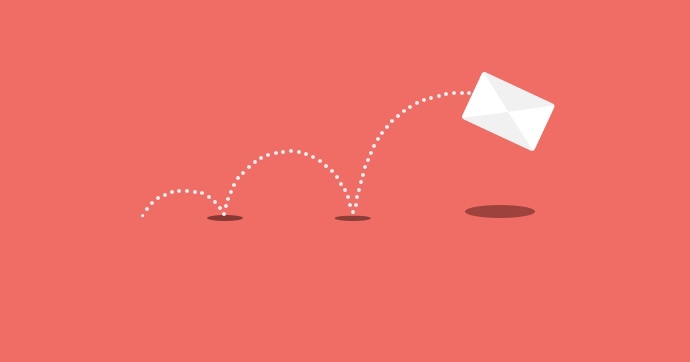
**Конспект лекций «Трансформация и анимация в CSS»**



Еще относительно недавно, чтобы реализовать любую анимацию на веб-странице, приходилось обращаться за помощью к языку JavaScript. Также для создания анимаций использовалась программа Adobe Flash. Но данные технологии весьма сложны и требуют отдельного изучения, поэтому в CSS3 появилась возможность создавать анимированные элементы без JS и сторонних программ.

Ранее мы видели, что CSS позволяет нам применять цвета, устанавливать шрифты, изменять настройки текста, позиционировать элементы, задавать для них пространство, украшать их и перемещать.

Преобразования в CSS представляют собой набор *функций*, которые позволяют определённым образом придавать элементу форму:

translate: перемещает элемент вдоль трёх осей (х, у и z);

rotate: вращает элемент вокруг центральной точки;

scale: изменяет размер элемента;

skew: искажает элемент.

Свойства трансформации

Есть три доступных свойства для трансформации:

transform определяет, какая *функция* будет применяться (translate, rotate, scale и др.);

transform-origin позволяет изменять точку начала преобразования (работает как background-position);

transform-style для настройки 3D.

Обратите внимание, что в отличие от background и border, свойство transform не является сокращённым.

Здесь мы будем использовать только transform.

Не прерывает поток

Чтобы предотвратить неожиданное поведение, трансформируемые элементы не влияют на поток. Независимо от вращения, масштабирования или перемещения, они не будут влиять на другие элементы.

translate

Функция translate() позволяет перемещать элемент в плоскости (по осям х и у). Она принимает либо:

один параметр: перемещает элемент вдоль оси х;

два параметра: первое значение для оси х, второе для оси у.

Это похоже на использование относительного позиционирования с помощью значений left и top

*CSS:*

p {

animation: translating 4s linear 0s infinite;

background: turquoise;

color: white;

height: 60px;

line-height: 60px;

margin: 0;

position: absolute;

text-align: center;

width: 60px;

}

@keyframes translating {

0% { transform: translate(0, 0);}

25% { transform: translate(240px, 0);}

50% { transform: translate(240px, 140px);}

75% { transform: translate(0, 140px);}

100%{ transform: translate(0, 0);}

}

*HTML:*

<p>Привет</p>

*Помните: transform — это стилевое свойство, translate() — это значение (и одновременно функция), применяемое к этому свойству.*

*Вы можете использовать translateX() и translateY(), чтобы переместить элемент вдоль только оси х и у, соответственно.*

rotate

Функция rotate() позволяет вращать элемент вокруг неподвижной точки. По умолчанию, вращение происходит вокруг центра элемента. Думайте об этом как о грампластинке, которая играет на проигрывателе.

rotate() принимает только один параметр, который является значением угла и определяется в градусах (deg), градах (grad), радианах (rad) или в оборотах (turn) (один оборот эквивалентен полному кругу).

*CSS:*

p {

animation: rotating 4s linear 0s infinite;

background: slateblue;

border-radius: 60px;

color: white;

height: 120px;

line-height: 120px;

margin: 0;

text-align: center;

width: 120px;

}

@keyframes rotating {

0% {

transform: rotate(0deg);

}

100% {

transform: rotate(360deg);

}

}

*HTML:*

<p>Винил</p>

scale

Функция scale() позволяет изменять размер элемента. Она может увеличить или уменьшить элемент. Функция принимает либо:

один параметр: изменение размеров элемента одинаково по высоте и ширине;

два параметра: первое значение изменяет размер элемента по горизонтали, второе по вертикали.

