JavaScript Events II: The Mouse and Keyboard

John M. Morrison

August 6, 2018

Contents

0 Introduction

The mouse and keyboard exhibit a wide array of ways for the user to send signals to a program. We will begin with an exploration of the mouse and its behaviors. We will then turn to looking at using keys to interact with programs.

1 The Mouse

Let us recall the events we showed in the last chapter that are spawned by the mouse.

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

We will begin with a very simple example where a mouse entering a `div` causes the `div`'s contents to change colors. Note that you cannot achieve this with hover, since the colors would revert when the mouse exited. We create this HTML first in the file `div.html`.

```
<!doctype html>
<html>
<head>
<title>div</title>
<meta charset="utf-8"/>
<link rel="stylesheet" href="div.css"/>
<script type="text/javascript" src="div.js"></script>
</head>
<body>
    <h2>Mouse-Sensitive <code>div</code> Demo</h2>

    <div id="mouseMe">
        <h2>Mouse Me!</h2>

        <p>This <code>div</code> will change color as the mouse
            enters and exits it.  </p>

        <ol>
            <li>The background changes color.</li>
            <li>The text changes color.</li>
            <li>Watch what happens to this list.</li>
        </ol>
```

```
    </div>

</body>
</html>
```

Next, we give it some basic style.

```
h1, h2, .display
{
    text-align:center;
}
body
{
    background-color:#FFF8E7;
    padding:1em;
}
#mouseMe
{
    width:80%;
    border:solid 1px black;
    padding:1em;
}
```

Next, some Javascript to make the `div` aware of `mouseenter` events

```
window.addEventListener("load", init);
function init()
{
    lee mm = document.getElementById("mouseMe");
    mm.addEventListener("mouseenter", e =>
    {
        mm.style.color = "red";
        mm.style.backgroundColor = "yellow"
        let list = document.querySelectorAll("#mouseMe ol");
        list[0].style.listStyleType="lower-roman";
    });
}
```

Notice the use of `document.querySelecttorAll` to find the ordered list. This method returns a nodelist of all nodes that would be selected by `#moouseMe ol`, which is all `ol` nodes that are descandants of the node having `id mouseMe`.

Programming Exercise

1. Give the `div` a `mouseleave` event to restore it to its original condition when the mouse exits it.

2. Give the `div` a `mousemove` event to make it react to a mouse moving by changing a a color.
3. Deal similarly with `mousepressed` and `mousereleased`.

2 Making a Canvas Sensitive to `click` events

Here is our aim. Have a canvas draw a disc of radius 50px centered at the point of the click of a random color. So let us begin by making an HTML file with a canvas in it.

```
<!doctype html>
<html>
<head>
<title>canvas</title>
<meta charset="utf-8"/>
<link rel="stylesheet" href="canvas.css"/>
<script type="text/javascript" src="canvas.js"></script>
</head>
<body>
    <h2>Mouse Clickable Canvas</h2>
    <p class="display">Click on the canvas to draw a ball in it.</p>

    <p class="display">
        <canvas width="700" height="500" id="surface">
            Get a modern browser that supports canvas, chump.
        </canvas>
    </p>

</body>
</html>
```

Now give it a little style.

```
h1, h2, .display
{
    text-align:center;
}
canvas
{
    border:solid 1px black;
    background-color:white;
}
body
```

```
{
    padding:1em;
    background-color#FFF8E7;
}
```

All that was easy. Now we get down to business and start making some JavaScript. Here we create a shell to get started.

```
window.addEventListener("load", init);
function init()
{
    let c = document.getElementById("surface");
    let pen = c.getContext("2d");
    c.addEventListener("click" e =>
    {
    });
}
```

Now we want to think about what needs to happen when the user clicks. The pen should be set to a random color, then a ball should be drawn centered at the click with radius 50 px.

