Лекція 9.
Анімація JavaScript

JavaScript-анімація застосовується там, де не підходить CSS.

Наприклад, по складній траєкторії, з тимчасовою функцією, що виходить за рамки кривих Безьє, на canvas. Іноді її використовують для анімації в старих IE.

setInterval

З точки зору HTML / CSS, анімація - це поступова зміна стилю DOM-елемента. Наприклад, збільшення координати style.left від 0px до 100px зрушує елемент.

Якщо збільшувати left від 0 до 100 за допомогою setInterval, роблячи по 50 змін в секунду, то це буде виглядати як плавне переміщення. Той же принцип, що і в кіно: для безперервної анімації досить 24 або більше викликів setInterval в секунду.

Псевдо-код для анімації виглядає так:

```
var fps = 50; // 50 кадрів в секунду
var timer = setInterval(function() {
    if (час вийшов) clearInterval(timer);
    else трохи збільшує left
}, 1000 / fps)
```

Більш повний приклад коду анімації:

```
var start = Date.now(); // зберегти час початку

var timer = setInterval(function() {

 // розрахувати скільки часу пройшло з початку анімації

 var timePassed = Date.now() - start;

 if (timePassed >= 2000) {

  clearInterval(timer); // конець через 2 секунди

  return;

 }
```

```
  // малює стан анімації, відповідає часу timePassed
  draw(timePassed);
}, 20);
// в той час як timePassed йде від 0 до 2000
// left приймає значення від 0 до 400px
function draw(timePassed) {
  train.style.left = timePassed / 5 + 'px';
}
```

requestAnimationFrame

Якщо у нас не один такий setInterval, а кілька в різних місцях коду, то браузеру потрібно в ті ж 20 мс працювати зі сторінкою вже кілька разів. Але ж крім setInterval є ще інші дії, наприклад, прокрутка сторінки, яку теж треба намалювати.

Якщо всі дії по перемальовуванні виробляти незалежно, то буде виконуватися багато подвійної роботи.

Набагато вигідніше з точки зору продуктивності - згрупувати всі перемальовування в одну і запускати їх централізовано, все разом.
Для цього в JavaScript-фреймворку, які підтримують анімацію, є єдиний таймер:

```
setInterval (function () {
/ * Відмальовує всі анімації * /
}, 20);
```

... Всі анімації, які запускає такий фреймворк, додаються до загального списку, і раз в 20 мс єдиний таймер перевіряє його, запускає поточні, видаляє закінчені.

Сучасні браузери, крім IE9-, підтримують стандарт Animation timing, який являє собою подальший крок у цьому напрямку. Він дозволяє синхронізувати наші анімації з вбудованими механізмами оновлення сторінки. Тобто, згруповані будуть не тільки наші, а й CSS-анімації та інші браузерні перемальовування.

При цьому графічний прискорювач буде використаний максимально ефективно, і виключена повторна обробка одних і тих же ділянок сторінки. А значить - менше буде завантаження CPU, та й сама анімація стане більш плавною.

Для цього використовується функція requestAnimationFrame.

синтаксис:

```
var requestId = requestAnimationFrame (callback)
```

Такий виклик планує запуск callback найближчим часом, коли браузер вважатиме можливим здійснити анімацію.

Якщо запланувати в callback якесь малювання, то воно буде згруповано з іншими requestAnimationFrame і з внутрішніми перемальовуваннями браузера.

Значення, що повертається requestId служить для скасування запуску:

```
// скасувати заплановане вище виконання callback
cancelAnimationFrame (requestId);
```

Функція callback отримує один аргумент - час, що минув з початку завантаження сторінки, результат виклику performance.now ().

Як правило, запуск callback відбувається дуже скоро. Якщо у процесора велике завантаження або батарея у ноутбука майже розряджена - то рідше.

```
<Script>
 var prev = performance.now ();
 var times = 0;

 requestAnimationFrame (function measure (time) {
  document.body.insertAdjacentHTML ( "beforeEnd", Math.floor (time - prev) + "");
  prev = time;

  if (times ++ <10) requestAnimationFrame (measure);
 })
</ Script>
```

Функція анімації на основі requestAnimationFrame:

```
// Малює функція draw
// Тривалість анімації duration
function animate (draw, duration) {
  var start = performance.now ();

  requestAnimationFrame (function animate (time) {
    // визначити, скільки минуло часу з початку анімації
    var timePassed = time - start;

    // можливе невелике перевищення часу, в цьому випадку зафіксувати кінець
    if (timePassed> duration) timePassed = duration;

    // намалювати стан анімації в момент timePassed
    draw (timePassed);

    // якщо час анімації не закінчилося - запланувати ще кадр
    if (timePassed <duration) {
      requestAnimationFrame (animate);
    }

  });
}
```

Структура анімації

На основі requestAnimationFrame можна спорудити і набагато більш потужну, але в той же час просту функцію анімації.

У анімації є три основних параметри:

duration

Загальний час, який має тривати анімація, в мс. Наприклад 1000.

timing (timeFraction)

Тимчасова функція, яка, за аналогією з CSS-властивість transition-timing-function, буде по поточному часу обчислювати стан анімації.

