

Geolocation, Scroll, Drag and Drop API

Урок 12





Кадочников Алексей

Frontend-разработчик

- ☀ Веб-разработчик со стажем более 9 лет
- ☀ Преподаватель GeekBrains с 2015 года
- ☀ Автор курсов по Frontend на портале Geekbrains
- ☀ Работал в таких компаниях, как VK и Wizard-C



План курса

1

Знакомство с JavaScript

2

Основы JavaScript

3

Функции в JavaScript

4

Циклы и массивы

5

Работа с объектами в JavaScript

6

Введение в DOM

7

Работа с DOM

8

Основы событий в JavaScript

9

Работа с событиями в JavaScript

10

Основы шаблонизации, работа с JSON

11





Работа с медиафайлами

12

Основы API, итоги курса



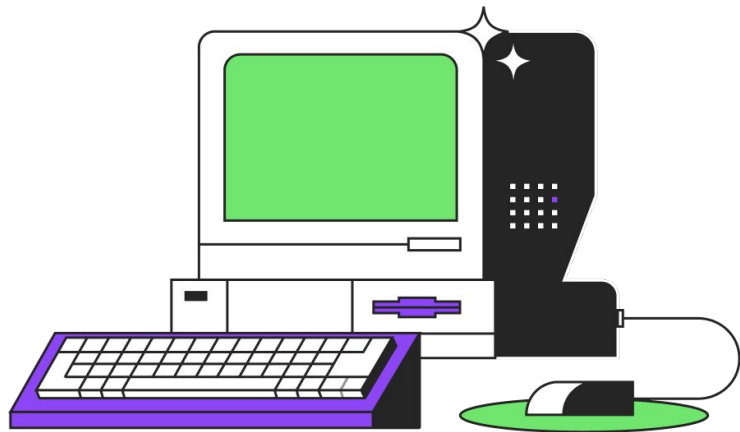
Что будет на уроке сегодня

-  История появления API устройств
-  Геолокация
-  Работа со скроллом
-  Использование Drag and Drop API



История появления API устройств

За более чем 25 лет с момента создания интернета мы стали свидетелями того, как он распространился на рабочие станции, настольные ПК, ноутбуки, мобильные телефоны, планшеты, а теперь даже на модные аксессуары, такие как часы. Нет никаких сомнений, что интернет никуда не денется.





История API

Естественно, интернет-технологии должны развиваться, чтобы лучше соответствовать возможностям современных устройств, на которых они теперь доступны.

На этом уроке мы познакомимся с некоторыми популярными JavaScript API устройств, способными расширять функциональность браузера и обладающими хорошим уровнем поддержки в современных браузерах.





Геолокация





Геолокация

API геолокации позволяет определять местоположение пользователя, используя возможности позиционирования его устройства. В большинстве случаев позиционирование выполняется с использованием GPS, но также могут использоваться менее точные методы, такие как определение местоположения на основе Wi-Fi.

API геолокации доступен через объект `navigator.geolocation`.





Геолокация

Если объект существует, функции определения местоположения доступны.

Проверить это можно следующим образом:

```
1 if ('geolocation' in navigator) {  
2   /* местоположение доступно */  
3 } else {  
4   /* местоположение недоступно */  
5 }  
6
```



Получение текущего местоположения

Для получения текущего местоположения пользователя надо вызвать метод `getCurrentPosition()`. Это инициирует асинхронный запрос для обнаружения местоположения пользователя и запрашивает аппаратные средства позиционирования, чтобы получить последнюю актуальную информацию.

```
1 navigator.geolocation.getCurrentPosition((position) => {  
2   const {latitude, longitude} = position.coords  
3   console.log('Географические координаты устройства', latitude,  
4     longitude)  
5 })
```



Наблюдение за текущим местоположением

Чтобы следить за изменениями местоположения устройства, используют функцию **watchPosition()**, которая имеет три входных параметра, похожих на **getCurrentPosition()**. Переданные ей в параметрах callback-функции вызываются много раз, позволяя браузеру обновлять данные о текущей локации либо во время движения, либо после уточнения информации о местоположении с применения более точных методов позиционирования.

