<html>	document.documentElement	The topmost tree nodes are available directly as document properties:
<body>	document.body	Another widely used DOM node is the <body> element – document.body. There's a catch: document.body can be null
<head>	document.head	

Element-only navigation

Navigation properties listed above refer to all nodes. For instance, in childNodes we can see both text nodes, element nodes, and even comment nodes if there exist.
But for many tasks we don't want text or comment nodes. We want to manipulate element nodes that represent tags and form the structure of the page

The links are similar to those given above, just with Element word inside:

- children – only those children that are element nodes.
- firstElementChild , lastElementChild – first and last element children.
- previousElementSibling , nextElementSibling – neighbor elements.
- parentElement – parent element.

Child nodes (or children) – elements that are direct children. In other words, they are nested exactly in the given one. For instance, <head> and <body> are children of <html> element.
The childNodes collection lists all child nodes, including text nodes.

Descendants – all elements that are nested in the given one, including children, their children and so on.
Properties firstChild and lastChild give fast access to the first and last children.

They are just shorthands. If there exist child nodes, then the following is always true:
1. elem.childNodes[0] === elem.firstChild
2. elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
There's also a special function elem.hasChildNodes() to check whether there are any child nodes.

The next sibling is in nextSibling property, and the previous one – in previousSibling .
The parent is available as parentNode .
For example:

```
// 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
```

Attributes and Properties

Summary
● Attributes – is what's written in HTML.
● Properties – is what's in DOM objects.
A small comparison:
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
Methods to work with attributes are:
● elem.hasAttribute(name) – to check for existence.
● elem.getAttribute(name) – to get the value.
● elem.setAttribute(name, value) – to set the value.
● elem.removeAttribute(name) – to remove the attribute.
● elem.attributes is a collection of all attributes.

For most situations using DOM properties is preferable. We should refer to attributes only when DOM properties do not suit us, when we need exactly attributes, for instance:
● We need a non-standard attribute. But if it starts with data- , then we should use dataset .
● We want to read the value "as written" in HTML. The value of the DOM property may be different, for instance the href property is always a full URL, and we may want to get the "original" value.

Node properties: type, tag and contents

Name	Value
EventTarget	The root "abstract" class. Objects of that class are never created. It serves as a base, so that all DOM nodes support so-called "events", we'll study them later.
Node	The an "abstract" class, serving as a base for DOM nodes. It provides the core tree functionality: parentNode, nextSibling, childNodes and so on (they are getters). Objects of Node class are never created. But there are concrete node classes that inherit from it, namely: Text for text nodes, Element for element nodes and more exotic ones like Comment for comment nodes.
Element	This is a base class for DOM elements. It provides element-level navigation like nextElementSibling , children and searching methods like getElementsByTagName , querySelector . A browser supports not only HTML, but also XML and SVG. The Element class serves as a base for more specific classes: SVGElement , XMLElement and HTMLElement.
HTMLElement	This is finally the basic class for all HTML elements.

The getter steps are to return the first matching statement, switching on this:

node	nodeType	nodeName
Element	ELEMENT_NODE (1)	Its HTML-uppercased qualified name.

Attr	ATTRIBUTE_NODE (2);	Its qualified name.
Text	TEXT_NODE (3);	"#text".
CDATASection	CDATA_SECTION_NODE (4);	"#cdata-section".
ProcessingInstruction	PROCESSING_INSTRUCTION_NODE (7);	Its target.
Comment	COMMENT_NODE (8);	"#comment".
Document	DOCUMENT_NODE (9);	"#document".
DocumentType	DOCUMENT_TYPE_NODE (10);	Its name.
DocumentFragment	DOCUMENT_FRAGMENT_NODE (11).	"#document-fragment".

Node properties

innerHTML	The innerHTML property allows to get the HTML inside the element as a string. We can also modify it. So it's one of the most powerful ways to change the page.
outerHTML	Full HTML of the element The outerHTML property contains the full HTML of the element. That's like innerHTML plus the element itself.
textContent	The textContent provides access to the text inside the element: only text, minus all <tags>.
nodeValue/data	The innerHTML property is only valid for element nodes.
The "hidden" property	The "hidden" attribute and the DOM property specifies whether the element is visible or not.

