Events and Event Registration

Tuesday, February 7, 2017 11:43 PM

Understanding events

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

Event Registration

- For telling browsers what to do when an event occurred, we have to do event registration.
- There are many ways of doing this, like

Using tag attributes

- We can simply add an event handler(present as tag-attribute) to any **tag.**

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<some-HTML-element some-event='some JavaScript'>
```

Example **- onclick**

```
<li><img src="images/pnk.gif" alt="pink" onclick="alert('pushed pink')"></li>
```

Using dot notation

- we can give an event to any tag using DOM.
- We first grab that element and then using dot(.) assign it's event handler some function.
- Grabbing of element can be done easily by giving id to that element.

Example -

```
<ul>
  <li><img src="images/pnk.gif" alt="pink" id="pink_image"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
</ul>

<script>
  document.getElementById('pink_image').onclick=function(){
   alert('pink is clicked');
  }
```

NOTE - We have added an id, so we can grab element easily, then assign function to onclick.

Using `addEventListener()`

- This is mode modern way of handling event.

syntax - addEventListener(eventy_type,JS_function,boolean propagation_type)

- We learn propagation type later.

```
    <li><img src="images/pnk.gif" alt="pink" id="pink"></li>
    <li><img src="images/grn.gif" alt="green"></li>
    <li><img src="images/ygr.gif" alt="ygreen"></li>
    </ul>
<script>
  document.getElementById('pink').addEventListener('click', function() {
    alert('clicked on pink');
  }, false);
</script>
```

- ADVANTAGE - allow check for many event for same function, this is called event propagation.

`addEventListener()` has several benefits over using the DOM event attributes:

- ✔ You can apply more than one event listener to an element.
- ✔ It works on any node in the DOM tree, not just on elements.
- ✔ It gives you more control over when it's activated.

The `addEventListener()` method is implemented by using three arguments.

The first argument is the event type. Unlike the other two event handling methods, `addEventListener()` just wants the name of the event, without the `on` prefix.

The second argument is the function to call when the event happens. As with the event properties method of event handling, it's important to not use the parentheses here in order for the function to be assigned to the event handler, rather than the result of running the function.

The third argument is a Boolean value (`true` or `false`) that indicates the order in which event handlers execute when an element with an event has a parent element that also is associated with an event.

Handling event in IE8 and older browsers

Wednesday, February 8, 2017 1:04 AM

Legacy Browsers

Support for legacy browsers
Special problem for events
IE8 and earlier use different model

- IE8 and older browsers use different model for event handling, so we have to take care of them specially.
- IE8 uses **attachEvent()** method instead of **addEventListener()**

IE9>-- use addEventListener()

```
<script>
  document.getElementById('pink').addEventListener('click', function() {
    alert('clicked on pink');
  }, false);
</script>
```

IE8-- use attachEvent()

```
<script>
  document.getElementById('pink').attachEvent('onclick', function() {
    alert('clicked on pink');
  });
</script>
```

- Main difference btw functions
  - There is full name of event in **attachEvent()** like 'onclick' , but only 'click' in **addEventListener().**
  - Event propagation is not passed in 'attachEvent()' , there is a different method of passing this.

Cross Platform Example

```
<script>
if (window.addEventListener) {
  document.getElementById('pink').addEventListener('click', function() {
    alert('clicked on pink');
    console.log('addEventListener');
  }, false);
} else if (window.attachEvent) {
  document.getElementById('pink').attachEvent('onclick', function() {
    alert('clicked on pink');
    console.log('attachEvent');
  });
}
</script>
```

- If 'addEventListener' is present than use this, and same for attachEvent.

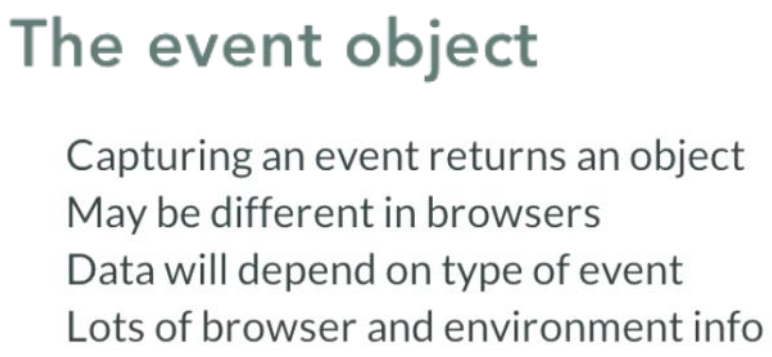
Using jQuery