Важно! *Использовать `watchPosition()` можно и без вызова `getCurrentPosition()`.*

```
1 let watchId = navigator.geolocation.watchPosition(({coords}) => {  
2   console.log('Устройство обновило местоположение', coords.latitude,  
3     coords.longitude)  
4 })
```



Наблюдение за текущим местоположением

Метод **watchPosition()** возвращает числовой идентификатор для использования вместе с методом **clearWatch()**, чтобы отписаться от получения новых данных о местоположении.

```
1 navigator.geolocation.clearWatch(watchId)  
2
```

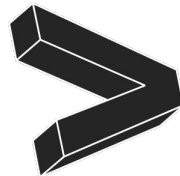
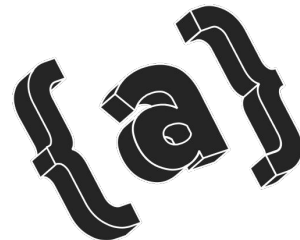




Точная настройка позиционирования

getCurrentPosition() и **watchPosition()** принимают третьим параметром необязательный объект **PositionOptions** со следующими полями:

1. **enableHighAccuracy (Boolean)** — высокая точность определения позиции.
2. **maximumAge (Number)** — максимальное время кеширования в миллисекундах. При повторных запросах, пока время не вышло, будет возвращаться кешированное значение, а после браузер будет запрашивать актуальные данные.
3. **timeout (Number)** — максимальное время ожидания ответа при определении позиции. Через сколько миллисекунд будет вызван обработчик ошибки.

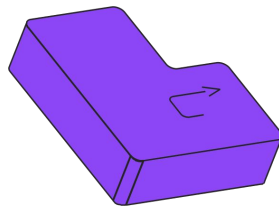




Обработка ошибок

Callback-функция обработки ошибок, если она была передана в `getCurrentPosition()` или `watchPosition()`, ожидает экземпляр объекта `GeolocationPositionError` в качестве первого аргумента. Он будет содержать два свойства:

- **code** — значение, которое указывает, какая именно ошибка произошла;
- **message** — понятное для человека описание значения `code`.





Работа со скроллом





Работа со скроллом

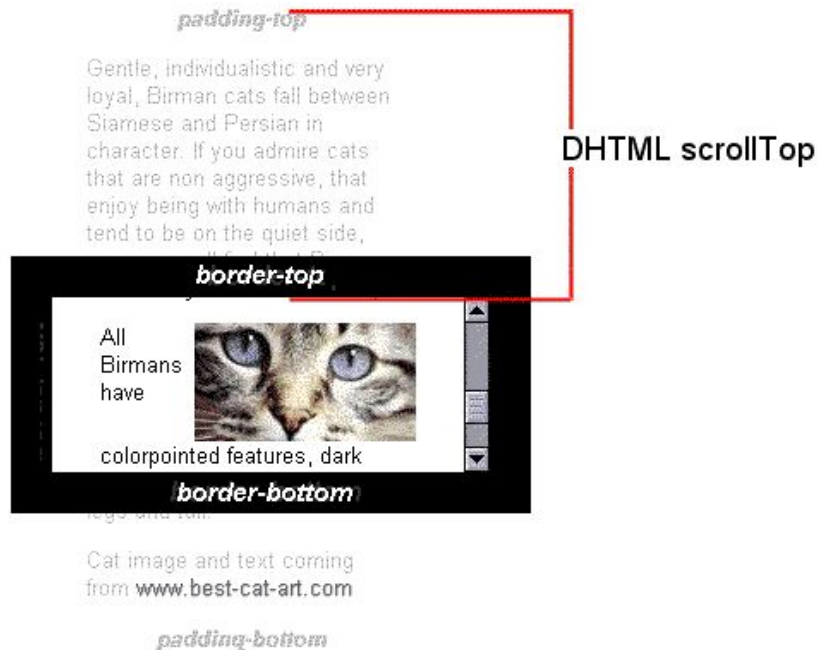
DOM-интерфейсы Window и Element содержат несколько полей и методов для работы с полосой прокрутки.





