нетология
университет интернет-профессий

СОЗДАНИЕ HTML С НУЛЯ

АЛЕКСАНДР ШЛЕЙКО / ЯНДЕКС

АЛЕКСАНДР ШЛЕЙКО

Разработчик интерфейсов в Яндекс

a.shleyko@yandex.ru

dustyo_O

vk.com/shleiko

ПЛАН ЗАНЯТИЯ

1. Создание HTML с нуля
   - Неоптимальный способ
   - Теория
   - `createElement`
   - Добавим привкус декларативности...
2. Стили и размеры
   - Колонки равной высоты
   - Вертикальное центрование
   - «Зависающая» шапка
   - Определение ширины документа

СОЗДАНИЕ HTML С НУЛЯ

КАТАЛОГ ТЕЛЕФОНОВ

Представим, что мы с сервера получаем XHR запросом JSON-данные об имеющихся в наличии телефонах и их ценах, и нам их нужно вывести на страницу в виде списка.

Вот данные (опустим сам XHR запрос):

```
const products = [
  {
    title: 'iPhone 8',
    price: 499
  },
  {
    title: 'iPhone 9',
    price: 799
  }
];
```

РАЗМЕТКА ДЛЯ ОБЪЕКТА

Напишем функцию, которая будет брать объект с данными продукта и возвращать всю HTML-разметку для это этого продукта:

```
function renderProduct(product) {
  return `<div class="product">
    <div class="product-title">${product.title}</div>
    <div class="product-price">${product.price}</div>
  </div>`;
}
```

ДОБАВЛЯЕМ РАЗМЕТКУ В DOM

Затем применим эту функцию к каждому товару при помощи `map` и слепим все в одну строку при помощи `join`, затем при помощи уже знакомого нам `innerHTML` добавим получившуюся разметку в DOM:

```
1 const renderedProducts = products.map(renderProduct);
2 const markup = renderedProducts.join('');
3 document.body.innerHTML = markup;
```

Live Demo

Что можно улучшить?

МОЖНО ЛУЧШЕ

Мы уже имеем структурированные данные.

Зачем тогда мы превращаем данные в строку, а затем снова заставляем браузер парсить эту строку и создавать из нее элементы.

А про угрозы XSS не забыли?

НЕМНОГО ТЕОРИИ

Существуют способ создавать новые элементы напрямую.

- `createElement('h1')` – создает новые DOM-элементы с заданным тегом.
- `createTextNode('some text')` – создает новый текстовый узел с заданным текстом. Без потерь заменяется на `.textContent`.
- `createDocumentFragment` – создает `DocumentFragment`, в который мы можем оборачивать несколько элементов для вставки разом. Подробнее на примере.
- `document.write` – динозавр, самый архаичный метод, очень редко используется. Дописывает HTML-код в момент начального парсинга страницы, еще до создания DOM-структуры. Позволяет добавлять невалидную HTML-разметку, все равно если бы мы дописали эту разметку в HTML-файл руками на этом месте.

ОПТИМИЗАЦИЯ: `createElement`

Напишем функцию, которая будет создавать для продукта дерево узлов, которые мы сможем добавть в DOM:

```
function createProductNode(product) {
  const title = document.createElement('div');
  title.className = 'product-title';
  title.textContent = product.title;
  const price = document.createElement('div');
  price.className = 'product-price';
  price.textContent = product.price;
  const productNode = document.createElement('div');
  productNode.className = 'product';
  productNode.appendChild(title);
  productNode.appendChild(price);
  return productNode;
}
```

БОЛЬШЕ КОДА!

Стало больше кода, верно? Это потому, что мы стали действовать императивно, то есть, стали пошагово объяснять браузеру, что ему нужно делать. Позже разберемся, что с этим делать.

ЗАМЕНЯЕМ ФУНКЦИЮ

Дальше заменим новой функцией функцию `renderProduct`, применим эту функцию к каждому продукту. А затем соберем все элементы продуктов в один фрагмент, и добавим его в DOM:

```
1  const productNodes = products.map(createProductNode);
2
3  const fragment = productNodes
4    .reduce((fragment, currentValue) => {
5      fragment.appendChild(currentValue);
6      return fragment;
7    }, document.createDocumentFragment());
8
9  document.body.appendChild(fragment);
```

Live Demo

ДОБАВИМ ПРИВКУС ДЕКЛАРАТИВНОСТИ...

```
function el(tagName, attributes, children) {
  const element = document
    .createElement(tagName);
  if (typeof attributes === 'object') {
    Object.keys(attributes).forEach(i => element
      .setAttribute(i, attributes[i]));
  }
  if (typeof children === 'string') {
    element.textContent = children;
  } else if (children instanceof Array) {
    children.forEach(child => element
      .appendChild(child));
  }
  return element;
}
```

