

More on JavaScript

JavaScript Frameworks

What is a framework?

- A framework is an application structure written in JavaScript. It describes “how you should present your code”.

e.g. Angular-s Angular.js->Angular2->Angular4,

How it differs from a JavaScript library?

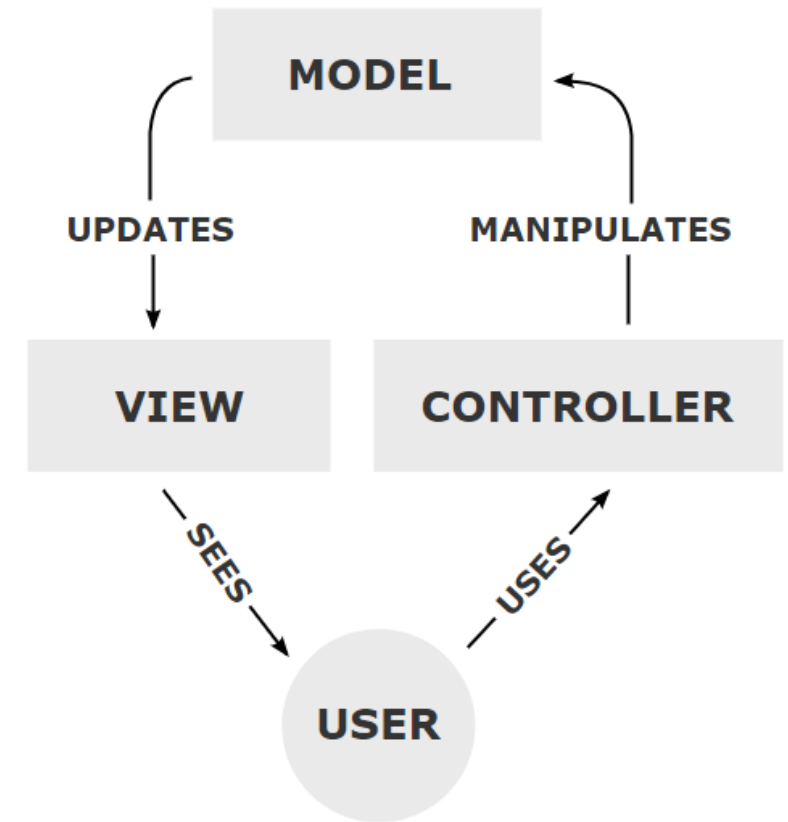
- A library has predefined functions to be called by its parent code whereas a framework defines the entire application design.

e.g. jquery, ReactJS

They are devised because JavaScript functions are too many and too hard to remember.

Frameworks *mostly* are based on MVC paradigm

- View- how the user interacts.
- Model- stores the data that is used by your application in SQLite database, HTML5 local storage...
- Controller- has an Application Programming interface to modify the underlying data.



Documents, Events and Interfaces

- DOM Specifications
 - Describes the document structure, manipulates and events.
- CSSOM Specifications
 - Describes the stylesheets, style rules, manipulations with them.
- HTML Specifications
 - Describes HTML language and also BOM.

Events

- An event is a signal that something has happened.
- All DOM nodes generate such signals (but events are not limited to DOM).

Mouse events:

- **click** – when a mouse clicks on an element (tap for touchscreen devices).
- **contextmenu** – when the mouse right-clicks on an element.
- **mouseover** / **mouseout** – when the mouse cursor comes over / leaves an element.
- **mousedown** / **mouseup** – when the mouse button is pressed / released over an element.
- **mousemove** – when the mouse is moved.

Events

Form element events:

- submit – when the visitor submits a `<form>`.
- focus – when the visitor focuses on an element, e.g. on an `<input>`.

Keyboard events:

- keydown and keyup – when the visitor presses and then releases the button.

Document events

- DOMContentLoaded – when the HTML is loaded and processed, DOM is fully built.

CSS events:

- transitionend – when a CSS-animation finishes.