Диапазон возможных значений:

1: элемент сохраняет свой первоначальный размер;

2: элемент удваивается в размере;

0.5: элемент уменьшается наполовину;

0: элемент в основном исчезает (так как его высота и ширина равны нулю);

-1: элемент отражается.

*CSS:*

p {

animation: scaling-toggle 10s steps(1) 0s infinite forwards;

background: white;

color: grey;

left: 2rem;

margin: 0;

opacity: 0;

padding: 0 0.5em;

position: absolute;

top: 2rem;

z-index: 1;

}

p:first-child {

left: 0;

position: relative;

top: 0;

}

p:nth-child(2) {

animation-delay: 2s;

}

p:nth-child(3) {

animation-delay: 4s;

}

p:nth-child(4) {

animation-delay: 6s;

}

p:nth-child(5) {

animation-delay: 8s;

}

p strong {

content: "";

display: inline-block;

}

div {

animation: scaling 10s linear 0s infinite;

background: goldenrod;

border-radius: 60px;

color: white;

height: 120px;

line-height: 120px;

margin-top: 2rem;

;

text-align: center;

transition: all 100ms linear;

width: 120px;

}

@keyframes scaling {

0% {

transform: scale(1);

}

19% {

transform: scale(1);

}

20% {

transform: scale(2);

}

39% {

transform: scale(2);

}

40% {

transform: scale(0.5);

}

59% {

transform: scale(0.5);

}

60% {

transform: scale(0);

}

79% {

transform: scale(0);

}

80% {

transform: scale(-1);

}

99% {

transform: scale(-1);

}

100% {

transform: scale(1);

}

}

@keyframes scaling-toggle {

0% {

opacity: 1;

}

20% {

opacity: 0;

}

100% {

opacity: 0;

}

}

*HTML:*

<p><strong>scale(1)</strong>: обычный размер</p>

<p><strong>scale(2)</strong>: двойной размер</p>

<p><strong>scale(0.5)</strong>: половина размера</p>

<p><strong>scale(0)</strong>: нулевой размер (невидим)</p>

<p><strong>scale(-1)</strong>: отражение</p>

<div>Масштаб</div>

Как и для translate(), у функция scale() есть версии для х и у: scaleX() и scaleY(), для изменения размера по горизонтали и вертикали, соответственно.

skew

Функция skew() позволяет искажать элемент, сдвигая его стороны вдоль линий.

Эта функция преобразования используется редко, поскольку её последствия весьма непредсказуемы, а результат не обязательно привлекателен. Тем не менее, давайте посмотрим, как это работает.

Как и scale(), функция skew() принимает либо:

один параметр: элемент искажается по горизонтали;

два параметра: первое значение искажает элемент по горизонтали, второе по вертикали.

Как и rotate(), функция skew() принимает только значения угла, такого как градусы (deg).

*CSS:*

@keyframes skewing {

0% { transform: skew(0deg); }

20% { transform: skew(10deg); }

40% { transform: skew(45deg); }

60% { transform: skew(90deg); }

80% { transform: skew(120deg); }

100% { transform: skew(0deg); }

}

*HTML:*

p { animation: skewing 10s steps(1) 0s infinite;

3D-функции

Мы видели, как функции трансформации работают на плоскости, вдоль осей х и у.

Например:

У translate() до двух параметров:translate(x)translate(x, y)

translateX() только для оси х

translateY() только для оси у

Но для всех этих функций также есть 3D-версии.

Например, для translate() есть версия translate3d(), которая выполняет преобразование в трёх измерениях, а это значит, что она также включает в себя ось z (кроме того существует отдельная функция translateZ).

Параметр z в основном заставляет элемент двигаться ближе и дальше, в зависимости от уменьшения или увеличения значения. Это как увеличение и уменьшение масштаба.

