С.Д. Приходченко ©

http://pzks.nmu.org.ua/ ©

Лабораторная работа №7

Анимация в JavaScript

Цель работы: Научиться создавать простейшие приложения при помощи JavaScript

Теоретические сведения

Основы анимации

С точки зрения HTML/CSS, анимация — это постепенное изменение стиля DOM-элемента. Например, увеличение координаты style.left от 0px до 100px сдвигает элемент.

Код, который производит изменение, вызывается таймером. Интервал таймера очень мал и поэтому анимация выглядит плавной. Это тот же принцип, что и в кино: для непрерывной анимации достаточно 24 или больше вызовов таймера в секунду.

Псевдо-код для анимации выглядит так:

```
var timer = setInterval(function() {
   показать новый кадр
   if (время вышло) clearInterval(timer);
}, 10)
```

Задержка между кадрами в данном случае составляет 10 ms, что означает 100 кадров в секунду.

В большинстве фреймворков, задержка по умолчанию составляет 10-15 мс. Меньшая задержка делает анимацию более плавной, но только в том случае, если браузер достаточно быстр, чтобы анимировать каждый шаг вовремя.

Если анимация требует большого количества вычислений, то нагрузка процессора может доходить до 100% и вызывать ощутимые «тормоза» в работе браузера. В

таком случае, задержку можно увеличить. Например, 40мс дадут нам 25 кадров в секунду, что очень близко к кинематографическому стандарту в 24 кадра.

setInterval вместо setTimeout

Мы используем setInterval, а не рекурсивный setTimeout, потому что нам нужен один кадр за промежуток времени, а не фиксированная задержка между кадрами.

Пример

Например, передвинем элемент путём изменения element.style.left от 0 до 100px. Изменение происходит на 1px каждые 10мс.

```
<script>
function move(elem) {
  var left = 0; // начальное значение
  function frame() { // функция для отрисовки
    left++;
    elem.style.left = left + 'px'
    if (left == 100) {
      clearInterval(timer); // завершить анимацию
    }
  }
  var timer = setInterval(frame, 10) // рисовать каждые 10мс
}
</script>
<div onclick="move(this.children[0])" class="example_path">
  <div class="example_block"></div>
</div>
```

Структура анимации

У анимации есть три основных параметра:

Delay Время между кадрами (в миллисекундах, т.е. 1/1000 секунды). Например, 10мс.

Duration Общее время, которое должна длиться анимация, в мс. Например, 1000мс.

step(progress) Функция step(progress) занимается отрисовкой состояния анимации, соответствующего времени progress.

Каждый кадр выполняется, сколько времени прошло: progress = (now-start)/duration. Значение progress меняется от 0 в начале анимации до 1 в конце. Так как вычисления с дробными числами не всегда точны, то в конце оно может быть даже немного больше 1. В этом случае мы уменьшаем его до 1 и завершаем анимацию.

Создадим функцию animate, которая получает объект со свойствами delay, duration, step и выполняет анимацию.

```
  function animate(opts) {
    var start = new Date; // сохранить время начала
```

```
  var timer = setInterval(function() {
    // вычислить сколько времени прошло
    var progress = (new Date - start) / opts.duration;
    if (progress > 1) progress = 1;
    // отрисовать анимацию
    opts.step(progress);
    if (progress == 1) clearInterval(timer); // конец :)
  }, opts.delay || 10); // по умолчанию кадр каждые 10мс }
```

Пример

Анимируем ширину элемента width от 0 до 100%, используя нашу функцию:

```
function stretch(elem) {
 animate({
   duration: 1000, // время на анимацию 1000 мс
   step: function(progress) {
     elem.style.width = progress*100 + '%';
   }
 });
}
```

Функция step может получать дополнительные параметры анимации из opts (через this) или через замыкание.

Следующий пример использует параметр to из замыкания для анимации бегунка:

```
function move(elem) {
  var to = 500;
  animate({
    duration: 1000,
    step: function(progress) {
      // progress меняется от 0 до 1, left от 0px до 500px
```

```
      elem.style.left = to*progress + "px";
    }
  });
}
```

Временная функция delta

В сложных анимациях свойства изменяются по определённому закону. Зачастую, он гораздо сложнее, чем простое равномерное возрастание/убывание.

