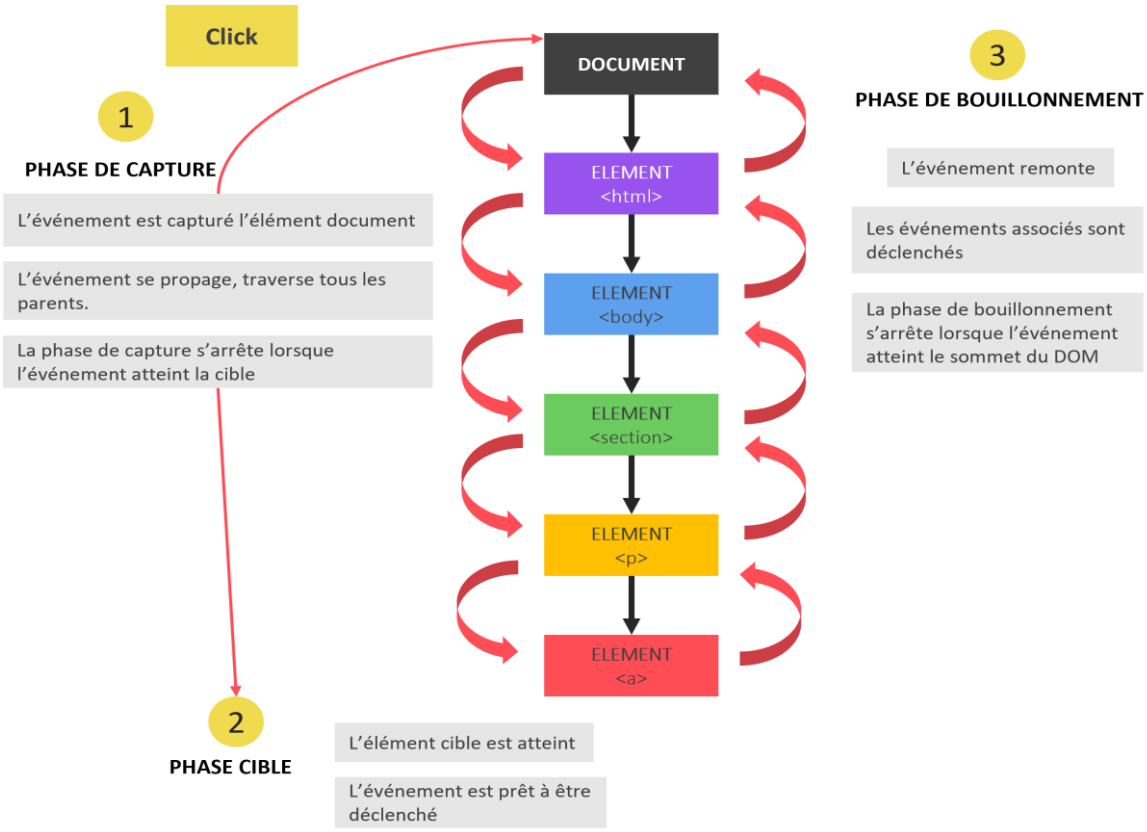
La Propagation des Événements en JavaScript



Lorsqu'un événement est déclenché sur un élément du DOM, il suit un processus en trois étapes appelé **propagation d'événements**. Cette propagation permet à l'événement de traverser différents niveaux d'éléments dans le DOM, depuis le document racine jusqu'à l'élément cible, puis de remonter en sens inverse. Voici les trois phases :

1. Phase de Capture

Lorsqu'un événement est déclenché (par exemple, un clic sur un bouton dans un paragraphe), il est d'abord capturé par le document, puis traverse les éléments parents, de la racine du DOM jusqu'à l'élément cible en descendant dans l'arborescence. Cette première phase est appelée **capture**.

2. Phase de la Cible

Une fois que l'événement atteint l'élément cible (c'est-à-dire l'élément exact sur lequel l'utilisateur a cliqué ou interagit), l'événement est prêt à être déclenché. À ce stade, bien que l'événement ait atteint la cible, il n'est pas encore déclenché. Le déclenchement réel de l'événement se produit au début de la phase suivante.

1. Phase de Bouillonnement

La **phase de bouillonnement** commence après que l'événement a atteint la cible. C'est au cours de cette phase que l'événement se déclenche réellement et remonte vers le document, en

traversant chaque élément parent de l'élément cible. Ainsi, après l'élément cible <button>, l'événement remonte successivement à travers <p>, <section>, <body>, <html>, et enfin le document. Pendant cette phase de bouillonnement, si des écouteurs d'événements du même type sont attachés aux éléments parents de l'élément cible, ils seront également exécutés.

Exemple : Démonstration des Phases de Propagation

Prenons la structure HTML suivante pour observer la propagation d'un événement :

```
<!DOCTYPE html>
<html>
<head>
   <title>Document</title>
</head>
<body>
   <section>
       <p>Un paragraphe avec un <button>Click</button>.</p>
   </section>
</body>
</html>
```

Pour voir comment un événement se propage dans cette structure, ajoutons des écouteurs d'événements aux éléments <section>, <p>, et <button>. Chaque écouteur enregistrera un message dans la console lorsqu'un événement se déclenche sur ces éléments.

