нетология
университет интернет-профессий

MEDIA И ПРОДВИНУТАЯ АНИМАЦИЯ

АЛЕКСАНДР ШЛЕЙКО / ЯНДЕКС

АЛЕКСАНДР ШЛЕЙКО

Разработчик интерфейсов в Яндекс

a.shleyko@yandex.ru

vk.com/shleiko

ПЛАН ЗАНЯТИЯ

1. Анимация
2. Основы Paper.js
3. Нелинейные анимации
4. Видеокамера и микрофон

АНИМАЦИЯ

ВЫБОР ПОДХОДЯЩЕГО СПОСОБА

В условиях современных реалий браузеров выполнять анимацию средствами JavaScript приходится довольно редко. Ещё несколько лет назад большинство анимаций выполнялись при помощью библиотек вроде **jQuery**, сегодня большую часть анимаций принято реализовывать средствами CSS, что позволяет достичь лучших результатов за счёт частичного использования аппаратного ускорения и более низкоуровневых языков.

Несмотря на всё это, по-прежнему существуют задачи, которые имеет смысл решать при помощи JavaScript.

`setTimeout()`

Любой тип анимации в JavaScript в конечном итоге сводится к тому, что вы задаёте начальное и конечное состояние того или иного объекта и покадрово асинхронно обновляете то самое состояние.

Во времена, когда jQuery только стал популярным, асинхронность достигалась за счёт того, что каждый последующий кадр анимации планировался при помощи `setTimeout`.

Учитывая особенности работы браузера и, в частности, `event loop`, это не очень эффективно, так как `setTimeout()` вызывается гораздо чаще, чем происходит перерисовка страницы. Соответственно, если вы будете перерассчитывать состояние внутри `setTimout()`, то около 2/3 вызовов будут бесполезными.

ВТОРОЙ АРГУМЕНТ

Вы можете указать, как часто вызывать `setTimeout`, за счёт второго аргумента `(setTimeout(() => {...}, 1000 / 60))`, но всё равно это решение неидеально, потому что у вас нет никаких гарантий, что очередной вызов `setTimeout` произойдёт непосредственно перед перерисовкой страницы. Поэтому в определённый момент web-разработчикам стало сильно не хватать способа решить все перечисленные выше проблемы.

`requestAnimationFrame`

На сегодняшний день для анимации принято использовать `requestAnimationFrame` вместо `setTimeout`.

`requestAnimationFrame` вызывается около 60 раз в секунду, но, что самое главное, непосредственно перед фазой перерисовки страницы. В дополнение ко всему этому, первым аргументом в ваш callback будет передан точный таймстемп (с момента загрузки страницы) вызова (речь идёт о `DOMHighResTimeStamp`), что позволит вам с большей точностью рассчитать состояние анимируемого объекта.

ОТ 0 ДО 10

```
function animateNumber(from, to, target, duration = 1) {
  const durationMs = duration * 1000;
  let start = null;
  let timer = null;

  // расчет состояния элемента и обновление DOM

  timer = requestAnimationFrame(tick);
}
```

ФУНКЦИЯ РАСЧЕТА И ОБНОВЛЕНИЯ

```
function tick(timestamp) {
  start = start || timestamp;
  const elapsedTime = timestamp - start;
  const progress = elapsedTime / durationMs;
  const value = from + (to - from) * progress;
  if(progress >= 1) {
    target.innerText = to;
    return cancelAnimationFrame(timer);
  }
  target.innerText = value.toFixed(3);
  timer = requestAnimationFrame(tick);
}
```

ПЛАВНАЯ АНИМАЦИЯ

```
const output = document.getElementById('output');
animateNumber(0, 10, output, 10);
```

Live Demo

КЛЮЧЕВЫЕ ЭТАПЫ

Здесь всего несколько ключевых этапов:

1. Запоминаем время, когда мы начали анимацию;
2. Покадрово перерассчитываем новое число и обновляем его на странице;
3. Проверяем, не закончилась ли анимация, и если это так, не планируем следующий кадр.

АНИМАЦИЯ БЕЗ ПОЛЬЗОВАТЕЛЯ

Частота вызова `requestAnimationFrame` непосредственно зависит от того, как часто перерисовывается текущая страница.

Таким образом, если пользователь не находится во вкладке, в которой вы ждёте следующего фрейма, современные браузеры снизят частоту перерисовки до 10-15 фреймов в секунду, а то и меньше.