So, let us create shells for some auxillary functions.

```
window.addEventListener("load", init)
function init()
{
    let c = document.getElementById("surface");
    let pen = c.getContext("2d");
    c.addEventListener("click", e =>
    {
    });
}
function drawBall(pen, x, y)
{
}
function generateRandomColor()
{
}
```

The function `drawBall` is low-hanging fruit if we use the `arc` method for the `pen` object.

```
function drawBall(pen, x, y)
{
```

```
    pen.beginPath()
    pen.arc(x, y, 0, 50, 0, 2*Math.PI);
    pen.fill();
}
```

So, let us add a call to this function in our event listener. Notice that our argument to our anonymous function passed to the listener is `e`; think of this as "event." In our case, it is the mouse event object that is broadcast. Here is a table of some important properties for mouse events

`x`	The x-coördinate of the click in pixels in the content pane of the browser.
`y`	The x-coördinate of the click in pixels in the content pane of the browser.
`offsetX`	The x-coördinate of the click in pixels in the element.
`offsetY`	The x-coördinate of the click in pixels in the element.
`screenX`	The x-coördinate of the click in pixels in the entire screen.
`screenY`	The x-coördinate of the click in pixels in the entire screen.

We now make the call using the canvas coördinates.

```
    c.addEventListener("click", e =>
    {
        drawBall(pen, e.offsetX, e.offsetY);
    });
```

Click in the canvas and see a black ball get drawn on the screen.

Now let us mix the random color. To add to the fun, we will give random transparency, too. To do this we use `rgba`. First, let's generate a random number 0-255

```
function randLevel()
{
    return 256*Math.random();
}
```

Now let's generate the color

```
function randColor()
{
```

```
    return "rgba(" + randLevel() + ", "
        + randLevel() + ", " + randLevel() + ", " + Math.random() + ")";
}
```

Now insert a color change like so.

```
    c.addEventListener("click", e =>
    {
        pen.fillStyle = randColor();
        drawBall(pen, e.offsetX, e.offsetY);
    });
```

Open the HTML file in your browser. When you click on the canvas, a random-colored ball of random opacity will appear on the canvas.

Programming Exercises

1. Put a "clear" button on the page that clears the canvas of balls.
2. Put a slider on the page to change the size of the ball. Have the current size of the next ball to be drawn show on the page.

3 Balls as Objects

Let us make a smarter ball. To think about creating a ball object we ask, "What does a a ball need to know?" and "What does a ball need to be able to do?"

A self-respecting ball needs to know everything necessary to draw itself when it's handed a pen. This is the role of the ball, so we will design it to do exactly that. A ball is drawn by its center, radius, and it needs to have a color specified. So, let's decide on some properties

`centerX`	the x-coördinate of the ball's center
`centerY`	the y-coördinate of the ball's center
`radius`	the radius of the ball
`color`	the color of the ball

Now we begin to create our class by giving it state.

```
class Ball
{
    constructor(centerX, centerY, radius, color)
    {
```

```
        this.centerX = centerX;
        this.centerY = centerY;
        this.radius = radius;
        this.color = color;
    }
}
```

Copy the class and paste it into an empty console window. Then we can test-drive it like so.

```
> b = new Ball(100,100,50,"red");
Ball{centerX: 100, centerY: 100, radius: 50, color: "red"}
```

To draw itself, our balls need a pen, which will be passed in by the application using balls. Let us put a method stub in for it.

```
class Ball
{
    constructor(centerX, centerY, radius, color)
    {
        this.centerX = centerX;
        this.centerY = centerY;
        this.radius = radius;
        this.color = color;
    }
    draw(pen)
    {
    }
}
```

Now we need to think about drawing ourselves if we are a ball. We have been handed a pen. So the first order of business is to make it our color.

```
pen.fillStyle = this.color;
```

Now we do the rest by having the pen execute and fill a path.

```
pen.beginPath();
pen.arc(this.centerX, this.centerY, this.radius, 0, 2*Math.PI);
pen.fill();
```

