Programmation événementielle en JS

SOMMAIRE

Définitions

Programmation événementielle

Paradigme de programmation fondé sur les événements

Analogie “paradigme de programmation”

Décider soi-même d’aller faire les courses → Programmation impérative
Aller faire les courses parce qu’ils manque des vivres → Programmation événementielle

Événement

Survenue de quelque chose

Événement → pages réactives

→ qui réagissent aux actions de l’utilisateur

Exemple d’événements en JS

Fin de chargement de la page, Touche du clavier pressée,
Double clic sur un bouton de la souris, Redimensionnement de la fenêtre du navigateur
Lecture ou mise en pause d’une vidéo, Envoi d’un formulaire...

Événements et DOM

Tous les noeuds du DOM émettent des événements

Les événements héritent de l’interface Event

Les événements héritent souvent de l’interface du type d'évènement dont ils font partie

Il existent des 100aine d’évènements différents et des dizaines de types d'événements

Quelques types d’évènements

MouseEvent → Événement de la souris
KeyboardEvent → Événement du clavier
FetchEvent → Événement de requête HTTP
DragEvent → Événement de glisser/déposer

Gestionnaire d’événement

Pour gérer les événements, il faut:
Les écouter → event listener
Y réagir → callback (gestionnaire d’événement)

Gestionnaire d’événement en pratique

```css
div {
  height: 300px;
  width: 300px;
  border: 1px solid gray;
  padding: 10px;
  font-family: sans-serif;
}
```

Les noeuds ont des propriétés on<event> (default value → null)

```js
const divTag = document.querySelector('div');

divTag.onclick = () => console.log('Event detected !');
divTag.ondblclick = () => console.log('Event detected !');
// div contenteditable
divTag.onkeydown = () => console.log('Event detected !');
divTag.onblur = () => console.log('Event detected !');
```

Gestionnaire d’événement en pratique

Gestionnaire écrasé !

```
const divTag = document.querySelector('div');

divTag.onclick = () => console.log('Event detected !');
divTag.onclick = () => console.log('Once again !');
```

Solution → addEventListener()

```
divTag.addEventListener('click', () => console.log('Event detected !'));
divTag.addEventListener('click', () => console.log('Once again !'));
```

addEventListener() en détail

Syntaxe

node.addEventListener(event, listener, options?);

Listener

<function> → Reçoit l'event en 1er argument
<object>.handleEvent() → Doit avoir une propriété handleEvent

Boolean | AddEventListenerOptions

BOOLEAN:
useCapture → Propagation de l'événement vers le noeud parent

OU OBJET:
once → Suppression du listener après son appel
capture → Propagation de l'événement vers le noeud parent
passive → Le listener n'appellera jamais preventDefault()

Gestionnaire multi event

```
const divTag = document.querySelector('div');

const MOUSE_ENTER_EVENT = 'mouseenter';
const MOUSE_LEAVE_EVENT = 'mouseleave';

const eventManager = {
  handleEvent(e) {
    switch (e.type) {
      case MOUSE_ENTER_EVENT: return divTag.innerText = 'Aaaah, une souris !';
      case MOUSE_LEAVE_EVENT: return divTag.innerText = 'Ouf, elle est partie !';
    }
  }
}

divTag.addEventListener(MOUSE_ENTER_EVENT, eventManager);
divTag.addEventListener(MOUSE_LEAVE_EVENT, eventManager);
```

Valeur de this

this → lexical environment

```
divTag.addEventListener('click', () => {
  console.log(this); // window
});
```

this → execution context

```
divTag.addEventListener('click', function() {
  console.dir(this); // div
});
```

this → listener

```
divTag.addEventListener('click', {
  handleEvent() { console.log(this) }
});
```

Suppression d’un gestionnaire d’événement

Syntaxe

node.removeEventListener(event, id-listener, options?);

id-listener

L’identifiant du gestionnaire d’événement

Boolean | AddEventListenerOptions

Permet une plus grande précision si le même gestionnaire est utilisé plusieurs fois pour le même événement

