Виды анимации на верстке: свойства, особенности, примеры кода**10 мин12K**CSS\*JavaScript\*

Виды анимаций

Самые простые и распространённые виды анимаций — CSS Transition и CSS Animation.CSS Transition — это сокращённое CSS-свойство, которое позволяет управлять плавным переходом между двумя состояниями элемента.

Свойство Transition состоит из:

transition-property — название свойства, к которому применяется эффект перехода.

transition-duration — длительность перехода.

transition-timing-function — временную функцию перехода (то, как будут определены промежуточные значения для свойства, которое участвует в переходе).

transition-delay — задержку перед началом перехода.

**button**:hover {  
 **transform**: scale(1.3);  
 **transition**: transform 300ms cubic-bezier(0.07, 0.76, 0.93, 0.59) 0.3s;  
}**Плюсы:** Это наилучший способ (с точки зрения простоты и быстродействия), когда есть всего два состояния элемента в зависимости от одного условия.

Например, при наведении мышкой, при нажатии на элемент, и т. п.

**Минусы:** transition не позволяет изменять свойства элемента, если состояний больше, чем два. Также невозможно делать бесконечные анимации.

**Особенности:** в значении свойства transition можно указывать как отдельные CSS-свойства со своими значениями, так и использовать ключевое слово all. Оно скажет браузеру использовать переход для всех свойств, которые есть внутри CSS-правила.

CSS Animation

Animation — сокращённое CSS-свойство, включающее в себя:

animation-name — название.

animation-duration — длительность.

animation-timing-function — временную функцию (по аналогии с transition-timing-function).

animation-delay — задержку перед началом.

animation-iteration-count — количество повторений.

animation-direction — направление (можно запустить анимацию наоборот, начиная с конца).

animation-fill-mode — то, как будут применяться стили ключевых кадров до и после окончания анимации.

animation-play-state — свойство, которое позволяет поставить анимацию на паузу и продолжить с того места, на котором она остановилась.

Чтобы создать анимацию, нужно описать директиву [@keyframes](https://habr.com/users/keyframes) с названием анимации и описанием ключевых кадров:

**@keyframes** shift {  
 10% { **transform**: translate3d(0, 0, 0); }  
 20% { **transform**: translate3d(-20px, 0, 0); }  
 50% { **transform**: translate3d(10px, 0, 0);}  
 75% { **transform**: translate3d(-7px, 0, 0); }  
 100% { **transform**: translate3d(0, 0, 0); }  
}Для того, чтобы использовать эту анимацию, достаточно добавить к элементу свойство animation с названием анимации и её опциями:

element.is-animated { **animation**: shift 1700ms linear 1000ms infinite;

В следующем простом примере при клике на элемент в JS добавляется и удаляется класс .is-animated, что включает и выключает анимацию

**Плюсы:** свойство animation позволяет делать сложные, повторяющиеся анимации и более точно управлять состоянием анимации через подробное описание ключевых кадров.

**Минусы:** если есть какие-то сложные сцены, которые используют много анимаций, включающихся подряд, то создание анимации с помощью animation становится сложным. Также сложно будет это поддерживать.

