Lecture 8: Events, Geolocation & LocalStorage

CS472 Web Programming

Maharishi University of Management

Department of Computer Science

Assistant Professor Asaad Saad

Maharishi University of Management - Fairfield, Iowa © 2016

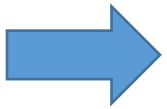
`.each()` loop

```
$("li").each(function(index, e) {
        e = $(e);
        // do stuff with e
});

$("li").each(function() {
        this = $(this);
        // do stuff with this
});
```

- return `false` to exit the loop early
- `e` is a plain old DOM object, We can upgrade it again using `$` if we want
- The `this` keyword refers to the same selected element as `e`
- (when a function is called by an object, that object becomes the context or 'this' parameter)

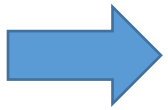
jQuery.each()

```
$.each([ 52, 97 ], function( index, value ) {
      console.log( index + ": " + value );
});
```



0: 52
1: 97

```
var obj = {
        "course": "CS472",
        "name": "WAP"
};
$.each( obj, function( key, value ) {
      console.log( key + ": " + value );
});
```



course: CS472
name: WAP

Attaching event handlers the jQuery way

To use jQuery's event features, you must pass the handler using the jQuery object's event method

```
DOMObject.onevent = function;
jQueryObject.event(function);

// call the playNewGame function when the Play button is clicked
$("#play").click(playNewGame);

function playNewGame(evt) {
        // respond to the click event
}
```

You can trigger the event manually by calling the same function with no parameters

```
$("#play").click();
```

The jQuery event object

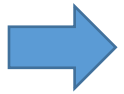
Event handlers can accept an optional parameter to represent the event that is occurring. Event objects have the following properties and methods:

```
function handler(event) {
        // an event handler function ...
}
```

target	The element on which the event handler was registered
preventDefault()	Prevents browser from performing its usual action in response to the event
stopPropagation()	Prevents the event from bubbling up further
stopImmediatePropagation()	Prevents the event from bubbling and prevents any other handlers from being executed

Example

```
<pre id="target">Move the mouse over me!</pre>

window.onload = function() {
        $("#target").mouseover(showCoords);
};

function showCoords(event) {
        $("#target").html(
                "page : (" + event.pageX + ", " + event.pageY + ")\n" +
                "screen : (" + event.screenX + ", " + event.screenY + ")\n" +
                "client : (" + event.clientX + ", " + event.clientY + ")"
         );
}
```



```
pointer: (172, 3519)
screen : (172, 505)
client : (172, 420)
```

Main Point

jQuery has a simple syntax for attaching events to individual or collections of DOM elements. **Science of Consciousness:** Our actions will be naturally efficient when they are spontaneously in accord with natural law.

jQuery and `this`

- All JavaScript code actually runs inside of an object
- By default, code runs in the global window object
  (so `this` === `window`)
- All global variables and functions you declare become part of `window`
- In jQuery the `this` keyword refers to the current object

Event handler binding

Event handlers attached unobtrusively are bound to the element. Inside the handler, that element becomes this (rather than the window)

```
<script>
window.onload = function() {
        $("#textbox").mouseout(sayHi); // bound to text box here
        $("#submit").click(sayHi); // bound to submit button here
};

function sayHi() { // sayHi knows what object it was called on
        this.value = "sayHi";
}
</script>

<div class="exampleoutput">
        <input id="textbox" onmouseout="this.value = 'sayHi';" />
        <input type="submit" id="save" value="Save"
                                        onclick="this.value = 'sayHi';">
</div>
```

Fixing redundant code with `this`

```
<fieldset>
	<label><input type="radio" name="ducks" value="Huey" /> Huey</label>
	<label><input type="radio" name="ducks" value="Dewey" /> Dewey</label>
	<label><input type="radio" name="ducks" value="Louie" /> Louie</label>
</fieldset>

$(":radio").click(processDucks);

function processDucks() {
	if ($("huey").checked) { alert("Huey is checked!"); }
	else if ($("dewey").checked) { alert("Dewey is checked!"); }
	else { alert("Louie is checked!"); }
	alert(this.value + " is checked!");
}
```

If the same function is assigned to multiple elements, each gets its own bound copy

Main Point

All Javascript code runs inside of some object and the *this* keyword always refers to the current object. Event handlers that are attached unobtrusively are bound to that element and inside the handler *this*references the bound DOM element. Usage of the *this* reference in event handlers is a commonly used Javascript event handling programming idiom which enables handlers to be reused across different kinds of elements. **Science of Consciousness:** We can think of the TM Technique as an event handler that gives the result of transcending and can be used by any individual person (element).

