Browser Events

* events
* respond to events
* data stored in events
* stop an event
* lifecycle of events

* Events

An event is a signal that something has happened. All DOM nodes generate such signals (but events are not limited to DOM).

Ex - click, submit, mouseover, etc.

→ moniterEvents() is a property by which we can check all the events that are happening at a particular time.

By using the unmonitorEvents() we can turn off the events.

* Event Target

It is a interface (blueprint) implemented by objects that can recieve & may have

listerers for them.

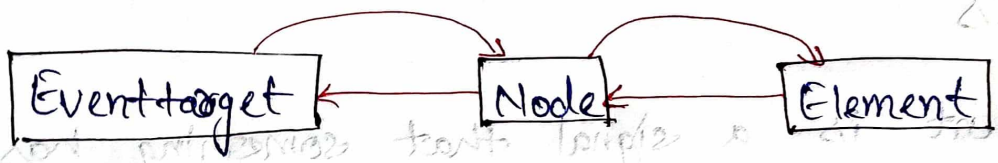
It consist of 3 methods

→ addEventListener()

→ removeEventListener()

→ dispatchEvent()

Event target is a top level interface.



→ Node is inheriting the properties of Event target

→ ~~Event~~ Element is inheriting the properties of Event target as well as Node.

* addEventListener()

→ Listen to a event

→ Respond to event

→ Hook into event

Pseudocode

<event.target>.addEventListener(<event-to-listen-for>,
<function-to-run-when-event-happened>);

⇒ To apply a event Listener we need to have a:

i) event-target → component (document, div, p, article)

ii) event-type → click, scroll

iii) function → defines what to do when event happened

Ex

```
document.addEventListener('click', function() {
    console.log('I clicked!');
});
```

* .removeEventListner()

This method of EventTarget interface removes an event listener previously registered with EventTarget.addEventListener() from the target.

Ex

```
document.removeEventListner('click', eventFunction)
```

It is a function which needs to be defined before execution.

In order to successfully run a removeEventListner() we need 3 conditions to be correct.

i) same ~~target~~
ii) same type
iii) same function

What does same function refer to?

```
document.addEventListner('click', function() {
                console.log('Hi');
});
```
①

```
document.removeEventListner('click', function() {
                console.log('Hi');
});
```
②

① & ② are different function and it will not able to remove eventlistener.

```
function print() {
        console.log('hi');
}
document.addEventListner('click', print);
document.removeEventListner('click', print);
```

It will work perfectly fine because here, the function is same that is being refered.

* .dispatchEvent()

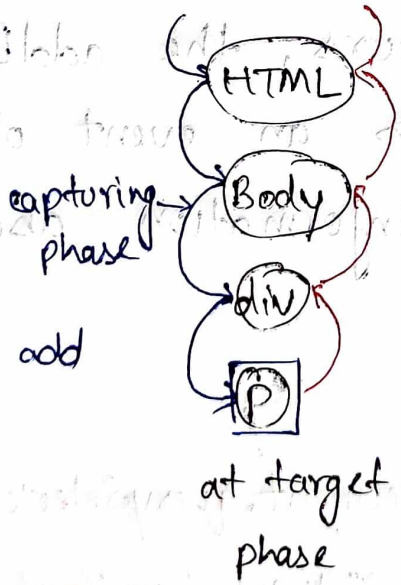
This method of eventtarget sends an Event to the object, invoking the affected EventListners in the appropriate order.

Calling dispatchEvent() is the last step in to firing an event. It should have already been created. and initialized.

=> .dispatchEvent(event)

* Phases of an Event
  * Capturing phase
  * At target phase
  * Bubbling phase

Ex



Here we are finding <p> to add a listner

Let say we want to add listener to <p>

```
<html>
<body>
  <div id="wrap">
    <p> ---- </p>   (Listner)
  </div>
</body>
</html>
```

By default addEventListener() will execute in the bubbling phase.

If I want to execute my event listner to the capturing phase.

Then I will use 3rd parameter.

.addEventListner('click', print, true)

here true value will turn on the capturing phase

NOTE – No we can't apply an event listner on Target phase.

* The Event Object

→ Browser sends the events

When an event occurs, the addEventListner( ) function gets an event object with a lot of information about that ~~object~~ event.

Ex

```
const content = document.querySelector('#wrappe');
content.addEventListner('click', function(event) {
    console.log(event);
});
```

Note: Here 'event' is just a name, you can write anything.

* The default action

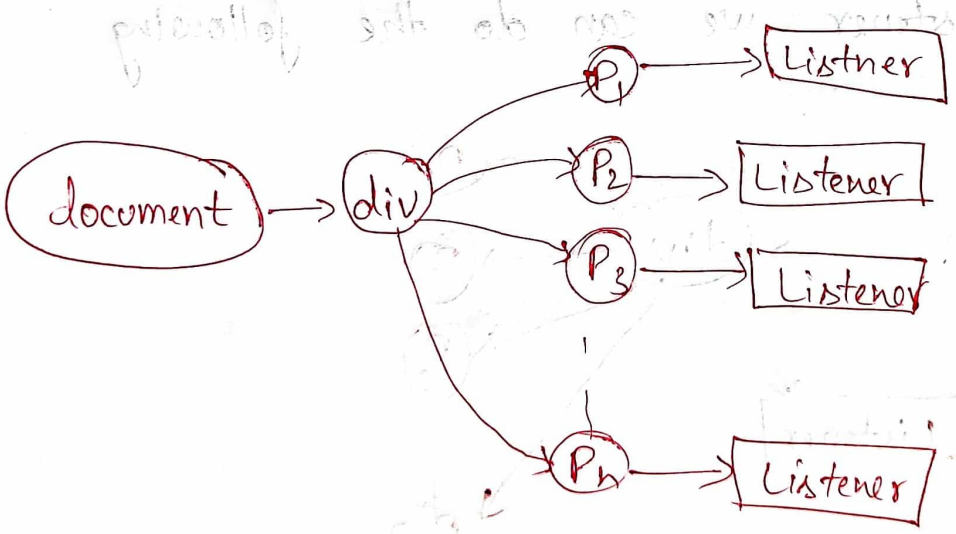
Let say if there is an <a> and its default action is to open a link.

