

Раздел

[Интерфейсные события](#)

Навигация по уроку

Алгоритм Drag'n'Drop

Правильное  
позиционирование


Цели переноса (draggable)

Итого

Задачи (2)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Браузер: документ, события, интерфейсы](#)  
[→ Интерфейсные события](#) 12-го сентября 2020

## Drag'n'Drop с событиями мыши

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»).

В современном стандарте HTML5 есть [раздел о Drag and Drop](#) – и там есть специальные события именно для Drag'n'Drop переноса, такие как `dragstart`, `dragend` и так далее.

Они интересны тем, что позволяют легко решать простые задачи. Например, можно перетащить файл в браузер, так что JS получит доступ к его содержимому.

Но у них есть и ограничения. Например, нельзя организовать перенос «только по горизонтали» или «только по вертикали». Также нельзя ограничить перенос внутри заданной зоны. Есть и другие интерфейсные задачи, которые такими встроенными событиями не реализуемы. Кроме того, мобильные устройства плохо их поддерживают.

Здесь мы будем рассматривать Drag'n'Drop при помощи событий мыши.

### Алгоритм Drag'n'Drop

Базовый алгоритм Drag'n'Drop выглядит так:

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

Это и есть основа Drag'n'Drop. Позже мы сможем расширить этот алгоритм, например, подсветив элементы при наведении на них мыши.

В следующем примере эти шаги реализованы для переноса мяча:

```
1 ball.onmousedown = function(event) { // (1) отследить н
2
3 // (2) подготовить к перемещению:
4 // разместить поверх остального содержимого и в абсо
5 ball.style.position = 'absolute';
6 ball.style.zIndex = 1000;
7 // переместим в body, чтобы мяч был точно не внутри р
8 document.body.append(ball);
9 // и установим абсолютно спозиционированный мяч под к
10
11 moveAt(event.pageX, event.pageY);
12
13 // передвинуть мяч под координаты курсора
14 // и сдвинуть на половину ширины/высоты для центриров
15 function moveAt(pageX, pageY) {
16     ball.style.left = pageX - ball.offsetWidth / 2 + 'p
17     ball.style.top = pageY - ball.offsetHeight / 2 + 'p
18 }
19
20 function onMouseMove(event) {
21     moveAt(event.pageX, event.pageY);
22 }
23
24 // (3) перемещать по экрану
25 document.addEventListener('mousemove', onMouseMove);
```

Раздел

Интерфейсные события

Навигация по уроку

Алгоритм Drag'n'Drop

Правильное  
позиционирование

Цели переноса (draggable)

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



```
26
27 // (4) положить мяч, удалить более ненужные обработчи
28 ball.onmouseup = function() {
29     document.removeEventListener('mousemove', onMouseMove);
30     ball.onmouseup = null;
31 };
32
33 };
```

Если запустить этот код, то мы заметим нечто странное. При начале переноса мяч «раздваивается» и переносится не сам мяч, а его «клон».

Это можно увидеть в действии:

Перетащите мяч.



Попробуйте перенести мяч мышкой и вы увидите описанное поведение.

Всё потому, что браузер имеет свой собственный Drag'n'Drop, который автоматически запускается и вступает в конфликт с нашим. Это происходит именно для картинок и некоторых других элементов.

Его нужно отключить:

```
1 ball.ondragstart = function() {
2     return false;
3 };
```



Теперь всё будет в порядке.

В действии:

Перетащите мяч.



Ещё одна деталь – событие `mousemove` отслеживается на `document`, а не на `ball`. С первого взгляда кажется, что мышь всегда над мячом и обработчик `mousemove` можно повесить на сам мяч, а не на документ.

Но, как мы помним, событие `mousemove` возникает хоть и часто, но не для каждого пикселя. Поэтому из-за быстрого движения указатель может спрыгнуть с мяча и оказаться где-нибудь в середине документа (или даже за пределами окна).

Вот почему мы должны отслеживать `mousemove` на всём `document`, чтобы поймать его.

## Правильное позиционирование

В примерах выше мяч позиционируется так, что его центр оказывается под указателем мыши:

```
1 ball.style.left = pageX - ball.offsetWidth / 2 + 'px';
2 ball.style.top = pageY - ball.offsetHeight / 2 + 'px';
```

Раздел

Интерфейсные события

Навигация по уроку

Алгоритм Drag'n'Drop

Правильное  
позиционирование

Цели переноса (draggable)

Итого

Задачи (2)

Комментарии

Поделиться



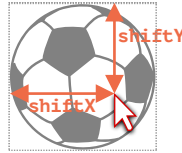
Редактировать на GitHub



Неплохо, но есть побочные эффекты. Мы, для начала переноса, можем нажать мышью на любом месте мяча. Если мячик «взят» за самый край – то в начале переноса он резко «прыгает», центрируясь под указателем мыши.

Было бы лучше, если бы изначальный сдвиг курсора относительно элемента сохранялся.

Где захватили, за ту «часть элемента» и переносим:



Обновим наш алгоритм:

1. Когда человек нажимает на мячик (mousedown) – запомним расстояние от курсора до левого верхнего угла шара в переменных shiftX/shiftY. Далее будем удерживать это расстояние при перетаскивании.

