Вращение

jQueryRotate — крутим картинки

Автор: Александр Головко Дата публикации: 09.11.2011

Плагин jQueryRotate позволяет поворачивать картинки (img) на заданный угол. Реализован как статический поворот, так и динамическое вращение.

По заверениям разработчиков, для современных браузеров используются CSS3-свойства и их вендорные аналоги, для старых браузеров используется рисование на канве или VML.

Демо-пример.

Проверено в:

IE 6-9

Firefox 3+

Opera 9+

Safari

Chrome

Что качать?

библиотеку jquery качаем или подключаем из хранилища.

jqueryrotate.2.1.js (10.69 Kb) — собственно плагин.

Быстрый старт

Подключаем библиотеки:

HTML1 <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js"></script>

2 <script type="text/javascript" src="js/jqueryrotate.2.1.js"></script>

Как-либо выделяем картинку, которую будем вращать. Например, присваиваем id.

HTML1 <img src="img/pic1.jpg" height="200" width="200" alt="" id="rotateImg"/>

Даем команду на поворот на 45 градусов:

Javascript1 <script type="text/javascript">

2 jQuery(document).ready(function() {

3 jQuery("#rotateImg").rotate(45);

4 });

5 </script>

А теперь подробнее

Итак, управление вращением происходит с помощью функции rotate(). Результат вращения определяется переданными ей параметрами.

Собственно, есть два варианта:

Вещественное число (как в примере выше) — угол в градусах, на который повернется картинка. Положительное число — поворот по часовой стрелке, отрицательное — против.

Объект, содержащий параметры вращения.

Параметры вращения:Название параметра Описание Значение по-умолчанию

angle Угол мгновенного поворота в градусах. 0

animateTo Угол анимированного плавного поворота в градусах. 0

duration Длительность анимации поворота при использовании animateTo, мс. 1000

easing При использовании плагина jQuery Easing, можно задать неравномерное вращение.

bind Объект, содержащий события, при которых необходимо вращение картинки.

callback Функция, вызывающаяся после завершение анимации.

Примеры

Поворот с использованием числового параметра и объекта (результат одинаков):

Javascript1 <script type="text/javascript">

2 jQuery("#rotateImg").rotate(45);

3

4 jQuery("#rotateImg").rotate({angle:45});

5 </script>

Поворот по наведению/убиранию мыши:

Javascript01 <script type="text/javascript">

02 jQuery("#rotateImg").rotate({

03 bind:

04 {

05 mouseover : function() {

06 jQuery(this).rotate({animateTo:180})

07 },

08 mouseout : function() {

09 jQuery(this).rotate({animateTo:0})

10 }

11 }

12 });

13 </script>

Постоянное вращение с использованием таймера:

Javascript1 <script type="text/javascript">

2 var angle = 0;

3 setInterval(function(){

4 angle+=3;

5 jQuery("#rotateImg").rotate(angle);

6 },50);

7 </script>

Постоянное вращение c использованием рекурсивного вызова функции:

Javascript1 <script type="text/javascript">

2 var rotation = function (){

3 jQuery("#rotateImg").rotate({

4 angle:0,

5 animateTo:360,

6 callback: rotation

7 });

8 }

9 rotation();

Вращение контента

Для нормальных современных браузеров плагин позволяет вращать не только картинки, но и другой обычный контент. Хотя красота отображения повернутых шрифтов в некоторых браузерах оставляет желать лучшего.

В старых версиях браузеров плагин, как минимум, не сможет повернуть не img (например IE7 лично у меня вообще повис наглухо).

Помни! Разработчики позиционируют свое детище именно, как плагин для поворота картинок. Только в этом случае гарантируется кроссбраузерность.

