



# Web Programming

**Lecturer:** Ung Văn Giàu  
**Email:** [giau.ung@eiu.edu.vn](mailto:giau.ung@eiu.edu.vn)



# Event Model

Touch, Mouse, Keys

# Contents

01

**JavaScript Event Model**

02

**Event Registration**

03

**The Event Object**

04

**Capturing and Bubbling Events**

05

**JSON**

06

**Fetch API**

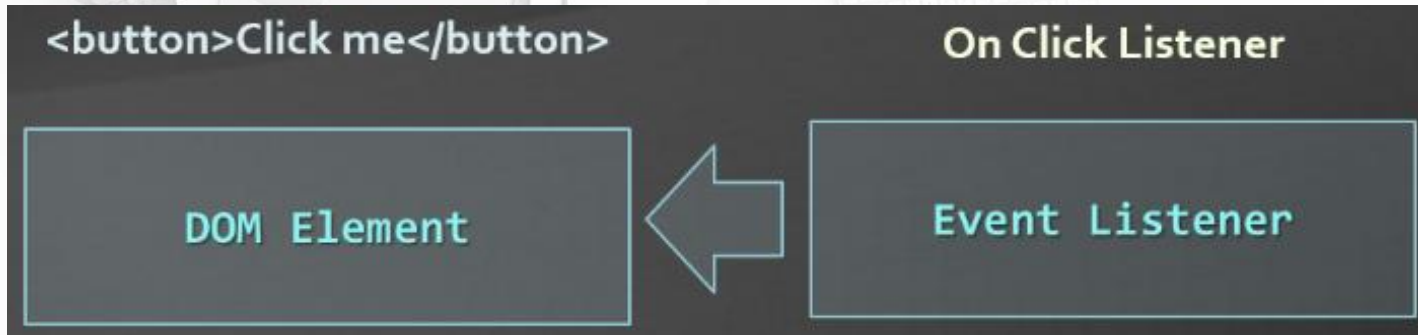




# 1. JavaScript Event Model

# JavaScript Event Model

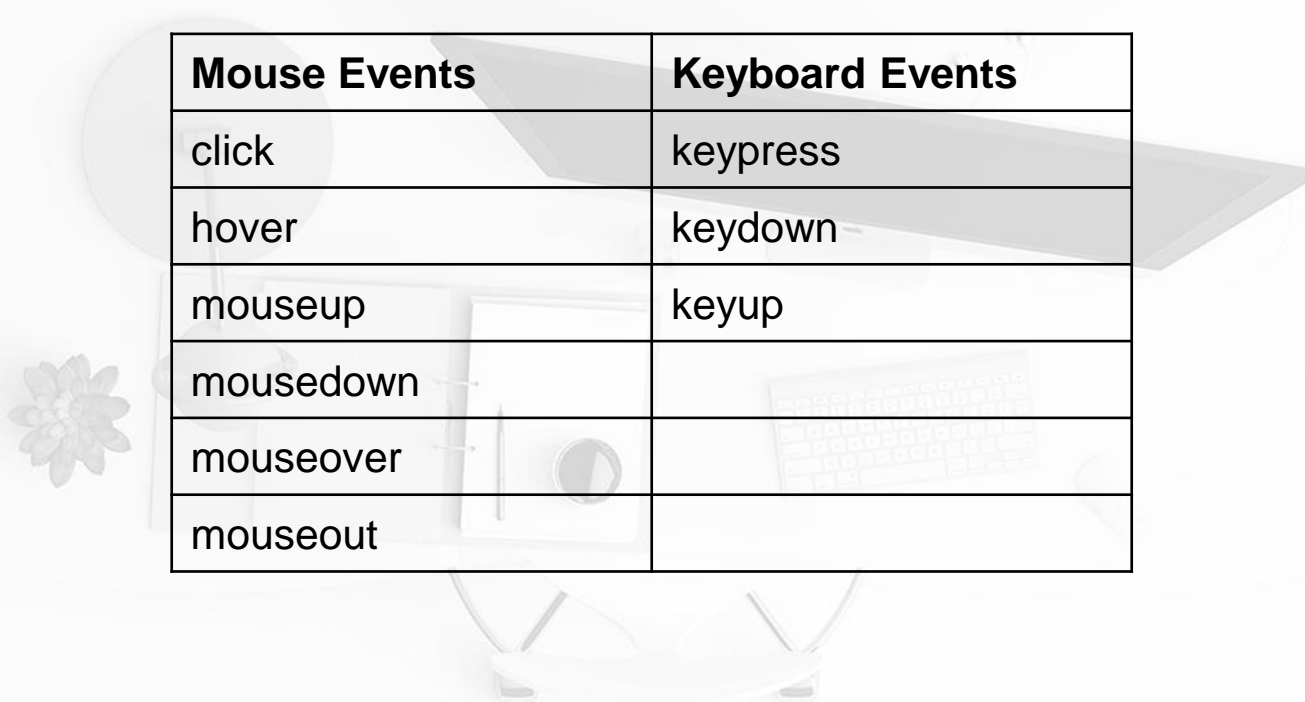
- The DOM event model **provides a way for the user to interact with the browser environment**
- The DOM event model **consists of events and event listeners** attached to the DOM objects



