JavaScript интерфейсные события

Основы событий мыши, Drag and Drop,
Прокрутка - кнопка вверх, Виртуальный скролл,
Простая TODO-шка на js, Крестики нолики игра

Основы событий мыши

- mousedown/mouseup - Кнопка мыши нажата/отпущена над элементом
- mouseover/mouseout - Курсор мыши появляется над элементом и уходит с него
- mousemove - Каждое движение мыши над элементом генерирует это событие
- click - Вызывается при mousedown , а затем mouseup над одним и тем же элементом, если использовалась левая кнопка мыши.
- dblclick - Вызывается двойным кликом на элементе.
- contextmenu - Вызывается при попытке открытия контекстного меню, как правило, нажатием правой кнопки мыши. Но, заметим, это не совсем событие мыши, оно может вызываться и специальной клавишей клавиатуры.

Модификаторы: shift, alt, ctrl и meta

- shiftKey: Shift
- altKey: Alt (или Opt для Mac)
- ctrlKey: Ctrl
- metaKey: Cmd для Mac

Они равны true, если во время события была нажата соответствующая клавиша.

```
<button id="button">Нажми Alt+Shift+Click на мне!</button>

<script>
  button.onclick = function(event) {
    if (event.altKey && event.shiftKey) {
      alert('Ура!');
    }
  };
</script>
```

Координаты: clientX/Y, pageX/Y

Все события мыши имеют координаты двух видов:

Относительно окна: clientX и clientY.

Относительно документа: pageX и pageY.

Если вкратце, то относительные координаты документа pageX/Y отсчитываются от левого верхнего угла документа и не меняются при прокрутке страницы, в то время как clientX/Y отсчитываются от левого верхнего угла текущего окна. Когда страница прокручивается, они меняются.

Например, если у нас есть окно размером 500x500, и курсор мыши находится в левом верхнем углу, то значения clientX и clientY равны 0, независимо от того, как прокручивается страница.

А если мышь находится в центре окна, то значения clientX и clientY равны 250 независимо от того, в каком месте документа она находится и до какого места документ прокручен. В этом они похожи на position:fixed.

Отключаем выделение

Двойной клик мыши имеет побочный эффект, который может быть неудобен в некоторых интерфейсах: он выделяет текст.

В данном случае самым разумным будет отменить действие браузера по умолчанию при событии mousedown, это отменит оба этих выделения:

```
<span ondblclick="alert('dblclick')">Сделайте двойной клик на мне</span>
```

```
До...
<b ondblclick="alert('Клик!')" onmousedown="return false">
  Сделайте двойной клик на мне
</b>
...После
```

Предотвращение копирования

Если мы хотим отключить выделение для защиты содержимого страницы от копирования, то мы можем использовать другое событие: oncopy.

Если вы попытаетесь скопировать текст в , у вас это не получится, потому что срабатывание события oncopy по умолчанию запрещено.

Конечно, пользователь имеет доступ к HTML-коду страницы и может взять текст оттуда, но не все знают, как это сделать.

```
<div oncopy="alert('Копирование запрещено!');return false">
  Уважаемый пользователь,
  Копирование информации запрещено для вас.
  Если вы знаете JS или HTML, вы можете найти всю нужную вам информацию
</div>
```

Drag and Drop

Drag’n’Drop – отличный способ улучшить интерфейс. Захват элемента мышкой и его перенос визуально упростят что угодно: от копирования и перемещения документов (как в файловых менеджерах) до оформления заказа («положить в корзину»).

Базовый алгоритм Drag’n’Drop выглядит так:

- При mousedown – готовим элемент к перемещению, если необходимо (например, создаём его копию).
- Затем при mousemove передвигаем элемент на новые координаты путём смены left/top и position:absolute.
- При mouseup – остановить перенос элемента и произвести все действия, связанные с окончанием Drag’n’Drop.