Searching

Metod	Action
querySelector	CSS-selector
querySelectorAll	CSS-selector
getElementById	id
getElementsByName	name

getElementsByTagName	tag or '*'
getElementsByClassName	Class name
mathes	The elem.matches(css) does not look for anything, it merely checks if elem matches the given CSS-selector. It returns true or false. The method comes in handy when we are iterating over elements (like in an array or something) and trying to filter out those that interest us.
closest	Ancestors of an element are: parent, the parent of parent, its parent and so on. The ancestors together form the chain of parents from the element to the top. The method elem.closest(css) looks the nearest ancestor that matches the CSS-selector. The elem itself is also included in the search. In other words, the method closest goes up from the element and checks each of parents. If it matches the selector, then the search stops, and the ancestor is returned.

Selectors:

Basic kinds of selectors are:

Argument	Value
*	Any element
TagName	Any elements with tagName
#id	An element with this id
.class	Any elements with such class
Attributes	Attribute with name and value Possible variants: [attr] — Attribute was set. [attr="val"] — Attribute equals val. [attr^="val"] — Attribute begins since val. [attr|="val"] — Attribute equals val or begins since val. [attr*="val"] — Attribute contains substring val. [attr~="val"] — Attribute contains substring val as one of possible values. For example: [attr~="delete"] for "edit delete" and not for "undelete"/"no-delete" . [attr$="val"] — Attribute ends on val.
.class1 .class2	Combine classes in selector field.
Tagname#id .class1 .class2	Combine tagname with identifier name and some style classes included

:pseudoclass	pseudoclass.
Tagname1 Tagname2	Select all nodes with tagname2 which are children of node with tagname1.
Tagname1 > Tagname2	Select all nodes with tagname2 which are first level childs of node with tagname1.
Tagname1 - Tagname2	Select all nodes with tagname2 which are right neighbors of node with tagname1 on the same level.
Tagname1 + Tagname2	Select node with tagname2 which are first right neighbor of node with tagname1 on the same level.
:first-child tagname:first-of-type	First child node of it`s parent. First child node of it`s parent with the same tagname.
:last-child tagname:last-of-type	Last child node of it`s parent. Last child node of it`s parent with the same tagname.
:only-child tagname:only-of-type	One child node of it`s parent if others are not exist. One child node of it`s parent if others are not exist with the same tagname.
:nth-child(n) tagname:nth-of-type tagname:nth-last-of-type	Child node of it`s parent number n. n > 1. Child node of it`s parent with the same tagname number n. n > 1.
:nth-child(an+b)	Allows get all child nodes of it`s parent. Examples: :nth-child(2n) evens order number. :nth-child(2n+1) odds order number.
:not(selector)	Select all nodes except that is relevant selector.
:focus	Select all nodes in focus.
:hover	Select node under mouse pointer.
:empty	Select all nodes without children, text nodes too.
:checked :disabled :enabled	Select all input elements.
:target	Select an element, which has an id === '#... ' current URL.

::before
::after
	<style>
	tagname::before {
	content: " [[";
	}
	tagname::after {
	content: "]] ";
	}
	</style>

Modifying document

Method	Action
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 and removal
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.
parent.appendChild(node) parent.insertBefore(node, nextSibling) parent.removeChild(node) parent.replaceChild(newElem, node)	
elem.insertAdjacentHTML(where, html) inserts it depending on the value of where: "beforebegin" – insert html right before elem , "afterbegin" – insert html into elem , at the beginning, "beforeend" – insert html into elem , at the end, "afterend" – insert html right after elem.	Given some HTML in html

Event	Description
DOMContentLoaded	The browser fully loaded HTML, and the DOM tree is built, but external resources like pictures <img> and stylesheets may be not yet loaded. DOM is ready, so the handler can lookup DOM nodes, initialize the interface.
load	Not only HTML is loaded, but also all the external resources: images, styles etc. External resources are loaded, so styles are applied, image sizes are known etc.
beforeunload/unload	The user is leaving the page. We can check if the user saved the changes and ask them whether they really want to leave.
unload	The user almost left, but we still can initiate some operations, such as sending out statistics. Let’s explore the details of these events.

