DOM events & Building front-ends

Maher Khan

Recitation 3
Date: 21st September, 2018

Table of Contents

DOM events

onload	when element is fully loaded
onclick	when element is clicked
onsubmit	when element is submitted
onmousehover	when mouse is on top of element
onkeydown	when a key is pressed while element is in focus

more: https://www.w3schools.com/jsref/dom_obj_event.asp

Listening and dispatching events

- Listen for the event

```
element.addEventListener(‘onSomething’, function(e){
        ...
  });
```

- Dispatch the event

```
element.dispatchEvent(new Event(‘onSomething'));
```

- Check example_1.html

Bubbling and capturing

Let’s start with an example.

```
<div onclick="alert('The handler!')">
    <em>If you click on <code>EM</code>, the handler on
        <code>DIV</code> runs.
    </em>
</div>
```

Isn’t it a bit strange? Why the handler on runs if the actual click was on ?

Bubbling

- Principle:

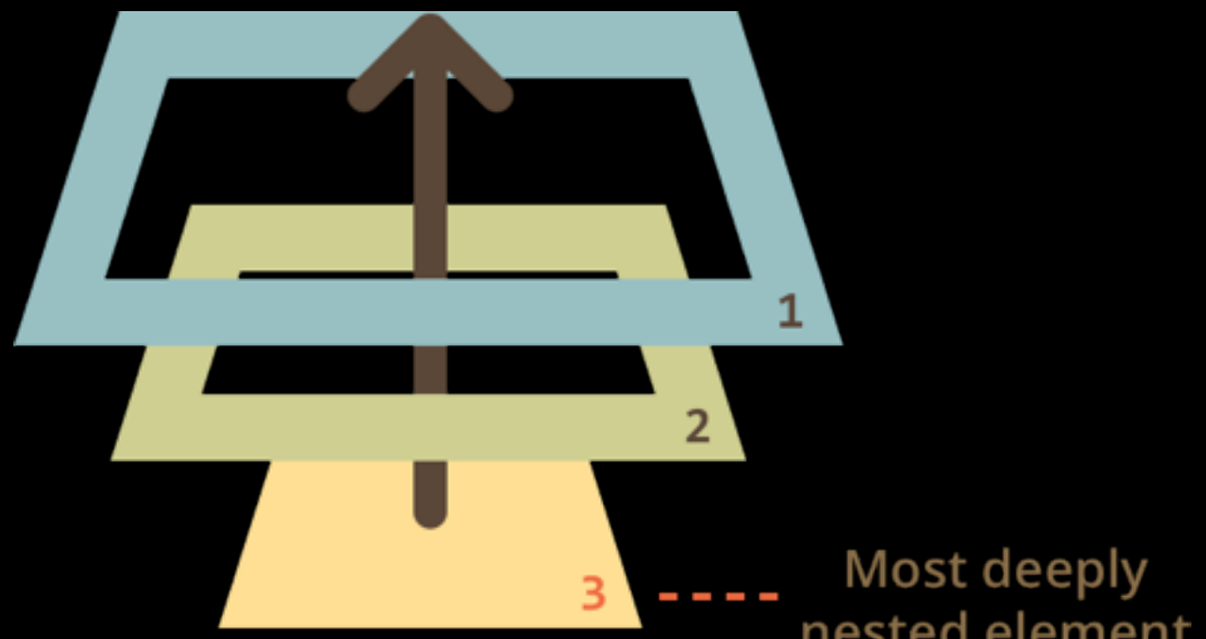
  When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.

- Check example_2.html

More about example_2.html

A click on the inner <p>
first runs onclick:

1. On that <p>.
2. Then on the outer .
3. Then on the outer <form>.
4. And so on upwards till the document object.



event.target vs this

- **event.target** – is the “target” element that initiated the event, it doesn’t change through the bubbling process.
- **this** – is the “current” element, the one that has a currently running handler on it.

Check example_3.html

- If we have a single handler **form.onclick**, then it can “catch” all clicks inside the form. No matter where the click happened, it bubbles up to <form> and runs the handler.

- In form.onclick handler:
  - **this** (=event.currentTarget) is the <form> element, because the handler runs on it.
  - **event.target** is the concrete element inside the form that actually was clicked.