Пример - ТЫК

```
ball.onmousedown = function(event) {

  let shiftX = event.clientX - ball.getBoundingClientRect().left;
  let shiftY = event.clientY - ball.getBoundingClientRect().top;

  ball.style.position = 'absolute';
  ball.style.zIndex = 1000;
  document.body.append(ball);

  moveAt(event.pageX, event.pageY);

  // переносит мяч на координаты (pageX, pageY),
  // дополнительно учитывая изначальный сдвиг относительно указателя мыши
  function moveAt(pageX, pageY) {
    ball.style.left = pageX - shiftX + 'px';
    ball.style.top = pageY - shiftY + 'px';
  }

  function onMouseMove(event) {
    moveAt(event.pageX, event.pageY);
  }

  // передвигаем мяч при событии mousemove
  document.addEventListener('mousemove', onMouseMove);

  // отпустить мяч, удалить ненужные обработчики
  ball.onmouseup = function() {
    document.removeEventListener('mousemove', onMouseMove);
    ball.onmouseup = null;
  };

};

ball.ondragstart = function() {
  return false;
};
```

Drag and Drop концепция

HTML разметка:

- Создаем элемент div с атрибутом draggable="true", который делает его перетаскиваемым.
- Создаем элемент div для зоны сброса.

CSS стили:

- Задаем стили для перетаскиваемого элемента и зоны сброса, чтобы они были видны и различимы.

JavaScript функциональность:

- dragstart: Когда начинается перетаскивание, мы сохраняем данные о перетаскиваемом элементе и временно скрываем его.
- dragend: Когда перетаскивание завершается, мы вновь отображаем элемент.
- dragover: Позволяем элементу быть сброшенным на зону, предотвращая стандартное поведение.
- drop: Получаем данные о перетаскиваемом элементе и перемещаем его в зону сброса.

Пример - ТЫК

```
// Получаем элемент, который можно перетаскивать
const draggable = document.getElementById('draggable');
// Получаем зону для перетаскивания
const dropzone = document.getElementById('dropzone');

// Добавляем обработчик событий для начала перетаскивания
draggable.addEventListener('dragstart', (event) => {
  event.dataTransfer.setData('text/plain', event.target.id);
  setTimeout(() => {
    event.target.style.display = 'none';
  }, 0);
});

// Добавляем обработчик событий для окончания перетаскивания
draggable.addEventListener('dragend', (event) => {
  event.target.style.display = 'block';
});

// Обработчик событий для перетаскивания над зоной
dropzone.addEventListener('dragover', (event) => {
  event.preventDefault();
});

// Обработчик событий для сброса элемента в зону
dropzone.addEventListener('drop', (event) => {
  event.preventDefault();
  const id = event.dataTransfer.getData('text/plain');
  const draggableElement = document.getElementById(id);
  dropzone.appendChild(draggableElement);
});
```

Drag and Drop упорядоченный список

HTML Разметка:

- Создаем список ul с элементами li, которые можно перетаскивать благодаря атрибуту draggable="true".

CSS Стили:

- Задаем стили для списка и элементов, добавляя немного визуальных эффектов.
- Добавляем класс dragging, чтобы элемент при перетаскивании становился полупрозрачным.

JavaScript Функциональность:

- dragstart: Когда начинается перетаскивание, сохраняем ссылку на текущий элемент и добавляем ему класс dragging.
- dragend: Когда перетаскивание завершается, удаляем класс dragging и сбрасываем переменную draggingItem.
- dragover: Позволяем элементу быть сброшенным на зону, предотвращая стандартное поведение. Определяем, куда вставить перетаскиваемый элемент, основываясь на его текущей позиции.
- getDragAfterElement: Функция определяет, какой элемент является ближайшим к позиции, где был сброшен перетаскиваемый элемент, чтобы правильно его вставить.