<table>
<tr><td>readyState</td><td>The current state of the document, changes can be tracked in the readystatechange event:

<table><tr><td>loading</td><td>The document is loading.</td></tr><tr><td>interactive</td><td>The document is parsed, happens at about the same time as DOMContentLoaded , but before it.</td></tr><tr><td>complete</td><td>The document and resources are loaded, happens at about the same time as window.onload , but before it.</td></tr></table></td></tr>
</table>

Elements and fonts properties

Name	Type	Usage
Pixels, px	Real number	Use it for putting screen resolution of some element
Em, em	Real number	Use it for putting font size value of some element

Text-align	Enumeration	Possible values:
		`text-align: left`
		`text-align: right`
		`text-align: center`
		`text-align: justify`
		`text-align: start`
		`text-align: end`

Margin	String type	Use it for putting margin between near elements. Possible alternative margin-*: left margin on the left side of an element. top margin on the top side of an element. right margin on the right side of an element. bottom margin on the bottom side of an element.
Overflow	Enumeration	Possible values: • visible; • hidden; • scroll; • auto; It manages content disposition if length of it more than limit size.
Position	Enumeration	Possible values: • static; • absolute; • relative; • fixed;
Dsiplay	Enumeration	Possible values: • none;

		• block; • inline; • inline-block; • table; • table-row; • table-header-group; • table-row-group; • table-footer-group; • table-column; • table-column-group; • table-cell; • table-caption;
Box-sizing	Enumeration	Possible values: • border-box; • content-box;
Font-size	Px	A size of font inside some element

Blob

Blobs allow you to construct file like objects on the client that you can pass to apis that expect urls instead of requiring the server provides the file. For example, you can construct a blob containing the data for an image, use `URL.createObjectURL()` to generate a url, and pass that url to `HTMLImageElement.src` to display the image you created without talking to a server.

<table>
<tr><td>new Blob()</td><td>Creates a new <code>Blob</code> with <code>size</code> set to <code>0</code>.</td></tr>
<tr><td>new Blob(blobParts: Array, [blobPropertyBag: Object]):Blob
blobPropertyBag:{
<table>
<tr><td>Type</td><td>String</td><td>A valid mime type such as <code>'text/plain'</code></td></tr>
<tr><td>endings</td><td>String</td><td>Must be either <code>'transparent'</code> or <code>'native'</code></td></tr>
</table>
}

// Create a new Blob object
var a = new Blob();
// Create a 1024-byte ArrayBuffer
// buffer could also come from reading a File
var buffer = new ArrayBuffer(1024);
// Create ArrayBufferView objects based on buffer
var shorts = new Uint16Array(buffer, 512, 128);
var bytes = new Uint8Array(buffer, shorts.byteOffset + shorts.byteLength
var b = new Blob(["foobarbazetcetc" + "birdiebirdieboo"], {type: "text/plain;ch</td><td>Creates a new <code>Blob</code>. The elements of <code>blobParts</code> must be of the types <code>ArrayBuffer</code>, <code>ArrayBufferView</code>, <code>Blob</code>, or <code>String</code>. If <code>ending</code> is set to <code>'native'</code>, the line endings in the blob will be converted to the system line endings, such as <code>'\r\n'</code> for Windows or <code>'\n'</code> for Mac.</td></tr>
</table>

var c = new Blob([b, shorts]);
var a = new Blob([b, c, bytes]);
var d = new Blob([buffer, b, c, bytes]);	

Size	the size of the blob in bytes. Read-Only. Number.
Type	the type of the blob. Read-Only. String
slice([start=0:Number, [end:Number, [contentType=":String]]]):Blob	Returns a new blob that contains the bytes `start` to `end - 1` from `this`. If `start` or `end` is negative, the value is added to `this.size` before performing the slice. If `end` is not specified, `this.size` is used. The returned blob's `type` will be `contentType` if specified, otherwise it will be `''`.