Материал:

jquery plugin :: jqueryrotate

По теме:

jquery.flip — плагин для переворота контента

Текст под углом

Вертикальный текст на CSS

IE's CSS3 Transforms Translator

Вращение элементов при прокрутке при помощи JavaScript

От автора: когда я решил переписать код и сменить бренд на сайте The New Code, я дал себе слово, что не буду использовать фреймворки: основной текст и примеры отображения кода остались нетронуты, однако под капотом теперь vanilla JavaScript. Я уже писал похожую статью про то, как вращать элементы во время скроллинга страницы при помощи JavaScript. В этой статье я представлю вам обновленную версию того демо, но только теперь на vanilla JavaScript … данный подход во многих отношениях проще и эффективнее

Создаем шестеренки

Как и в предыдущем примере, шестеренки размещаются на странице каждая со своим идентификатором. Наши шестеренки это векторные изображения, так что можно представить их в SVG:

view source

print

?

1.

<div id="gearbox">

2.

<img src="gear.svg" alt id="leftgear">

3.

<img src="gear.svg" alt id="rightgear">

4.

</div>

Разместить шестеренки на странице можно как угодно: очень часто можно встретить фиксированное положение; также можно воспользоваться новым CSS свойством position: sticky. Для размещения элементов я воспользовался flexbox, в качестве единиц измерения выбрал vw; сами шестерни имеют фиксированное позиционирование, а значит, при прокрутке двигаются вместе со страницей.

view source

print

?

01.

#gearbox {

02.

display: flex;

03.

justify-content: space-between;

04.

}

05.

#leftgear, #rightgear {

06.

width: 20vw;

07.

max-width: 20%;

08.

height: auto;

09.

}

Вращаем шестеренки

Задачка довольно простая: внизу страницы необходимо добавить следующий скрипт.

view source

print

?

1.

var leftgear = document.getElementById("leftgear"),

2.

rightgear = document.getElementById("rightgear");

3.

4.

window.addEventListener("scroll", function() {

5.

leftgear.style.transform = "rotate("+window.pageYOffset+"deg)";

6.

rightgear.style.transform = "rotate(-"+window.pageYOffset+"deg)";

7.

});

Шестеренки прокручиваются на значение, равное количеству прокрчуенных пикселей экрана, конвертированных в градусы. Левая вращается по часовой стрелке, правая – против часовой. Для ускорения или замедления вращения можно воспользоваться математическим выражением для window.pageYOffset: к примеру, умножив его на 2 или разделив.

Избавляемся от подергиваний во время анимации

При использовании скроллинга часто возникает проблема, что манипулируемый элемент начинает дергаться: во время перерисовки браузером элемента тот начинает дергаться, анимация пытается подстроиться под текущие значения. Избавиться от такого поведения можно, заставив элемент двигаться только в те моменты, когда браузер к этому готов, т.е. при помощи requestAnimationFrame. Для этого необходимо изменить addEventListener на пользовательское событие:

view source

print

?

1.

window.addEventListener("optimizedScroll", function() {

2.

...

3.

})

Над обработчиком события добавьте анонимную функцию:

view source

print

?

01.

;(function() {

02.

var throttle = function(type, name, obj) {

03.

var obj = obj || window;

04.

var running = false;

05.

var func = function() {

06.

if (running) { return; }

07.

running = true;

08.

requestAnimationFrame(function() {

09.

obj.dispatchEvent(new CustomEvent(name));

10.

running = false;

11.

});

12.

};

13.

obj.addEventListener(type, func);

14.

};

15.

throttle ("scroll", "optimizedScroll");

16.

})();

В следующих статьях я более подробно расскажу про requestAnimationFrame.

Источник: http://thenewcode.com/

Редакция: Команда webformyself.

CSS-анимации

Все современные браузеры, кроме IE9- поддерживают CSS transitions и CSS animations, которые позволяют реализовать анимацию средствами CSS, без привлечения JavaScript.

Однако, как мы увидим далее, для более тонкого контроля анимации JavaScript вовсе не будет лишним.

CSS transitions

Идея проста. Мы указываем, что некоторое свойство будет анимироваться при помощи специальных CSS-правил. Далее, при изменении этого свойства, браузер сам обработает анимацию.

Например, CSS, представленный ниже, 3 секунды анимирует свойство background-color.

.animated {

transition-property: background-color;

transition-duration: 3s;

}

Теперь любое изменение фонового цвета будет анимироваться в течение 3х секунд.