Пример - ТЫК

```
const list = document.getElementById('sortable-list');
let draggingItem = null;

list.addEventListener('dragstart', (event) => {
    draggingItem = event.target;
    event.target.classList.add('dragging');
});

list.addEventListener('dragend', (event) => {
    event.target.classList.remove('dragging');
    draggingItem = null;
});

list.addEventListener('dragover', (event) => {
    event.preventDefault();
    const afterElement = getDragAfterElement(list, event.clientY);
    if (afterElement == null) {
        list.appendChild(draggingItem);
    } else {
        list.insertBefore(draggingItem, afterElement);
    }
});

function getDragAfterElement(list, y) {
    const draggableElements = [...list.querySelectorAll('.sortable-item:not(.dragging)')];

    return draggableElements.reduce((closest, child) => {
        const box = child.getBoundingClientRect();
        const offset = y - box.top - box.height / 2;
        if (offset < 0 && offset > closest.offset) {
            return { offset: offset, element: child };
        } else {
            return closest;
        }
    }, { offset: Number.NEGATIVE_INFINITY }).element;
}
```

Прокрутка - кнопка вверх

Высота страницы и текущая позиция прокрутки:

const scrollTotal = document.documentElement.scrollHeight - window.innerHeight;: Вычисляем общую высоту страницы за вычетом высоты видимой области.

const scrollY = window.scrollY;: Получаем текущую позицию прокрутки.

Проверка прокрутки:

if (scrollY / scrollTotal > 0.2): Если текущая позиция прокрутки превышает 20% от общей высоты страницы, показываем кнопку.

Если меньше, скрываем кнопку.

Обработчик нажатия:

Без изменений, прокручиваем страницу вверх при нажатии на кнопку.

Пример - ТЫК

```
const scrollToTopBtn = document.getElementById('scrollToTopBtn');

window.addEventListener('scroll', () => {
  const scrollTotal =
    document.documentElement.scrollHeight - window.innerHeight;
  const scrollY = window.scrollY;

  if (scrollY / scrollTotal > 0.2) {
    scrollToTopBtn.style.display = 'block';
  } else {
    scrollToTopBtn.style.display = 'none';
  }
});

scrollToTopBtn.addEventListener('click', () => {
  window.scrollTo({
    top: 0,
    behavior: 'smooth',
  });
});
```

Виртуальный скролл

Задача - сделать страницу новостей с динамической подгрузкой новостей при скроле (бесконечный скрол)

Этот код делает следующее:

generateNews(count): Создает массив новостей с заданным количеством элементов, каждая новость содержит заголовок, описание и уникальный цвет.

renderNewsBatch(): Отрисовывает порцию новостей в контейнере newsContainer, каждая новость представлена в виде карточки с уникальным цветом фона.

loadMoreNews(): Проверяет, есть ли еще новости для загрузки, и при необходимости вызывает renderNewsBatch().

window.addEventListener('scroll', ...): Слушает событие скроллинга страницы и при достижении конца страницы загружает новую порцию новостей.

renderNewsBatch() вызывается при инициализации страницы для загрузки первой порции новостей.

Таким образом, пользователь видит постепенную загрузку новостей по мере скроллинга, что улучшает производительность и опыт использования веб-страницы.

РЕШЕНИЕ ЗАДАЧКИ - ТЫК

Простая TODO-шка на js

Добавление задачи: при нажатии на кнопку "Добавить задачу" новая задача добавляется в список. Каждая задача представлена элементом <li>, содержащим название задачи, кнопки "Изменить" и "Удалить", а также чекбокс для отметки выполнения.

Удаление задачи: при нажатии на кнопку "Удалить" задача удаляется из списка.

Изменение задачи: при нажатии на кнопку "Изменить" текст задачи становится редактируемым. После внесения изменений можно сохранить новое название задачи или отменить изменения.

Отметка выполнения: при клике на чекбокс "Задача выполнена" задача либо отмечается как выполненная (с текстом, перечеркнутым и серым), либо снимается отметка выполнения.

Пример - ТЫК

Крестики нолики игра

Доска 3x3.

Возможность сделать ход как крестиками (X), так и ноликами (O).

Отображение текущего игрока.

Проверка победителя после каждого хода.

Возможность начать новую игру.

Пример реализации данной игры - ТЫК

Ресурсы

Основы событий мыши - ТЫК

Drag and drop - ТЫК

Прокрутка - ТЫК