Event Handlers

- To react to an event, we assign an event handler.

An event handler is a function that is run in case of an event.

How to assign a handler?

One of the simplest- on attributes

A handler can be set in HTML with an attribute named on<event>

For instance, to assign a click handler for an input, we can use onclick:

```
<input value="Click me" onclick="alert('Click!')" type="button">
```

Writing Functions to be called on events

- An HTML-attribute is not a right place to write a lot of code, so we'd better create a JavaScript function and call it there.
- Here a click runs the function countPeople():

```
<script>
function countPeople() {
  for(let i=1; i<=3; i++) {
    alert("People number " + i);
  }
}
</script>
<input type="button" onclick="countPeople()" value="Count People!">
```


document.getElementById

```
<section id="elem">
```

```
  <article id="elem-content">Element</article>
```

```
</article>
```

```
</section>
```

```
<script>
```

```
  alert(elem);
```

```
  alert(window.elem);
```

```
  alert(window['elem-content']);
```

```
</script>
```

Better approach is to use: `getElementById`

```
<section id="elem">
```

```
  <article id="elem-content">Element</article>
```

```
</section>
```

```
<script>
```

```
  alert(getElementById(elem));
```

```
</script>
```

Other methods to look for nodes:

- `getElementsByTagName(tag)`

```
<table id="table">
  <tr>
    <td>Your age:</td>
    <td>
      <label>
        <input type="radio" name="age" value="young" checked> less than 18
      </label>
      <label>
        <input type="radio" name="age" value="mature"> from 18 to 50
      </label>
      <label>
        <input type="radio" name="age" value="senior"> more than 60
      </label>
    </td>
  </tr>
</table>
<script>
  let inputs = table.getElementsByTagName('input');
  for (let input of inputs) {
    alert( input.value + ': ' + input.checked );
  }
</script>
```

getElementsByTagName

- It returns a collection, not an element!
- `//` doesn't work

`document.getElementsByTagName('input').value = 5;`

- That won't work, because it takes a collection of inputs and assigns the value to it, rather to elements inside it.
- We should either iterate over the collection or get an element by the number, and then assign, like this:
- `//` should work (if there's an input)

`document.getElementsByTagName('input')[0].value = 5;`

document.getElementsByName and getElementsByClassName

- `<form name="my-form">`
- `<div class="article">Article</div>`
- `<div class="long article">Long article</div>`
- `</form>`

- `<script>`
- `// find by name attribute`
- `let form = document.getElementsByName('my-form')[0];`

- `// find by class inside the form`
- `let articles = form.getElementsByClassName('article');`
- `alert(articles.length); // 2, found two elements with class "article"`
- `</script>`

querySelectorAll

- The call to `elem.querySelectorAll(css)` returns all elements inside `elem` matching the given CSS selector.
- ``
- `The`
- `test`
- ``
- ``
- `has`
- `passed`
- ``
- **`<script>`**
- **`let elements = document.querySelectorAll('ul > li:last-child');`**
- **`for (let elem of elements) {`**
- **`alert(elem.innerHTML); // "test", "passed"`**
- **`}`**
- **`</script>`**

querySelector

- The call to `elem.querySelector(css)` returns the first element for the given CSS selector.
- In other words, the result is the same as **`elem.querySelectorAll(css)[0]`**, but the latter is looking for all elements and picking one, while `elem.querySelector` just looks for one. So it's faster and shorter to write.

matches

```
<a href="http://anyfile.com">...</a>
<a href="http://ya.ru">...</a>
<script>
  // can be any collection instead of
  document.body.children
  for (let elem of document.body.children) {
    if (elem.matches('a[href$="zip"]')) {
      alert("The archive reference: " +
        elem.href );
    }
  }
</script>
```

- Previous methods were searching the DOM.
- 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 handy when we are iterating over elements (like in array or something) and trying to filter those that interest us.

closest

- Ancestors 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.