```
<script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
<script>
  $('#pink').on('click', function(event) {
      alert('clicked on pink');
  });
</script>
```

- We can also use Jquery to handle the events, so we do not need to care about the problem.

Event Properties

Wednesday, February 8, 2017 1:14 AM



- Event object can be used to get the properties of an event.
- If we pass a variable in the **handler function** of **addEventListener() only,** then this object will automatically be passed to the function. which we can use for some query.

```
<ul>

  <li><img src="images/pnk.gif" alt="pink" id="pink"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
</ul>
<script>
  document.getElementById('pink').addEventListener('click', function(e) {
    console.log(e);
  }, false);
```

- Now here 'e' is passed to handler function, so when it event occurred information of event is passed into this variable 'e'.

OUTPUT -

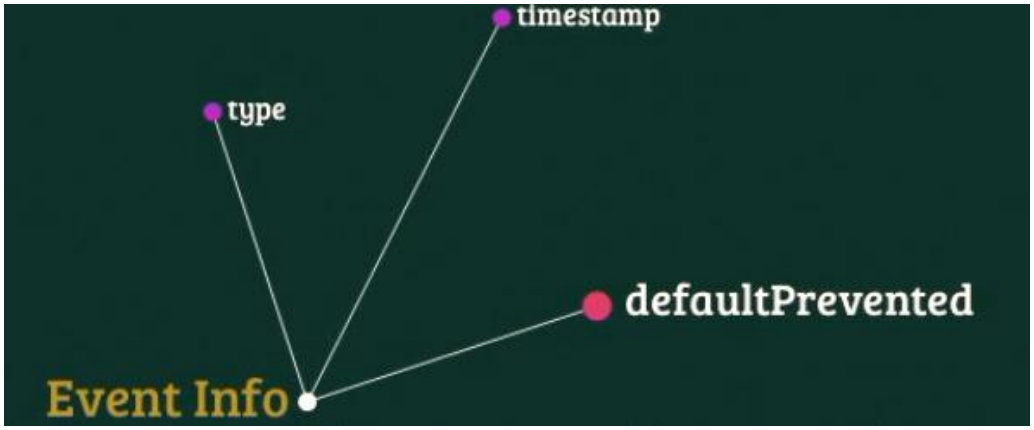
- Each time we click we get the following output, which contain all information about the event.

```
                                                                    index.html:17
▶ MouseEvent {isTrusted: true, screenX: 254, screenY: 153, clientX: 254, clientY: 87…}
                                                                    index.html:17
▶ MouseEvent {isTrusted: true, screenX: 224, screenY: 158, clientX: 224, clientY: 92…}
                                                                    index.html:17
▶ MouseEvent {isTrusted: true, screenX: 223, screenY: 158, clientX: 223, clientY: 92…}
>
```

Getting Information

- We can get many type of information from 'event object', some of them are as follows.

Event Info



Type - event type ex-click(whatever you have passed as first argument in addActionEvent()
TimeStamp- when something happened
DefaultPrevented- we can prevent default behaviour of event.

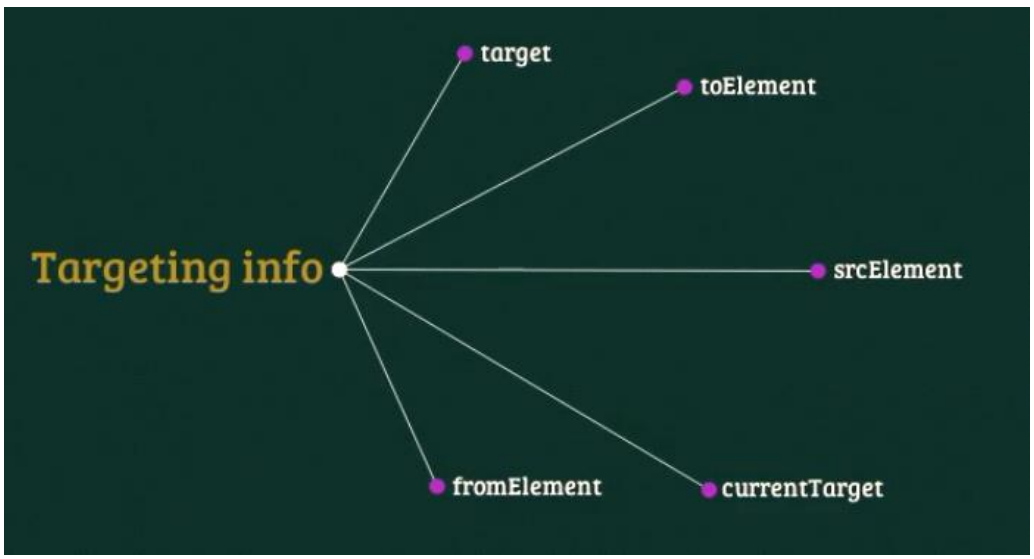
Example -