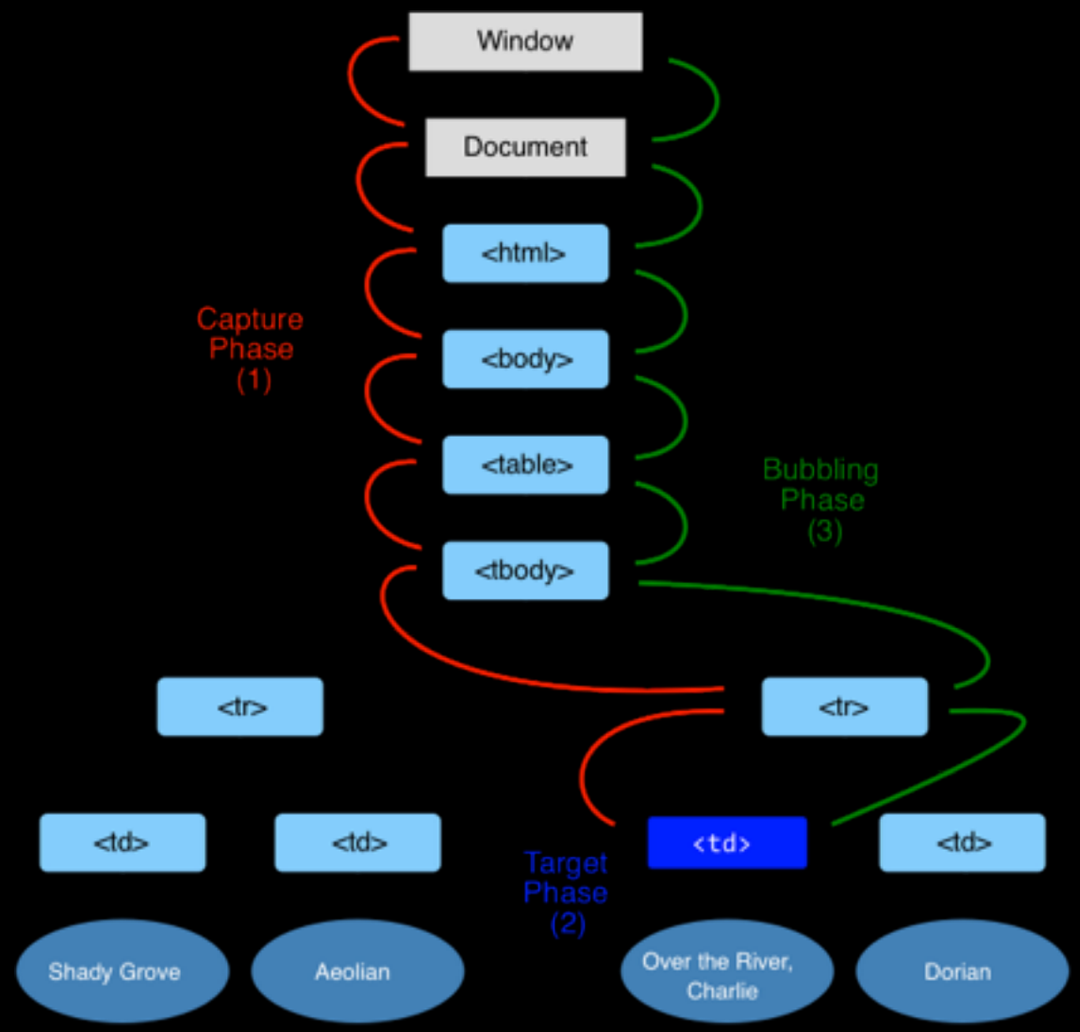
Stop bubbling!

- A bubbling event goes from the target element straight up.
- Normally it goes upwards till <html>, and then to document object.
- Sometimes this may cause problems! How?
- To stop bubbling:

  event.stopPropagation()
- Check example_4.html

Capturing vs bubbling

- **Capturing phase** – the event goes down to the element.
- Target phase – the event reached the target element.
- **Bubbling phase** – the event bubbles up from the element.



How to capture?

- To catch an event on the capturing phase, we need to set the 3rd argument of **addEventListener** to true.

element.addEventListener(‘onSomething’, *function*, **true**)

Let’s check example_5.html

- The code sets click handlers on *every* element in the document to see which ones are working.

- If you click on <p>, then the sequence is:
  1. capturing:
     HTML → BODY → FORM → DIV → P
  2. bubbling:
     P → DIV → FORM → BODY → HTML

Building front-ends

Recipes to become a good front-end developer

- Load Javascript code efficiently
- Ensure the DOM is loaded properly
- Write good Javascript code
- Encapsulate Javascript in closures
- Create a Frontend API

The problem with Javascript interpreters

Good Javascript is interpreted by browsers in a consistent way

Bad javascript code is loosely interpreted by browsers in an inconsistent way

Force the browser to validate Javascript against the standard:

```
"use strict";
var doSomething = function() {
    // this runs in strict mode
}
```

Dynamically raises errors (or warnings) in the console when the code is not compliant with the standard

Problem with DOM not loading properly

Execute code when all DOM content is loaded including fonts, images, etc:

```
window.addEventListener('load', function() {
    console.log('All assets are loaded')
})
```