**Особенности:** чтобы применить анимацию к элементу, достаточно указать всего два значения: имя и длительность анимации. При этом порядок значений практически не важен. Важно только то, что первое значение времени — это длительность, а второе — задержка перед началом анимации.Описывая следующие виды анимаций, я не буду вдаваться в подробности их реализации. Каждый из них — отдельный, огромный мир, который заслуживает нескольких больших статей. Расскажу про основные моменты, которые их отличают, где и для чего они используются, и приведу несколько примеров, чтоб вы могли наглядно увидеть как они работают, в чём их плюсы и минусы.SVG animationSVG можно анимировать двумя способами: с помощью CSS-анимации SMIL-анимации.Анимация SVG с помощью CSS ничем не отличается от обычной animation. Мы также с помощью [@keyframes](https://habr.com/users/keyframes) создаём анимацию, описываем в ней свойства, которые изменяем, и используем её при необходимости.Есть три «но»:если в SVG что-то не получилось анимировать с помощью CSS-анимации,или нужно устанавливать зависимости между разными анимациями (например, запустить вторую анимацию через 3 секунды после того, как закончится первая анимация),либо нужно плавно превратить один вид SVG в другой, во всех этих (и некоторых других) случаях необходимо использовать SMIL-анимацию для SVG.Так выглядит простая SMIL-анимация движения круга по клику мышки:<svg **width**="400" **height**="80">  
 <circle id="circle" r="30" cx="40" cy="40" fill="blue"/>  
 <animate  
 xlink:href="#circle"  
 attributeName="cx"  
 from="50"  
 to="370"  
 dur="700ms"  
 begin="click"  
 fill="freeze" />  
</svg>Примеры:1) Анимация SMIL + CSS. На CSS анимируется изменение цвета и перемещение фигуры, с помощью SMIL фигура изменяет свой контур.2) Здесь анимация полностью сделана с помощью SMIL3) Ещё один пример морфинга (изменение контура SVG)**Плюсы:**SMIL анимация будет работать даже если вам нужно вставить SVG с помощью тега img, либо через background-image в CSS. С помощью SMIL анимации можно анимировать те атрибуты, которые не доступны для CSS-анимации. Например, анимировать точки в теге <path>, чтоб создать эффект плавного перехода от одной фигуры к другой.**Минусы:**По сути SMIL-анимация — это отдельная спецификация со своими правилами и тонкостями, которую необходимо изучать отдельно от CSS.  
**Особенности:** У SMIL-анимации очень много тонкостей, знать которые можно только если вплотную занимаешься именно этим видом анимации.JS-АнимацияГоворя простым языком, это плавное изменение стилей элемента через JavaScript. С помощью JS анимации можно контролировать все стили элемента на любом этапе анимации, менять их в зависимости от любых условий. С помощью чистого JS делаются анимации переключения слайдов, раскрытие-закрытие аккордеона, различные перемещения блоков по определённым условиям, и многое многое другое.Примеры:1) Довольно необычный слайдер2) Анимация эффекта при движении мышкой**Плюсы:** Анимация с помощью JavaScript позволяет сделать то, на что не способна CSS-анимация. Можно как угодно манипулировать элементами. Также в любом месте анимации её можно замедлять, приостанавливать, и т.п.**Минусы:**JS-анимация часто более ресурсозатратна для браузеров.**Особенности:**если написать JS-анимацию неправильно, то есть большой шанс, что на слабых устройствах она будет тормозить. Как этого избежать поговорим ниже.для JS-анимаций есть большое количество готовых библиотек (например, anime.js), которые позволяют создавать анимации быстрее и эффективнее (особенно сложные), нежели писать их руками.JS canvasCanvas — это холст. То есть всего лишь один тег в html-коде. При создании анимации внутри canvas все изменения происходят внутри одного тега. Всё, что находится внутри canvas, не существует в DOM-дереве. Но с помощью JS мы можем создавать, перемещать и изменять любые элементы, которые будут находиться в canvas.**Плюсы:**при создании анимации внутри canvas браузеру не нужно постоянно изменять и пересчитывать всю страницу, расположение элементов относительно друг от друга, и т.п.**Минусы:** как и все предыдущие виды анимаций, canvas позволяет реализовать только 2D-графику.Примеры:1) Анимация снегопада2) Игра змейка на canvasJS. WebGL / ThreeJsWebGL — технология, которая добавляет в браузер 3D-графику. Самая популярная библиотека для работы с WebGL (на данный момент) — three.js. Она существенно упрощает порог вхождения в технологию WebGL. Благодаря поддержке 3d, с помощью three.js часто делаются браузерные 3d игры.Примеры:На этом [сайте](https://freefrontend.com/three-js-games/) приведены примеры игр, созданных с помощью three.js[Земля](https://2050.earth/ru) (шар), на которой есть интерактивные точки. При нажатии на точки открываются описания.**Плюсы:** это единственная возможность создавать полноценную 3D-анимацию в браузере.**Минусы:**даже учитывая, что three.js оптимизирует анимацию, чаще всего анимация на three.js — это огромная нагрузка на браузер, и на средних и слабых компьютерах анимация будет тормозить.Как работают анимации в браузереЧто происходит в браузере, когда мы запускаем анимацию? Каким образом браузер перерисовывает все эти кадры? Какие процессы происходят кроме просто перерисовки стилей? Зная ответы на эти вопросы, можно грамотно оптимизировать анимацию.Вот 5 основных этапов, которые проходит браузер во время анимации:1) JavaScript. Браузер смотрит влияние JavaScript на стили на странице.2) Style. Из CSS для каждого элемента рассчитываются стили, которые отвечают за внешний вид (размеры, отступы, цвета).3) Layout. Браузер рассчитывает местоположение каждого элемента относительно друг друга. А также как они влияют друг на друга.На всех этих этапах браузер не отрисовывает элементы. Он всего лишь считывает их характеристики.4) Paint. На данном этапе уже прорисовываются все элементы. Каждый пиксель каждого элемента.5) Composite. Происходит проверка, какой элемент на какой наложился сверху. Например, снизу фон, а поверх него остальные элементы.Все 5 этапов проходят по умолчанию при загрузке страницы.Во время анимации в браузере могут быть три варианта:Все эти 5 этапов проходят тогда, когда мы изменяем CSS-свойства, которые рассчитываются на процессоре (CPU) и влияют на размеры элемента, его местоположение, внешние и внутренние отступы. Это худший вариант.Если мы изменяем свойства, которые не влияют на расположение элемента (фон, тени, цвет, и т. п.), то браузер проходит 4 этапа — 1, 2, 4, 5 (без третьего, одного из самых затратных, — этапа Layout).Лучшим вариантом анимаций с точки зрения работы браузера будет вариант, когда мы проходим только 1, 2, и 5 этап. (без этапов Layout и Paint). Это возможно, если использовать свойства opacity и transform. opacity — это просто прозрачность, а transform фиксирует область за элементом, и для него перестаёт просчитываться взаимодействие с другими элементами. Это позволяет не перерисовывать всю страницу.Основные правила для успешной работы анимацийСамая сложная и важная часть веб-анимации — добиться её плавности. То есть сделать так, чтоб количество кадров анимации приблизилось к 60 fps. Существует множество факторов, которые влияют на плавность анимации — это и мощность устройства, и размеры экрана, и количество пикселей на экране, и многое другое.Приведу несколько способов оптимизации анимаций.С transition лучше не использовать ключевое слово all, а задавать конкретные свойства, поскольку в CSS-правиле в дальнейшем могут добавиться другие свойства, переход между которыми нам не нужен. Также, когда используется ключевое all, браузер проходит по всем свойствам в CSS-правиле, определяя, какие можно анимировать, а какие нет. Это создаёт лишнюю нагрузку, что незначительно, но всё же влияет на быстродействие анимации.Использовать свойства, изменение которых не заставляет браузер перерисовывать всю страницу, особенно всё, что относится к размерам.Выносить анимированные элементы на отдельный слой, чтобы анимация происходила за с помощью видеокарты. (transform: translateZ(0); или transform: translate3d(0, 0, 0); )использовать свойство will-change, которое заранее говорит браузеру о тех свойствах, которые будут анимированы. При этом браузер оптимизирует анимацию, и, если нужно, то переносит рендеринг видеокарту. (Не следует использовать для простой анимации, чтоб не путать браузер на что нужно, а на что не стоит тратить особенное внимание).Если элемент визуально отсутствует на странице после или до анимации, то необходимо его скрыть с помощью display: none или visibility: hidden, для того, чтоб освободить ресурсы браузера. Это полностью выключит считывание событий и считывание анимаций.Если анимаций много, то лучше запускать их последовательно. Лучше для этого не использовать delay, потому что это равнозначно одновременному запуску нескольких анимаций, что съедает ресурсы.При начале разработки анимацию следует тестировать на разных, особенно слабых, устройствах. Если анимация тормозит, можноизменить подход или как-то оптимизировать анимацию. Если же это тестировать в конце, то возможно придётся всё переписывать, что увеличит время разработки.Если используем JS-анимацию, то обязательно нужно использовать функцию window.requestAnimationFrame(). В коде это выглядит так:function step() {  
// код функции, который анимирует элемент   
window.requestAnimationFrame(step);  
}  
window.requestAnimationFrame(step);

