Gestion avancée des événements

Déléguer des événements

Le délégation d'événements est une technique qui consiste à ajouter un seul écouteur d'événement à un élément parent pour gérer les événements de tous les éléments enfants. Cela permet de réduire le nombre d'écouteurs d'événements et d'améliorer les performances de l'application.

Bubble et capture

Lorsqu'un événement est déclenché sur un élément HTML, il se propage à travers l'arbre DOM. Il existe deux phases de propagation : la phase de capture et la phase de bouillonnement (bubble).

Phase de capture

Lorsqu'un événement est déclenché sur un élément, l'événement est d'abord capturé par les éléments parents de l'élément cible. Cela signifie que l'événement est d'abord déclenché sur l'élément le plus haut dans l'arbre DOM, puis descend progressivement jusqu'à l'élément cible. Une fois l'élément cible atteint, l'événement passe à la phase de bouillonnement. Pour activer la phase de capture, il faut passer `true` comme troisième argument à la méthode `addEventListener`.

Cette phase est rarement utilisée, car la phase de bouillonnement est plus courante et plus intuitive.

```
element.addEventListener(
    "click",
    function (event) {
        // Gérer l'événement
    },
    true
);
```

Phase de bouillonnement

Lorsque l'événement atteint l'élément cible, il remonte à travers l'arbre DOM en commençant par l'élément cible et en remontant jusqu'à l'élément le plus haut dans l'arbre DOM. Cela signifie que l'événement est d'abord déclenché sur l'élément cible, puis remonte progressivement jusqu'à l'élément le plus haut dans l'arbre DOM.

Arrêter la propagation

Il est possible d'arrêter la propagation de l'événement en utilisant la méthode `stopPropagation()` de l'objet `Event`. Cela empêche l'événement de se propager à travers l'arbre DOM.

Contexte d'utilisation

Cela peut être utile lorsque vous avez une liste d'éléments et que vous souhaitez ajouter un écouteur d'événement à chaque élément de la liste. En utilisant la propagation des événements, vous pouvez ajouter un seul écouteur d'événement à un élément parent et gérer les événements de tous les éléments enfants.

```
<ul id="liste">
    <li class="item">Élément 1</li>
    <li class="item">Élément 2</li>
    <li class="item">Élément 3</li>
</ul>
```

```
const liste = document.getElementById("liste");

liste.addEventListener("click", function (event) {
    if (event.target.closest(".item")) {
        // Gérer l'événement
    }
});
```

Target vs CurrentTarget

Lorsque vous gérez un événement, vous avez accès à deux propriétés de l'objet `Event` : `target` et `currentTarget`. La propriété `target` fait référence à l'élément sur lequel l'événement a été déclenché, tandis que la propriété `currentTarget` fait référence à l'élément sur lequel l'écouteur d'événement est attaché.

Donc dans le cas du bouillonnement, `target` sera l'élément sur lequel l'événement a été déclenché, tandis que `currentTarget` sera l'élément parent sur lequel l'écouteur d'événement est attaché.

Dans l'exemple plus haut, currentTarget sera l'élément `liste`, tandis que target sera l'élément `li` sur lequel l'événement a été déclenché.

Retirer un écouteur d'événement par programmation

Il est possible de retirer un écouteur d'événement en utilisant la méthode `removeEventListener`. **Il est important de noter que pour retirer un écouteur d'événement, la fonction de rappel doit être nommée. Vous ne pouvez pas utiliser de fonction anonyme**

Vous devez fournir le même type d'événement et la même fonction de rappel que vous avez utilisé pour ajouter l'écouteur d'événement.

```
function handleClick(evenement) {
    console.log("Clic !");
    evenement.currentTarget.removeEventListener("click", handleClick); // Retire
l'écouteur d'événement donc le clic ne sera plus géré
```

```
}

element.addEventListener("click", handleClick);
```