Événement perso avec constructeur générique

```
const div = document.querySelector('div');
const p = document.querySelector('p');
const bt = document.querySelector('button');

div.addEventListener('click', () => console.log('DIV'));
p.addEventListener('click', () => console.log('P'));
bt.addEventListener('click', () => console.log('BT'));

const myEvent = new Event( // Generic constructor
  'click',
  {bubbles: false, cancelable: false} // Default values
);

setTimeout(() => {
  bt.dispatchEvent(myEvent);
}, 1000);
```

Event → constructeur générique
1er param → type (natif ou perso)
2ème param → options (optionnelles)

Options
bubbles → Active la propagation cible → parent
cancelable →

node.dispatchEvent(customEvent)
Génère l'événement

Événement perso

(type natif)

```
const div = document.querySelector('div');

div.addEventListener('click', (e) => console.log(e.clientX));

const myEvent = new MouseEvent( // Specific constructor
  'click',
  {
    bubbles: false,
    cancelable: false,
    clientX: 10,
    clientY: 10,
  }
);

setTimeout(() => {
  div.dispatchEvent(myEvent);
}, 1000);
```

<Type>Event → constructeur spécifique
Différence avec Event → propriétés spécifiques

Options
bubbles → Active la propagation cible → parent
cancelable →

node.dispatchEvent(customEvent)
Génère l'événement

```
const div = document.querySelector('div');

div.addEventListener('hello', (e) => {
  console.log(
    `Salut %c ${e.detail.firstname} %c !`,
    'background: red; color: white',
    null
  );
});

const myEvent = new CustomEvent(
  'hello',
  {
    bubbles: false,
    cancelable: false,
    detail: {
      firstname: 'Coco'
    }
  }
);

setTimeout(() => {
  div.dispatchEvent(myEvent);
}, 1000);
```

Événement perso

(type perso)

CustomEvent
Permet de passer des données perso (propriété detail)

Options
bubbles → Active la propagation cible → parent
cancelable →

node.dispatchEvent(customEvent)
Génère l'événement

Bouillonnement et Capture

Problématique

Lorsque plusieurs noeuds imbriqués ont chacun un écouteur d'événement
quels listeners seront déclenchés et dans quel ordre

```html
<div>
  <p>
    <button>TEST</button>
  </p>
</div>
```

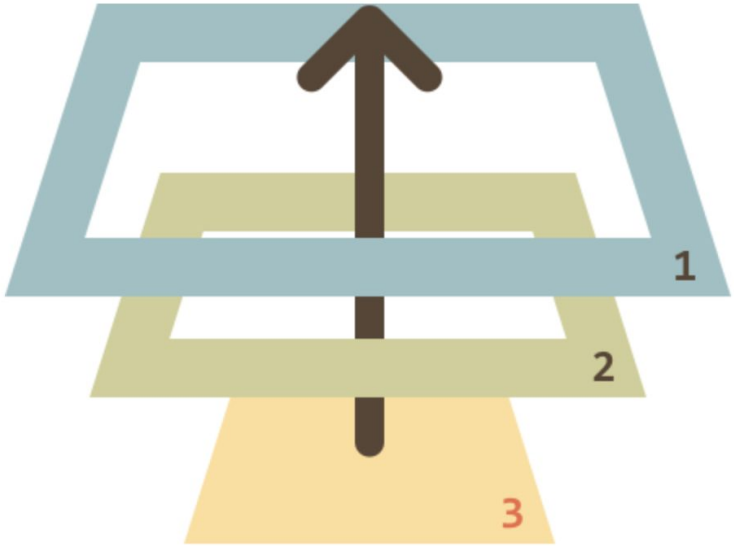
```js
const div = document.querySelector('div');
const p = document.querySelector('p');
const bt = document.querySelector('button');

div.addEventListener('click', e => console.log(e.currentTarget.nodeName));
p.addEventListener('click', e => console.log(e.currentTarget.nodeName));
bt.addEventListener('click', e => console.log(e.currentTarget.nodeName));
```

Bouillonnement et Capture

Phase de bouillonnement

Tous les écouteurs sont déclenchés
depuis le noeud qui déclenche l'événement (le + imbriqué) vers le parent
(Comportement par défaut de la plupart des événements)



