

**Я**ндекс

**Я**ндекс

**События**

# Обработка события

Чтобы обработать событие, нужно в свойство `on<событие>` записать функцию. В момент возникновения факта события браузер запустит эту функцию.

```
document.getElementById('button').onclick = function() {  
    console.log('somebody clicked me!')  
}
```

Браузер понимает, по какому прямоугольнику был выполнен клик, какой объект связан с этим прямоугольником и, если при клике у этого объекта есть свойство `onclick` и это функция, она будет запущена.

# Программный вызов события

Самый простой способ программно вызвать событие - это вызвать метод `<событие>()` на объекте-узле.

**`document.getElementById('button').click();`**

Есть и более сложные (и гибкие) способы, но нам пока хватит.

Вообще вызывать стандартные события на элементах - странная идея и у вас должна быть веская причина так делать.

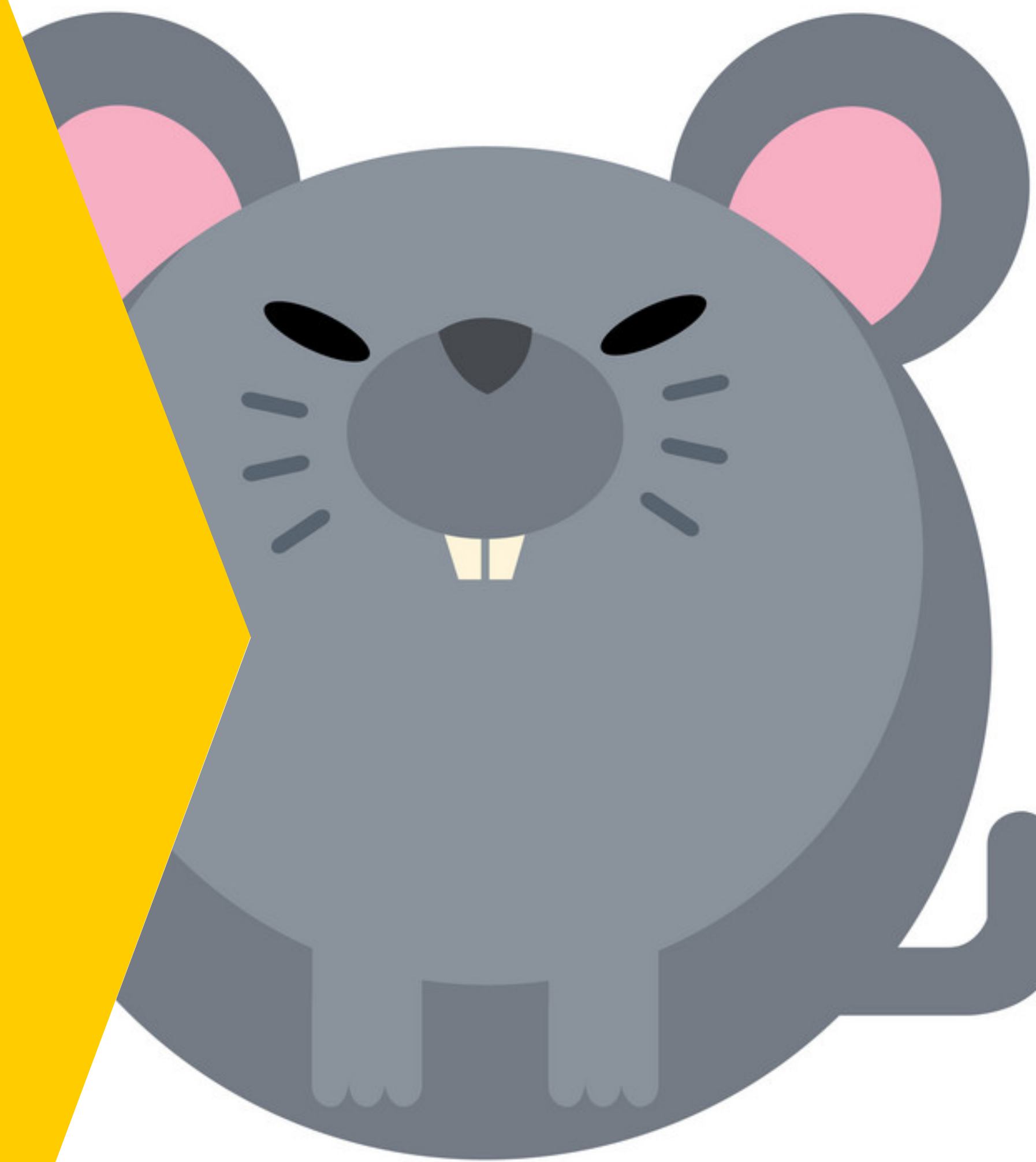
# Объект события

При этом, браузер передаст в функцию объект в качестве аргумента. Принимая этот объект, мы получаем возможность узнать много всего интересного про случившееся событие. Но самое главное и частое, что мы будем делать - это отменять поведения браузера по умолчанию при этом событии

```
<a href="https://google.com" id="google">google</a>
```

```
document.getElementById('google').onclick = function(event) {  
    event.preventDefault();  
}
```

# Событія мышИ



| click знаем

# contextmenu

Событие, возникающее при клике правой клавишей мыши (если речь о правшах). Той самой, которая открывает контекстное меню на элементе

**Посмотрим пример...**

В современных интерфейсах используется не так часто, подумайте почему?