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

section.addEventListener('click', function() {
   console.log("Événement déclenché sur l'élément <section>");
});

const p = document.querySelector('p');

p.addEventListener('click', function() {
   console.log("Événement déclenché sur l'élément <p>");
});

const button = document.querySelector('button');

button.addEventListener('click', function() {
   console.log("Événement déclenché sur l'élément <button>");
});
```

Lorsque vous cliquez sur le bouton <button>, vous verrez l'ordre suivant dans la console :

```
console.log("Événement déclenché sur l'élément <button>");
console.log("Événement déclenché sur l'élément <p>");
console.log("Événement déclenché sur l'élément <section>");
```

Cela montre la phase de **bouillonnement** : l'événement se déclenche d'abord sur l'élément enfant (<button>) et remonte ensuite à travers chaque parent dans l'ordre : <p>, puis <section>.

Ce phénomène de remontée, appelé **bouillonnement**, signifie que l'événement, après avoir été déclenché sur l'élément cible, remonte dans la hiérarchie du DOM et déclenche les événements associés sur les éléments parents, dans l'ordre de l'enfant vers le parent.

Dans cet exemple :

1. L'événement se déclenche sur le bouton <button>, où il est capté en premier.
2. Il remonte ensuite au paragraphe <p>, où le message correspondant est affiché.
3. Enfin, l'événement atteint l'élément <section>, où le dernier message est affiché.

Ainsi, cet exemple illustre bien le fonctionnement de la phase de bouillonnement, où les événements se propagent de l'élément cible vers ses ancêtres dans le DOM.

Délégation d'événements : Principe et Utilité

La délégation d'événements consiste à attacher un écouteur d'événements à un élément parent pour gérer les événements de plusieurs éléments enfants. Cela réduit le nombre d'écouteurs d'événements nécessaires et améliore les performances, surtout dans des cas où de nombreux éléments doivent réagir aux mêmes interactions. Voici les deux principaux cas où la délégation d'événements est particulièrement utile :

Cas 1 : Attacher le même événement à plusieurs éléments HTML

Si plusieurs éléments enfants nécessitent un même comportement — par exemple, si tous les boutons d'une liste doivent afficher une alerte au clic — on peut ajouter un seul écouteur d'événements sur le parent. Cet écouteur se déclenchera chaque fois qu'un clic se produit sur l'un des enfants.

Prenons l'exemple suivant :

```
<section>
   <p>Un paragraphe avec un <button>Click</button>.</p>
</section>
```

En plaçant un écouteur d'événement click sur <section>, celui-ci se déclenchera lorsqu'un clic se produira sur le paragraphe <p> ou sur le bouton <button>. Cependant, si nous voulons que

l’événement se déclenche uniquement lorsque l'utilisateur clique sur le bouton, nous pouvons utiliser la propriété target de l’objet event fourni par addEventListener. Cela permet de cibler précisément l'élément qui a déclenché l'événement et d'ajouter une condition pour que le code ne s'exécute que sur l’élément voulu.

Par exemple, en vérifiant la propriété tagName de event.target, qui retourne le nom de la balise HTML (en majuscule) :

```
element.addEventListener("click", function(event){
   if(event.target.tagName == "BUTTON"){
      // Le code à exécuter
   }
})
```

Dans cet exemple, le code dans le bloc if ne s'exécutera que si le clic se produit sur un élément <button>. Il est également possible de faire une vérification en utilisant classList.contains pour vérifier si l’élément possède une classe spécifique.

Cas 2 : Gérer les éléments créés dynamiquement

La délégation d'événements est également utile lorsqu'on ajoute des éléments dynamiquement au DOM. Par exemple, si un élément HTML est créé après le chargement initial de la page, comme lorsqu'un formulaire est affiché suite à un clic. Dans ce cas, si vous essayez de sélectionner cet élément nouvellement créé pour lui ajouter un événement, cela ne fonctionnera pas, car le navigateur ne le reconnaît pas encore au moment de l'initialisation de la page.

Pour contourner ce problème, on peut attacher l'écouteur d'événements au parent existant, comme dans le cas précédent. Étant donné que le parent est présent dans le DOM dès le chargement de la page, le navigateur le reconnaît, et vous pouvez ainsi utiliser la délégation d'événements pour gérer les actions sur les éléments enfants, même s'ils ont été ajoutés après le chargement initial.

TP : Propagation et Délégation d'Événements

Ce TP vous guidera pour explorer les concepts de propagation d'événements et de délégation d'événements dans le DOM en JavaScript.

Objectifs :
1. Comprendre et observer les phases de propagation d'un événement.
2. Mettre en place la délégation d'événements pour des éléments existants et dynamiquement créés.

Étape 1 : Structure HTML de Base

Commencez par créer une structure HTML simple avec une section contenant un paragraphe et un bouton, ainsi qu’un bouton en dehors de la section pour ajouter des éléments dynamiques.