Свойства `Element.scrollTop` и `Element.scrollLeft`

Свойства **`Element.scrollTop`** и **`Element.scrollLeft`** возвращают или устанавливают расстояние от начальной точки содержимого элемента (`padding-box` элемента) до начальной точки его видимого контента. Когда контент элемента не создаёт полосу прокрутки, его **`scrollTop`** или **`scrollLeft`** равен нулю.





Работа со скроллом

Значения свойств `Element.scrollTop` и `Element.scrollLeft` могут быть любым целым числом, но с определёнными оговорками:

1. Если элемент не прокручивается, у него нет переполнения или мы не прокручиваем элемент, `scrollTop` (`scrollLeft`) устанавливается в 0.
2. Если значение меньше нуля, `scrollTop` (`scrollLeft`) устанавливается в 0.
3. Если установленное значение больше максимума прокручиваемого контента, `scrollTop` (`scrollLeft`) устанавливается в максимум.

```
1 const box = document.querySelector('#box')
2 console.log(box.scrollTop, box.scrollLeft)
3
4 // Устанавливаем количество прокрученных пикселей
5 box.scrollTop = 500
```



Свойства Element.scrollHeight и Element.scrollWidth

Свойства Element.scrollHeight и Element.scrollWidth (только чтение) содержат высоту (ширину для scrollWidth) содержимого элемента, включая содержимое, невидимое из-за прокрутки. Значение scrollHeight (scrollWidth) равно минимальному clientHeight (clientWidth), которое потребуется элементу, чтобы поместить всё содержимое в видимую область, не используя полосу прокрутки. Оно включает в себя padding элемента, но не его margin.



DHTML scrollHeight



Методы `scroll`, `scrollTo`, `scrollBy`

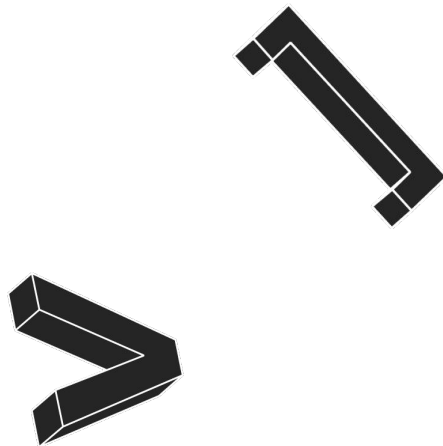
Есть три основных метода для программного управления полосой прокрутки.

1. **x-coord** — координаты пикселя по горизонтальной оси документа или элемента, который надо отобразить вверху слева.
2. **y-coord** — координаты пикселя по вертикальной оси документа или элемента, который надо отобразить вверху слева.
3. **options** — объект с тремя возможными параметрами:
 - *top* — то же, что и y-coord;
 - *left* — то же, что и x-coord;
 - *behavior* — строка, содержащая либо smooth, instant, либо auto (по умолчанию — auto).



Методы `scroll`, `scrollTo`, `scrollBy`

Методы `scroll` и `scrollTo` абсолютно идентичны: мы можем использовать любой из них. Но `scrollBy` использует относительные координаты, а `scroll` и `scrollTo` — абсолютные. То есть, последовательно вызывая `scrollBy` с одними и теми же параметрами, мы будем менять положение полосы прокрутки на значение, переданное в параметрах. В случае со `scroll` и `scrollTo` изменения произойдут лишь при первом вызове.