# Наведение: `mouseover`, `mouseout`, `mouseenter`, `mouseleave`

Группа событий, возникающая при перемещении курсора мыши по элементам документа.

**`mouseover`, `mouseenter` - события про перемещение курсора внутрь блока**

**`mouseout`, `mouseleave` - события про перемещения курсора из блока наружу**

Пара `mouseover/mouseout` реагирует на перемещения курсора на дочерние элементы блока, а `mouseenter/mouseleave` - нет.

# mousemove

Событие, возникающее по мере перемещения курсора по видимой части элемента документа

**Самое интересное тут в объекте события - там придут данные о текущей позиции курсора,**



# Drag & Drop

# События клавиатуры



# keydown/keypress

События, которое происходит «за миллисекунду» до того, как нажата клавиша на клавиатуре. Особенность событий в том, что вы еще можете успеть «отменить» нажатие с помощью `event.preventDefault()`.

Угадайте, для чего это может быть полезно?

Отличия `keypress` от `keydown` в том, что `keypress` не возникает для управляющих клавиш (Esc, F1...F12, и т.д.), но зато в `keypress` есть информация о вводимом символе (в `keydown` только о клавише)

**Смотрим пример...**

# keyup

Событие возникает сразу после нажатия клавиши, когда браузер уже успел обработать нажатие. Отменить таким образом сделанное уже нельзя.

Угадайте, для чего это полезно?

**Смотрим пример...**



# События в элементах форм



# focus / blur

**focus** - событие, которое происходит в момент получения фокуса элементом.

Можно дать возможность получить фокус любому элементу документа (div, span...), если у него указан `tabindex`

**blur** - событие потери фокуса элементом

`event.preventDefault()` не позволит заблокировать получение/потерю фокуса, так как события происходят уже после того как фокус получен/потерян



# change

Событие возникает после окончания взаимодействия с элементом формы (в момент blur), если значение элемента было изменено.


**Таким образом, событие не возникает на любое нажатие клавиши в поле ввода**

**Важно, чтобы содержимое в конечном итоге стало другим, а не просто менялось в процессе**

# input

Универсальное, самое удобное событие для отлова изменений в поле ввода. Срабатывает на нажатие клавиш, cut/paste/очистку поля ввода. Можно в live режиме получать данные из поля ввода.

**Как правило, использовать событие input более правильно, нежели события клавиатуры**



highlighted text

# submit

Срабатывает перед отправкой формы. Если вы не хотите отпускать пользователя со страницы, то можете отменить обработку события браузером с помощью `event.preventDefault()`;

**Это именно та точка, где можно проверить правильность ввода данных в форму в целом.**

**Вы можете, например, делать `preventDefault` при неправильном вводе, и не делать, давая браузеру таким образом отправить данные если все ок**