File

File is a Blob that represents a file from the filesystem. You can get Files from the HTMLInputElement.files property or the DataTransferItem.getAsFile() method. Use FileReader to read the contents of a File.

new File(fileParts : Array, name : String, [filePropertyBag : Object]) : File filePropertyBag : {typeStringA valid mime type such as `'text/plain'`endingsStringMust be either `'transparent'` or `'nativ'`}	Creates a new `File`. The elements of `fileParts` must be of the types `ArrayBuffer`, `ArrayBufferView`, `Blob`, or `String`. If `ending` is set to `'native'`, the line endings in the file will be converted to the system line endings, such as `'\r\n'` for Windows or `'\n'` for Mac.

<input type="file" id="input" (multiple)>
Accessing one file
document.getElementById('input').files[0];
 	
var inputElement = document.getElementById("input");
inputElement.addEventListener("change", function() {
var fileList = this.files;
}, false);
	Input field type file to get one or multiple files

	var file = new File(['foo', 'bar'], 'foobar.txt');
	
	console.log('size=' + file.size);
	console.log('type=' + file.type);
	console.log('name=' + file.name);

	var testEndings = function(string, endings) {
	  var file = new File([string],
	{ type: 'plain/text', endings: endings });
	  var reader = new FileReader();
	  reader.onload = function(event){
	  console.log(endings + ' of ' +
	JSON.stringify(string) +
	' => ' + JSON.stringify(reader.result));
	  };
	  reader.readAsText(file);
	};
	
	testEndings('foo\nbar', 'native');
	testEndings('foo\r\nbar', 'native');
	testEndings('foo\nbar', 'transparent');
	testEndings('foo\r\nbar', 'transparent');

lastModifiedDate : Date readonly	The last time the file was modified.

	<input type='file' onchange='openFile(event)'>
	<script>
	  var openFile = function(event) {
	  var input = event.target;
	
	  var file = input.files[0];
	  console.log(file.lastModifiedDate);
	  };
	</script>

name : String readonly	The name of the file.

	<input type='file'
	onchange='onFilePicked(event)'>
	<script>
	  var onFilePicked = function(event) {
	  var input = event.target;
	
	  var file = input.files[0];
	  console.log(file.name);
	  };
	</script>

FileReader

FileReader is used to read the contents of a `Blob` or `File`.is used to read the contents of a `Blob` or `File`.

new FileReader() : FileReader	Constructs a new FileReader.
	<input type='file' accept='image/*'

	onchange='openFile(event)'>
	
	<img id='output'>
	<script>
	  var openFile = function(event) {
	  var input = event.target;
	
	  var reader = new FileReader();
	  reader.onload = function(){
	  var dataURL = reader.result;
	  var output = document.getElementById('output');
	  output.src = dataURL;
	  };
	  reader.readAsDataURL(input.files[0]);
	  };
	</script>

error : Error readonly	The error encountered during load.

	<input type='file' onchange='openFile(event)'>
	<script>
	  var openFile = function(event) {
	  var input = event.target;
	
	  var reader = new FileReader();
	  reader.onloadstart = function() {
	  reader.abort();
	  };
	
	  reader.onloadend = function() {
	  console.log(reader.error.message);
	  };
	
	  reader.readAsDataURL(input.files[0]);
	  };
	</script>

readyState : Number readonly	The current state of the reader. Will be one of EMPTY, LOADING, or DONE.

	<input type='file' onchange='openFile(event)'>
	<script>
	  var stateNames = {};
	  stateNames[FileReader.EMPTY] = 'EMPTY';
	  stateNames[FileReader.LOADING] = 'LOADING';
	  stateNames[FileReader.DONE] = 'DONE';
	
	  var openFile = function(event) {
	  var input = event.target;

	var reader = new FileReader();
	reader.onload = function(){
	console.log('After load: ' + stateNames[reader.readyState]);
	};
	
	console.log('Before read: ' + stateNames[reader.readyState]);
	reader.readAsDataURL(input.files[0]);
	console.log('After read: ' + stateNames[reader.readyState]);
	};
	</script>

result : Object readonly	The result from the previous read. The result will be either a `String` or an `ArrayBuffer`. The result is only available after the `load` event fires.