ПЕРЕПИШЕМ `createProductNode`

```
function createProductNode(product) {
  return el('div', {class: 'product'}, [
    el('div', {class: 'product-title'}, product.title),
    el('div', {class: 'product-price'}, product.price)
  ]);
}
```

Live Demo

Стало меньше кода, не правда ли? И если прищуриться, можно почти представить, что это HTML-код =)

СТИЛИ И РАЗМЕРЫ

ВЗАИМОДЕЙСТВИЕ СО СТИЛЯМИ

В обычной ситуации мы динамически изменяем внешний вид элементов при помощи присвоения классов. Но иногда нам нужно как-то изменить стиль элемента при помощи данных, рассчитанных в JavaScript, и для этого мы можем взаимодействовать со стилями элемента напрямую.

ЗАДАЧА С КОЛОНКАМИ

Возьмем простую задачку. У нас есть несколько блоков с текстом:

```
<div data-equalize>
  <div class="column">
    Константин Константинопольский
  </div>
  <div class="column">
    Nullam ut arcu id lacus blandit finibus.
    Vivamus volutpat felis nec urna congue tincidunt.
  </div>
</div>
```

СТИЛИ ДЛЯ КОЛОНОК

```
.column {
  box-sizing: border-box;
  float: left;
  width: 50%;
  border: 1px solid rebeccapurple;
}
```

ПОСТАНОВКА ЗАДАЧИ

Нам нужно сделать колонки равной высоты по размеру бóльшего из блоков. Как это сделать?

ПЕРВОЕ РЕШЕНИЕ

Правильно, flexbox. А если нет flexbox, как мы сможем сделать это при помощи JavaScript?

`element.style`

`element.style` для каждого элемента дает нам доступ к объекту типа `CSSStyleDeclaration`, содержащего пары «*свойствоВТакомВотКейсе* - стиль» для данного элемента (например `element.style.marginTop`).

Но он содержит не все стили, а только те, которые были заданы через атрибут `style`.

Также при помощи этого свойства можно задавать стили для выбранного элемента. Эффект от них будет такой же, как от инлайн-стилей.

ФУНКЦИЯ `getComputedStyle()`

Но поскольку свойство `style` содержит только инлайн-стили, как же нам узнать реальные применяемые стили для выбранного элемента, с учетом CSS-каскада?

Для этого есть функция `getComputedStyle(element[, ':pseudo'])`, которая возвращает вычисленные значения стилей для элемента. Для `getComputedStyle` важно указвать полное название свойства, например, `marginTop` вместо `margin`.

РЕАЛЬНЫЕ РАЗМЕРЫ ЭЛЕМЕНТА

Но есть одно но. Использовать `getComputedStyle` для считывания геометрии объекта не лучшая идея.

Например, полученная таким образом ширина будет зависеть от `box-sizing`; будет выдавать `auto` для инлайн-элементов и тому подобное.

Правильный способ узнать размеры элемента – это `element.clientWidht` / `element.clientHeight`. Эти свойства возвращают размеры элемента с полями, но без границ.

ПРИМЕНЯЕМ ТЕОРИЮ НА ПРАКТИКЕ

```
function equalize(rootNode) {
  const children = Array
    .from(rootNode.children);
  const maxHeight = Math
    .max(...children.map(child => child.clientHeight));
  children.forEach(child => child.style.height =
    `${maxHeight}px`);
}
Array.from(document.querySelectorAll('[data-equalize]'))
  .forEach(equalize);
```