При клике на эту кнопку происходит анимация её фона: ✍

<button id="color">Кликни меня</button>

<style>

#color {

transition-property: background-color;

transition-duration: 3s;

}

</style>

<script>

color.onclick = function() {

this.style.backgroundColor = 'red';

}

</script>

Есть всего 5 свойств, задающих анимацию:

●

transition-property

●

transition-duration

●

transition-timing-function

●

transition-delay

Далее мы изучим их все, пока лишь заметим, что общее свойство transition может перечислять их все, в порядке: property duration timing-function delay, а также задавать анимацию нескольких свойств сразу.

Например, при клике на эту кнопку анимируются одновременно цвет и размер шрифта: ✍

<button id="growing">Кликни меня</button>

<style>

#growing {

transition: font-size 3s, color 2s;

}

</style>

<script>

growing.onclick = function() {

this.style.fontSize='36px';

this.style.color='red';

}

</script>

Далее мы рассмотрим свойства анимации по отдельности.

transition-property

Список свойств, которые будут анимироваться, например: left, margin-left, height, color.

Анимировать можно не все свойства, но многие. Значение all означает «анимировать все свойства».

transition-duration

Продолжительность анимации, задаётся в формате CSS time, то есть в секундах s или ms.

transition-delay

Задержка до анимации. Например, если transition-delay: 1s, то анимация начнётся через 1 секунду после смены свойства.

Возможны отрицательные значения, при этом анимация начнётся с середины.

Например, вот анимация цифр от 0 до 9:Результат

script.js

style.css

index.html 

✍

Она осуществляется сменой margin-left у элемента с цифрами, примерно так:

#stripe.animate {

margin-left: -174px;

transition-property: margin-left;

transition-duration: 9s;

}

В примере выше JavaScript просто добавляет элементу класс – и анимация стартует:

digit.classList.add('animate');1

Можно стартовать её «с середины», с нужной цифры, например соответствующей текущей секунде, при помощи отрицательного transition-delay.

В примере ниже при клике на цифру она начнёт двигаться с текущей секунды:Результат

script.js

style.css

index.html 

✍

В JavaScript это делается дополнительной строкой:

stripe.onclick = function() {

var sec = new Date().getSeconds() % 10;

// например, значение -3s начнёт анимацию с 3-й секунды

stripe.style.transitionDelay = '-' + sec + 's';

stripe.classList.add('animate');

};

transition-timing-function

Временнáя функция, которая задаёт, как процесс анимации будет распределён во времени, например начнётся ли анимация медленно, чтобы потом ускориться или наоборот.

Самое сложное, но при небольшом изучении – вполне очевидное свойство.

У него есть два основных вида значения: кривая Безье и по шагам. Начнём с первого.

Кривая Безье

В качестве временной функции можно выбрать любую кривую Безье с 4 опорными точками, удовлетворяющую условиям:

Начальная точка (0,0).

Конечная точка (1,1).

Для промежуточных точек значения x должны быть в интервале 0..1, y – любыми.

Синтаксис для задания кривой Безье в CSS: cubic-bezier(x2, y2, x3, y3). В нём указываются координаты только двух точек: второй и третьей, так как первая и последняя фиксированы.

Она указывает, как быстро развивается процесс анимации во времени.

●

По оси x идёт время: 0 – начальный момент, 1 – конец времени transition-duration.

●

По оси y – завершённость процесса: 0 – начальное значение анимируемого свойства, 1 – конечное.

Самый простой вариант – это когда процесс развивается равномерно, «линейно» по времени. Это можно задать кривой Безье cubic-bezier(0, 0, 1, 1).

График этой «кривой» таков:

…Как видно, это просто прямая. По мере того, как проходит время x, завершённость анимации y равномерно приближается от 0 к 1.

Поезд в примере ниже с постоянной скоростью «едет» слева направо, используя такую временную функцию:Результат

style.css

index.html 

✍

CSS для анимации:

.train {

left: 0;

transition: left 5s cubic-bezier(0, 0, 1, 1);

/\* JavaScript ставит значение left: 450px \*/

}

Как нам показать, что поезд тормозит?

Для этого используем кривую Безье: cubic-bezier(0.0, 0.5, 0.5 ,1.0).