Here is our complete class

```
class Ball
{
```

```
    constructor(centerX, centerY, radius, color)
    {
        this.centerX = centerX;
        this.centerY = centerY;
        this.radius = radius;
        this.color = color;
    }
    draw(pen)
    {
        pen.fillStyle = this.color;
        pen.beginPath();
        pen.arc(this.centerX, this.centerY, this.radius, 0, 2*Math.PI);
        pen.fill();
    }
}
```

Now we will refactor our code in the app we just built to use this class. Modify the head of your HTML document as follows.

```
<head>
    <title>canvas</title>
    <meta charset="utf-8"/>
    <link rel="stylesheet" href="canvas.css"/>
    <script src="Ball.js"></script>
    <script src="canvas.js"></script>
</head>
```

We just added a new line to load the file `Ball.js` into our page's memory. We will now take advantage of our new Ball class. Open the file `canvas.js` and modify it as follows.

```
window.addEventListener("load", init)
function init()
{
    let c = document.getElementById("surface");
    let pen = c.getContext("2d");
    c.addEventListener("click", e =>
    {
        b = new Ball(e.offsetX, e.offsetY, 50, generateRandomColor());
        b.draw(pen);
    });
}
function randLevel()
{
    return 256*Math.random();
```

```
}
function generateRandomColor()
{
    return "rgba(" + randLevel() + ", "
        + randLevel() + ", " + randLevel() + ", " + Math.random() + ")";
}
```

We get rid of `drawBall` since we taught a ball to draw itself. Our event listener simply makes a new ball in the desired location and has it draw itself.

4 Creating Application State

So far, our application as a whole is not very self-aware. It passively plops a ball on a page when we click on the canvas. We might like to be able to change the background color or the size of the next ball. We might want to be able to undo the last ball we placed.

If we change the background color, we will need to redraw the balls we placed on the page. We need to keep track of them. This is also necessary for having an undo feature. So, let's list some properties to make this process go.

`balls`	an array of all of the balls we have created
`bgColor`	the color of the canvas's background

Let us add these to our `init` method.

```
function init()
{
    let c = document.getElementById("surface");
    let pen = c.getContext("2d");
    let balls = [];
    let bgColor = "white";
    c.addEventListener("click", e =>
    {
        b = new Ball(e.offsetX, e.offsetY, 50, generateRandomColor());
        b.draw(pen);
    });
}
```

Next, we will go into our event listener and, when a click occurs, add the new ball to the array. Let us get rid of `b.draw(pen)`, as we will now write a new method to refresh the canvas. You can see we added that calll.

```
c.addEventListener("click", e =>
{
    balls.push(new Ball(e.offsetX, e.offsetY, 50, generateRandomColor()));
    refresh();
});
```

Now implement `refresh`; this function must be nested inside of `init` so it can see application state. Notice the use of the `for of` loop for traversing the collection of balls. If the array is empty, the loop will be skipped, as it should be.

```
function refresh()
{
    pen.fillStyle = bgColor;
    pen.fillRect(0, 0, c.width, c.height);
    for(let b of balls)
    {
        b.draw(pen);
    }
}
```

Now let us implement an undo feature to erase the last ball. We need to add a button to the page. Add this after the canvas element in `canvas.html`

```
<p class="display"><button id="undoButton">Undo Last</button></p>
```

The next order of business is in the `.js` file; we need to create a pointer to the button and give it a listener. So let us add this line

```
let undoer = document.getElementById("undoButton");
```

The listener should pop the array and then refresh the canvas if the array is not empty. If the array is empty, do nothing.

```
undoer.addEventListener("click", e =>
{
    if(balls.length > 0)
    {
        balls.pop();
        refresh();
    }
});
```

Programming Exercises

1. Add a color input to the page to change the background color of the canvas.
2. Add a slider to control the size of the ball. Note you will need to add a variable to keep track of the size of the next ball.

5 Key Event Handling