Here if we want we can prevent the default of the <a>, not to open the link.

⟹ .preventDefault()

With the use of this method of the Event interface we can prevent any default action of the tag.

* How to avoid too many Events?

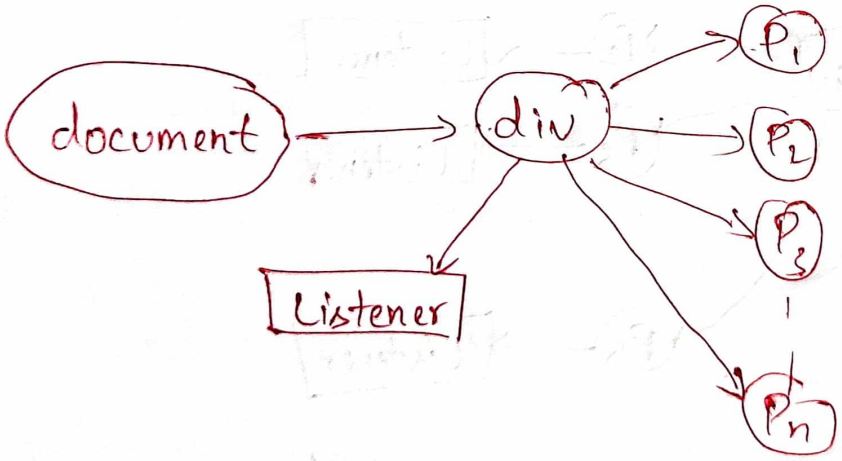


In the above diagram same listner is attached to many paragraph with different objects, so the memory usage is high.

Code

```
let myDiv = document.createElement('div');
for (let i=1 ; i<=100 ; i++) {
    let newElement = document.createElement('p');
    newElement.textContent = 'This is para '+i;
    newElement.addEventListner('click', function(event) {
        console.log('I clicked para');
    });
    myDiv.appendChild(newElement);
}
document.body.appendChild(myDiv);
```

* To avoid creating multiple objects for a same listener we can do the following



Instead of creating multiple mappings we created only 1 mapping for div.

But here is a issue, we cannot access each paragraph individually.

Now if we click anywhere in the , the listener will execute.

We lost the individuality of the paragraph.

⟹ Now we will make use of phase to avoid the issue of lost of individuality.

Code

```
let myDiv = document.createElement('div');

function paraStatus(event) {
        console.log('I clicked para');
}
myDiv.addEventListener('click', paraStatus);
for(let i=0; i<=100; i++) {
        let newElement = document.createElement('p');
        newElement.textContent = 'This is para ' + i;

        myDiv.appendChild(newElement);
}
document.body.appendChild(myDiv);
```

* The target Property

To gain the individuality of the paragraph we will use the target property

Definition

The target property returns the element where the event occured.

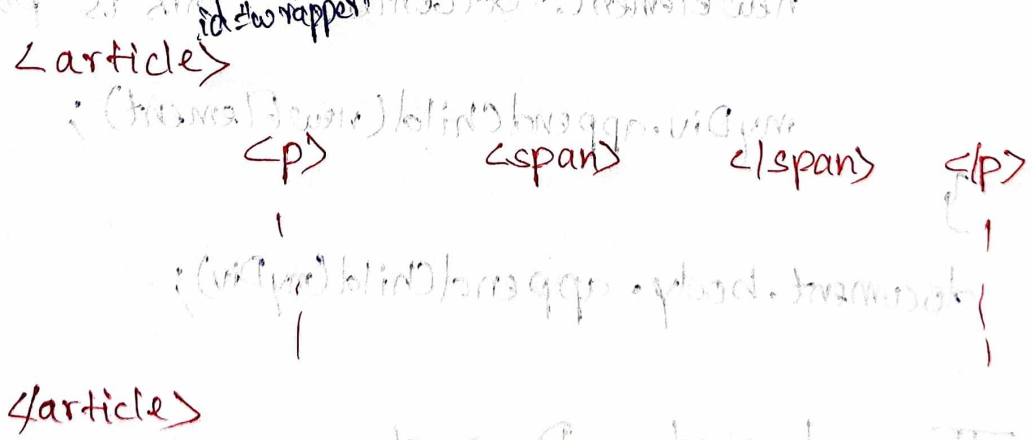
Code

Here the entire code will be same, the only change occur in function.

```
function paraStatus(event) {
    console.log("Para ", event.target.textContent);
}
```

Now we can access individual para by using eventListener on a div.

But there is a catch in it if we modify it little bit.



The issue in above structure is that we want code to run when we click only on span but when we click on para the code is still running.

We don't want listener to execute when we click on <p>.

We can avoid the above problem by using the property .nodeName

Code (code with problem)

```
let element = document.querySelector("#wrapper");

element.addEventListner('click', function(event) {
        console.log('I clicked span' + event.target.textContent);
});
// Here the code will run in both cases i.e. <p> & <span>
```

Code (without problem)

```
let element = document.querySelector('#wrapper');

element.addEventListener('click', function(event) {
    if (event.target.nodeName === 'SPAN') {
        console.log('I clicked span' + event.target.textContent);
    }
}); // Here the code will run only when we click on <span> tag.
```

NOTE: Always use <script> tags in the last of <body> tag as it is a best practice.