График этой кривой:

Как видно, процесс вначале развивается быстро – кривая резко идёт вверх, а затем всё медленнее, медленнее.

Вы можете увидеть эту временную функцию в действии, кликнув на поезд:Результат

style.css

index.html 

✍

CSS для анимации:

.train {

left: 0;

transition: left 5s cubic-bezier(0, .5, .5, 1);

/\* JavaScript ставит значение left: 450px \*/

}

Существует несколько стандартных обозначений кривых: linear, ease, ease-in, ease-out и ease-in-out.

Значение linear – это прямая, мы её уже видели.

Остальные кривые являются короткой записью следующих cubic-bezier:ease\* ease-in ease-out ease-in-out

(0.25, 0.1, 0.25, 1.0) (0.42, 0, 1.0, 1.0) (0, 0, 0.58, 1.0) (0.42, 0, 0.58, 1.0)

\* – По умолчанию, если никакой временной функции не указано, используется ease.

Кривая Безье может заставить анимацию «выпрыгивать» за пределы диапазона.

Допустимо указывать для кривой Безье как отрицательные y, так и сколь угодно большие. При этом кривая Безье будет также по y выскакивать за пределы диапазона 0..1, представляющего собой начало-конец значения.

В примере ниже CSS-код анимации таков:

.train {

left: 100px;

transition: left 5s cubic-bezier(.5, -1, .5, 2);

/\* JavaScript поменяет left на 400px \*/

}

Свойство left должно меняться от 100px до 400px.

Однако, если кликнуть на поезд, то мы увидим, что:

●

Он едет сначала назад, то есть left становится меньше 100px.

●

Затем вперёд, причём выезжает за назначенные 400px.

●

А затем опять назад – до 400px.Результат

style.css

index.html 

✍

Почему так происходит – отлично видно, если взглянуть на кривую Безье с указанными опорными точками:

Мы вынесли координату y для второй опорной точки на 1 ниже нуля, а для третьей опорной точки – на 1 выше единицы, поэтому и кривая вышла за границы «обычного» квадрата. Её значения по y вышли из стандартного диапазона 0..1.

Как мы помним, значению y = 0 соответствует «нулевое» положение анимации, а y = 1 – конечное. Получается, что значения y<0 двинули поезд назад, меньше исходного left, а значения y>1 – больше итогового left.

Это, конечно, «мягкий» вариант. Если поставить значения y порядка -99, 99, то поезд будет куда более сильно выпрыгивать за диапазон.

Итак, кривая Безье позволяет задавать «плавное»" течение анимации. Подобрать кривую Безье вручную можно на сайте http://cubic-bezier.com/.

Шаги steps

Временная функция steps(количество шагов[, start/end]) позволяет разбить анимацию на чёткое количество шагов.

Проще всего это увидеть на примере. Выше мы видели плавную анимацию цифр от 0 до 9 при помощи смены margin-left у элемента, содержащего 0123456789.

Чтобы цифры сдвигались не плавно, а шли чётко и раздельно, одна за другой – мы разобьём анимацию на 9 шагов:

#stripe.animate {

margin-left: -174px;

transition: margin-left 9s steps(9, start);

}

В действии step(9, start):Результат

style.css

index.html 

✍

Первый аргумент steps – количество шагов, то есть изменение margin-left разделить на 9 частей, получается примерно по 19px. На то же количество частей делится и временной интервал, то есть по 1s.

start – означает, что при начале анимации нужно сразу применить первое изменение. Это проявляется тем, что при нажатии на цифру она меняется на 1 (первое изменение margin-left) мгновенно, а затем в начале каждой следующей секунды.

То есть, процесс развивается так:

●

0s – -19px (первое изменение в начале 1-й секунды, сразу при нажатии)

●

1s – -38px

●

…

●

8s – -174px

●

(на протяжении последней секунды видно окончательное значение).

Альтернативное значение end означало бы, что изменения нужно применять не в начале, а в конце каждой секунды, то есть так:

●

0s – 0

●

1s – -19px (первое изменение в конце 1-й секунды)

●

2s – -38px

●

…

●

9s – -174px

В действии step(9, end):Результат

style.css

index.html 

✍

Также есть сокращённые значения:

●

step-start – то же, что steps(1, start), то есть завершить анимацию в 1 шаг сразу.