	<input type='file' accept='image/*' onchange='openFile(event)'>
	
	<script>
	var openFile = function(event) {
	var input = event.target;
	
	var reader = new FileReader();
	reader.onload = function(){
	var dataURL = reader.result;
	var output = document.getElementById('output');
	output.src = dataURL;
	};
	reader.readAsDataURL(input.files[0]);
	};
	</script>

abort() : undefined	Stops the current read operation.
readAsArrayBuffer(blob : Blob) : undefined	Begins reading from `blob` as an `ArrayBuffer`. The result will be stored on `this.result` after the `'load'` event fires.

	<input type='file' onchange='openFile(event)'>
	<script>
	var openFile = function(event) {
	var input = event.target;
	
	var reader = new FileReader();
	reader.onload = function(){
	var arrayBuffer = reader.result;

	console.log(arrayBuffer.byteLength);
	};
	reader.readAsArrayBuffer(input.files[0]);
	};
	</script>

readAsDataURL(blob : Blob) : undefined	Begins reading from `blob` as a `'data:'` url string. The result will be stored on `this.result` after the `'load'` event fires.

	<input type='file' accept='image/*'
	onchange='openFile(event)'>
	
	<script>
	var openFile = function(event) {
	var input = event.target;
	
	var reader = new FileReader();
	reader.onload = function(){
	var dataURL = reader.result;
	var output =
	document.getElementById('output');
	output.src = dataURL;
	};
	reader.readAsDataURL(input.files[0]);
	};
	</script>

readAsText(blob : Blob, [encoding : String]) : undefined	Begins reading from `blob` as a string. The result will be stored on `this.result` after the `'load'` event fires. For the valid values of `encoding`, see character sets.

	<input type='file' accept='text/plain'
	onchange='openFile(event)'>
	
	<script>
	var openFile = function(event) {
	var input = event.target;
	
	var reader = new FileReader();
	reader.onload = function(){
	var text = reader.result;
	console.log(reader.result.substring(0, 200));
	};
	reader.readAsText(input.files[0]);
	};
	</script>

onloadstart / 'loadstart' event listener(event : ProgressEvent) : undefined	Called after starting a read operation.

	<input type='file' onchange='openFile(event)'>
	<script>
	  var openFile = function(event) {
	  console.log('entering openFile()');
	  var input = event.target;
	
	  var printEventType = function(event) {
	  console.log('got event: ' + event.type);
	  };
	
	  var reader = new FileReader();
	  reader.onloadstart = printEventType;
	  reader.onprogress = printEventType;
	  reader.onload = printEventType;
	  reader.onloadend = printEventType;
	
	  console.log(' starting read');
	  reader.readAsDataURL(input.files[0]);
	  console.log('leaving openFile()');
	  };
	</script>

onprogress / 'progress' event listener(event : ProgressEvent) : undefined	Called during a read operation to report the current progress.

	<input type='file' onchange='openFile(event)'>
	
	<script>
	  var openFile = function(event) {
	  console.log('entering openFile()');
	  var input = event.target;
	
	  var printEventType = function(event) {
	  console.log('got event: ' + event.type);
	  };
	
	  var reader = new FileReader();
	  reader.onloadstart = printEventType;
	  reader.onprogress = printEventType;
	  reader.onload = printEventType;
	  reader.onloadend = printEventType;
	
	  console.log(' starting read');
	  reader.readAsDataURL(input.files[0]);
	  console.log('leaving openFile()');
	  };

	</script>
onload / 'load' event listener(event : ProgressEvent) : undefined	Called when a read operation successfully completes.