Метод `Element.scrollToView`

Метод `Element.scrollToView()` интерфейса `Element` прокручивает текущий контейнер родителя элемента так, чтобы элемент, на котором вызван `scrollIntoView()`, был видим пользователю. Этот метод принимает два типа параметров:

1. **`alignToTop`** — необязательный аргумент типа `Boolean` со следующими возможными значениями:
 - a. *`true`* — верхняя граница элемента выравнивается по верхней границе видимой части окна прокручиваемой области. Соответствует `scrollIntoViewOptions: {block: "start", inline: "nearest"}`. Значение по умолчанию.
 - b. *`false`* — нижняя граница элемента выравнивается по нижней границе видимой части окна прокручиваемой области. Соответствует конфигурации `scrollIntoViewOptions: {block: "end", inline: "nearest"}`



Метод `Element.scrollToView`

1. **`scrollIntoViewOptions`** — необязательный аргумент типа `object` со следующим набором необязательных полей:
 - a. **`behavior`** — определяет анимацию скrolла. Принимает значение `auto` или `smooth`. По умолчанию — `auto`.
 - b. **`block`** — вертикальное выравнивание. Варианты значений: `start`, `center`, `end` или `nearest`. По умолчанию — `start`.
 - c. **`inline`** — горизонтальное выравнивание. Варианты значений: `start`, `center`, `end` или `nearest`. По умолчанию — `nearest`.



Использование Drag and Drop API





Использование Drag and Drop API

API перетаскивания HTML5 DnD API позволяет сделать практически любой элемент на нашей странице перетаскиваемым. В большинстве браузеров выделенный текст, изображения и ссылки по умолчанию перетаскиваются.





Использование Drag and Drop API

В нашем примере мы создадим интерфейс для перестановки некоторых столбцов, размещённых посредством CSS Grid. Базовая разметка для столбцов выглядит так: для каждого столбца установлен draggable-атрибут true.

```
1 <div class="container">
2   <div draggable="true" class="box">A</div>
3   <div draggable="true" class="box">B</div>
4   <div draggable="true" class="box">C</div>
5 </div>
6
```



Использование Drag and Drop API

Ниже вы видите CSS для элементов контейнера и боксов. Обратите внимание, что единственное CSS-правило, связанное с функциональностью DnD, — это свойство `cursor: move`. Остальное просто управляет компоновкой и стилями контейнера и боксов.

```
1 .container {  
2   display: grid;  
3   grid-template-columns: repeat(5, 1fr);  
4   gap: 10px;  
5 }  
6  
7 .box {  
8   border: 3px solid #666;  
9   background-color: #ddd;  
10  border-radius: .5em;  
11  padding: 10px;  
12  cursor: move;  
13 }
```



Прослушивание событий перетаскивания

Есть ряд событий, к которым можно привязать мониторинг всего процесса перетаскивания:

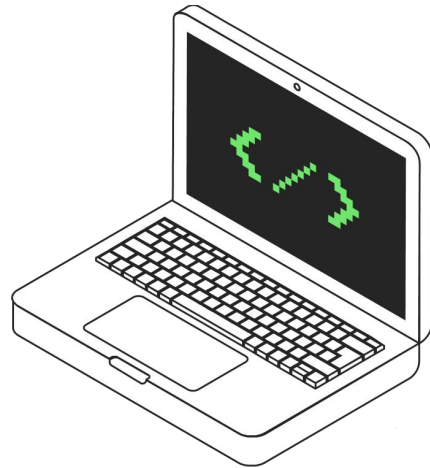
1. **dragstart** — срабатывает, когда пользователь начал перетаскивать элемент.
2. **drag** — срабатывает каждые несколько сотен миллисекунд, пока длится перетаскивание элемента или выделение текста.
3. **dragenter** — срабатывает, когда перетаскиваемый элемент попадает в допустимую цель сброса.
4. **dragleave** — срабатывает, когда перетаскиваемый элемент покидает допустимую цель сброса.
5. **dragover** — срабатывает каждые несколько сотен миллисекунд, когда элемент перетаскивается над допустимой целью сброса.
6. **drop** — срабатывает, когда элемент сбрасывается в допустимую зону сброса.
7. **dragend** — срабатывает в момент завершения перетаскивания, например, при отпускании кнопки мыши или по нажатию Escape.



Прослушивание событий перетаскивания

Чтобы корректно обрабатывать DnD-процесс, нам потребуется:

- исходный элемент (source element), откуда начинается перетаскивание;
- объект с данными (data payload) — структура, которую мы перетаскиваем;
- целевой элемент (target) — область, в которую мы «ловим» переташенный объект.