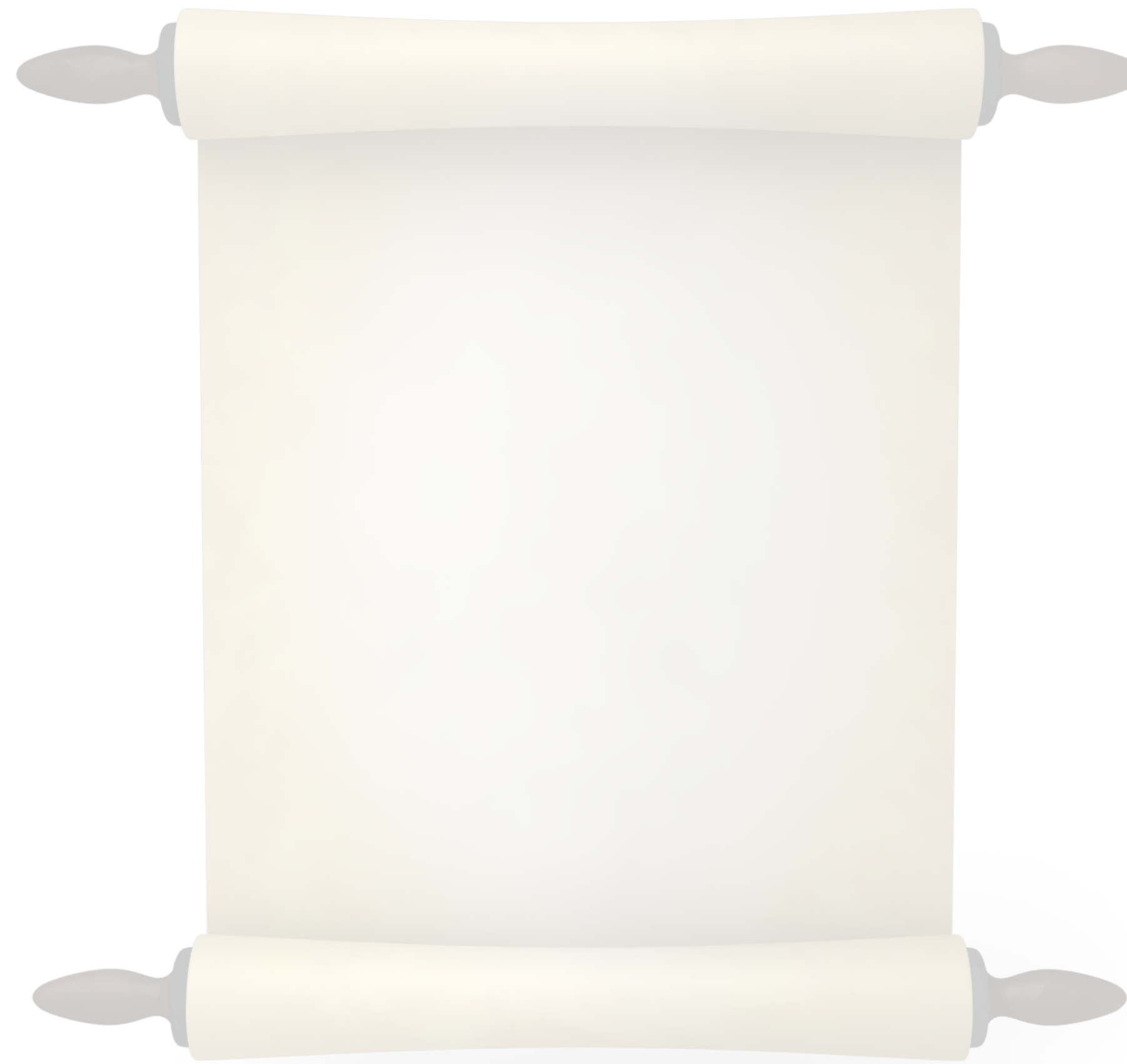
# События документа и его элементов



# scroll

Возникает по мере прокрутки документа. Позволяет реагировать на прокрутку, делать какие-то метаморфозы в документе по мере скролла - панорамные, анимированные фоны, показ баннеров на обратный скролл и т.п. механики

**Смотрим пример...**



# load

`window.onload` срабатывает когда вся страница загрузится - включая все картинки и прочие внешние ресурсы. С учетом особенностей web, событие возникает довольно редко.

Для картинок (а также других блоков, для которых есть внешняя ссылка в `src`) событие срабатывает в момент загрузки содержимого по ссылке.

**Смотрим пример...**

# error

Для картинки, событие «ошибки» возникнет, если картинка не загрузилась. Причины могут быть разными. Возможно, хорошая идея при такой ситуации отобразить какую-то заглушку

**Смотрим пример...**



# DOMContentLoaded

Событие `document`, говорящее о том, что документ загружен, распарсен, все объекты узлы созданы и готовы к работе. Именно после этого события безопасно обращаться к любым узлам документа.

**Поля `onDOMContentLoaded` нет и единственный способ подписаться на это событие - чуть более сложный синтаксис `addEventListener`**



# addEventListener

Более гибкий метод, позволяющий назначать несколько обработчиков одного события, подписываться на всякие несуществующие, кастомные события, отменять подписку и т.п.

```
document.addEventListener('DOMContentLoaded', function() {  
    // весь код пишем здесь  
});
```

# Всплытие

Если представить документ как доску с разложенными на нем прямоугольниками, то клик как будто выстрел - поражает сперва ближний прямоугольник, который как бы «сверху».

«Простреливая» его - вызывая на нем событие клик, он затем вызывает это событие на всех его родителях по очереди, как бы прошивая и их тоже.

Это и называется всплытием

Любой обработчик клика может отменить всплытие с помощью `event.stopPropagation()`. `event.stopImmediatePropagation()` также заблокирует любые другие обработчики этого события на этом же блоке

# Делегирование

На стыке всплытия и `event.target` появилось так называемое «делегирование» событие.

Если у вас контейнер с кучей одинаковых блоков, то логичнее не слушать клики на всех потомках контейнера, а слушать все клики, которые всплывут в контейнер, а затем, смотря в `event.target` принимать решение, что делать.



# Спасибо

**Шлейко Александр**

Разработчик интерфейсов



dusty@yandex-team.ru



@dustyo\_O