Stopping an event's browser behavior

To abort a form submit or another event's default browser behavior, call jQuery's **`preventDefault()`** method on the event

```
<form id="exampleform" action="">...</form>

$(function() {
	$("#exampleform").submit(checkData);
});

function checkData(event) {
	if ($("#city").val() == "" || $("#state").val().length != 2) {
		alert("Error, invalid city/state."); // show error message
		event.preventDefault();
	}
}
```

Which element gets the event?

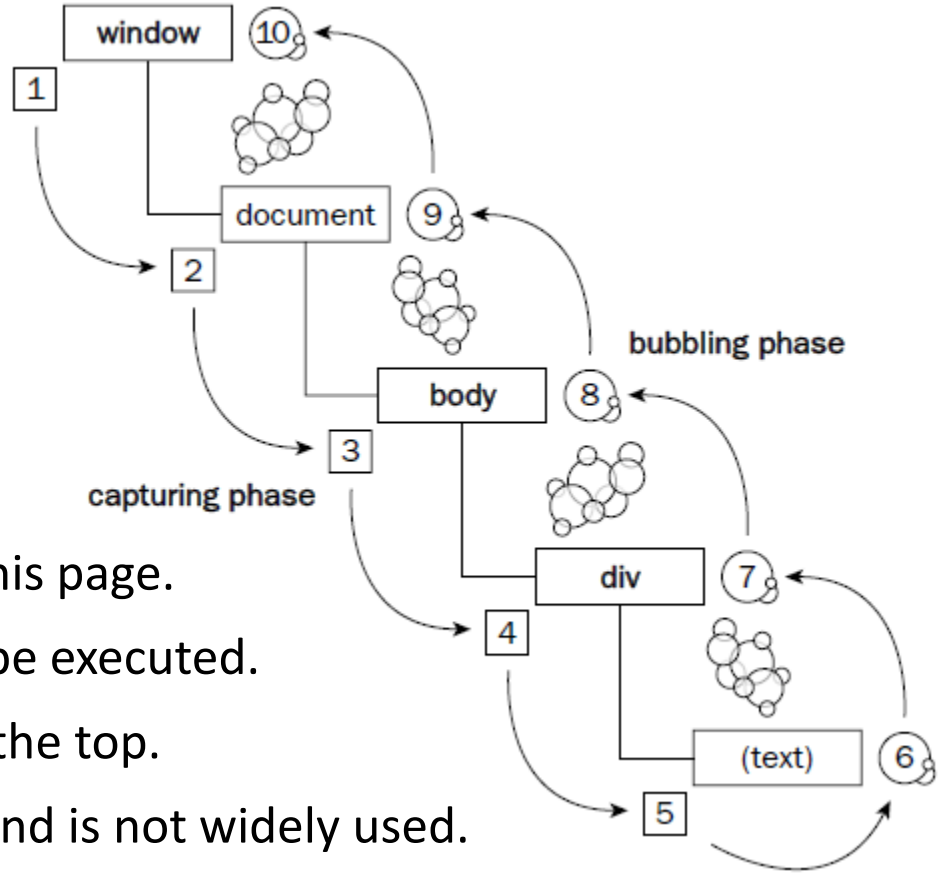
```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
</body>

$(function() {
      $("body, div, p, em").click(hello);
});

function hello() {
      alert("You clicked on the " + $(this).attr("tag"));
}
```

What happens when I click on the span? Which element will get the event?

Answer: All of them!

Event Bubbling

```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
</body>
```



- Clicking the span is actually a click on every element in this page.
- Therefore it was decided that all of the handlers should be executed.
- The events **bubble** from the bottom of the DOM tree to the top.
- The opposite model (top to bottom) is called **capturing** and is not widely used.