```
<ul>
  <li><img src="images/pnk.gif" alt="pink" id="pink"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
</ul>
<script>
  document.getElementById('pink').addEventListener('click', function(e) {
    console.log(e);
    console.log(e.type);
    console.log(e.timeStamp);
    console.log(e.defaultPrevented);
  }, false);
</script>
```

Output-

```
▶ MouseEvent {isTrusted: true, screenX: 553, screenY: 189, clientX: 553, clientY: 123…}
click
1570.845
false
```

Targeting info



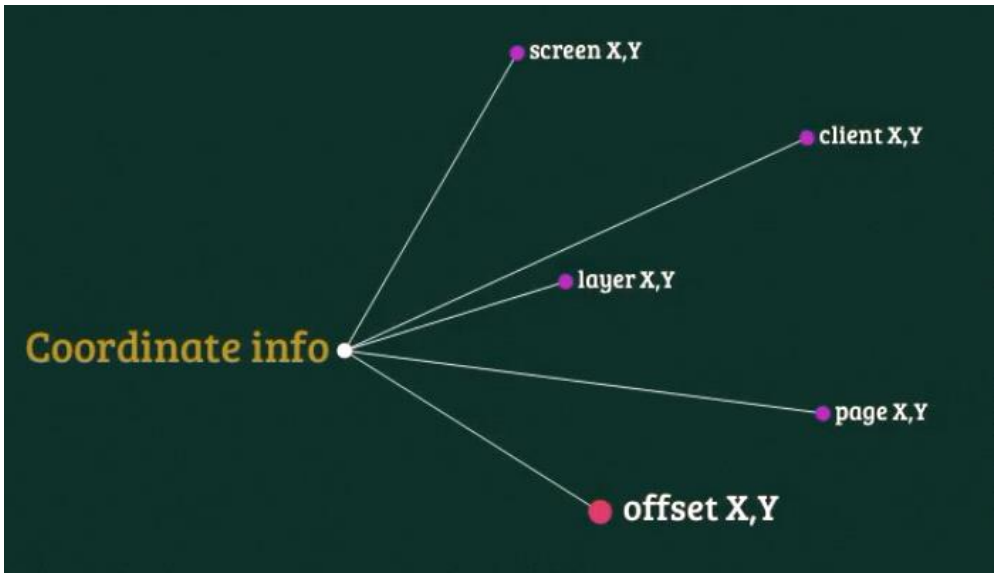
- Target gives node which has generated event.

- We can use it to check type at run time, like

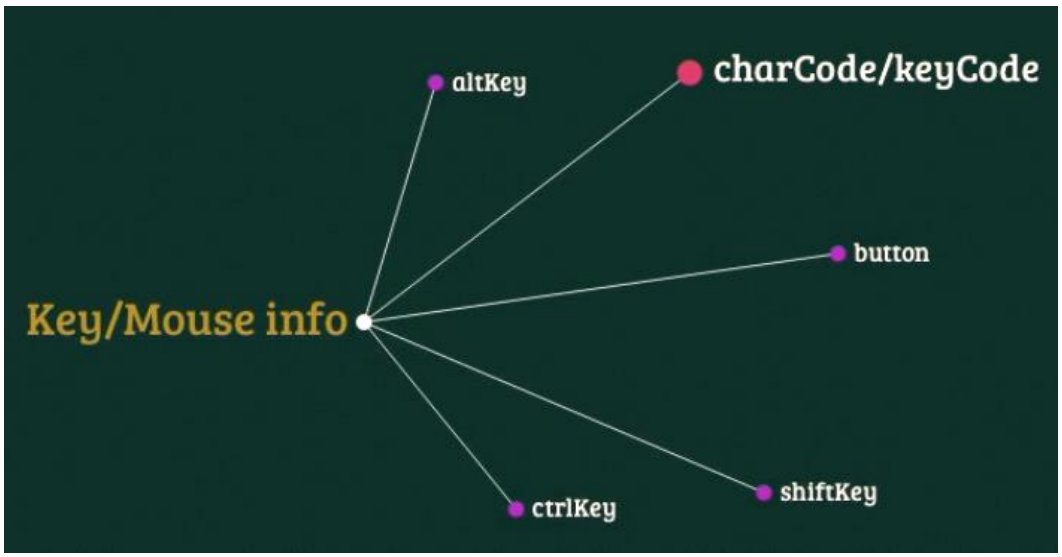
```
if (e.target.tagName === 'IMG') {
```

- tagName will give name of tag.

Coordinate info



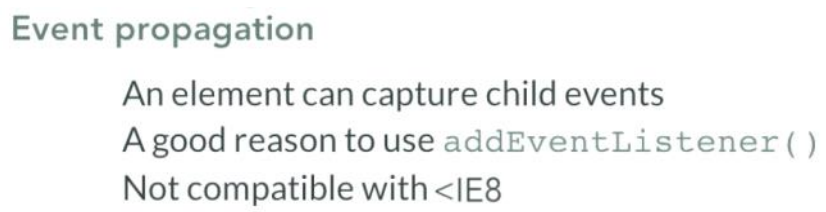
Key and Mouse info



Event Propagation

Wednesday, February 8, 2017 2:19 AM

- Event propagation allows an element to handle events of its child.



- Event propagation works with **addEventListener()** only.

Example -