Для того, чтобы можно было задать более хитрые виды анимации, в алгоритм добавляется дополнительная функция delta(progress), которая вычисляет текущее состояние анимации от 0 до 1, а step использует её значение вместо progress.

В animate изменится всего одна строчка. Было: ...

```
opts.step(progress);
...
```

Станет: ...

```
opts.step( opts.delta(progress) );
...
```

Такое небольшое изменение добавляет много гибкости. Функция step занимается всего лишь отрисовкой текущего состояния анимации, а само состояние по времени определяется в delta.

Разные значения delta заставляют скорость анимации, ускорение и другие параметры вести себя абсолютно по-разному.

Рассмотрим примеры анимации движения с использованием различных delta.

Самая простая функция delta — это та, которая просто возвращает progress.

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

То есть, как будто никакой delta нет. Состояние анимации (которое при передвижении отображается как координата left) зависит от времени линейно.

Пример:

```
<div onclick="move(this.children[0], linear)" class="example_path">
  <div class="example_block"></div>
</div>
```

Здесь и далее функция move будет такой:

```
function move(elem, delta, duration) {
  var to = 500;
  animate({
    delay: 10,
    duration: duration || 1000,
    delta: delta,
    step: function(delta) {
      elem.style.left = to*delta + "px"
    }
  });
}
```

Вот еще один простой случай. delta - это progress в n-й степени . Частные случаи - квадратичная, кубическая функции и т.д. Увеличение степени влияет на ускорение.

Для квадратичной функции:

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

Эта функция работает по принципу лука: сначала мы «натягиваем тетиву», а затем «стреляем».

В отличие от предыдущих функций, эта зависит от дополнительного параметра x, который является «коэффициентом упругости». Он определяет расстояние, на которое «оттягивается тетива». Её код:

```
function back(progress, x) {
   return Math.pow(progress, 2) * ((x + 1) * progress - x)
}
```

Обычно, JavaScript-фреймворк предоставляет несколько delta-функций. Их прямое использование называется «easeIn».

Иногда нужно показать анимацию в обратном режиме. Преобразование функции, которое даёт такой эффект, называется «easeOut».

easeOut

В режиме «easeOut», значение delta вычисляется так:

deltaEaseOut(progress) = 1 - delta(1 - progress)

Например, функция bounce в режиме «easeOut»:

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

Функция highlight, представленная ниже, анимирует изменение цвета.

```
function highlight(elem) {
  var from = [255,0,0], to = [255,255,255]
  animate({
    delay: 10,
    duration: 1000,
    delta: linear,
```

```
    step: function(delta) {
      elem.style.backgroundColor = 'rgb(' +
        Math.max(Math.min(parseInt((delta * (to[0]-from[0])) + from[0], 10), 255), 0) + ',' +
        Math.max(Math.min(parseInt((delta * (to[1]-from[1])) + from[1], 10), 255), 0) + ',' +
        Math.max(Math.min(parseInt((delta * (to[2]-from[2])) + from[2], 10), 255), 0) + ')'
    }
  })
}
```

Вы можете создавать интересные анимации, как, например, набор текста в «скачущем» режиме (нужна область Textarea с каким-нибудь текстом, и кнопка, по нажатию на которую происходит віполнение следующей функции):

```
function animateText(textArea) {
  var text = textArea.value
  var to = text.length, from = 0
  animate({
    delay: 20,
    duration: 5000,
    delta: bounce,
    step: function(delta) {
      var result = (to-from) * delta + from
      textArea.value = text.substr(0, Math.ceil(result))
    }
  })
}
```

Задания:

1. Полет мяча (bounce) ; Набор текста с замедляющимся выводом.
2. Полет снежинки (синусоида); Набор текста с ускоряющимся выводом.
3. Полет мухи (random); Набор текста с замедляющимся выводом.

4. Рисование заданного графика (sin(x)); Набор текста с ускоряющимся выводом.
5. Рисование заданного графика (x^2); Набор текста с замедляющимся выводом.
6. Полет мяча (bounce) ; Набор текста с ускоряющимся выводом.
7. Полет снежинки (синусоида); Набор текста с замедляющимся выводом.
8. Полет мухи (random); Набор текста с ускоряющимся выводом.
9. Рисование заданного графика (sin(x)); Набор текста с замедляющимся выводом.
10. Рисование заданного графика (x^2); Набор текста с ускоряющимся выводом.