Вона отримує на вхід безперервно зростаюче число timeFraction - від 0 до 1, де 0 означає самий початок анімації, а 1 - її кінець.

Її результатом має бути значення завершеності анімації, якому в CSS transitions на кривих Безьє відповідає координата y.

Також за аналогією з transition-timing-function повинні дотримуватися умови:

• timing (0) = 0
• timing (1) = 1

... Тобто, анімація починається в точці (0,0) - нульовий час і нульовий прогрес і закінчується в (1, 1) - пройшло повне час, і процес завершено.

Наприклад, функція-пряма означає рівномірний розвиток процесу:

```
function linear (timeFraction) {
  return timeFraction;
}
```

Є й інші, більш цікаві варіанти, ми розглянемо їх трохи пізніше.

```
draw (progress)
```

Функція, яка отримує стан завершеності анімації і малює його. Значенням progress = 0 відповідає початкова точка анімації, progress = 1 - кінцева.

Саме ця функція і здійснює, власне, анімацію.

Наприклад, може рухати елемент:

```
function draw (progress) {
  train.style.left = progress + 'px';
}
```

Можливі будь-які варіанти, анімувати можна що завгодно і як завгодно.

Анімуємо ширину елемента width від 0 до 100%, використовуючи нашу функцію.

Код для запуску анімації:

```
animate ({
  duration: 1000,
  timing: function (timeFraction) {
```

```
  return timeFraction;
 },
 draw: function (progress) {
  elem.style.width = progress * 100 + '%';
 }
});
```

Тимчасові функції

Вище ми бачили найпростішу, лінійну тимчасову функцію.

Розглянемо приклади анімації руху з використанням різних timing.

В степені n

Ось ще один простий випадок - progress в ступеня n. Окремі випадки - квадратична, кубічна функції і т.д.

Для квадратичної функції:

```
function quad(progress) {
 return Math.pow(progress, 2)
}
```

Відскік bounce

Уявіть, що ми відпускаємо м'яч, він падає на підлогу, кілька разів відскакує і зупиняється.

Функція bounce робить те ж саме, тільки навпаки: «підстрибування» починається відразу.

Ця функція трохи складніше попередніх і використовує спеціальні коефіцієнти:

```
function bounce (timeFraction) {
  for (var a = 0, b = 1, result; 1; a + = b, b / = 2) {
   if (timeFraction> = (7 - 4 * a) / 11) {
```

```
    return -Math.pow ((11 - 6 * a - 11 * timeFraction) / 4, 2) + Math.pow (b, 2)
   }
  }
}
```

Реверсивні функції ease

Отже, у нас є колекція тимчасових функцій.

Їх пряме використання називається «easeIn».

Іноді потрібно показати анімацію в зворотному режимі. Перетворення функції, яке дає такий ефект, називається «easeOut».

easeOut

У режимі «easeOut», значення timing обчислюється за формулою:

```
timingEaseOut (timeFraction) = 1 - timing (1 - timeFraction)
```

Наприклад, функція bounce в режимі «easeOut»:

```
// звичайний варіант
function bounce (timeFraction) {
  for (var a = 0, b = 1, result; 1; a + = b, b / = 2) {
    if (timeFraction> = (7 - 4 * a) / 11) {
      return -Math.pow ((11 - 6 * a - 11 * timeFraction) / 4, 2) + Math.pow (b, 2);
    }
  }
}

// перетворювач в easeOut
function makeEaseOut (timing) {
  return function (timeFraction) {
    return 1 - timing (1 - timeFraction);
  }
}

var bounceEaseOut = makeEaseOut (bounce);
```

Повний приклад - відскік в bounceEaseOut тепер не на початку, а в кінці:

Якщо є анімаційний ефект, такий як підстрибування - він буде показаний в кінці, а не на початку (або навпаки, на початку, а не в кінці).

• Зазвичай аніміруемий об'єкт спочатку повільно скаче внизу, а потім, в кінці, різко досягає верху ...

• А після easeOut - він спочатку стрибає вгору, а потім повільно скаче вниз.

easeInOut

А ще можна зробити так, щоб показати эффект і в початку і в кінці анімації. Відповідне перетворення називається «easeInOut».

Його код виглядає так:

```
if (timeFraction <= 0.5) { // перша половина анімації)
  return timing(2 * timeFraction) / 2;
} else { // друга половина
  return (2 - timing(2 * (1 - timeFraction))) / 2;
}
```

Код, який трансформує timing:

```
function makeEaseInOut(timing) {
  return function(timeFraction) {
    if (timeFraction < .5)
      return timing(2 * timeFraction) / 2;
    else
      return (2 - timing(2 * (1 - timeFraction))) / 2;
  }}
bounceEaseInOut = makeEaseInOut(bounce);
```

Трансформація «easeInOut» об'єднує в собі два графіка в один: easeIn для першої половини анімації і easeOut - для другої.

Процес анімації повністю в ваших руках завдяки timing. Її можна зробити настільки реалістичною, наскільки захочеться.

Втім, виходячи з практики, можна сказати, що варіанти timing, описані вище, покривають 95% потреб в анімації.