Live Demo

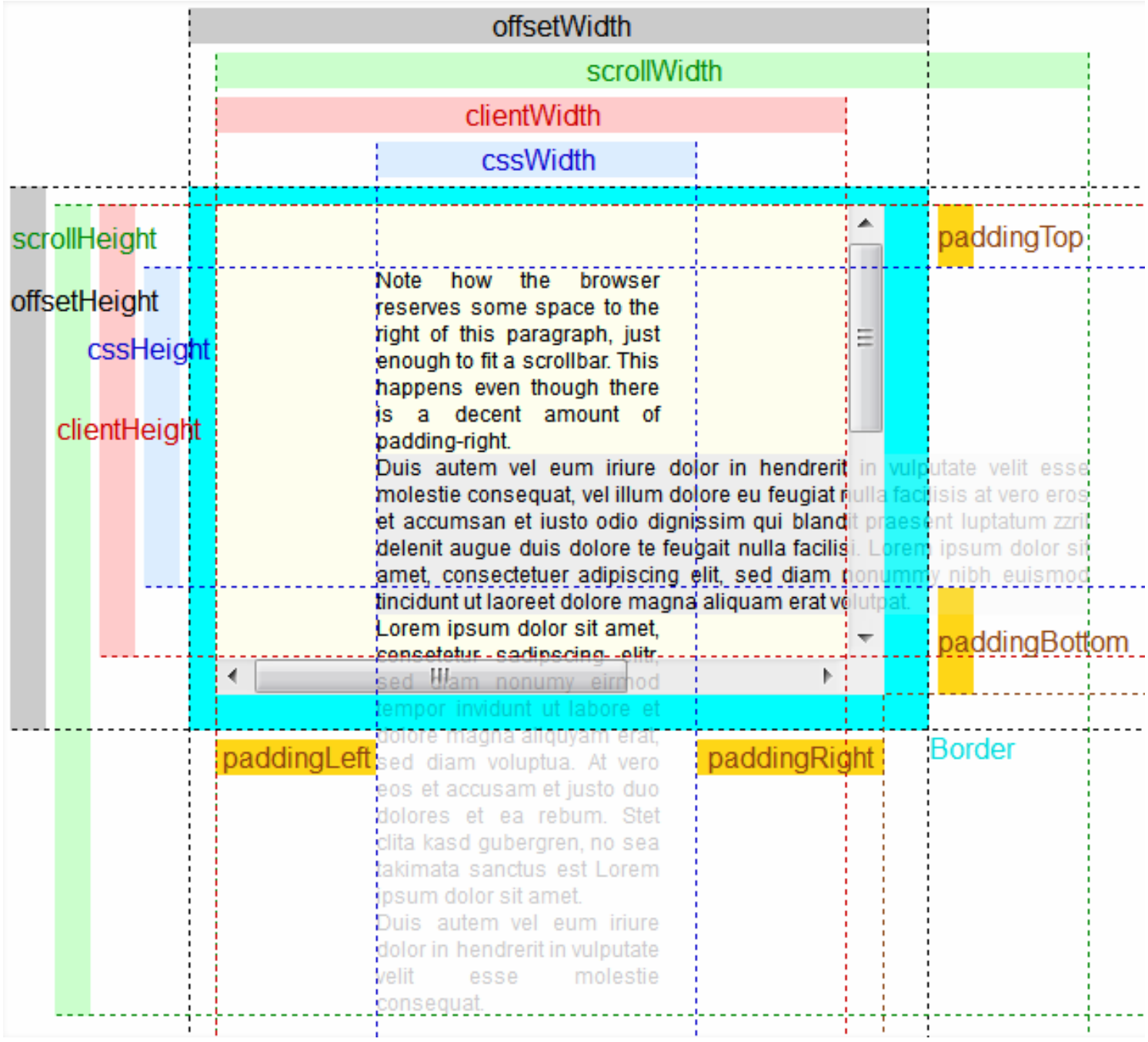
Обратите внимание, что присваивать размеры без единицы измерения нельзя.

ВЕРТИКАЛЬНОЕ ЦЕНТРОВАНИЕ

Мы хотим отцентровать один элемент внутри другого. Как это сделать без flexbox?

РАЗМЕТКА ДЛЯ ЗАДАЧИ

```
<style>
.outerBox {
  position: absolute;
  width: 200px;
  height: 200px;
  border: 10px solid red;
}
.innerBox {
  position: absolute;
  width: 100px;
  height: 100px;
  border: 10px solid blue;
}
</style>
<div class="outerBox"><div class="innerBox"></div></div>
```


offsetWidth
scrollWidth
clientWidth
cssWidth
scrollHeight
offsetHeight
cssHeight
clientHeight
paddingTop
paddingBottom
paddingLeft
paddingRight
Border
Note how the browser reserves some space to the right of this paragraph, just enough to fit a scrollbar. This happens even though there is a decent amount of padding-right.
Duis autem vel eum iriure dolor in hendrerit
molestie consequat, vel illum dolore eu feugiat
et accumsan et iusto odio dignissim qui bland
delenit augue duis dolore te feugait nulla facilis
amet, consectetuer adipiscing elit, sed diam
tincidunt ut laoreet dolore magna aliquam erat
Lorem ipsum dolor sit amet,

ГЕОМЕТРИЯ ЭЛЕМЕНТА

- `clientWidth` / `clientHeight` – ширина элемента + `padding`.
- `offsetWidth` / `offsetHeight` – ширина элемента + `padding` + `border` + `scrollbars`.
- `scrollWidth` / `scrollHeight` – размеры элемента с учетом прокрутки.