ОСНОВЫ PAPER.JS

«

Paper.js

Paper.js *представляет собой огромный набор различных инструментов для работы с Canvas.*

Основной изюминкой библиотеки яляется объектно-ориентированный подход, в отличие от того, что предлагает Canvas API.

Paper.js также предлагает свой собственный синтаксис, PaperScript, который позволяет проще использовать математические операции.

ПРИМИТИВНАЯ ЛИНЕЙНАЯ АНИМАЦИЯ

В качестве иллюстрации возможностей Paper.js нарисуем синий круг и проанимируем его движение вдоль оси x на 260px.

```
function animate(callback, duration = 1) {
  const durationMs = duration * 1000;

  return new Promise((resolve, reject) => {
    let start = null;
    let timer = null;

    // tick

    timer = requestAnimationFrame(tick);
  });
}
```

ФУНКЦИЯ `tick`

```
function tick(timestamp) {
  start = start || timestamp;

  const elapsedTime = timestamp - start;
  const progress = elapsedTime / durationMs;

  if(progress >= 1) {
    callback(1);
    resolve();
    return;
  }

  callback(progress);
  timer = requestAnimationFrame(tick);
}
```

ОБНОВЛЕННАЯ ФУНКЦИЯ

Заметьте, что мы используем слегка изменённую версию функции `animateNumber`, которую мы написали ранее в этой лекции и теперь переименовали в просто `animate`. Вся разница заключается лишь в том, что новая функция ничего сама по себе не изменяет, а лишь вызывает callback, который ей передаётся первым аргументом.

ОБРАТАТЫВАЕМ CANVAS

```
1  const { Path, Point, view } = paper;
2  const canvas = document.getElementById('canvas');
3
4  paper.setup(canvas);
5  const radius = 20;
6  const position = new Point(20, 50);
7  const circle = new Path.Circle(position, radius);
8  circle.fillColor = 'blue';
```

АНИМИРУЕМ ДВИЖЕНИЕ

```
const forward = animate(
  (progress) => {
    position.x = 20 + progress * 260;
    circle.position = position;
  }
, 3);

forward.then(() => animate(
  (progress) => {
    position.x = 20 + (1 - progress) * 260;
    circle.position = position;
  }
, 3));
```

Live Demo

БЕЗ ЛИШНИХ НАПОМИНАНИЙ

Примечательно то, что Paper.js не нужно явно указывать на то, что пора бы перерисовать всю сцену. Напротив, библиотека сама отслеживает изменения и перерисовывает, когда это необходимо.

НЕУДОБСТВО CANVAS

Вспомните лекцию про Canvas и то, как неудобно было организовывать слои при помощи функционального (практически) API Canvas. Когда количество слоёв превысит 10, работать без библиотек вроде Paper.js станет невыносимо сложно.

НЕЛИНЕЙНЫЕ АНИМАЦИИ

ПОД КАПОТОМ АНИМАЦИЙ

За практически любыми анимациями в JavaScript стоят функции либо методы, которые принимают реальное число ($\mathbb{R}$) от 0 до 1 и преобразовывают его в набор значений, которые характеризуют состояние того или иного объекта на странице.

ПАРАМЕТРИЧЕСКАЯ ФОРМА

Исходя из предыдущей идеи, можно заключить, что множество нелинейных паттернов движений проще задавать при помощи параметрической формы. Это значит, что вместо `y = f(x)` мы будем писать следующее:

```
x = f(t);
y = g(t);
```

Где `t` будет тем самым реальным числом от 0 до 1, характеризующим прогресс анимации.

ТРИГОНОМЕТРИЧЕСКИЕ ФУНКЦИИ

Функции, такие как `sin`, `cos` и им подобные, являются ключевыми при описании многих движений, в том числе потому, что их значения удобно варьируются между -1 и 1 и повторяются через определённый промежуток.