# Event Types

- 
- DOM provides a set of common event types that are used in 99% of the time
    - Mouse events
    - Keyboard events
    - UI events
    - Input events
    - Focus events
  - Full list of all DOM event types:  
<http://www.w3.org/TR/DOM-Level-3-Events/#event-types-list>
  - You could also define Custom Event Types

# Common Event Types



Mouse Events	Keyboard Events
click	keypress
hover	keydown
mouseup	keyup
mousedown	
mouseover	
mouseout	

# Common Event Types

UI Events	Focus Events
load	blur
abort	focus
select	focusin
resize	focusout
change	
input	





## 2. Event Registration

# Event Handlers

- The developer could register an event handler/listener for a specific event type and DOM element
- The registration can be performed with:
  - **HTML Attributes:** onEventName = “JavaScript code”
  - Using **DOM** element **properties**
  - Using DOM event **handler**

# As HTML Attribute

Event handlers can be attached by simply setting a value to the handler attribute

This value is pure JavaScript and is not always a function

## HTML

```
<button onclick="buttonClickFunction()">Click Me</button>
```

## JavaScript

```
function buttonClickFunction() {  
    console.log("You click the Button");  
}
```

# As HTML Attribute

## Form Validation

```
<form name="myForm" onsubmit="return validateForm()" method="get">
  <input type="text" name="fname" id="fname" placeholder="Enter your name">
  <input type="submit" value="Submit">
</form>
```

```
<script type="text/javascript">
  function validateForm() {
    let fname = document.getElementById("fname").value;
    // let fname = document.forms["myForm"]["fname"].value;
    if (fname == "") {
      alert("Name must be filled out");
      return false;
    }
    return true;
  }
</script>
```

# Preview an image using JavaScript

```
<input type="file" accept="image/*" onchange="loadFile(event)">
<img id="output"/>

<script>
    var loadFile = function(event) {
        var output = document.getElementById('output');
        output.src = URL.createObjectURL(event.target.files[0]);
    };
</script>
```

# Using DOM Element Properties

Use standard DOM events on certain DOM element and assign a reference to a function

Can be anonymous

**HTML:**      `<button id="click-button">Click me</button>`

**JavaScript:**

```
var button = document.getElementById("click-button");
button.onclick = function onButtonClick() {
    console.log("You clicked the button");
}
```

# Using DOM Event Listeners

The standard way for attaching event handlers to DOM

- The Basic Syntax is:

```
domElement.addEventListener(eventType,  
                             eventHandler,  
                             isCaptureEvent)
```

- Example:

```
var button = document.getElementById("click-button");  
button.addEventListener("click", function () {  
    console.log("You clicked me");  
}, false);
```



# 3. The Event Object

Get the Event Data



# Event Object

- The event handlers have access to the event object passed as function parameter
- The event object contains information about:
  - The **type** of the event
  - The **target** of the event
  - The **key that was pressed** when a keyboard event was fired
  - The **mouse button that was pressed** when a mouse event was fired
  - The **position** of the mouse on the screen

# Event Object

- The event object is accessible as the only argument of the function handler

```
function onClick(event) {  
    console.log(event.target);  
    console.log(event.type);  
    console.log(event.clientX, event.clientY);  
}  
button.addEventListener("click", onClick, false);
```

- Yet, there is IE - it does not pass event object
  - Keeps the event object in window.event
  - Fortunately, there is a simple fix

```
function onClick(event) {  
    if(!event) event = window.event;  
    // Your code...  
}
```