Bubbling Example

```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
</body>
```

```
$(function() {
        $("body, div, p, em").click(hello);
});

function hello() {
        alert("You clicked on the " + this.nodeName);
}
```

Run Example

Stopping an event from bubbling

Use the **`stopPropagation()`** method of the jQuery event to stop it form bubbling up.

```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
</body>
```

```
$(function() {
      $("body, div, p, em").click(hello);
});

function hello(evt) {
      alert("You clicked on the " + this.nodeName);
      evt.stopPropagation();
}
```

Run Example

Multiple handlers

```
<body>
    <div>
        <p> Events are <em>crazy</em>! </p>
    </div>
    <p>Another paragraph!</p>
</body>
```

```
$(function() {
        $("body, div, p, em").click(hello);
        $("div > p").click(anotherHandler);
});

function hello() { alert("You clicked on the " + this.nodeName); }
function anotherHandler() { alert("You clicked on the inner P tag"); }
```

What happens when the first p tag is clicked? Run Example

Stopping an event right now

- Use **`stopImmediatePropagation()`** to prevent any further handlers from being executed.
- Handlers of the same kind on the same element are otherwise executed in the order in which they were bound.

```
function anotherHandler(evt) {
        alert("You clicked on the inner P tag");
        evt.stopImmediatePropagation();
}
```

Run Example

stopImmediatePropogation()

```
$("div a").click(function() {
                    // Do something
                    });
$("div a").click(function(evt) {
                    // Do something else
                    evt.stopImmediatePropagation();
                    });
$("div a").click(function() {
                    // THIS NEVER FIRES
                    });
$("div").click(function() {
                    // THIS NEVER FIRES
                    });
```

Only the first two handlers will ever run when the `anchor` tag is clicked.

jQuery handler return value

jQuery does something special if you return `false` in your event handler

1. prevents the default browser action, eg evt.preventDefault()
2. stops the event from bubbling, eg evt.stopPropagation()

```
<form id="exampleform"> ... <button>Done</button> </form>
$(function() {
        $("#exampleform").submit(cleanUpData);
        $("button").click(checkData);
});
function checkData() {
        if ($("#city").val() == "" || $("#state").val().length != 2) {
                alert("Error, invalid city/state."); // show error message
                return false;
        }
}
```

Main Point

Events bubble from the bottom of the DOM tree to the top. The jQuery stopPropagation method prevents bubbling up the tree. jQuery's stopImmediatePropagation method prevents any other handlers that might be attached to the current element from being executed. **Science of Consciousness:** Everything in the universe is connected and it is impossible to intellectually predict all the possible ramifications of an action. If our thoughts are connected to the home of all the laws of nature then our actions will spontaneously be in accord with the entire environment.

Geolocation API

The user’s location can be requested using the geolocation API.

Location data is provided in the form of longitude and latitude points.

Browsers determine locations by:

- IP address
- Wiresless network connection
- Cell towers
- GPS hardware

Using geolocation API

```
getCurrentPosition(success(Position), fail(PositionError));
```

- Position.coords.latitude
- Position.coords.longitude
- Position.coords.altitude
- Position.coords.speed
- PositionError.code
- PositionError.message

Geolocation Example

```
navigator.geolocation.getCurrentPosition(success, fail);

function success(position) {
	console.log('Longitude:' + position.coords.longitude );
	console.log('Latitude:' + position.coords.latitude );
}

function fail(msg) {
	console.log(msg.code + msg.message); // Log the error
}
```

`localStorage` and `sessionStorage`

Problems with cookies:

- Not able to hold much data
- Sent to the server every time you request a page from the domain
- Not secure

The Storage API

- In Storage object browsers store 5MB of data per domain
- Data is stored in key/value pairs
- To protect the information browsers employ same origin policy