ПО ЧАСОВОЙ СТРЕЛКЕ

В качестве примера заставим квадрат двигаться по часовой стрелке вдоль заданной нами окружности.

```
class RectGoingAroundCircle {
  // конструктор
  x(t) {
    return this.center.x + this.radius * cos(t);
  }
  y(t) {
    return this.center.y + this.radius * sin(t);
  }
  get progress() { return this.t; }
  // сеттер
}
```

Live Demo

КОНСТРУКТОР

```
constructor(center, radius, rectSize) {
  this.t = 0;
  this.center = center;
  this.radius = radius;
  this.rectSize = rectSize;
  this.rect = new Path
    .Rectangle(new Point(0, 0), this.rectSize);
  this.rect.fillColor = 'blue';
}
```

СЕТТЕР

Здесь мы используем параметрическое уравнение окружности (`x(t) = r * cos(t); y(t) = r * sin(t)`), что позволяет перейти от реального числа к координатам нашего квадрата в плоскости ($\mathbb{R}$ -> $\mathbb{R}^2$). Этот метод-мутатор является ключевым в примере

```
set progress(t) {
  this.t = t;
  const x = this.x(2 * PI * t);
  const y = this.y(2 * PI * t);
  this.rect.position = new Point(x, y);
}
```

ТИК

```
let lastTick = 0;
const rect =
 new RectGoingAroundCircle(new Point(100, 100), 80, 30);
function tick(timestamp) {
  if (timestamp) {
    const deltaSec = (timestamp - lastTick) / 1000;
    const delta = deltaSec / 2;
    const progress = (rect.progress + delta) % 1;
    rect.progress = progress;
    lastTick = timestamp;
  }
  requestAnimationFrame(tick);
}
tick();
```

ВИДЕОКАМЕРА И МИКРОФОН

СОВРЕМЕННЫЕ ВОЗМОЖНОСТИ

Современные бразуеры позволяют разработчику захватывать картинку и звук с устройства пользователя. Дальше у вас очень много разных вариантов того, что сделать с записью: от простого отображения до сохранения и преобразования. В этой лекции мы не будем фокусироваться на работе с микрофоном, а остановимся лишь на камере.

ПОЛУЧАЕМ ДОСТУП

Перед тем, как что-либо сделать с камерой, необходимо получить доступ. Если вы запросите разрешение использовать камеру, то пользователь увидит уведомление с запросом, и в зависимости от того, разрешит он или запретит доступ, ваша программа будет уведомлена соответствующим образом, через promise.

Если всё пройдёт успешно, вам будет передана ссылка на объект класса `MediaStream`, который вы сможете далее использовать по своему усмотрению.

`getUserMedia`

Метод `getUserMedia` возвращает объект `promise`.

Если браузеру не удастся найти устройство, которое не будет противоречить вашим критериям, либо у пользователя вообще нет камеры и микрофона, то `promise` будет отклонён и вы сможете узнать об этом, поймав ошибку в `catch()`.

ЗАПРОС ДОСТУПА

```
navigator.mediaDevices
  .getUserMedia({video: true, audio: false})
  .then(stream => console.log('yay!', stream))
  .catch(err => console.warn('oh noes'));
```

Внутри `then` вы уже получите ссылку на поток.

Обратите внимание на первый аргумент, переданный в `getUserMedia`. Он отвечает за ограничения, возлагаемые вами на устройство. Вы можете передать объект класса `MediaStreamConstraints`, но если вы этого не сделаете, `getUserMedia` обернёт ваш простой объект за вас. Своего рода autoboxing.

ИСПОЛЬЗОВАННЫЕ ОБЪЕКТЫ

Мы использовали объекты `navigator` и `mediaDevices`. `navigator` содержит в себе различную информацию о текущем браузере, а также через этот объект принято организовывать взаимодействие с посторонними приложениями, интегрирующимися тем или иным образом с браузером. `mediaDevices` выступает в роли интерфейса к мультимедиа устройствам ввода.

ВЫВОДИМ ВИДЕО НА ЭКРАН

Чтобы вывести видео с камеры на экран, достаточно лишь воспользоваться тегом `<video>` и передать в него ссылку на Blob, которую вы можете сгенерировать при помощи `URL.createObjectURL`:

```
<video id="video"></video>
```

```
navigator.mediaDevices
  .getUserMedia({video: true, audio: false})
  .then((stream) => {
    const video = document.getElementById('video');
    video.src = URL.createObjectURL(stream);
  })
  .catch(err => console.warn('oh noes'));
```

ОБРАБОТКА КАДРОВ

Если впоследствии вы захотите сделать что-то необычное с кадрами видео, проще всего будет создать элемент `<canvas>`, использовать его метод `drawImage` и в качестве картинки передать DOM-объект, указывающий на ваш видеотег.

К такому примеру мы вернёмся в конце лекции.

ЗАПИСЬ ПОТОКА

Для записи видео и звука существует класс `MediaRecorder`, который принимает поток и оповещает, когда будет доступна следующая порция потока.