Чтобы получить этот сдвиг, мы можем вычесть координаты:

```
1 // onmousedown
2 let shiftX = event.clientX - ball.getBoundingClientRect().left
3 let shiftY = event.clientY - ball.getBoundingClientRect().top
```

2. Далее при переносе мяча мы позиционируем его с тем же сдвигом относительно указателя мыши, вот так:

```
1 // onmousemove
2 // ball has position: absolute
3 ball.style.left = event.pageX - shiftX + 'px';
4 ball.style.top = event.pageY - shiftY + 'px';
```

Итоговый код с правильным позиционированием:

```
1 ball.onmousedown = function(event) {
2
3   let shiftX = event.clientX - ball.getBoundingClientRect().left
4   let shiftY = event.clientY - ball.getBoundingClientRect().top
5
6   ball.style.position = 'absolute';
7   ball.style.zIndex = 1000;
8   document.body.appendChild(ball);
9
10  moveAt(event.pageX, event.pageY);
11
12  // переносит мяч на координаты (pageX, pageY),
13  // дополнительно учитывая изначальный сдвиг относительно
14  function moveAt(pageX, pageY) {
15    ball.style.left = event.pageX - shiftX + 'px';
16    ball.style.top = event.pageY - shiftY + 'px';
17  }
18
19  function onMouseMove(event) {
20    moveAt(event.pageX, event.pageY);
21  }
22
23  // передвигаем мяч при событии mousemove
24  document.addEventListener('mousemove', onMouseMove);
25
26  // отпустить мяч, удалить ненужные обработчики
27  ball.onmouseup = function() {
28    document.removeEventListener('mousemove', onMouseMove);
29    ball.onmouseup = null;
30  };
31}
```

Раздел

Интерфейсные события

Навигация по уроку

Алгоритм Drag'n'Drop

Правильное  
позиционирование

Цели переноса (draggable)

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



```
32 };
33
34 ball.ondragstart = function() {
35     return false;
36 };
```

В действии (внутри ифрейма):

Перетащите мяч.



Различие особенно заметно, если захватить мяч за правый нижний угол. В предыдущем примере мячик «прыгнет» серединой под курсор, в этом – будет плавно переноситься с текущей позиции.

## Цели переноса (draggable)

В предыдущих примерах мяч можно было бросить просто где угодно в пределах окна. В реальности мы обычно берём один элемент и перетаскиваем в другой. Например, «файл» в «папку» или что-то ещё.

Абстрактно говоря, мы берём перетаскиваемый (draggable) элемент и помещаем его в другой элемент «цель переноса» (draggable).

Нам нужно знать:

- куда пользователь положил элемент в конце переноса, чтобы обработать его окончание
- и, желательно, над какой потенциальной целью (элемент, куда можно положить, например, изображение папки) он находится в процессе переноса, чтобы подсветить её.

Решение довольно интересное и немного хитрое, давайте рассмотрим его.

Какой может быть первая мысль? Возможно, установить обработчики событий `mouseover` / `mouseup` на элемент – потенциальную цель переноса?

Но это не работает.

Проблема в том, что при перемещении перетаскиваемый элемент всегда находится поверх других элементов. А события мыши срабатывают только на верхнем элементе, но не на нижнем.

Например, у нас есть два элемента `<div>`: красный поверх синего (полностью перекрывает). Не получится поймать событие на синем, потому что красный сверху:

```
1 <style>
2   div {
3     width: 50px;
4     height: 50px;
5     position: absolute;
6     top: 0;
7   }
8 </style>
9 <div style="background:blue" onmouseover="alert('никогд
10 <div style="background:red" onmouseover="alert('над кра
```

То же самое с перетаскиваемым элементом. Мяч всегда находится поверх других элементов, поэтому события срабатывают на нём. Какие бы

Раздел

[Интерфейсные события](#)

Навигация по уроку

Алгоритм Drag'n'Drop

Правильное  
позиционирование

Цели переноса (draggable)

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



обработчики мы ни ставили на нижние элементы, они не будут выполнены.

Вот почему первоначальная идея поставить обработчики на потенциальные цели переноса нереализуема. Обработчики не сработают.

Так что же делать?

Существует метод `document.elementFromPoint(clientX, clientY)`. Он возвращает наиболее глубоко вложенный элемент по заданным координатам окна (или `null`, если указанные координаты находятся за пределами окна).

Мы можем использовать его, чтобы из любого обработчика событий мыши выяснить, над какой мы потенциальной целью переноса, вот так:

```
1 // внутри обработчика события мыши
2 ball.hidden = true; // (*) прячем переносимый элемент
3
4 let elemBelow = document.elementFromPoint(event.clientX
5 // elemBelow - элемент под мячом (возможная цель переноса)
6
7 ball.hidden = false;
```

Заметим, нам нужно спрятать мяч перед вызовом функции `(*)`. В противном случае по этим координатам мы будем получать мяч, ведь это и есть элемент непосредственно под указателем: `elemBelow=ball`. Так что мы прячем его и тут же показываем обратно.