Обработка начала и завершения перетаскивания

После того как мы указали атрибуты `draggable = "true"` для нашего контента, надо подписаться на событие **dragstart**, чтобы запустить последовательность DnD для каждого столбца.

Следующий код установит прозрачность столбца на 40%, когда пользователь начнёт его перетаскивать, а затем вернёт её на 100% по завершении перетаскивания:

```
1 const items = document.querySelectorAll('.container .box')
2
3 const handleDragStart = ({target}) => {
4   target.style.opacity = '0.4'
5 }
6
7 const handleDragEnd = ({target}) => {
8   target.style.opacity = '1'
9 }
10
11 items.forEach((item) => {
12   item.addEventListener('dragstart', handleDragStart)
13   item.addEventListener('dragend', handleDragEnd)
14 })
15
```



Обработка начала и завершения перетаскивания

Чтобы помочь пользователю понять, как взаимодействовать с интерфейсом, используют дополнительные стили, которые мы установим в обработчиках событий `dragenter`, `dragover` и `dragleave`. В примере ниже столбцы не только перетаскиваются, но и служат целями для перетаскивания. Мы можем помочь пользователю понять это, сделав границу пунктирной, когда он удерживает перетаскиваемый элемент над столбцом. Например, добавим в наш CSS класс `over` для представления элементов, которые считаются целями перетаскивания:

```
1 .box.over {  
2   border: 3px dotted #666;  
3 }  
4
```



Обработка начала и завершения перетаскивания

Затем в нашем JavaScript настроим обработчики событий:

- для добавления класса `over`, когда столбец перетаскивается над областью, занимаемой элементом;
- для удаления класса при покидании области.

В обработчике **dragend** мы также убираем классы в конце перетаскивания.



Обработка завершения перетаскивания

Чтобы обработать момент, когда пользователь отпускает перетаскиваемый объект над целевым элементом, надо подписаться на событие **drop**. В обработчике события drop нам потребуется предотвратить поведение браузера по умолчанию, которое обычно представляет собой перенаправление.

Предотвратим это поведение, вызвав `event.stopPropagation()`.





Обработка завершения перетаскивания

В `dataTransfer` происходит вся магия DnD. Он содержит фрагмент данных, отправленный при перетаскивании. Объект `dataTransfer` устанавливается в событии **dragstart** и читается (обрабатывается) в других событиях перетаскивания. Вызов `e.dataTransfer.setData(mimeType, dataPayload)` позволяет установить MIME-тип объекта и добавить необходимые данные.

В этом примере мы позволим пользователям изменять порядок столбцов. Для этого сначала сохраним HTML-код исходного элемента при старте перетаскивания:

```
1 const handleDragStart = ({target}) => {  
2   target.style.opacity = '0.4'  
3  
4   e.dataTransfer.effectAllowed = 'move'  
5   e.dataTransfer.setData('text/html', target.innerHTML)  
6 }
```







Обработка завершения перетаскивания

В обработчике `drop`, где мы обрабатываем отпускание столбца, заменим HTML-код целевого столбца на код исходного столбца, перетаскиваемый пользователем, предварительно проверив, не происходит ли отпускание на тот же столбец, из которого оно началось.

```
1 const handleDrop = (event) => {  
2   event.stopPropagation()  
3   const sourceElementHtml = e.dataTransfer.getData('text/html')  
4  
5   if (event.target.innerHTML === sourceElementHtml) {  
6     return  
7   }  
8   event.target.innerHTML = sourceElementHtml  
9  
10  return false  
11 }  
12  
13
```



Итоги урока

-  История появления API устройств
-  Геолокация
-  Работа со скроллом
-  Использование Drag and Drop API



План курса

1

Знакомство с JavaScript

2

Основы JavaScript

3

Функции в JavaScript

4

Циклы и массивы

5

Работа с объектами в JavaScript

6

Введение в DOM

7

Работа с DOM

8

Основы событий в JavaScript

9

Работа с событиями в JavaScript

10


Основы шаблонизации, работа с JSON

11

Работа с медиафайлами

12

Основы API, итоги курса

Спасибо 
за внимание