	<input type='file' onchange='openFile(event)'>
	<script>
	  var openFile = function(event) {
	  console.log('entering openFile()');
	  var input = event.target;
	
	  var printEventType = function(event) {
	  console.log('got event: ' + event.type);
	  };
	
	  var reader = new FileReader();
	  reader.onloadstart = printEventType;
	  reader.onprogress = printEventType;
	  reader.onload = printEventType;
	  reader.onloadend = printEventType;
	
	  console.log(' starting read');
	  reader.readAsDataURL(input.files[0]);
	  console.log('leaving openFile()');
	  };
	</script>

onabort / 'abort' event listener(event : ProgressEvent) : undefined	Called when the read is aborted with `abort()`.

	<input type='file' onchange='openFile(event)'>
	<script>
	  var openFile = function(event) {
	  console.log('entering openFile()');
	  var input = event.target;
	
	  var printEventType = function(event) {
	  console.log('got event: ' + event.type);
	
	  if (event.type === 'loadstart') {
	  reader.abort();
	  }
	  };
	
	  var reader = new FileReader();
	  reader.onloadstart = printEventType;
	  reader.onprogress = printEventType;
	  reader.onload = printEventType;
	  reader.onloadend = printEventType;
	  reader.onabort = printEventType;
	  reader.onerror = printEventType;

	console.log(' starting read');
	reader.readAsDataURL(input.files[0]);
	console.log('leaving openFile()');
	};
	</script>

onerror / 'error' event listener(event : ProgressEvent) : undefined	Called when there is an error during the load.

	<input type='file' onchange='openFile(event)'>
	<script>
	var openFile = function(event) {
	console.log('entering openFile()');
	var input = event.target;
	
	var printEventType = function(event) {
	console.log('got event: ' + event.type);
	
	if (event.type === 'loadstart') {
	reader.abort();
	}
	};
	
	var reader = new FileReader();
	reader.onloadstart = printEventType;
	reader.onprogress = printEventType;
	reader.onload = printEventType;
	reader.onloadend = printEventType;
	reader.onabort = printEventType;
	reader.onerror = printEventType;
	
	console.log(' starting read');
	reader.readAsDataURL(input.files[0]);
	console.log('leaving openFile()');
	};
	</script>

onloadend / 'loadend' event listener(event : ProgressEvent) : undefined	Called after a read completes (either successfully or unsuccessfully).

	<input type='file' onchange='openFile(event)'>
	<script>
	var openFile = function(event) {
	console.log('entering openFile()');
	var input = event.target;
	
	var printEventType = function(event) {
	console.log('got event: ' + event.type);
	};

	var reader = new FileReader();
	reader.onloadstart = printEventType;
	reader.onprogress = printEventType;
	reader.onload = printEventType;
	reader.onloadend = printEventType;
	
	console.log(' starting read');
	reader.readAsDataURL(input.files[0]);
	console.log('leaving openFile()');
	};
	</script>

DONE : Number readonly value = 2	The value returned by `readyState` after the `load` event has fired.
EMPTY : Number readonly value = 0	The value returned by `readyState` before the one of the read methods has been called.
LOADING : Number readonly value = 1	The value returned by `readyState` after one of the read methods has been called but before the `load` event has fired.

URL

Provides methods to generate a url for a `Blob` so locally generated content can be passed to APIs that accept urls.

createObjectURL(blob : Blob) : String	Creates a url for the specified `blob` that can be passed to methods that expect a url. When done with the returned url, call `revokeObjectURL()` to free the resources associated with the created url.

	<!-- Creates a Worker using a local script instead of a remote url -->
	<script id='code' type='text/plain'>
	postMessage('foo');
	</script>
	<script>
	var code = document.getElementById('code').textContent;
	var blob = new Blob([code], { type: 'application/javascript' });
	var url = URL.createObjectURL(blob);
	var worker = new Worker(url);
	URL.revokeObjectURL(url);

	worker.onmessage = function(e) {
	  console.log('worker returned: ', e.data);
	 };
	</script>

revokeObjectURL(url : String) : undefined	Frees the resources associated with the `url` created by `createObjectURL()`.