*CSS:*

#zoom-animation {

perspective: 500px;

}

#zoom-animation div,

#zoom-animation p {

background: midnightblue;

color: white;

height: 200px;

line-height: 200px;

margin: 0;

position: relative;

text-align: center;

width: 200px;

}

#zoom-animation p {

animation: zooming 5s alternate infinite both;

background: limegreen;

left: 2rem;

opacity: 0.7;

position: absolute;

top: 2rem;

}

@keyframes zooming {

0% {

transform: translate3d(0, 0, 0);

}

100% {

transform: translate3d(0, 0, 200px);

}

}

*HTML:*

<div id="zoom-animation">

<div>Оригинал</div>

<p>Преобразование</p>

</div>

Зелёный блок поднимается на 200px «*вверх*» по оси z, как будто становясь ближе к нам.

*К родительскому элементу требуется применить perspective: 500px, чтобы трёхмерное пространство стало заметным. В качестве альтернативы также может быть использовано transform: perspective(500px).*

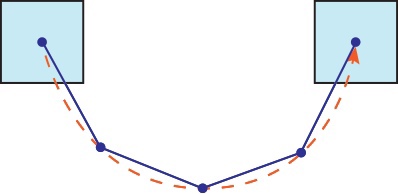
Анимация

Анимация бывает двух видов — покадровая анимация и анимация по ключевым кадрам. При покадровой анимации важным показателем является частота кадров в секунду. Для ориентира — в кинотеатрах используется 24 кадра в секунду, на телевидении — 25, современные камеры умеют снимать с частотой 50 и более кадров в секунду. Соответственно, чтобы сделать рисованный ролик продолжительностью полминуты и с частотой 24 кадра, потребуется 720 изображений. Некоторые мультипликаторы идут на хитрость и чтобы не рисовать огромное количество картинок сокращают их число в два раза, при этом повторяя каждый кадр дважды.

Анимация по ключевым кадрам использует совершенно иной подход. Например, нам нужно сделать перемещение квадрата из одной точки в другую. Достаточно задать исходные, конечные координаты квадрата и время перемещения. Расчёт промежуточных кадров берёт на себя программа, в нашем случае браузер, раз мы говорим о веб-технологиях.



Понятно, что это сильно упрощает процесс создания анимации, поскольку число кадров существенно сокращается, а всякая рутина вычисляется автоматически. Вместе с тем есть и ряд ограничений. К примеру, если требуется сделать движение объекта по дуге. Продвинутые программы вроде Adobe Flash позволяют перемещать объекты вдоль произвольной кривой, но браузеры пока сильно уступают им в этом плане. Движение может происходить только линейно, поэтому кривую мы заменяем приближённой ломаной. В математике этот метод называется аппроксимацией. Чем больше точек мы поставим, тем ближе ломаная будет к исходной кривой и тем плавнее движение, но с другой стороны это усложняет работу, фактически приближая её к покадровой анимации. Так что важно соблюдать баланс между качеством и затратами труда.



Итак, переходы в CSS являются *специфическим* видом анимации, где:

есть только два состояния: начало и конец;

анимация не зациклена;

промежуточные состояния управляются только функцией времени.

Но что если вы хотите:

иметь контроль над промежуточными состояниями?

зациклить анимацию?

сделать разные виды анимаций для *одного* элемента?

анимировать определённое свойство только на половину пути?

имитировать различные функции времени для разных свойств?

Анимация в CSS позволяет всё это, и не только.

Анимация как мини-фильм, где вы в качестве режиссёра даёте инструкции (стилевые правила) вашим актёрам (элементам HTML) для разных сцен (ключевые кадры).

Свойства анимации

Как и transition, свойство animation является сокращённым для нескольких других:

animation-name: название анимации;

animation-duration: как долго длится анимация;

animation-timing-function: как вычисляются промежуточные состояния;

animation-delay: анимация начинается спустя некоторое время;

animation-iteration-count: сколько раз должна выполняться анимация;

animation-direction: должно движение идти в обратную сторону или нет;

animation-fill-mode: какие стили применяются до начала анимации и после её завершения.