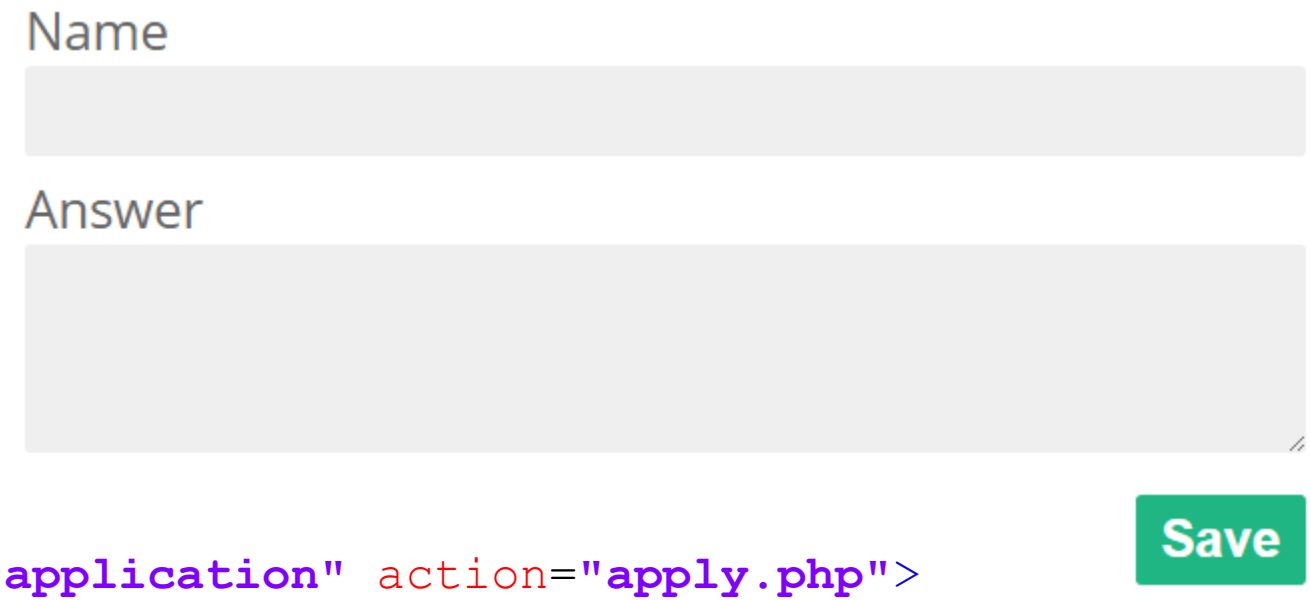
Storage API

- Methods
  - **`setItem (key, value)`**
  - **`getItem(key)`**
  - **`removeItem(key)`**
  - **`clear()`**
- Property
  - **`length`** – number of keys

We can also use the Storage object directly:

```
localStorage.name = 'Asaad'; // Store information
var name = localStorage.name; // Access information
var items = localStorage.length; // 1
```

Web Storage API Example




```
<form id="application" action="apply.php">
        <label for="username">Name</label>
        <input type="text" id="username" name="username" /><br>
        <label for="answer">Answer</label>
        <textarea id="answer" name="answer"></textarea>
        <input type="submit" value="Save" />
</form>
```

Using Web Storage API

```
var txtUsername = document.getElementById('username'); // Get form elements
var txtAnswer = document.getElementById('answer');

txtUsername.value = localStorage.getItem('username'); // by localStorage
txtAnswer.value = localStorage.getItem('answer');
txtUsername.value = sessionStorage.getItem('username'); // by sessionStorage
txtAnswer.value = sessionStorage.getItem('answer');

txtUsername.addEventListener('input', function () { // Data saved on keyup
      localStorage.setItem('username', txtUsername.value);
      sessionStorage.setItem('username', txtUsername.value); }, false);

txtAnswer.addEventListener('input', function () { // Data saved on keyup
      localStorage.setItem('answer', txtAnswer.value);
      sessionStorage.setItem('answer', txtAnswer.value); }, false);
```

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

JavaScript Events, Ajax and Advanced techniques

1. Client side programming with JavaScript is useful for making web applications highly responsive.
2. Ajax allows JavaScript to access the server in a very efficient manner using asynchronous messaging and partial page refreshing.

---

3. **Transcendental consciousness** is the experience of the home of all the laws of nature where all information is available at every point. MAIN POINTS
4. **Impulses within the transcendental field**: Communication at this level is instantaneous and effortless.
5. **Wholeness moving within itself**: In unity consciousness daily life is experienced in terms of this frictionless and effortless flow of information.