	<!-- Creates a Worker using a local script instead of a remote url -->
	<script id='code' type='text/plain'>
	 postMessage('foo');
	</script>
	<script>
	 var code = document.getElementById('code').textContent;
	 var blob = new Blob([code], { type: 'application/javascript' });
	 var url = URL.createObjectURL(blob);
	 var worker = new Worker(url);
	 URL.revokeObjectURL(url);
	
	 worker.onmessage = function(e) {
	  console.log('worker returned: ', e.data);
	 };
	</script>

ArrayBuffers

ArrayBuffers are fixed length buffer of bytes. The bytes in an ArrayBuffer are only accessible through a `DataView` (for heterogenous data) or one of the typed arrays (for homogeneous data): `Float32Array`, `Float64Array`, `Int8Array`, `Int16Array`, `Int32Array`, `Uint8Array`, `Uint8ClampedArray`, `Uint16Array`, `Uint32Array`. Multiple DataView and typed arrays can be applied to one ArrayBuffer and changes to one view can be seen in the others immediately.

new ArrayBuffer(byteLength: Number):ArrayBuffer	Allocates a new `ArrayBuffer` of the specified length where each byte starts as `0`.

	var buffer = new ArrayBuffer(12);
	var dataView = new DataView(buffer);
	var int8View = new Int8Array(buffer);
	dataView.setInt32(0, 0x1234ABCD);
	console.log(dataView.getInt32(0).toString(16));
	console.log(dataView.getInt8(0).toString(16));
	console.log(int8View[0].toString(16));

byteLength:Number readonly	The length of `this` in bytes.
	var buffer = new ArrayBuffer(12);

	console.log(buffer.byteLength);
slice(beginByte:Number, [endByte:Number]):ArrayBuffer	Creates a new ArrayBuffer with a copy of the bytes of `this` between `beginByte` (inclusive) and `endByte` (exclusive). If `endByte` is not specified, `this.byteLength` is used. Changes to `this` do not affect the copy returned by `slice`.

	var buffer = new ArrayBuffer(12);
	var x = new Int32Array(buffer);
	x[1] = 1234;
	var slice = buffer.slice(4);
	var y = new Int32Array(slice);
	console.log(x[1]);
	console.log(y[0]);
	x[1] = 6789;
	console.log(x[1]);
	console.log(y[0]);

isView(value : Object) : Boolean	Returns `true` if `value` is an ArrayBufferView.

	console.log(ArrayBuffer.isView(new Int32Array()));
	console.log(ArrayBuffer.isView(new Float64Array()));
	console.log(ArrayBuffer.isView(new Array()));

DataView : ArrayBufferView

DataViews allow heterogeneous access to data stored in an `ArrayBuffer`. Values can be read and stored at any byte offset without alignment constraints.

new DataView(buffer : ArrayBuffer, [byteOffset = 0 : Number, [byteLength : Number]]) : DataView	Creates a new `DataView` for `buffer` at the specified offset. If `length` is not specified, `buffer.byteLength - byteOffset` will be used.

	var buffer = new ArrayBuffer(12);
	var x = new DataView(buffer, 0);
	x.setInt8(0, 22);
	x.setFloat32(1, Math.PI);
	console.log(x.getInt8(0));
	console.log(x.getFloat32(1));

buffer : ArrayBuffer	Returns the underlying buffer for `this`.

	var buffer = new ArrayBuffer(12);
	var x = new DataView(buffer);
	console.log(x.buffer === buffer);

byteLength : Number	The length of `this` in bytes.

	var buffer = new ArrayBuffer(12);
	var x = new DataView(buffer, 4, 2);
	console.log(x.byteLength);

byteOffset : Number	The offset into `this.buffer` where the view starts.

	var buffer = new ArrayBuffer(12);
	var x = new DataView(buffer, 4, 2);
	console.log(x.byteOffset);

getFloat32(byteOffset:Number, [littleEndian=false:Boolean]):Number	Returns a 32 bit floating point number out of `this` at the specified offset. If `littleEndian` is `true`, the value will be read as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 3`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setFloat32(1, Math.PI);
	console.log(x.getFloat32(1));
	console.log(Math.PI);

getFloat64(byteOffset : Number, [littleEndian = false : Boolean]) : Number	Returns a 64 bit floating point number out of `this` at the specified offset. If `littleEndian` is `true`, the value will be read as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 7`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setFloat64(1, Math.PI);
	console.log(x.getFloat64(1));
	console.log(Math.PI);