closest

- `<h1>Contents</h1>`
- `<div class="contents">`
- `<ul class="book">`
- `<li class="chapter">Chapter 1`
- `<li class="chapter">Chapter 1`
- ``
- `</div>`
- **`<script>`**
- **`let chapter = document.querySelector('.chapter'); // LI`**
- **`alert(chapter.closest('.book')); // UL`**
- **`alert(chapter.closest('.contents')); // DIV`**
- **`alert(chapter.closest('h1')); // null (because h1 is not an ancestor)`**
- **`</script>`**

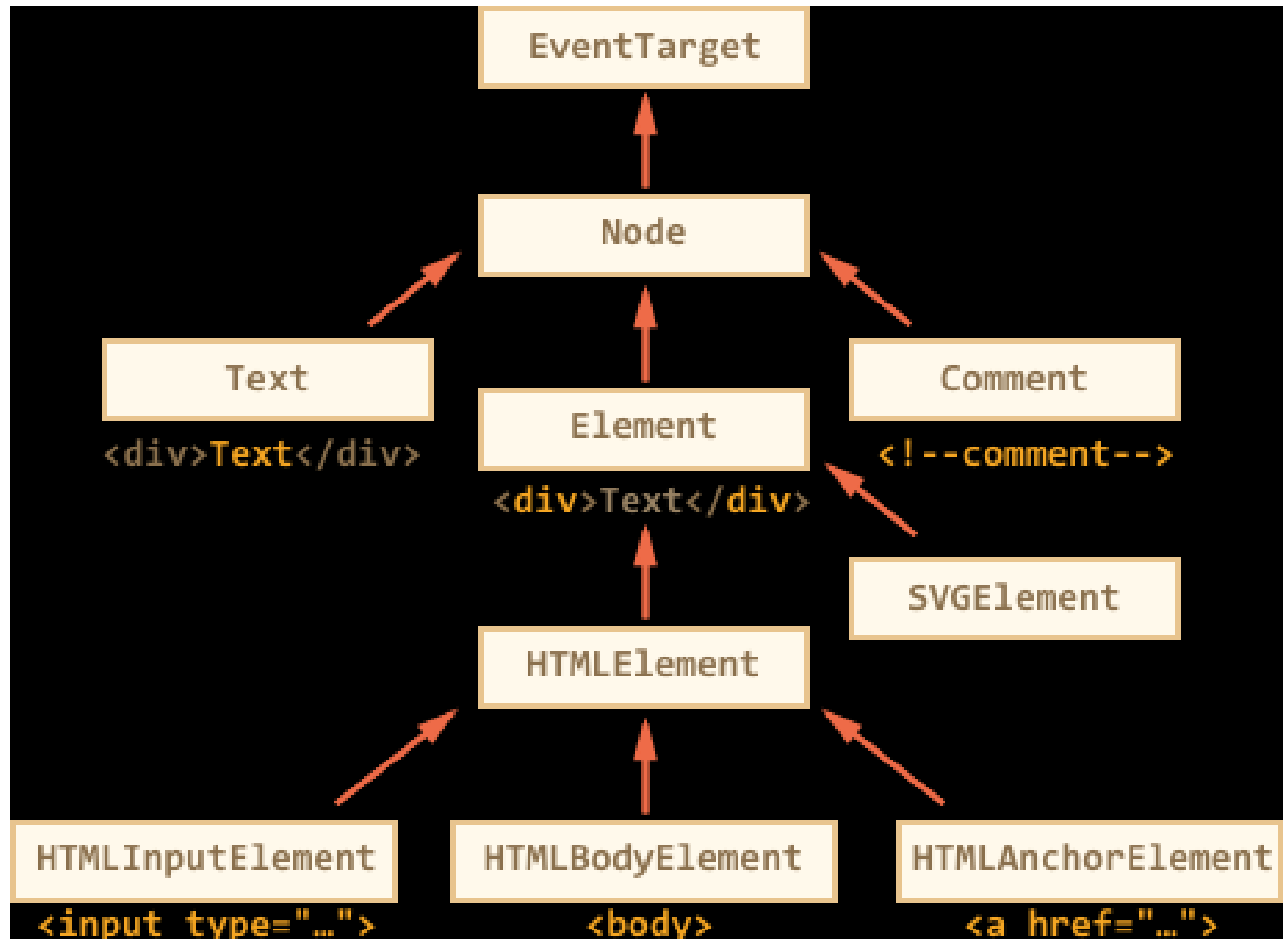
One more method here to check for the child-parent relationship: `elemA.contains(elemB)` returns true if `elemB` is inside `elemA` (a descendant of `elemA`) or when `elemA==elemB`.

