JavaScript Events

John M. Morrison

January 2, 2017

Contents

0 Introduction

By and large, the *raison d'etre* for JavaScript is to make web pages interactive. We have all seen pages that respond to interaction with page elements such as menus or buttons, and behavior induced by the mouse entering, clicking, dragging, or leaving page elements.

Other events inlude such things page loading being completed, the page closing, or the page being resized. Events are objects; depending on the source, they contain information about themselves. For example keybord events can tell you what key was hit, and mouse events can tell you where they occur on the screen.

Some events are discrete, in that they happen in a single moment. An example of this is the event of mouse being clicked. Others "poll" and are fired at close intervals over a period of time, such as when a key is pressed and held down. This will broadcast events at a short interval until the user releases the key.

Event handling allows you, the web developer, to respond to thes occurrence and to execute code in response to them.

Remember, we have said that what computers do most of the time is to wait for user instruction. Modern computers are event-driven machines; we can use JavaScript event handling to create the kinds of features you would expect on a web page.

We will deal with a few major types of events. The ones we will look at come in several categories. First, there are browser-spawned events.

Event Name	**When it is fired**
`load`	When the page is finished loading
`unload`	When the document is being unloaded

The `unload` event is often used to remind the user that he has not finished his busineess on a pages. For example, changing pages might cause a post to be lost, so the page warns the user with an `alert` box.

The `load` event is fired when all of the page's elements are finished loading. This is important because, if you are using `document.getElementById`, you want the element being fetechd to exist before you try to act on it. The most common place you will see it is in the `body` tag.

There are events spawned the mouse. Here is a sampling of some of the most important ones.

Event Name	**When it is fired**
`mouseenter`	the mouse enters an element with a listener attached
`mouseover`	the mouse is over an element with a listener attached, or one of its children
`mousemove`	the mouse is moving over an element with a listener attached, or one of its children This event is fired continuously (at the browser's polling rate) until the mouse is no longer over the element.
`mousedown`	any mouse button is pressed on an element
`mouseup`	any mouse button is released on an element
`click`	the mouse is clicked on an element
`dblclick`	the mouse is double-clicked on an element

There are also key events, triggered by actions of the user on the keyboard.

Event Name	When it is fired
`keydown`	any key is pressed on an element
`keyup`	any key is released on an element
`keypress`	any key is pressed on an element

1 Making Elements Event-Aware in HTML

Suppose you want to make a button perform an action when clicked. The following HTML code does this, and causes an alert box to be popped up when the user clicks.

```
<button onclick="alert("I've been clicked!");">Click me!</button>
```

Let us discuss in detail what happens here.

1. The user clicks on the button.
2. The JavaScript code in the `onclick` property is wrapped in an anonymous function.
3. That function is called, and its code executes, causing an alert box to pop up on the screen.
4. The user dismisses the box by hitting the OK button because the browser session spawning the alert box is blocked until he clicks away the box.

Observe we did not type

```
<button click="alert('I\'ve been clicked!');">Click me!</button>
```

or JavaScript would not carry out the action requested. If you want to specify event-handing code in HTML, you must prepend the event's name with `on`.

Warning! Do not attempt to manipulate global variables in your JavaScript using this mechanism. Remember, the code you specify is *wrapped in an anonymous function* and all of those variables become local to that function.

2 Using `onload`

Imagine you are the webmaster for the Greasy Spoon Café and that you are writing up the menu. At the top goes a yummy lunch special of Taylor Ham and Eggs. You start the page like this. The name of this file is `delay.html`.

```
<!doctype html>
<html>
<head>
<title>delay</title>
<link rel="stylesheet" href="delay.css"/>
<script type="text/javascript">
var handle = document.getElementById("special");
handle.innerHTML = "Today's lunch special is Taylor Ham with Eggs.";
</script>
</head>
<body>
<h2>The Greasy Spoon Caf&eacute;</h2>

<p><b>Lunch special</b> <span id="special"></span></p>
</body>

</html>
```

You save the file, refresh the browser and BAM, no lunch special. Everythiing looks fine. What went wrong?

Pages load from top to bottom. Your JavaScript loaded in the head element of the page. You direct JS to get an element by ID that *does not yet exist*. Oops. So, it appears we need to delay the running of the JavaScript until the page loads. So, when the `load` event runs, we will execute our JavaScript.