getInt16(byteOffset : Number, [littleEndian = false : Boolean]) : Number	Returns a signed 16 bit integer out of `this` at the specified offset. If `littleEndian` is `true`, the value will be read as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 1`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setInt16(1, 1234);
	console.log(x.getInt16(1));

getInt32(byteOffset : Number, [littleEndian = false : Boolean]) : Number	Returns a signed 32 bit integer out of `this` at the specified offset. If `littleEndian` is `true`, the value will be read as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 3`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setInt32(1, 1234);
	console.log(x.getInt32(1));

getInt8(byteOffset : Number) : Number	Returns a signed byte out of `this` at the specified offset.

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setInt8(1, 123);
	console.log(x.getInt8(1));

getUint16(byteOffset : Number, [littleEndian = false : Boolean]) : Number	Returns an unsigned 16 bit integer out of `this` at the specified offset. If `littleEndian` is `true`, the value will be read as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 1`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setUint16(1, 1234);
	console.log(x.getUint16(1));

getUint32(byteOffset : Number, [littleEndian = false : Boolean]) : Number	Returns an usigned 32 bit integer out of `this` at the specified offset. If `littleEndian` is `true`, the value will be read as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 3`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setUint32(1, 1234);
	console.log(x.getUint32(1));

getUint8(byteOffset : Number) : Number	Returns an unsigned byte out of `this` at the specified offset.

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setUint8(1, 123);
	console.log(x.getUint8(1));

setFloat32(byteOffset : Number, value : Number, [littleEndian = false : Boolean]) : undefined	Converts `value` to a 32 bit floating point number and stores it into `this` at the specified offset. If `littleEndian` is `true`, the value will be stored as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 3`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setFloat32(1, Math.PI); console.log(x.getFloat32(1));
	console.log(Math.PI);

setFloat64(byteOffset : Number, value : Number, [littleEndian = false : Boolean]) : undefined	Stores `value` as a 64 bit floating point number in `this` at the specified offset. If `littleEndian` is `true`, the value will be stored as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 7`).
	var x = new DataView(new ArrayBuffer(12), 0); x.setFloat64(1, Math.PI);

	console.log(x.getFloat64(1)); console.log(Math.PI);
setInt16(byteOffset : Number, value : Number, [littleEndian = false : Boolean]) : undefined	Stores a signed 16 bit integer into `this` at the specified offset. If `littleEndian` is `true`, the value will be stored as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 1`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setInt16(1, 1234);
	console.log(x.getInt16(1));

setInt32(byteOffset : Number, value : Number, [littleEndian = false : Boolean]) : undefined	Stores a signed 32 bit integer into `this` at the specified offset. If `littleEndian` is `true`, the value will be stored as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 3`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setInt32(1, 1234);
	console.log(x.getInt32(1));

setInt8(byteOffset : Number, value : Number) : undefined	Stores a signed byte into `this` at the specified offset.

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setInt8(1, 123);
	console.log(x.getInt8(1));

setUint16(byteOffset : Number, value : Number, [littleEndian = false : Boolean]) : undefined	Stores an unsigned 16 bit integer into `this` at the specified offset. If `littleEndian` is `true`, the value will be stored as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 1`).

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setUint16(1, 1234);
	console.log(x.getUint16(1));

setUint32(byteOffset : Number, value : Number, [littleEndian = false : Boolean]) : undefined	Stores an unsigned 32 bit integer into `this` at the specified offset. If `littleEndian` is `true`, the value will be stored as little endian (least significant byte is at `byteOffset` and most significant at `byteOffset + 3`).
	var x = new DataView(new ArrayBuffer(12), 0); x.setUint32(1, 1234); console.log(x.getUint32(1));
setUint8(byteOffset : Number, value : Number) : undefined	Stores an unsigned byte into `this` at the specified offset.

	var x = new DataView(new ArrayBuffer(12), 0);
	x.setUint8(1, 123);
	console.log(x.getUint8(1));