getElementsByTagName

- All methods “getElementsByTagName” return the current state of document and auto-updates with the changes.

Method	Searches by...
getElementById	id
getElementsByTagName	name
getElementsByTagName	tag or '*'
getElementsByTagName	class
querySelector	CSS-selector
querySelectorAll	CSS-selector

Node Properties: type, tag and contents



Node Properties

- Lets Consider the DOM object for an <input> element. It belongs to HTMLInputElement class. It gets properties and methods as a superposition of:
 - **HTMLInputElement** – this class provides input-specific properties, and inherits from...
 - **HTMLElement** – it provides common HTML element methods (and getters/setters) and inherits from...
 - **Element** – provides generic element methods and inherits from...
 - **Node** – provides common DOM node properties and inherits from...
 - **EventTarget** – gives the support for events (to be covered),
 - ...and finally it inherits from Object, so “pure object” methods like hasOwnProperty are also available.

DOM node class name

- To see the DOM node class name, we can reference to the class constructor, and `constructor.name` is its name:

```
alert( document.body.constructor.name ); // HTMLBodyElement
```

...Or we can just `toString` it:

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

We can also use `instanceof` to check the inheritance.

```
alert( document.body instanceof HTMLBodyElement ); // true
```

```
alert( document.body instanceof HTMLElement ); // true
```

```
alert( document.body instanceof Element ); // true
```

```
alert( document.body instanceof Node ); // true
```

```
alert( document.body instanceof EventTarget ); // true
```

innerHTML: the contents

The innerHTML property allows to get the HTML inside the element as a string.

The example shows the contents of document.body and then replaces it completely:

```
<body> <p>A paragraph</p>  
  <div>A div</div>  
  <script>  
    alert( document.body.innerHTML ); // read the  
current contents  
    document.body.innerHTML = 'The new BODY!';  
// replace it  
  </script> </body>
```

textContent

- The `textContent` provides access to the text inside the element: only text, minus all `<tags>`.
- As we can see, only text is returned, as if all `<tags>` were cut out, but the text in them remained.
- In practice, reading such text is rarely needed.
- Writing to `textContent` is much more useful, because it allows to write text the “safe way”.
- Let’s say we have an arbitrary string, for instance entered by a user, and want to show it.
- With **`innerHTML`** we’ll have it inserted “as HTML”, with all HTML tags.
- With **`textContent`** we’ll have it inserted “as text”, all symbols are treated literally.

The hidden property

- The “hidden” attribute and the DOM property specifies whether the element is visible or not.
- We can use it in HTML or assign using JavaScript, like this:

```
<div>Both divs below are hidden</div>  
<div hidden>With the attribute "hidden"</div>  
<div id="elem">JavaScript assigned the property "hidden"</div>  
<script> elem.hidden = true; </script>
```


Attributes and Properties

- DOM nodes are regular JavaScript objects. We can alter them.
- Attributes – is what's written in HTML.
- Properties – is what's in DOM objects.

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.