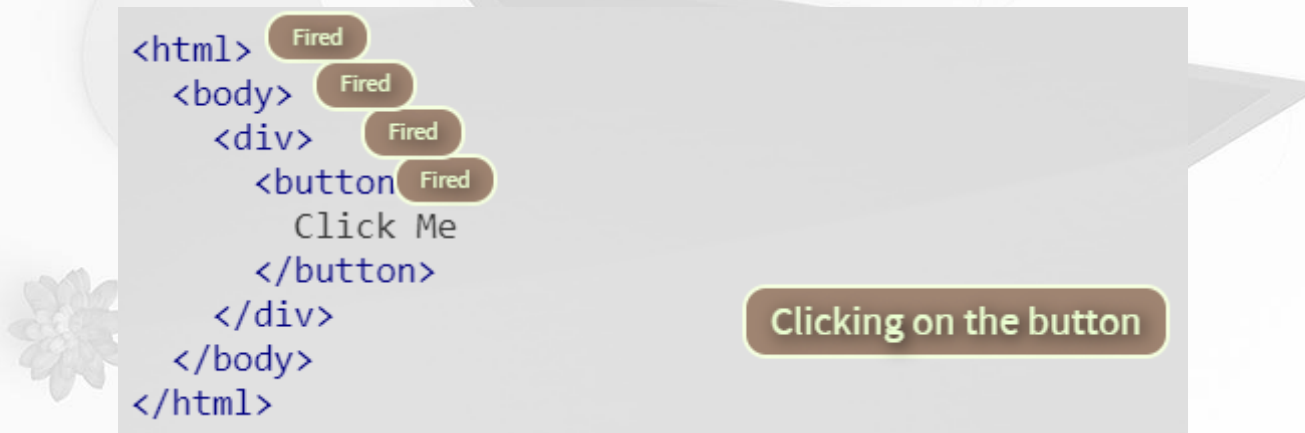


## 4. Capturing and Bubbling Events

Top to Bottom and the other way around

# The Event Chain

- When the user clicks on an HTML element, the event is also fired on all of its parents



```
<html>
  <body>
    <div>
      <button>
        Click Me
      </button>
    </div>
  </body>
</html>
```

The diagram illustrates the event chain for a click on a button. The HTML structure is shown in a light blue box. The button contains the text "Click Me". Four brown oval labels with the word "Fired" are positioned above the HTML tags: one above `<html>`, one above `<body>`, one above `<div>`, and one above `<button>`. This indicates that the event bubbles up from the button to each of its parent elements. A large, light gray arrow points from the button towards the top of the page, representing the direction of the event propagation.

Clicking on the button

- The **button is still the target**, but the click event is fired on all of its parents  
An event is fired on all elements in the chain

# Two Types of Event Chains

- There are two types of event chains

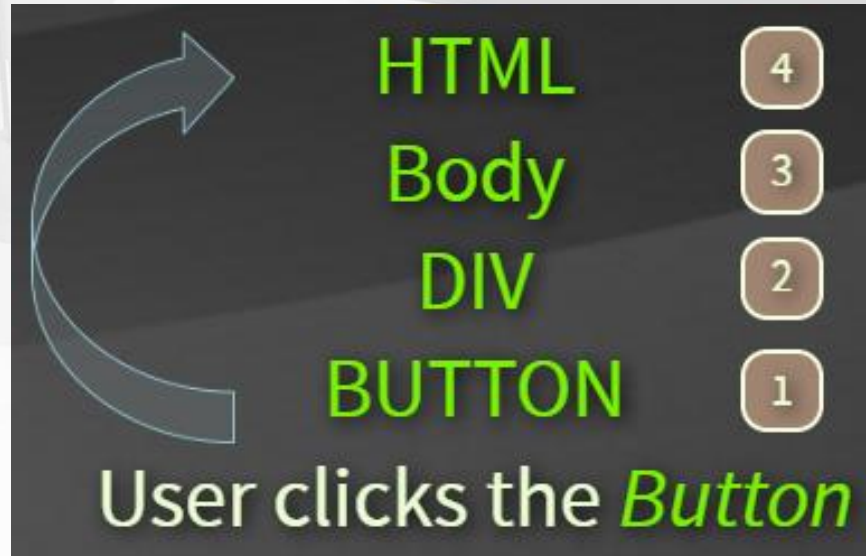
Capturing and Bubbling

- **Bubbling** handlers **bubble up** the chain
  - The first executed handler is on the target
  - Then its parent's, and its parent's, etc.
- **Capturing** handlers **go down** the chain
  - The first executed handler is on the parent of all
  - The last executed handler is on the target

# Bubbling Event Chain

Bubbling bubbles up the event chain

The first executed handler is the one on the target



# Capturing Event Chain

Capturing goes down the event chain

The first executed handler is the one of the parent of all





## 5. JSON



# JSON

- JSON is a format for storing and transporting data.
- JSON is often used when data is sent from a server to a web page.
- JSON stands for JavaScript Object Notation
- JSON is a lightweight data interchange format
- JSON is language independent \*
- JSON is “self-describing” and easy to understand

# JSON

- JSON Example:

This JSON syntax defines an employees object: an array of 3 employee records (objects)

```
{  
  "employees": [  
    {"firstName": "John", "lastName": "Doe"},  
    {"firstName": "Anna", "lastName": "Smith"},  
    {"firstName": "Peter", "lastName": "Jones"}  
  ]  
}
```