СКРИПТ ДЛЯ ВЕРТИКАЛЬНОГО ЦЕНТРИРОВАНИЯ

```
const innerBox = document.querySelector('.innerBox');
const outerBox = document.querySelector('.outerBox');
innerBox.style.left =`${outerBox.clientWidth / 2 -
 innerBox.offsetWidth / 2}px`;
innerBox.style.top =`${outerBox.clientHeight / 2 -
 innerBox.offsetHeight / 2}px`;
```

Live Demo

ЗАВИСАЮЩАЯ ШАПКА

Имитируем поведение свойства `position: sticky;` при помощи JavaScript. Шапка сайта при прокрутке на высоту, бóльшую, чем высота шапки, будет «прилипать» к верхней границе окна:

```
<header>Шапка</header>
<div class="content">Контент сайта</div>
```

СТИЛИ ДЛЯ ШАПКИ

```
.content {
  height: 3000px;
  background: gray;
}
header {
  height: 200px;
  background: teal;
  width: 100%;
}
header.fixed {
  position: fixed;
  height: 100px;
}
```

ОТСЛЕЖИВАЕМ ПРОКРУТКУ ПРАВИЛЬНО

Нас интересует прокрутка всей страницы, поэтому было бы логично обратиться к родительскому элементу в документе: `document.documentElement.scrollTop`?

Но не тут-то было, во многих браузерах элементом прокрутки является `body`! Опять кроссбраузерные проблемы.

Чтобы не думать обо всем этом, для определения основной прокрутки лучше использовать свойства `window.pageXOffset`/`window.pageYOffset`.

ПИШЕМ СКРИПТ

```
const header = document.querySelector('header');
document.addEventListener('scroll', event => {
  if (window.pageYOffset > header.clientHeight) {
    header.classList.add('fixed');
  } else {
    header.classList.remove('fixed');
  }
});
```

Live Demo

Этот код будет работать, но лучше для таких вещей использовать `requestAnimationFrame` для более плавной работы. Об этом будет в следующих лекциях.

ОПРЕДЕЛЕНИЕ ШИРИНЫ ДОКУМЕНТА

Теперь доработаем прошлую задачу, чтобы шапка зависала только при ширине окна больше `640px` (на мобильных она мешается).

К тому же шапка на мобильном будет другого цвета, при помощи такого CSS-кода:

```
@media (max-width: 640px) {
  header {
    background: red;
  }
}
```

ПРОВЕРКА КОРНЕВОГО ЭЛЕМЕНТА

Казалось бы, можно просто проверить размер корневого элемента?

```
if (document.documentElement.clientWidth > 640) {
  // Включить зависание
}
```

Но не тут-то было! Ведь при наличии полосы прокрутки `clientWidth` будет отдавать размер за вычетом прокрутки, а медиавыражения на прокрутку внимания не обращают!

`.innerWidth` / `.innerHeight`

Но нас спасут свойства `window.innerWidth` / `window.innerHeight`, которые отдают размеры окна, не обращая внимание на наличие прокрутки.

```
if (window.innerWidth > 640) {
  // Включить зависание
}
```

Теперь JavaScript-код будет работать синхронно с медиавыражениями.

ИТОГИ

СОЗДАЕМ DOM

- Для создания любого тега используем `createElement('h1')`.
- Для создания текста используем `createTextNode('some text')` или `.textContent` – они равнозначны.
- `createDocumentFragment` для создания разметки из более чем одного элемента.
- Нужна невалидная разметка? `document.write` в помощь.

СТИЛИ ЭЛЕМЕНТОВ

— `element.style` позволяет обратиться к стилям элемента. Просто допишите в конце полное название свойства в lowerCamelCase. Но так можно обратиться только к стилям в атрибуте `style`.

— Функция `getComputedStyle(element[, ':pseudo'])` возвращает вычислинные значения стилей элемента. Но у нее есть свои недостатки.

РАЗМЕРЫ СТРАНИЦЫ

- `element.clientWidht` / `element.clientHeight` — возвращают размеры элемента с полями, но без границ.
- Геометрия:
  - `clientWidth` / `clientHeight` – ширина элемента + `padding`.
  - `offsetWidth` /offsetHeight – ширина элемента + `padding` + `border` + `scrollbars`.
  - `scrollWidth` / `scrollHeight` – размеры элемента с учетом прокрутки.
- Для правильного отслеживания прокрутки `window.pageXOffset` / `window.pageYOffset`.
- `window.innerWidth` / `window.innerHeight` возвращают размеры окна без учета прокрутки.

нетология
университет интернет-профессий

Задавайте вопросы и напишите отзыв о лекции!

АЛЕКСАНДР ШЛЕЙКО

 a.shleyko@yandex.ru

 dustyo_O

vk.com/shleiko