The problem with scoping

Solution : encapsulate Javascript in a closure

```
(function() {
        "use strict";
        var private_function = function() {
            // private is not available from outside
        }
}());
```

Better solution

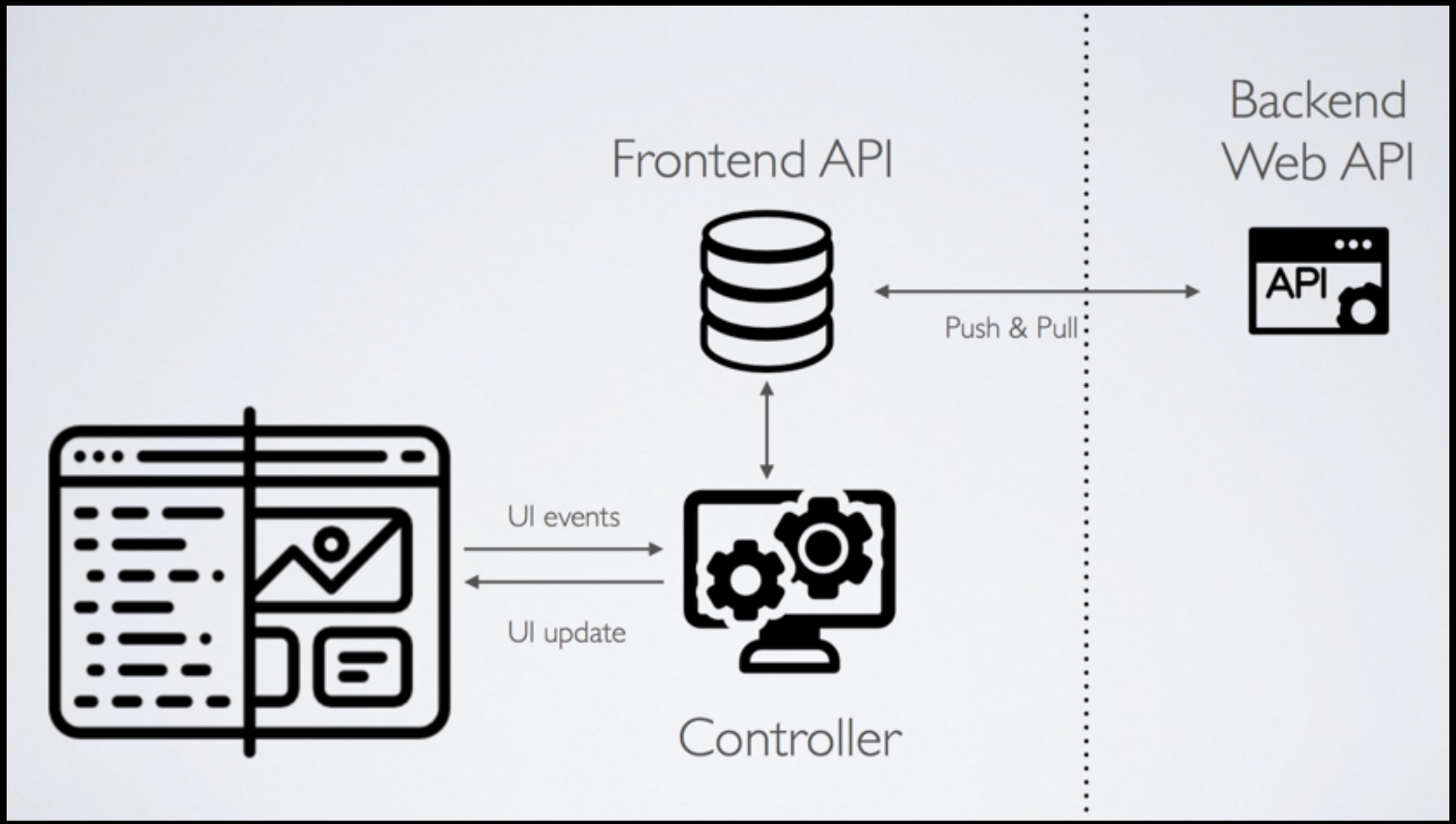
Solution : encapsulate and **export the namespace**

```
API = (function() {
    "use strict";

    var export_list = {};

    var private_function = function(){
        // private is not available from outside
    }

    export_list.public_function = function(){
        // public is available from outside
    }

    return export_list;
}());
```

Placing all these together…

```
window.addEventListener('load', function() {

    console.log('All assets are loaded')

    API = (function() {
        "use strict";

        var export_list = {};

        var private_function = function(){
            // private is not available from outside
        }

        export_list.public_function = function(){
            // public is available from outside
        }

        return export_list;
    }());
})
```

Structuring the front-end



That’s all! Questions?