●

step-end – то же, что steps(1, end), то есть завершить анимацию в 1 шаг по истечении transition-duration.

Такие значения востребованы редко, так как это даже и не анимация почти, но тоже бывают полезны.

Событие transitionend

На конец CSS-анимации можно повесить обработчик на событие transitionend.

Это широко используется, чтобы после анимации сделать какое-то действие или объединить несколько анимаций в одну.

Например, лодочка в примере ниже при клике начинает плавать туда-обратно, с каждым разом уплывая всё дальше вправо: ✍

Её анимация осуществляется функцией go, которая перезапускается по окончании, с переворотом через CSS:

boat.onclick = function() {

//...

var times = 1;

function go() {

if (times % 2) {

// плывём вправо

boat.classList.remove('back');

boat.style.marginLeft = 100 \* times + 200 + 'px';

} else {

// плывём влево

boat.classList.add('back');

boat.style.marginLeft = 100 \* times - 200 + 'px';

}

}

go();

boat.addEventListener('transitionend', function() {

times++;

go();

});

};

Объект события transitionend содержит специфические свойства:

propertyName

Свойство, анимация которого завершилась.

elapsedTime

Время (в секундах), которое заняла анимация, без учета transition-delay.

Свойство propertyName может быть полезно при одновременной анимации нескольких свойств. Каждое свойство даст своё событие, и можно решить, что с ним делать дальше.

CSS animations

Более сложные анимации делаются объединением простых при помощи CSS-правила @keyframes.

В нём задаётся «имя» анимации и правила: что, откуда и куда анимировать. Затем при помощи свойства animation: имя параметры эта анимация подключается к элементу, задаётся время анимации и дополнительные параметры, как её применять.

Пример с пояснениями в комментарии: ✍

<div class="progress"></div>

<style>

/\* Современные браузеры, кроме Chrome, Opera, Safari \*/

@keyframes go-left-right { /\* назовём анимацию: "go-left-right" \*/

from {

left: 0px; /\* от: left: 0px \*/

}

to {

left: calc(100% - 50px); /\* до: left: 100%-50px \*/

}

}

/\* Префикс для Chrome, Opera, Safari \*/

@-webkit-keyframes go-left-right {

from {

left: 0px;

}

to {

left: calc(100% - 50px);

}

}

.progress {

/\* применить анимацию go-left-right \*/

/\* продолжительность 3s \*/

/\* количество раз: бесконечное (infinite) \*/

/\* менять направление анимации каждый раз (alternate) \*/

animation: go-left-right 3s infinite alternate;

-webkit-animation: go-left-right 3s infinite alternate;

position: relative;

border: 2px solid green;

width: 50px;

height: 20px;

background: lime;

}

</style>

Этот стандарт пока в черновике, поэтому в Chrome, Safari, Opera нужен префикс -webkit.

Статей про CSS animations достаточно много, посмотрите, например:

●

Статья про CSS Animation.

●

Пример бесконечной подпрыгивающей анимации на CSS Animation и кривых Безье.

Итого

CSS-анимации позволяют плавно или не очень менять одно или несколько свойств.

Альтернатива им – плавное изменение значений свойств через JavaScript, мы рассмотрим подробности далее.

Ограничения и достоинства CSS-анимаций по сравнению с JavaScript:

Недостатки

●

Временная функция может быть задана кривой Безье или через шаги. Более сложные анимации, состоящие из нескольких кривых, реализуются их комбинацией при помощи CSS animations, но JavaScript-функции всегда гибче.

●

CSS-анимации касаются только свойств, а в JavaScript можно делать всё, что угодно, удалять элементы, создавать новые.

●

Отсутствует поддержка в IE9-

Достоинства

●

Простые вещи делаются просто.

●

«Легче» для процессора, чем анимации JavaScript, лучше используется графический ускоритель. Это очень важно для мобильных устройств.

Подавляющее большинство анимаций делается через CSS.