Let us put our JavaScript in a function to make things simple.

```
function showLunch()
{
    var handle = document.getElementById("special");
    handle.innerHTML =
        "Today's lunch special is Taylor Ham with Eggs.";
}
```

Then, in the `body` tag, place the following.

```
<body onload="showLunch();">
```

Do this, and what once was lost now is found. Here is the final appearance of the HTML

```
<!doctype html>
<html>
<head>
```

```
<title>delay</title>
<link rel="stylesheet" href="delay.css"/>
<script type="text/javascript">
function showLunch()
{
    var handle = document.getElementById("special");
    handle.innerHTML =
        "Today's lunch special is Taylor Ham with Eggs.";
}
</script>
</head>
<body onload="showLunch();">
<h2>The Greasy Spoon Caf&eacute;</h2>

<p><b>Lunch special</b> <span id="special"></span></p>
</body>

</html>
```

Now let us nag the user when he tries to leave our page. To do this we use `onunload`.

```
<!doctype html>
<html>
<head>
<title>delay</title>
<link rel="stylesheet" href="delay.css"/>
<script type="text/javascript">
function showLunch()
{
    var handle = document.getElementById("special");
    handle.innerHTML =
        "Today's lunch special is Taylor Ham with Eggs.";
}
function nag()
{
    alert("Your mama told you not to skip lunch!");
}
</script>
</head>
<body onload="showLunch();" onunload="nag();">
<h2>The Greasy Spoon Caf&eacute;</h2>

<p><b>Lunch special</b> <span id="special"></span></p>
</body>
```

```
</html>
```

You may not see the alert. In this event, look in the console and you will see this.

```
Blocked alert('Your mama told you not to skip lunch!') during unload.
```

People have abused this feature and now browsers are suppressing alerts fired on unload. This occurred whilst using Chrome.

However, it is the `onload` property for the body element that is far more important.

3 Cat and Mouse Games: Mouse Events

We can make elements on a page sensitive to mouse events. Here is where we start. Make this file, `mouseOne.html`.

```
<!doctype html>
<!--Author: Morrison-->

<html>
<head>
<title>mouseOne</title>
<link rel="stylesheet" href="mouseOne.css"/>
<script type="text/javascript" src="mouseOne.js">
</script>
</head>
<body onload="main();">
<h2>Mouse Event Demonstration</h2>

<div id="colorBlock">
<p>This div contains text</p>
</div>

</body>
</html>
```

Now let us give it a little style with the file `mouseOne.css`.

```
h1, h2, .display
{
```

```
    text-align:center;
}
#colorBlock
{
    width:80%;
    padding:1em;
    border:solid 1px black;
}
```

Open the HTML file with your browser. You will see a `div` with a black outline containing some text. Now we will make this `div` be mouse-aware.

Next, make this file, `mouseOne.js`

```
/*Author: Morrison*/
var block;
function handleClicked()
{
    block.style.color="red";
}
function main()
{
    block = document.getElementById("colorBlock")
    block.onclick = handleClicked;
}
```

Now let us write code to react to the mouse entering; we will cause the text to turn blue when this happens. Modify your JavaScript file to look like this

```
/*Author: Morrison*/
var block;
function handleClicked()
{
    block.style.color="red";
}
function handleOver()
{
    block.style.color="blue";
}
function main()
{
    block = document.getElementById("colorBlock")
    block.onclick = handleClicked;
    block.onmouseover = handleOver;
}
```

Now, as soon as your mouse enters the box, the text turns blue. When you click in the box, the text will turn red. Now let us restore it to black when we leave. To do this, add this function

```
function handleOut()
{
    block.style.color="black";
}
```

and this line of code to `main`

```
    block.onmouseout = handleOut;
```

Now, when your mouse enters the box, the text turns blue and when you exit i it, the text is black again. If you click in the box, the text turns red until you exit the box, at which time it becomes black again.

Let us now turn to a slightly less artificial and more useful example. Let us see how to make an expandable list of items using mouse events. For this purpose we shall repair to the Greasy spoon Café.

Here is the main idea. Each portion of the menu will have a heading. When this heading is moused over, the contents of that area of the menu become visible. When the mouse leaves the menu contents, they cease to be displayed.

We can achieve this using the style property `style.display`. Setting this property to `block` causes it to display normally. Setting this to `none` will cause it to be hidden.

