Contents

[JS Event target versus currentTarget 1](#_Toc137012483)

[New keyword 2](#_Toc137012484)

[Prototype 2](#_Toc137012485)

Notes

# JS Event target versus currentTarget

Events triggered in child elements move to the parent elements. This is called bubbling.

Target will refer to the element that fired the event.

* target is the element that triggered the event (e.g., the user clicked on)
* currentTarget is the element that the event listener is attached to.

Youtube: [JS Event target versus currentTarget - YouTube](https://www.youtube.com/watch?v=SpatM1W5wRQ)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Target and CurrentTarget</title>

    <meta name="viewport" content="width=device-width">

    <style>

        main{

            background-color: rebeccapurple;

            color: white;

            padding: 1rem;

        }

        div{

            background-color: #bada55;

            color: #333;

            padding: 1rem;

        }

    </style>

    <!--  -->

</head>

<body>

    <h1>event.target versus event.currentTarget</h1>

    <main>

        This text is inside the &lt;main&gt; element.

        <div>This text is inside the &lt;div&gt; element.</div>

    </main>

    <script>

        document.querySelector('main').addEventListener('click', clicked);

        document.querySelector('div').addEventListener('click', clicked);

        function clicked(ev){

            console.log('The click that was attached to', ev.currentTarget.tagName, 'is currently at', ev.target.tagName);

        }

    </script>

</body>

</html>

# New keyword

**New keyword** in JavaScript is used to create an instance of an object that has a constructor function. On calling the constructor function with ‘new’ operator, the following actions are taken:

* A new empty object is created.
* The new object’s internal ‘Prototype’ property (\_\_proto\_\_) is set the same as the prototype of the constructing function.
* The ‘this’ variable is made to point to the newly created object. It binds the property which is declared with ‘this’ keyword to the new object.
* About the returned value, there are three situations below.

1. If the constructor function returns a non-primitive value (Object, array, etc), the constructor function still returns that value. Which means the new operator won’t change the returned value.
2. If the constructor function returns nothing, ‘this’ is return;
3. If the constructor function returns a primitive value,  it will be ignored, and ‘this’ is returned.

# Prototype

[All you need to know to understand JavaScript’s Prototype](file:///E:\FWD\ELZERO\Challenges\Notes\External\1-%20All%20you%20need%20to%20know%20to%20understand%20JavaScript’s%20Prototype.html)

// first method that uses a lot of memory

var foo = function(name) {

    this.myName = name;

    this.tellMyName = function() {

        console.log(this.myName);

    }

}

var fooObj1 = new foo('James');

fooObj1.tellMyName();

//second method that saves memory

var foo = function(name) {

    this.myName = name;

}

foo.prototype.tellMyName = function() {

     console.log(this.myName);

    }

var fooObj1 = new foo('James');

fooObj1.tellMyName(); // will print James

var fooObj2 = new foo('Mike');

fooObj2.tellMyName(); // will print Mike

**The above example shows two methods. In the first code the “tell my name” is an object literal but in the second code the “tell my name” is a prototype.**

**The object literal will use the additional amount of space when creating a new object. On the other hand, the prototype will now use any additional space. (see figure 2 and 3 in the** [link](file:///E:\FWD\ELZERO\Challenges\Notes\External\1-%20All%20you%20need%20to%20know%20to%20understand%20JavaScript’s%20Prototype.html)**)**