Быстрый пример

Для оживления кнопки загрузки, вы можете написать анимацию подпрыгивания:

*CSS:*

@keyframes bouncing{

0% { bottom: 0; box-shadow: 0 0 5px rgba(0,0,0,0.5);}

100%{ bottom: 50px; box-shadow: 0 50px 50px rgba(0,0,0,0.1);}

}

a {

animation: bouncing 0.5s cubic-bezier(.1,.25,.1,1) 0s infinite alternate both;

background: lightslategrey;

border-radius: 2px;

display: inline-block;

color: white;

cursor: pointer;

font-size: 0.75rem;

font-weight: 300;

letter-spacing: 0.2em;

padding: 1em 2em 1.1em;

position: relative;

text-decoration: none;

text-transform: uppercase;

vertical-align: top;

margin-top: 50px;

}

*HTML:*

<a>Загрузка</a>

Сначала нужно написать реальную анимацию подпрыгивания с помощью @keyframes и назвать её bouncing. *Затем* вы можете использовать эту анимацию, применяя её к loading-button.

Я использовал сокращенное свойство animation и включил все возможные варианты:

animation-name: bouncing (совпадает с названием ключевых кадров)

animation-duration: 0.5s (полсекунды)

animation-timing-function: [cubic-bezier(0.1,0.25,0.1,1)](https://vk.com/away.php?to=http%3A%2F%2Fcubic-bezier.com%2F%23.1%2C.25%2C.1%2C1&cc_key=)

animation-delay: 0s (без задержки)

animation-iteration-count: infinite (воспроизводится бесконечно)

animation-direction: alternate (идёт назад и вперёд)

animation-fill-mode: both

@keyframes

Перед применением анимации к элементам HTML, вам требуется написать анимацию с помощью ключевых кадров. Вообще, ключевые кадры — это каждый промежуточный шаг в анимации. Они определяются с помощью процентов:

0% — первый шаг анимации;

50% — шаг на полпути в анимации;

100% — последний шаг.

Можно также использовать ключевые слова from и to вместо 0% и 100%, соответственно.

Вы можете определить столько ключевых кадров, сколько хотите, вроде 33%, 4% или даже 29.86%. На практике вы будете писать только некоторые из них.

Каждый ключевой кадр является правилом CSS, это означает, что вы можете писать свойства CSS как обычно.

Чтобы определить анимацию, просто напишите ключевое слово @keyframes с его названием:

*CSS:*

@keyframes around {

0% {

left: 0;

top: 0;

}

25% {

left: 240px;

top: 0;

}

50% {

left: 240px;

top: 140px;

}

75% {

left: 0;

top: 140px;

}

100% {

left: 0;

top: 0;

}

}

p {

animation: around 4s linear 0s infinite;

background: turquoise;

color: white;

height: 60px;

line-height: 60px;

margin: 0;

position: absolute;

text-align: center;

width: 60px;

}

*HTML:*

<p>

Привет

</p>

Обратите внимание, что начало 0% и конец 100% содержат одинаковые правила CSS. Это гарантирует, что анимация зациклится идеально. Поскольку счётчик итераций установлен как infinite, то анимация будет идти от 0% до 100%, а затем немедленно *обратно* к 0% и так бесконечно.

animation-name

Название анимации используется, по крайней мере, дважды:

при написании анимации с помощью @keframes;

при использовании анимации с помощью свойства animation-name (или через сокращённое свойство animation).

@keyframes whatever { /\* ... \*/ } .selector { animation-name: whatever; }

Подобно именам классов CSS, название анимации может включать в себя только:

буквы (a-z);

цифры (0-9);

подчёркивание (\_);

дефис (-).

Название не может начинаться с цифры или с двух дефисов.

animation-duration

Как и [длительность перехода](https://vk.com/away.php?to=https%3A%2F%2Fwebref.ru%2Fcourse%2Fcss-advanced%2Ftransitions&cc_key=), длительность анимации может быть установлена в секундах (1s) или миллисекундах (200ms).

.selector { animation-duration: 0.5s; }

Значение по умолчанию равно 0s, что означает отсутствие анимации вообще.

animation-timing-function

Подобно [функциям времени для переходов](https://vk.com/away.php?to=https%3A%2F%2Fwebref.ru%2Fcourse%2Fcss-advanced%2Ftransitions&cc_key=), функции времени для анимации могут использовать ключевые слова, такие как linear, ease-out или могут быть определены с помощью произвольных кривых Безье.

.selector { animation-timing-function: ease-in-out; }

Значение по умолчанию: ease.