Эта функция вызывается браузером каждый раз, когда он приступает к пересчёту местоположения всех элементов, пересмотру влияния JS, то есть когда проходит полный рендер. В идеальных условиях это 60 раз в секунду. Но интервалы между этими вызовами рассчитываются браузером, и зависят от нагрузки на устройство, от заряда батареи и т.п. Благодаря тому, что браузер сам вызывает эту функцию, когда ему нужно, оптимизируется быстродействие анимации.Всегда просить коллег посмотреть и саму анимацию, и код, для того, чтобы можно было что-то оптимизировать. При долгой работе над анимациями замыливается глаз. Очень важен взгляд со стороны.ИтогХочу обратить особенное внимание на то, что при создании веб-анимации важно думать о том, как с этим будет работать браузер, что именно он будет делать. Например, сколько раз и за какое время он будет вызывать события для запуска анимации. И о том, насколько анимация затрагивает перерисовку блоков или даже всей страницы. За счёт каких ресурсов компьютера будет исполняться код (CPU или GPU). По возможности не стоит переусложнять анимацию. Чем проще анимация, тем она производительнее. Анимация должна быть украшением, но никак не мешать пользователям пользоваться сайтом или приложением.Инструментов для написания веб-анимации много. Очень важно использовать каждый из них именно для тех задач, для которых они подходят лучше всего.