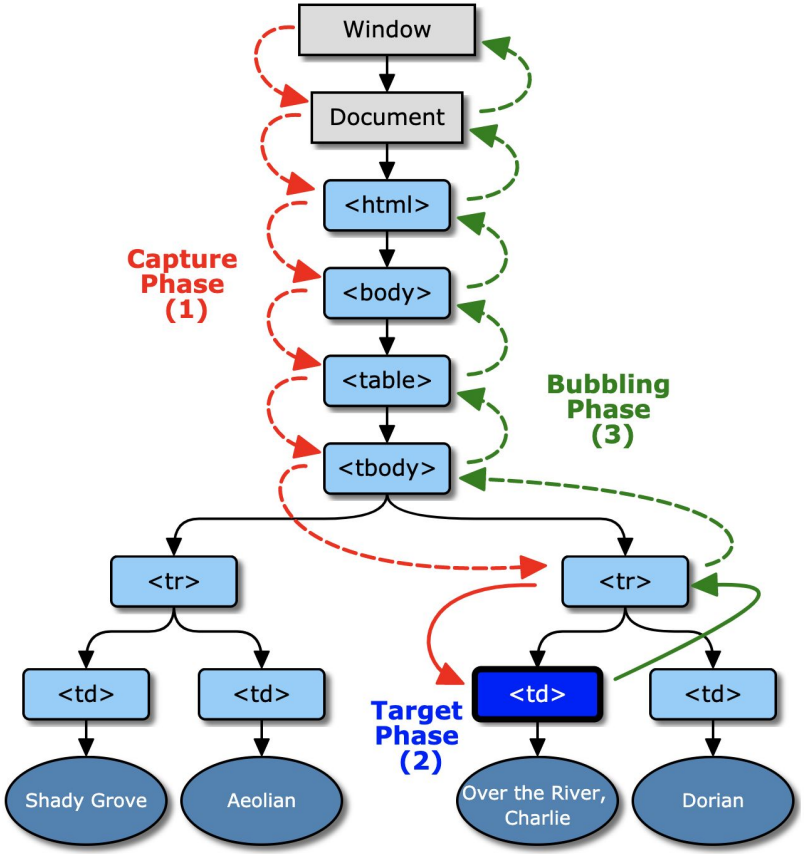
Bouillonnement et Capture

Phase de capture

Tous les écouteurs sont déclenchés
depuis le noeud parent vers le noeud enfant
(Rarement utilisée)

```
div.addEventListener('click',
  e => console.log(e.currentTarget.nodeName), {
    capture: true
  }
);
p.addEventListener('click', e => console.log(e.currentTarget.nodeName));
bt.addEventListener('click', e => console.log(e.currentTarget.nodeName));
```

Bubbling phase vs Capture phase



Graphical representation of an event dispatched in a DOM tree using the DOM event flow

Pattern de délégation d'événement

1 seul gestionnaire sur 1 seul type d'événement → gestion de plusieurs éléments

```
<div class="container">
  (section>h2{Titre $}+(p>lorem)+button>{Supprimer})*3
</div>
```

```
const container = document.querySelector('.container');

container.addEventListener('click', e => {
  if (e.target.nodeName !== 'BUTTON') return;
  e.target.closest('section').remove();
});
```

Propriétés / méthodes de l’objet event

type → Type de l’événement

target → Référence à l’élément émetteur le plus imbriqué dans la phase de bouillonnement

currentTarget → Référence à l’élément dont le gestionnaire est en cours d’exécution

isTrusted → True si l’événement a été déclenché par l’utilisateur

eventPhase → Phase courante (enum: NONE, CAPTURING_PHASE, AT_TARGET, BUBBLING_PHASE)

defaultPrevented → True si le comportement par défaut a été annulé (e.preventDefault())

timeStamp → Date exacte de la création de l’événement (ms avec une précision de 5 µs)

stopPropagation() → Stoppe la propagation de l’événement

preventDefault() → Empêche le comportement par défaut du navigateur

```
const link = document.querySelector('a');

link.addEventListener('click', e => {
  e.preventDefault();
});
```

FIN