Attributes

- Attributes are in the HTML itself, rather than in the DOM. They are very similar to properties, but not quite as good. When a property is available it's recommended that you work with properties rather than attributes.
- An attribute is only ever a string, no other type.
- Their name is case-insensitive (that's HTML: id is same as ID).

```
<body>
  <div id="elem" about="Elephant"></div>
  <script>
    alert( elem.getAttribute('About') ); // (1) 'Elephant', reading
    elem.setAttribute('Test', 123);      // (2), writing
    alert( elem.outerHTML );             // (3), see it's there
    for (let attr of elem.attributes) {  // (4) list all
      alert( attr.name + " = " + attr.value );
    }
  </script>
</body>
```

DOM Properties

- DOM properties are not always strings. For instance, `input.checked` property (for checkboxes) is boolean:

```
<input id="input" type="checkbox" checked> checkbox  
<script>
```

```
  alert(input.getAttribute('checked')); // the attribute value is:  
  empty string
```

```
  alert(input.checked); // the property value is: true  
</script>
```

```
<div id="div" style="color:red;font-size:120%">Hello</div>  
<script>  
  // string  
  alert(div.getAttribute('style')); // color:red;font-size:120%  
  // object  
  alert(div.style); // [object CSSStyleDeclaration]  
  alert(div.style.color); // red  
</script>
```

Custom-made attributes, dataset

- `<!-- mark the div to show "name" here -->`
- `<div show-info="name"></div>`
- `<!-- and age here -->`
- `<div show-info="age"></div>`
- `<script>`
- `// the code finds an element with the mark and shows what's requested`
- `let user = {`
- `name: "youName",`
- `age: 20`
- `};`
- `for(let div of document.querySelectorAll('[show-info])) {`
- `// insert the corresponding info into the field`
- `let field = div.getAttribute('show-info');`
- `div.innerHTML = user[field]; // yourName, then age`
- `}`
- `</script>`

data-

- All attributes starting with "data-" are reserved for programmers' use. They are available in dataset property.
- For instance, if an elem has an attribute named "data-about", it's available as elem.dataset.about.
- `<body data-about="Elephants">`
- `<script>`
- `alert(document.body.dataset.about); // Elephants`
- `</script>`

Non-standard

```
<style>
  .order[order-state="new"] {
    color: green;
  }
  .order[order-state="pending"] {
    color: blue;
  }
  .order[order-state="canceled"] {
    color: red;
  } </style>
<div class="order" order-state="new">  A new order.
</div>
<div class="order" order-state="pending">  A pending
order.
</div>
<div class="order" order-state="canceled">  A
canceled order.
</div>
```

Use of Data with non-standard

```
<style>
  .order[data-order-state="new"] {
    color: green;
  }
  .order[data-order-state="pending"] {
    color: blue;
  }
  .order[data-order-state="canceled"] {
    color: red;
  } </style>
<div id="order" class="order" data-order-
state="new">
  A new order.
</div>
<script>  // read
  alert(order.dataset.orderState); // new
  // modify
  order.dataset.orderState = "pending"; // (*)
</script>
```

Creating an element

- There are two methods:

```
document.createElement(tag)
```

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

Creating a text node:

```
let textNode = document.createTextNode('Here I am');
```

Creating the message:

```
let div = document.createElement('div');  
div.className = "alert alert-success";  
div.innerHTML = "<strong>Hi there!</strong> You've read  
an important message.";
```

Insertion methods

- To make the div show up, we need to insert it somewhere into document. For instance, in document.body.
- There's a special method for that: document.body.appendChild(div).

```
<style>
.alert {
  padding: 15px;
  border: 1px solid #d6e9c6;
  border-radius: 4px;
  color: #3c763d;
  background-color: #dff0d8;
}
</style>
<script>
  let div = document.createElement('div');
  div.className = "alert alert-success";
  div.innerHTML = "<strong>Hi
there!</strong> You've read an important
message.";
  document.body.appendChild(div);
</script>
```

```
<ol id="list">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>

<script>
  let newLi =
document.createElement('li');
  newLi.innerHTML = 'Hello, world!';

  list.appendChild(newLi);
</script>
```