```
<ul>
  <li><img src="images/ylw.gif" alt="yellow"></li>
  <li><img src="images/org.gif" alt="orange"></li>
  <li><img src="images/ppl.gif" alt="purple"></li>
  <li><img src="images/blu.gif" alt="blue"></li>
  <li><img src="images/pnk.gif" alt="pink" id="pink"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
  <li><img src="images/gry.gif" alt="gray"></li>
  <li><img src="images/red.gif" alt="red"></li>
</ul>
<script>
  document.getElementById('pink').addEventListener('click', function(e) {
    console.log(e);
  }, false);
</script>
```

- Here have event for **'#pink**' only, so if we wanted to have same event to all images, we have two ways
  - Make for every one
  - Use event propagation i.e. give event to parent instead.

event propagation

```
<ul id="grid">
  <li><img src="images/ylw.gif" alt="yellow"></li>
  <li><img src="images/org.gif" alt="orange"></li>
  <li><img src="images/ppl.gif" alt="purple"></li>
  <li><img src="images/blu.gif" alt="blue"></li>
  <li><img src="images/pnk.gif" alt="pink"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
  <li><img src="images/gry.gif" alt="gray"></li>
  <li><img src="images/red.gif" alt="red"></li>
</ul>
<script>
  document.getElementById('grid').addEventListener('click', function(e) {
    console.log(e);
  }, false);
</script>
```

- Now we have make that event to all's parent, which will automatically propagates to its children.

  OUTPUT - on clicking any of the image.

```
▶ MouseEvent {dataTransfer: null, toElement: img, fromElement: null, y: 216, x: 613…}
```

- Notice here the attribute **'toElement'** , which contain all information about the event invoker.

