JavaScript Events

JavaScript Events

* JavaScript Events are items that transpire based on an action. A document event is the loading of an HTML document. A form event is the clicking on a button. Events are objects with properties.
* An event occurs when something happens in a browser window.
* The kinds of events that might occur are due to:
* A document loading
* The user clicking a mouse button
* The browser screen changing size
* When a function is assigned to an event handler, that function is run when that event occurs.
* If third argument is false that means event bubbling & if true i.e. event capturing.

Advanced Event Handling consists of three sections:

* Cancelling Events
* Event Bubbling
* Mouse Capture
* Cancelling Events stops them from happening. So, if you press a key, it cancels your onkeypress event.
* Event Bubbling is slightly more involved, and uses the hierarchical nature of a web document. Elements have owners, or parents. An image on a document is owned by the document. This means that an event on the image creates an event for the document, and so on.
* Mouse Capture is when we transfer all the events for a document to a single element, giving that element the capture.

return false

* Using return false inside an event handler cancels the event.

event.returnValue

* An event can be cancelled by setting the event.returnValue property from a function. If we set it to true, the event continues as normal, if we set it to false, the event is cancelled.

Events

* JavaScript code runs sequentially
* Events can trigger scripts
* Browsers trigger events
* Events may happen anytime

For example when the page loading a user moves mouse on a link, when a video is finished loading or even a form is submitted.

Events Registration

* Telling the browser we want do something when an events take place.
* Several ways to doing that :   
  a. Using tag attributes (That’s may problem when we want to update the script) Example - <li><img src="images/artists/Bellingham.jpg" alt="Photo of Barot Bellingham" onclik = "alert('pushed pink')"></li>
* This is considered extremely bad practice because its makes our code hard to update. It’s much better to place the JavaScript outside the html tag.

b. Using dot notation (That’s means which element from the dom we want to get to and then specify which event we want to track)

<li><img src="images/artists/Bellingham.jpg" alt="Photo of Barot Bellingham" id = "pink"></li>

<script>

document.getElementById('pink').onclick=function(){

alert('Clicked on pink');

}

</script>

c. Using addEventListener() (The most modern way to handle the events)

- addEventListener() has three parts first it takes the element, second it asks the function third it takes true or false that means event propagation.

<li><img src="images/artists/Bellingham.jpg" alt="Photo of Barot Bellingham" id = "pink"></li>

<script>

document.getElementById('pink').addEventListener('click', function(){

alert('Clicked on pink');

}, false);

</script>

* The huge advantage of addEventListener is allowing us to check multiple events with a single call. This is known as event propagation.
* The other advantage is event can trigger by other non-dom element.

The event object

* Capturing an event returns an object – we receive an event object that back from the browser
* The object might look different depending on the browser versions.
* Data will depend on type of event
* Lot of info that common for all events
* Lots of browser and environment info

Event Properties

Event Info

* Type – The type of event that we getting like click event
* Timestamp – The time which the events happen
* defaultPrevented – preventing default behaviors happening or preventing default actions.

Targeting Info

* Target – The target element is that the originated from.
* srcElement – Actual element that fires the event.
* currentTarget – The element that was event assigned to.
* fromElement & toElement
* This two element are related with one other they handling mouseOver and mouseOut

Reference : [www.javascriptkit.com/jsref/events.html](http://www.javascriptkit.com/jsref/events.html)

Coordinate info

* Screen X,Y – perception relative to the user screen
* Offset X,Y – relative to the X and Y positions relative to element that fire the event
* Page X,Y – relative to the html document
* Client X,Y - relative to window
* Layer X, Y – position of the event element relative to the other event element. This are not available in all browser

Reference

* To check browser compatibility
* www.quirksmode.org

Key/Mouse info

* altKey
* button
* shiftKey
* ctrlKey
* charCode/keyCode – Tracking which character is pressed

Reference

* www.unixpapa.com/js/key.html

addEventListener

* The EventTarget.addEventListener() method registers the specified listener on the EventTarget it's called on. The event target may be an Element in a document, the Document itself, a Window, or any other object that supports events (such as XMLHttpRequest).

Event Propagation

* A single element capture all the children elements
* An element can capture child events
* A good reason to use addEventListener
* Not compatible with older versions of IE

Dark Side of Propagation Model

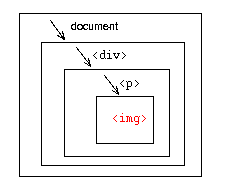
* Different browser supports different propagation model
* The browser catch upon the events in one of two ways
* Browsers disagree on the order
* Capturing goes down the DOM
* Bubbling goes up the DOM

Event Capture and Bubbling

* The DOM has two ways for objects to detect events: from the top down, and from the bottom up. The first method is known as event capture, the second is called event bubbling.

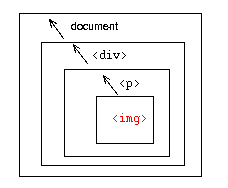
Event Capture

* The event handler of outer-div fires first, then event handler of inner-div fires last.
* Let's say that your document contains a <div> which contains a <p> which contains an <img>. Further, let's say you've added an event listener to all of them. When a user clicks on the image, a mouseclick event occurs.
* With capturing, the event is first captured by the outermost element and propagated to the inner elements.



Bubbling

* The event handler of the inner-div fires first, the event handler of outer-div fires last.
* With bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.



Event capturing

When you use event capturing

| |

---------------| |-----------------

| element1 | | |

| ----------- | |----------- |

| |element2 \ / | |

| ------------------------- |

| Event CAPTURING |

* The event handler of element1 fires first, the event handler of element2 fires last.

Event bubbling

When you use event bubbling

/ \

---------------| |-----------------

| element1 | | |

| ----------- | |----------- |

| |element2 | | | |

| ------------------------- |

| Event BUBBLING |

-----------------------------------

* The event handler of element2 fires first, the event handler of element1 fires last.

Event Propagation

* If its last argument is true the event handler is set for the capturing phase, if it is false the event handler is set for the bubbling phase.

Stopping Propagation

* Propagation saves time
* By capturing parent element instead of capturing individual element
* It can be stopped
* Use the stopPropagation() method
* Cancelbubble = true for <IE8

Preventing default behavior

* Events can have consequences
* Clicking on links that the default behavior follow the link until probably the website
* Submitting forms the default behavior takes the form data and pass it to the form processing
* Prevent a behavior with preventDefault()
* preventDefault is useful for form processing and form validation

Preventing default behavior

* preventDefault is a complementary method that can be used to prevent the default action of the event from happening.
* Lots of events have default behavior that takes place when events happen
* Events can have consequences
* For example when Clicking on links the default behavior follow the link until the website
* Submitting forms the default behavior takes the form and pass it to form processing
* Prevent a behavior with preventDefault()

Notes about Events

* Every DOM element has its own addEventListener method, which allows you to listen specifically on that element.

Event.stopPropagation()

* Prevents further propagation of the current event.
* The event.stopPropagation() method stops the bubbling of an event to parent elements, preventing any parent event handlers from being executed.
* Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.
* stopPropagation stops the event from bubbling up the event chain.
* preventDefault prevents the default action the browser makes on that event.
* With stopPropagation only the buttons click handler is called and the divs click handler never fires.
* Where as if you just preventDefault only the browsers default action is stopped but the div's click handler still fires.