We will need to create an HTML file that gives and ID to each menu heading as well as the menu heading's contents.

```
<!doctype html>
<html>
<head>
<title>expandalbeMenus</title>
<meta charset="utf-8"/>
<link rel="stylesheet" href="expand.css.css"/>
<script type="text/javascript" href="expand.js"/>
</head>
<body onload="main();">
<h2>The Greasy Spoon Caf&eacute;</h2>

<p><b>Lunch Special</b> <span id="special"></span></p>

<p><b id="appsHead">Appetizers</b></p>
<div id="apps">
```

```
    <ul>
        <li>House Salad $3</li>
        <li>Tomato Soup $2</li>
        <li>Garlic Knots $3</li>
        <li>Chips and Salsa $3</li>
    </ul>
</div>

<p><b id="entreesHead">Entrees</b></p>
<div id="entrees">
    <ul>
        <li>Hot Roast Beef Sandwich $3</li>
        <li>Tuna Melt $2</li>
        <li>Fried Chicken $3</li>
        <li>Spaghetti and Meatballs $3</li>
    </ul>
</div>

<p><b id="dessertsHead">Desserts</b></p>
<div id="desserts">
    <ul>
        <li>Apple Pie $3</li>
        <li>Cherry Pie $3</li>
        <li>Chocolate  Cake $3</li>
        <li>Banana Pudding $2</li>
        <li>Scoop of ice cream with any pie $1</li>
    </ul>
</div>
</body>
</html>
```

Now create this style file, `expand.css`

```
/*Author: Morrison*/
h1, h2, .display
{
    text-align:center;
}
```

Finally, create `expand.js`.

```
function main()
{
}
```

Open the `.html` file with your browser. You should see a page containing the complete menu for the Greasy Spoon Café.

Now let us begin to work on our JavaScript file. We will begin by creating handles to all of our IDed elements.

```
funtion main()
{
    apps = document.getElementById("apps");
    appsH = document.getElementById("appsHead");
    entrees = document.getElementById("entrees");
    entreesH = document.getElementById("entreesHead");
    desserts = document.getElementById("desserts");
    dessertsH = document.getElementById("dessertsHead");
    special=document.getElementById("special");
}
```

Next, we put in today's special and we suppress the display of the three elements containing menu contents.

```
function main()
{
    apps = document.getElementById("apps");
    appsH = document.getElementById("appsHead");
    entrees = document.getElementById("entrees");
    entreesH = document.getElementById("entreesHead");
    desserts = document.getElementById("desserts");
    dessertsH = document.getElementById("dessertsHead");
    special = document.getElementById("special");
    special.innerHTML = "Tomato Soup and Grilled Cheese Sandwich, $7.";
    apps.style.display = "none";
    entrees.style.display = "none";
    desserts.style.display = "none";
}
```

Refresh. You should now see the daily speical posted and thhe three main menu heading with no items below them.

Using `addEventListener` This is a newer feature in JavaScript. Its usage looks like this; we will just use `false` for now for the third argument and discuss its purpose later.

```
item.addEventListener(eventType, someFunction, false);
```

Here is what happens. When the event `eventType` occurs in the element `item`, the function `someFunction` is called.

What we will do is when one of the menu headings is moused over, we will display the items constituting the contents of that heading. When we leave the contents, we will no longer display them. This creates the effect of "expandable menus" in JavaScript.

Here is how we will get the appetizers to display. Note the use of an anonymous function to trigger the display of the appetizers.

```
appsH.addEventListener("mouseover",
       function(){apps.style.display="block";}, false);
```

Refresh. When you mouse over the heading for the appetizers, they appetizers now display. When you move away from them, they persist. Now let us make them disappear.

```
apps.addEventListener("mouseout",
       function(){apps.style.display="none";}, false);
```

Now our function looks like this.

```
/******* Autor: Morrison *********/
function main()
{
    apps = document.getElementById("apps");
    appsH = document.getElementById("appsHead");
    entrees = document.getElementById("entrees");
    entreesH = document.getElementById("entreesHead");
    desserts = document.getElementById("desserts");
    dessertsH = document.getElementById("dessertsHead");
    special = document.getElementById("special");
    special.innerHTML = "Tomato Soup and Grilled Cheese Sandwich, $7.";
    apps.style.display = "none";
    entrees.style.display = "none";
    desserts.style.display = "none";
    appsH.addEventListener("mouseover",
            function(){apps.style.display="block";}, false);
    apps.addEventListener("mouseout",
                function(){apps.style.display="none";}, false);
}
```

Next, deal similarly with the entrees and desserts.

```
    entreesH.addEventListener("mouseover",
                function(){entrees.style.display="block";}, false);
```

```
entrees.addEventListener("mouseout",
            function(){entrees.style.display="none";}, false);
dessertsH.addEventListener("mouseover",
            function(){desserts.style.display="block";}, false);
desserts.addEventListener("mouseout",
            function(){desserts.style.display="none";}, false);
```

Programming Exercises