```
▼ toElement: img
    accessKey: ""
    align: ""
    alt: "ygreen"
  ▶ attributes: NamedNodeMap
    baseURI: "file:///Users/rayvillalobos/Desktop/events/index.html"
    border: ""
    childElementCount: 0
```

- Here we can see we can get information about the actual invoker using property inside **toElemenet.**

```
<ul id="grid">
  <li><img src="images/ylw.gif" alt="yellow"></li>
  <li><img src="images/org.gif" alt="orange"></li>
  <li><img src="images/ppl.gif" alt="purple"></li>
  <li><img src="images/blu.gif" alt="blue"></li>
  <li><img src="images/pnk.gif" alt="pink"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
  <li><img src="images/gry.gif" alt="gray"></li>
  <li><img src="images/red.gif" alt="red"></li>
</ul>
<script>
  document.getElementById('grid').addEventListener('click', function(e) {
    console.log(e.toElement.alt);
  }, false);
</script>
```

- Here we are accessing the 'alt' of element which has encountered the event.

PROBLEM WITH EVENT PROPAGATION

Capturing vs Bubbling

Browsers disagree on the order

Capturing goes down the DOM
Bubbling goes up the DOM

- Different browsers have different machanism for event propagation.
  1- Capturing (event noticed at uppermost level and goes down)
  2- Bubbling (event noticed at the lowermost level and goes up)

- For example in example above '**capturing**' starts goes in following order
  'ul->li->img'
- In **'Bubbling'** it goes from lowermost label to the uppermost side.
  'img->li->ul'

SOME THEORY

The `addEventListener()` method is implemented by using three arguments.

The first argument is the event type. Unlike the other two event handling methods, `addEventListener()` just wants the name of the event, without the `on` prefix.

The second argument is the function to call when the event happens. As with the event properties method of event handling, it's important to not use the parentheses here in order for the function to be assigned to the event handler, rather than the result of running the function.

The third argument is a Boolean value (`true` or `false`) that indicates the order in which event handlers execute when an element with an event has a parent element that also is associated with an event.

When elements are nested, it's important to know which one will happen first. Figure 11-2 illustrates a common problem: The outer square is clickable, but so is the inner circle. When you click on the inner circle, should the event attached to the square happen first, or should the event attached to the circle happen first?

Most people would say that it makes sense that the circle event should happen first. However, when Microsoft implemented its version of events in Internet Explorer, it decided that the outer event (the square) should happen first.

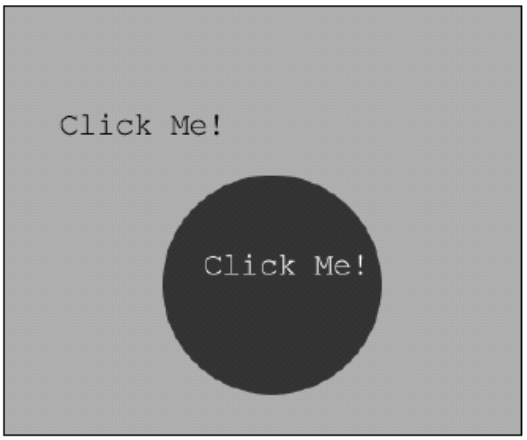


Figure 11-2: Events within events.

- Microsoft IE follow **capturing** by default(when we use other methods to register other than addEventListener()), and chrome uses **bubbling** by default so it is good practice to handle it by ourself using **addEventListerner**'s third argument.
  - False - bubbling up (going upward toward parents)
  - True - capturing (going down)

The most common way for events to be handled in a situation like the one in Figure 11-2 is called *bubbling up*. Events on the inside-most element happen first and then bubble up to the outermost elements. To use the bubble up method, set the last argument of the `addEventListener()` method to false, which is also the default value.

The other way to handle this scenario is called the *capture* method. In capture mode, the outermost events happen first, and the innermost events happen last.

Example - Bubbling

- Of course the problem happens only when there is nested event registration for same event, then we have to decide which one to invoke first.