Поскольку анимация в CSS использует ключевые кадры, вы можете установить линейную функцию времени и моделировать конкретную кривую Безье, определяя множество очень специфичных ключевых кадров. Посмотрите [Bounce.js](https://vk.com/away.php?to=http%3A%2F%2Fbouncejs.com%2F&cc_key=) для создания передовой анимации.

animation-delay

Как и с [задержкой перехода](https://vk.com/away.php?to=https%3A%2F%2Fwebref.ru%2Fcourse%2Fcss-advanced%2Ftransitions&cc_key=), задержка анимации может быть установлена в секундах (1s) или миллисекундах (200ms).

По умолчанию равно 0s, что означает отсутствие любой задержки.

Полезно использовать, когда включается несколько анимаций в серии.

.a, .b, .c { animation: bouncing 1s; } .b { animation-delay: 0.25s; } .c { animation-delay: 0.5s; }

animation-iteration-count

По умолчанию, анимация воспроизводится только один раз (значение 1). Вы можете установить три типа значений:

целые числа, вроде 2 или 3;

дробные числа, вроде 0.5, которые будут воспроизводить только половину анимации;

ключевое слово infinite, которое будет повторять анимацию бесконечно.

.selector { animation-iteration-count: infinite; }

animation-direction

Свойство animation-direction определяет, *в каком порядке* читаются ключевые кадры.

normal: начинается с 0%, заканчивается на 100%, начинается с 0% снова.

reverse: начинается со 100%, заканчивается на 0%, начинается со 100% снова.

alternate: начинается с 0%, идёт до 100%, возвращается на 0%.

alternate-reverse: начинается со 100%, идёт до 0%, возвращается на 100%.

Это легче представить, если счётчик итераций анимации установлен как infinite.

*CSS:*

p{ color: grey;}

.normal div{ animation-direction: normal;}

.reverse div{ animation-direction: reverse;}

.alternate div{ animation-direction: alternate;}

.alternate-reverse div{ animation-direction: alternate-reverse;}

p:nth-child(1) strong{ color: crimson;}

div:nth-child(2) div{ background: crimson;}

p:nth-child(3) strong{ color: midnightblue;}

div:nth-child(4) div{ background: midnightblue;}

p:nth-child(5) strong{ color: mediumseagreen;}

div:nth-child(6) div{ background: mediumseagreen;}

p:nth-child(7) strong{ color: goldenrod;}

div:nth-child(8) div{ background: goldenrod;}

div {

background: white;

height: 20px;

width: 220px;

}

div div {

animation: swipe 2s linear infinite;

background: crimson;

height: 20px;

left: 0;

margin-top: -1rem;

position: relative;

transition: 1s;

width: 20px;

}

@keyframes swipe {

0% { left: 0;}

100%{ left: 200px;}

}

*HTML:*

<p><strong>normal</strong>: 0% --> 100%</p>

<div class="normal">

<div></div>

</div>

<p><strong>reverse</strong>: 100% --> 0%</p>

<div class="reverse">

<div></div>

</div>

<p><strong>alternate</strong>: 0% <--> 100%</p>

<div class="alternate">

<div></div>

</div>

<p><strong>alternate-reverse</strong>: 100% <--> 0%</p>

<div class="alternate-reverse">

<div></div>

</div>

animation-fill-mode

Свойство animation-fill-mode определяет, что происходит *перед* началом анимации и *после* её завершения.

При определении ключевых кадров можно указать правила CSS, которые будут применяться на разных шагах анимации. Теперь эти правила CSS могут *столкнуться* с теми, которые *уже применяются* к анимируемым элементам.

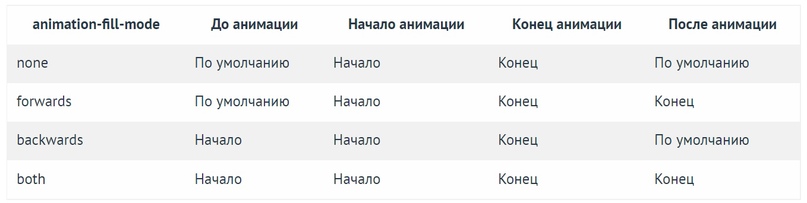
animation-fill-mode позволяет сообщить браузеру, если *стили анимации* также должны применяться за пределами анимации.