Мы можем использовать этот код для проверки того, над каким элементом мы «летим», в любое время. И обработать окончание переноса, когда оно случится.

Расширенный код `onMouseMove` с поиском потенциальных целей переноса:

```
1 // потенциальная цель переноса, над которой мы пролетаем
2 let currentDraggable = null;
3
4 function onMouseMove(event) {
5   moveAt(event.pageX, event.pageY);
6
7   ball.hidden = true;
8   let elemBelow = document.elementFromPoint(event.clientX, event.clientY);
9   ball.hidden = false;
10
11   // событие mousemove может произойти и когда указатель
12   // (мяч перетаскивали за пределы экрана)
13
14   // если clientX/clientY за пределами окна, elementFromPoint
15   if (!elemBelow) return;
16
17   // потенциальные цели переноса помечены классом draggable
18   let draggableBelow = elemBelow.closest('.draggable');
19
20   if (currentDraggable !== draggableBelow) {
21     // мы либо залетаем на цель, либо улетаем из неё
22     // внимание: оба значения могут быть null
23     //   currentDraggable=null,
24     //   если мы были не над draggable до этого события
25     //   draggableBelow=null,
26     //   если мы не над draggable именно сейчас, во в
27
28     if (currentDraggable) {
29       // логика обработки процесса "вылета" из draggable
30       leaveDraggable(currentDraggable);
31     }
32     currentDraggable = draggableBelow;
33     if (currentDraggable) {
34       // логика обработки процесса, когда мы "влетаем"
35       enterDraggable(currentDraggable);
36     }
37   }
```

```
38    }  
    }
```

Раздел

[Интерфейсные события](#)

Навигация по уроку

Алгоритм Drag'n'Drop

Правильное  
позиционирование

Цели переноса (draggable)

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)



В приведённом ниже примере, когда мяч перетаскивается через футбольные ворота, ворота подсвечиваются.



Теперь в течение всего процесса в переменной `currentDroppable` мы храним текущую потенциальную цель переноса, над которой мы сейчас, можем её подсветить или сделать что-то ещё.

## Итого

Мы рассмотрели основной алгоритм Drag'n'Drop.

Ключевые идеи:

1. Поток событий: `ball.mousedown` → `document.mousemove` → `ball.mouseup` (не забудьте отменить браузерный `ondragstart`).
2. В начале перетаскивания: запоминаем начальное смещение указателя относительно элемента: `shiftX/shiftY` – и сохраняем его при перетаскивании.
3. Выявляем потенциальные цели переноса под указателем с помощью `document.elementFromPoint`.

На этой основе можно сделать многое.

- На `mouseup` – по-разному завершать перенос: изменять данные, перемещать элементы.
- Можно подсвечивать элементы, пока мышь «пролетает» над ними.
- Можно ограничить перетаскивание определённой областью или направлением.
- Можно использовать делегирование событий для `mousedown/up`. Один обработчик событий на большой зоне, который проверяет `event.target`, может управлять Drag'n'Drop для сотен элементов.
- И так далее.

Существуют фреймворки, которые строят архитектуру поверх этого алгоритма, создавая такие классы, как `DragZone`, `Droppable`, `Draggable`. Большинство из них делают вещи, аналогичные описанным выше. Вы можете и создать вашу собственную реализацию переноса, как видите, это достаточно просто, возможно, проще, чем адаптация чего-то готового.

## ✓ Задачи

### Слайдер

важность: 5

Создайте слайдер:



Захватите мышкой синий бегунок и двигайте его.

Раздел

[Интерфейсные события](#)

Навигация по уроку

Алгоритм Drag'n'Drop

Правильное  
позиционирование

Цели переноса (draggable)

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



Важные детали:

- Слайдер должен нормально работать при резком движении мыши влево или вправо за пределы полосы. При этом бегунок должен останавливаться чётко в нужном конце полосы.
- При нажатом бегунке мышь может выходить за пределы полосы слайдера, но слайдер пусть все равно работает (это удобно для пользователя).

[Открыть песочницу для задачи.](#)

решение

## Расставить супергероев по полю

важность: 5

В этой задаче вы можете проверить своё понимание сразу нескольких аспектов Drag'n'Drop и DOM.

Сделайте так, чтобы элементы с классом `draggable` можно было переносить мышкой. Как мяч в этой главе.

Требования к реализации:

- Используйте делегирование событий для отслеживания начала перетаскивания: только один обработчик событий `mousedown` на документе.
- Если элементы подносят к верхней/нижней границе окна – оно должно прокручиваться вверх/вниз, чтобы позволить дальнейшее перетаскивание.
- Горизонтальная прокрутка отсутствует (чуть-чуть упрощает задачу, её просто добавить).
- Элемент при переносе, даже при резких движениях мышкой, не должен даже частично попасть вне окна.

Демо слишком велико для размещения здесь, перейдите по ссылке ниже.

[Демо в новом окне](#)

[Открыть песочницу для задачи.](#)

решение

Проводим [курсы по JavaScript и фреймворкам.](#) 

## Комментарии

перед тем как писать...