```
<ul id="grid">
  <li><img src="images/ylw.gif" alt="yellow"></li>
  <li><img src="images/org.gif" alt="orange"></li>
  <li><img src="images/ppl.gif" alt="purple"></li>
  <li><img src="images/blu.gif" alt="blue"></li>
  <li><img src="images/pnk.gif" alt="pink" id="pink"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
  <li><img src="images/gry.gif" alt="gray"></li>
  <li><img src="images/red.gif" alt="red"></li>
</ul>
<script>
  document.getElementById('grid').addEventListener('click', function(e) {
    console.log("Clicked inside the UL");
  }, false);

  document.getElementById('pink').addEventListener('click', function(e) {
    console.log(e.toElement.alt);
  }, false);

</script>
```

- Now as third element is false, so bubbling will happen, as 'click' is here for grid and pink.
- So when click occurs it first come to img and then come to #grid

OUTPUT -

Case 1 - when element other than #pink is clicked

```
Clicked inside the UL
>
```

Case 2 - when element #pink is clicked(bubbling will happen)

```
pink
Clicked inside the UL
>
```

Example - Capturing

- By setting parameters to 'true'.

```
<ul id="grid">
  <li><img src="images/ylw.gif" alt="yellow"></li>
  <li><img src="images/org.gif" alt="orange"></li>
  <li><img src="images/ppl.gif" alt="purple"></li>
  <li><img src="images/blu.gif" alt="blue"></li>
  <li><img src="images/pnk.gif" alt="pink" id="pink"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
  <li><img src="images/gry.gif" alt="gray"></li>
  <li><img src="images/red.gif" alt="red"></li>
</ul>
<script>
  document.getElementById('grid').addEventListener('click', function(e) {
    console.log("Clicked inside the UL");
  }, true);

  document.getElementById('pink').addEventListener('click', function(e) {
    console.log(e.toElement.alt);
  }, true);

</script>
```

OUTPUT -

Case 1 - when element other than #pink is clicked

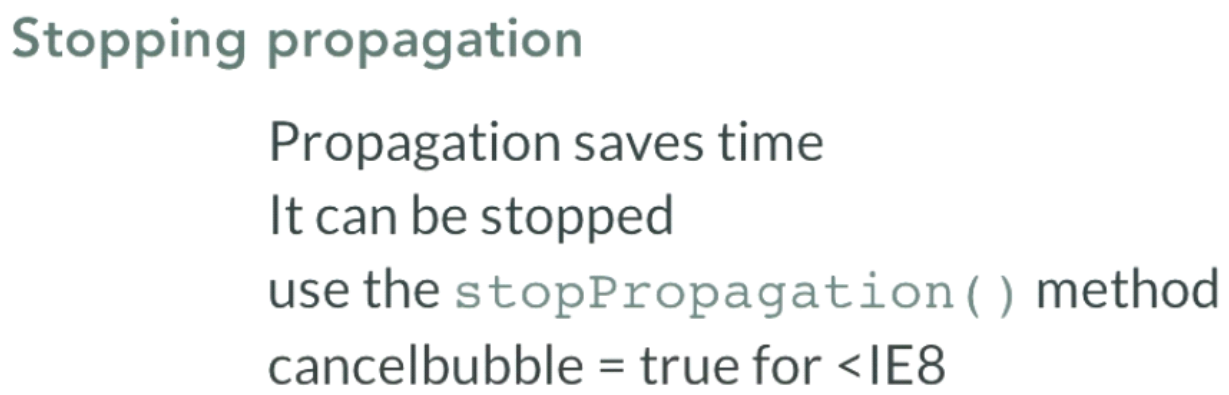
```
Clicked inside the UL
>
```

Case 2 - when element #pink is clicked

```
Clicked inside the UL
pink
```

Stopping Propagation

Thursday, February 9, 2017 2:03 AM



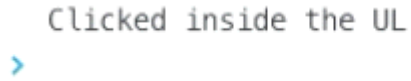
Example -

```
<ul id="grid">
  <li><img src="images/ylw.gif" alt="yellow"></li>
  <li><img src="images/org.gif" alt="orange"></li>
  <li><img src="images/ppl.gif" alt="purple"></li>
  <li><img src="images/blu.gif" alt="blue"></li>
  <li><img src="images/pnk.gif" alt="pink" id="pink"></li>
  <li><img src="images/grn.gif" alt="green"></li>
  <li><img src="images/ygr.gif" alt="ygreen"></li>
  <li><img src="images/gry.gif" alt="gray"></li>
  <li><img src="images/red.gif" alt="red"></li>
</ul>
script>
document.getElementById('grid').addEventListener('click', function(e) {
  console.log("Clicked inside the UL");
}, false);