ЗАПИСЬ И ВОСПРОИЗВЕДЕНИЕ

Рассмотрим пример с записью видео и его дальнейшим воспроизведением (Live Demo):

```
recorder = new MediaRecorder(stream);
let chunks = [];
recorder.addEventListener('dataavailable', (e) => chunks
  .push(e.data));
recorder.addEventListener('stop', (e) => {
  const recorded =
    new Blob(chunks, { 'type' : recorder.mimeType });
  chunks = null;
  recorder = stream = null;
  videoEl.src = URL.createObjectURL(recorded);
});
recorder.start();
```

ПРИНЦИП РАБОТЫ

Событие `dataavailable`, на которое мы и подписываемся, будет вызываться каждый раз, когда `MediaRecorder` удобно скормить нам следующую порцию потока. Вы также можете контролировать частоту отправки вам порций. Также вы можете вручную вынудить `MediaRecorder` отдать вам порцию, вызвав метод `requestData`.

По окончании записи мы собираем порции в один большой Blob, который впоследствии может быть использован где-либо: передан на сервер или назначен в виде источника к `<video>`.

ВЗАИМОДЕЙСТВИЕ CANVAS, VIDEO, И MEDIA

Ранее мы упоминали, что в случае, если стандартных средств браузера было недостаточно для вашей задачи, стоило использовать комбинацию `<video>` + `<canvas>`.

Создадим простую страничку, которая по нажатию кнопки будет фотографировать вас и сохранять в картинку.

МОМЕНТАЛЬНОЕ ФОТО

```
navigator.mediaDevices
  .getUserMedia({video: true, audio: false})
  .then((stream) => {
    video.src = URL.createObjectURL(stream);
    video.addEventListener('canplay', (evt) => {
     setTimeout(() => {
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      ctx.drawImage(video, 0, 0);
      image.src = canvas.toDataURL();
      stream.getVideoTracks().map(track => track.stop());
     }, 100);
    });
   })
   .catch((err) => {});
```

КЛЮЧЕВОЙ МОМЕНТ

Live Demo

Ключевым моментом здесь является вызов метода `drawImage`, который забирает текущую картинку из видеотега, на который транслируется поток с вашей камерой, и рисует его на Canvas.

ИТОГИ

АНИМАЦИЯ

— Большая часть анимаций в современном интерфейсе реализована при помощи CSS.

— `setTimeout()` неэффективен, вызывается чаще, чем это нужно.

— `requestAnimationFrame` вызывается непосредственно перед отрисовкой страницы, порядка 60 раз в секунду.

— Первым аргументом в callback передается таймстеп вызова, что увеличивает точность расчета состояния элемента.

— Если вкладка не активна, то частота отрисовки автоматически снижается до менее чем 10-15 фреймов в секунду.

PAPER.JS

- Paper.js – сборник объектно-ориентированных инструментов для работы с Canvas.
- Библиотека сама отслеживает изменения и перерисовывает состояние по необходимости.
- Paper.js пригождается, когда слоев в canvas становится больше 10.

НЕЛИНЕЙНЫЕ АНИМАЦИИ

— Нелинейное движение проще описывать параметрической формулой.

— Для нелинейных анимаций очень пригождаются функции типа `sin`, `cos` и им подобные.

ВИДЕОПОТОК

– Перед началом работы с камерой нужно получить разрешение пользователя.

– В случае согласия пользователя передается ссылка на объект класса `MediaStream`.

– Метод `getUserMedia` возвращает объект `promise`.

– Отлавливайте ошибки при помощи `catch()`. Пригодится, если камера или микрофон не найдены или не подходят по параметрам.

– Объект `navigator` содержит информацию о браузере и помогает работать с посторонними приложениями.

– `mediaDevices` работает как интерфейс к мультимедиа устройствам ввода.

– Видео на экран выводим в теге `<video>` с ссылкой на Blob в атрибуте `src`. Ссылку генерируем при помощи `URL.createObjectURL`.

ЗАПИСЬ

- Класс `MediaRecorder` дает возможность записывать видео.
- `drawImage` забирает текущий кадр и видеотега и вставляет его в Canvas.

нетология
университет интернет-профессий

Задавайте вопросы и напишите отзыв о лекции!

АЛЕКСАНДР ШЛЕЙКО

 a.shleyko@yandex.ru

 dustyo_O

vk.com/shleiko