Давайте представим себе кнопку, которая является:

красной по умолчанию;

становится синей в начале анимации;

и в итоге зелёной, когда анимация завершена.



*CSS:*

p {

color: grey;

margin: 0;

position: relative;

}

p:first-child {

margin-bottom: 1rem;

}

p span {

animation: toggle 6s infinite both;

left: 0;

position: absolute;

}

p:first-child span {

animation-name: tabs;

margin-right: 1rem;

position: static;

}

p span:nth-child(1) {

animation-delay: 0;

position: static;

}

p span:nth-child(2) {

animation-delay: 2s;

}

p span:nth-child(3) {

animation-delay: 4s;

}

div {

background: white;

height: 20px;

margin-bottom: 1rem;

width: 220px;

}

div div {

animation: race-none 6s linear infinite;

background: crimson;

height: 20px;

margin-bottom: 0;

position: relative;

width: 20px;

}

div:nth-child(7) div {

animation-name: race-forwards;

}

div:nth-child(10) div {

animation-name: race-backwards;

}

div:nth-child(13) div {

animation-name: race-both;

}

@keyframes tabs {

0% {

opacity: 0.2;

}

1% {

opacity: 1;

}

33% {

opacity: 1;

}

34% {

opacity: 0.2;

}

100% {

opacity: 0.2;

}

}

@keyframes toggle {

0% {

opacity: 0;

}

1% {

opacity: 1;

}

33% {

opacity: 1;

}

34% {

opacity: 0;

}

100% {

opacity: 0;

}

}

@keyframes race-none {

0% {

background: crimson;

left: 0;

}

33% {

background: crimson;

left: 0;

}

34% {

background: midnightblue;

left: 100px;

}

66% {

background: mediumseagreen;

left: 200px

}

67% {

background: crimson;

left: 0;

}

100% {

background: crimson;

left: 0;

}

}

@keyframes race-forwards {

0% {

background: crimson;

left: 0;

}

33% {

background: crimson;

left: 0;

}

34% {

background: midnightblue;

left: 100px;

}

66% {

background: mediumseagreen;

left: 200px

}

67% {

background: mediumseagreen;

left: 200px;

}

100% {

background: mediumseagreen;

left: 200px;

}

}

@keyframes race-backwards {

0% {

background: midnightblue;

left: 100px;

}

33% {

background: midnightblue;

left: 100px;

}

34% {

background: midnightblue;

left: 100px;

}

66% {

background: mediumseagreen;

left: 200px

}

67% {

background: crimson;

left: 0;

}

100% {

background: crimson;

left: 0;

}

}

@keyframes race-both {

0% {

background: midnightblue;

left: 100px;

}

33% {

background: midnightblue;

left: 100px;

}

34% {

background: midnightblue;

left: 100px;

}

66% {

background: mediumseagreen;

left: 200px

}

67% {

background: mediumseagreen;

left: 200px;

}

100% {

background: mediumseagreen;

left: 200px;

}

}

*HTML:*

<p>

<span><strong>1.</strong> До анимации</span>

<span><strong>2.</strong> В процессе анимации</span>

<span><strong>3.</strong> После анимации</span>

</p>

<p><strong>none</strong>: стили анимации не влияют на стиль по умолчанию.</p>

<p>

<span>Красный по умолчанию</span>

<span>От синего к зелёному</span>

<span>Обратно к красному</span>

</p>

<div class="none">

<div></div>

</div>

<p><strong>forwards</strong>: последние стили, применяемые в конце анимации, сохраняются впоследствии.</p>

<p>

<span>Красный по умолчанию</span>

<span>От синего к зелёному</span>

<span>Остаётся зелёным</span>

</p>

<div class="forwards">

<div></div>

</div>

<p><strong>backwards</strong>: стили анимации будут применяться до того, как на самом деле начинается анимация.</p>

<p>

<span>Уже синий</span>

<span>От синего к зелёному</span>

<span>Обратно к красному</span>

</p>

<div class="backwards">

<div></div>

</div>

<p><strong>Both</strong>: стили применяются до и после воспроизведения анимации.</p>

<p>

<span>Уже синий</span>

<span>От синего к зелёному</span>

<span>Остаётся зелёным</span>

</p>

<div class="both">

<div></div>

</div>