# JSON Syntax Rules

- Data is in name/value pairs (JSON names require double quotes)

`"firstName": "John"`

- Data is separated by commas
- Curly braces hold objects

`{"firstName": "John", "lastName": "Doe"}`

- Square brackets hold arrays

# Converting a JSON Text to a JavaScript Object

Use the JavaScript built-in function **JSON.parse()** to convert the string into a JavaScript object.

```
const obj = JSON.parse(jsonString);
```

Example:

```
var obj = JSON.parse('{ "name": "John", "age": 30, "city": "New York" }');
```



## 6. Fetch API

# Fetch API

- The Fetch API provides a JavaScript interface for making HTTP requests and processing the responses.
- With the Fetch API, you make a request by calling **fetch()**, which is available as a global function

# fetch() method

## ▪ Syntax

- `fetch(resource)`
- `fetch(resource, options)`

## ▪ Parameters

- Resource: defines the resource that you wish to fetch
  - ✓ A **string** containing the or any other object with a stringifier — including a URL object
  - ✓ A **Request** object
- Options (optional):
  - A **RequestInit** object containing any custom settings that you want to apply to the request.

## ▪ Return value

A **Promise** that resolves to a **Response** object.

# fetch() method

Synchronous fetch

Here's a minimal function that uses fetch() to retrieve some JSON data from a server:

```
function getData() {  
  fetch("https://example.org/products.json")  
    .then((response) => {  
    if(!response.ok) {  
      throw new Error(`Response status: ${response.status}`);  
    }  
    return response.json();  
  })  
    .then((data) => {  
    // Your code  
  })  
}
```



# fetch() method

Asynchronous fetch

Here's a minimal function that uses fetch() to retrieve some JSON data from a server:

```
async function getData() {  
  const url = "https://example.org/products.json";  
  try {  
    const response = await fetch(url);  
    if (!response.ok) {  
      throw new Error(`Response status: ${response.status}`);  
    }  
  
    const json = await response.json();  
    console.log(json);  
  } catch (error) {  
    console.error(error.message);  
  }  
}
```

# fetch() method

Setting the method

By default, `fetch()` makes a GET request, but you can use the `method` option to use a different request method:

```
const response = await fetch("https://example.org/post", {  
  method: "POST",  
  // ...  
});
```

# fetch() method

Setting a body

- The request **body** is the **payload of the request**: it's the thing the client is sending to the server.
- You cannot include a body with GET requests, but it's useful for requests that send content to the server, such as **POST** or **PUT** requests.
- To set a request body, pass it as the body option:

```
const response = await fetch("https://example.org/post", {  
  body: JSON.stringify({ username: "example" }),  
  // ...  
});
```

# fetch() method

Setting a body

- You can supply the body as an instance of any of the following types:
  - a string
  - ArrayBuffer
  - TypedArray
  - DataView
  - Blob
  - File
  - URLSearchParams
  - FormData

# fetch() method

## Setting headers

- **Request headers** give the server information about the request: for example, the Content-Type header tells the server the format of the request's body.
- To set request headers, assign them to the headers option.

```
const response = await fetch("https://example.org/post", {  
  headers: {  
    "Content-Type": "application/json",  
  },  
  // ...  
});
```

# fetch() method

Post request example

```
fetch(  
  "https://example.com/getdata.php",  
  {  
    method: 'POST',  
    headers: {  
      'Content-Type': 'application/x-www-form-urlencoded'  
    },  
    body: new URLSearchParams({  
      'parameterName_1': 'value_1',  
      'parameterName_2': 'value_2'  
    })  
  })  
)  
  .then((response) => {  
    if (!response.ok) {  
      throw new Error("Error");  
    }  
    return response.json();  
  })  
  .then((data) => {  
    // Your code  
  })  
)
```

# fetch() method

## Post request example

```
let params = {
  'Type': 'golf-service'
};

let formBody = [];
for (let property in params) {
  let encodedKey = encodeURIComponent(property);
  let encodedValue = encodeURIComponent(params[property]);
  formBody.push(encodedKey + "=" + encodedValue);
}
formBody = formBody.join("&");

fetch("http://127.0.0.1/api/getGolfService.php", {
  method: "POST",
  // body: "Type=golf-service",
  body: formBody,
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded'
  },
}).then((res)=> {
  return res.json();
}).then((data) => {
  console.log(data)
})
```



# Summary



# Exercise

- **Exercise 1:** Write code to append a 4-product list to a product section when clicking on a “View More” button.

**Note:** use layout in your Bootstrap Exercise

- **Exercise 2:** Write code to get values of a form when clicking on a “Send” button. Note: the form contains “Full Name”, “Birthday”, “Address”, “Email” fields.



**Q&A**