document.getElementById('pink').addEventListener('click', function(e) {
  console.log(e.toElement.alt);
  e.stopPropagation();
}, false);
```

- Here we have added **stopPropagation()** on event object.

OUTPUT -
Case 1 - when element other than #pink is clicked



Case 2 - when element #pink is clicked(bubbling will happen)

- So we have successfully stopped the propagation of event.

Prevent Default Behaviour

Thursday, February 9, 2017 2:07 AM

Preventing default behavior

Events can have consequences
Clicking on links
Submitting forms
Prevent a behavior with `preventDefault()`

- Many items have default behaviour like
  - Link - took us to that website/target
  - Submit Form - take data and pass to Post
  - When we right-click it comes default contex-menu
- Actually when we click link, event is occurred and that event have default behaviour of going to that target.
- Some time we wanted to prevent default behaviour of events.

Example -

```
<ul id="grid">
  <li><a href="http://iviewsource.com"><img src="images/ylw.gif" alt="yellow"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/org.gif" alt="orange"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/ppl.gif" alt="purple"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/blu.gif" alt="blue"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/pnk.gif" alt="pink" id="pink"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/grn.gif" alt="green"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/ygr.gif" alt="ygreen"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/gry.gif" alt="gray"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/red.gif" alt="red"></a></li>
</ul>
<script>
document.getElementById('grid').addEventListener('click', function(e) {
  console.log("Clicked inside the UL");
}, false);

document.getElementById('pink').addEventListener('click', function(e) {
  console.log(e.toElement.alt);
  e.stopPropagation();
}, false);
```

- Now if we click the image it will take us to the website , we will prevent It by telling to prevent default behaviour.

```
<ul id="grid">
  <li><a href="http://iviewsource.com"><img src="images/ylw.gif" alt="yellow"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/org.gif" alt="orange"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/ppl.gif" alt="purple"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/blu.gif" alt="blue"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/pnk.gif" alt="pink" id="pink"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/grn.gif" alt="green"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/ygr.gif" alt="ygreen"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/gry.gif" alt="gray"></a></li>
  <li><a href="http://iviewsource.com"><img src="images/red.gif" alt="red"></a></li>
</ul>
script>
document.getElementById('grid').addEventListener('click', function(e) {
  e.preventDefault();
  console.log("Clicked inside the UL");
}, false);

document.getElementById('pink').addEventListener('click', function(e) {
  e.preventDefault();
  console.log(e.toElement.alt);
  e.stopPropagation();
}, false);
```

Removing Events

Thursday, February 9, 2017 9:13 PM

HTML DOM removeEventListener() Method

Example

Remove a "mousemove" event that has been attached with the addEventListener() method:

```
// Attach an event handler to <div>
document.getElementById("myDIV").addEventListener("mousemove", myFunction);

// Remove the event handler from <div>
document.getElementById("myDIV").removeEventListener("mousemove", myFunction);
```

Definition and Usage

The removeEventListener() method removes an event handler that has been attached with the addEventListener() method.

Note: To remove event handlers, the function specified with the addEventListener() method must be an external function, like in the example above (myFunction).

Anonymous functions, like "*element*.removeEventListener("*event*", function(){ *myScript* });" will not work.

NOTE -

- So for working of removeEventListener(), the function passed to it should be same as addEventListener() , so obviously it need to be named otherwise we can't pass as we will loose it's refrence.
- So the parameters of **addEventListener()** and **removeEventListener()** need to be matched **perfectly.**

Syntax

```
element.removeEventListener(event, function, useCapture)
```

Parameter Values

Parameter	Description
event	Required. A String that specifies the name of the event to remove. **Note:** Do not use the "on" prefix. For example, use "click" instead of "onclick".
function	Required. Specifies the function to remove.
useCapture	Optional. A Boolean value that specifies the event phase to remove the event handler from.

Removing DOM elements with events

Thursday, February 9, 2017 8:40 PM

See video in which we see how to remove element with click

Creating Element with event

Thursday, February 9, 2017 8:58 PM

See video in which we see how to create lement with click

SEE VIDEO ONWARDS