L'objet Event

Lorsqu'un événement est déclenché, un objet `Event` est créé et passé en argument à la fonction de rappel de l'écouteur d'événement. Cet objet contient des informations sur l'événement, telles que le type de l'événement, l'élément sur lequel l'événement a été déclenché, etc.

```
element.addEventListener("click", function (evenement) {
    console.log(evenement.type); // Affiche "click"
    console.log(evenement.target); // Affiche l'élément sur lequel l'événement a été
déclenché
});
```

Autres propriétés intéressantes de l'objet Event

- Récupérer le type de l'événement: `event.type`
- Récupérer l'élément sur lequel l'événement a été déclenché: `event.target`
- Récupérer l'élément sur lequel l'écouteur d'événement est attaché: `event.currentTarget`
- Récupérer le bouton de la souris qui a été cliqué: `event.button`
- Récupérer la position de la souris par rapport à la fenêtre du navigateur (sans la barre de navigation): `event.clientX` et `event.clientY`
- Récupérer la position de la souris par rapport à l'écran complet: `event.screenX` et `event.screenY`
- Récupérer la position de la souris par rapport à l'élément sur lequel l'événement a été déclenché: `event.offsetX` et `event.offsetY`
- Récupérer la position de la souris par rapport au document: `event.pageX` et `event.pageY`
- Récupérer la direction du mouvement de la souris: `event.movementX` et `event.movementY`
- Récupérer les doigts qui ont été utilisés pour l'événement (mobile): `event.touches`
- Récupérer les doigts qui sont actuellement en contact avec l'écran (mobile): `event.changedTouches`
- Récupérer les touches du clavier qui ont été appuyées: `event.key`
- Récupérer si la touche Ctrl a été appuyée: `event.ctrlKey`
- Récupérer si la touche Shift a été appuyée: `event.shiftKey`

- Récupérer si la touche Alt a été appuyée: `event.altKey`
- Récupérer la quantité de défilement horizontal et vertical: `event.deltaX` et `event.deltaY`
- Récupérer la direction du défilement: `event.deltaMode`
- Savoir si l'événement est en cours de propagation: `event.bubbles`
- Savoir l'état de l'événement: `event.defaultPrevented`
- Arrêter la propagation de l'événement: `event.stopPropagation()`

Pour plus d'informations sur l'objet `Event`, vous pouvez consulter la documentation MDN. https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent https://developer.mozilla.org/en-US/docs/Web/API/WheelEvent https://developer.mozilla.org/en-US/docs/Web/API/UIEvent

Créer un événement personnalisé

Il est possible de créer des événements personnalisés en utilisant la méthode `CustomEvent`. Cela peut être utile pour créer des événements personnalisés qui ne sont pas nativement supportés par le navigateur.

La méthode `CustomEvent` prend deux arguments : le nom de l'événement et un objet contenant des données supplémentaires à passer à l'événement. Cet objet doit contenir obligatoirement une propriété `detail` qui contient les données supplémentaires à passer à l'événement.

```
const evenement = new CustomEvent("monEvenement", {
    detail: {
        message: "Ceci est un événement personnalisé",
    },
});
```

Pour déclencher l'événement, vous pouvez utiliser la méthode `dispatchEvent` sur l'élément sur lequel vous souhaitez déclencher l'événement. Assurez-vous d

```
document.dispatchEvent(evenement);
```

Il faut ensuite attacher l'événement au même élément que le dispatchEvent en utilisant la méthode `addEventListener`.

```
document.addEventListener("monEvenement", function (event) {
    console.log(event.detail.message); // Affiche "Ceci est un événement
personnalisé"
});
```

Quand utiliser des événements personnalisés

Les événements personnalisés peuvent être utiles dans les cas suivants :

- Créer des événements personnalisés pour des composants réutilisables qui n'ont pas d'événements natifs ('click', 'change', etc.)
- Créer des événements personnalisés pour des interactions complexes qui nécessitent des données supplémentaires
- Créer des événements personnalisés pour des interactions asynchrones qui nécessitent une communication entre différents composants