```
<section class="section">
  <p class="paragraph">
      Un paragraphe avec un <button class="button">Bouton</button>
   </p>
</section>

<button id="addButton">Ajouter un bouton dynamique</button>
```

Étape 2 : Observer la Phase de Bouillonnement

Dans cette étape, vous allez ajouter des écouteurs d'événements sur différents éléments pour observer la phase de bouillonnement, c'est-à-dire la remontée de l'événement à travers les éléments parents.

```
const section = document.querySelector('.section');
const paragraph = document.querySelector('.paragraph');
const button = document.querySelector('.button');

// Ajoute un écouteur d'événement à la section
section.addEventListener('click', function() {
   console.log("Événement déclenché sur l'élément <section>");
});

// Ajoute un écouteur d'événement au paragraphe
paragraph.addEventListener('click', function() {
   console.log("Événement déclenché sur l'élément <p>");
});

// Ajoute un écouteur d'événement au bouton
button.addEventListener('click', function() {
   console.log("Événement déclenché sur l'élément <button>");
});
```

Testez la propagation :

- Ouvrez la console de votre navigateur.
- Cliquez sur le bouton et observez l'ordre d'affichage dans la console.

Analyse :

- Notez comment l'événement est capté sur l'élément cible (<button>) et remonte en déclenchant les écouteurs de <p> et <section>, illustrant la phase de bouillonnement.

Étape 3 : Testez les Événements sur les Éléments Créés Dynamiquement sans délégation

Dans cette étape, vous allez ajouter un nouveau bouton dynamiquement et essayer de lui attacher un écouteur d'événement directement. Cela vous permettra d'observer les limites de cette approche lorsque des éléments sont créés après le chargement initial de la page.

Commentez l' écouteur d'événements sur la section que nous avons ajouté au debut du TP. Cela nous permettra de tester uniquement l'ajout direct d'événements sur des éléments dynamiques.

```
// section.addEventListener('click', function() {
//     console.log("Événement déclenché sur l'élément <section>");
// });
```

Ajoutez le Code pour Créer un Nouveau Bouton Dynamique :

```
const addButton = document.querySelector('#addButton');

// Événement pour ajouter un nouveau bouton dynamiquement
addButton.addEventListener('click', function() {
   // Crée un nouveau bouton avec une classe spécifique
   const newButton = document.createElement('button');
   newButton.textContent = "Nouveau Bouton";
   newButton.classList.add('new-button');

   // Ajoute le nouveau bouton à la section
   const section = document.querySelector('.section');
   section.append(newButton);
});

// Tente de sélectionner le nouveau bouton et d'ajouter un événement directement
const newBtn = document.querySelector('.new-button');

newBtn.addEventListener('click', function() {
   console.log("Clic sur le nouveau bouton");
});
```

Testez cette Configuration

- Cliquez sur **"Ajouter un bouton dynamique"** pour insérer le nouveau bouton dans la section.
- Cliquez sur le nouveau bouton et observez la console pour voir si le message "Clic sur le nouveau bouton" s'affiche.

Analyse des Limites

Vous remarquerez que le message "Clic sur le nouveau bouton" n'apparaît pas dans la console lorsque vous cliquez sur le bouton.

Pourquoi ?

L'événement click n'est pas appliqué au bouton nouvellement créé, car la ligne const **newBtn = document.querySelector('.new-button');** est exécutée une seule fois, au chargement de la page. Lorsque vous cliquez sur **"Ajouter un bouton dynamique",** un nouveau bouton est créé, mais il n'est pas pris en compte par le code JavaScript existant. Cette limitation met en évidence la difficulté d'ajouter des événements directement aux éléments dynamiques sans utiliser de délégation d'événements.

Étape 4 : Mise en Place de la Délégation d'Événements

Pour simplifier et optimiser la gestion des événements sur les éléments créés dynamiquement, configurez la délégation d'événements.

Décommentez le code d'événements sur section pour gérer les clics sur les boutons dans la section, y compris ceux ajoutés dynamiquement.

Modifiez le code comme suit :

```
section.addEventListener('click', function(event) {
   if (event.target.tagName == "BUTTON") {
       console.log("Clic sur : " + event.target.textContent);
   }
});
```

Commentez les écouteurs ajoutés spécifiquement pour le paragraphe (<p>) et le bouton (<button>) dans les premières étapes pour éviter les conflits.

- Cliquez sur **"Ajouter un bouton dynamique"** pour insérer de nouveaux boutons.
- Cliquez sur les nouveaux boutons pour vérifier que l'événement fonctionne sans ajouter de nouveaux écouteurs.

Dans ce TP, vous avez appris :

- Les phases de propagation d'un événement dans le DOM : capture, cible et bouillonnement.
- Les limitations de la gestion d'événements sans délégation pour des éléments dynamiques.
- L'utilisation de la délégation d'événements pour optimiser la gestion des événements dans des scénarios où plusieurs éléments enfants partagent le même comportement.