Flexbox. Анимация

Основные преимущества flexbox

1. Все блоки очень легко делаются “резиновым”, что уже следует из названия “flex”. Элементы могут сжиматься и растягиваться по заданным правилам, занимая нужное пространство.
2. Выравнивание по вертикали и горизонтали, базовой линии текста работает шикарно.
3. Расположение элементов в html не имеет решающего значения. Его можно поменять в CSS. Это особенно важно для некоторых аспектов responsive верстки.
4. Элементы могут автоматически выстраиваться в несколько строк/столбцов, занимая все предоставленное место.
5. Множество языков в мире используют написание справа налево rtl (right-to-left), в отличии от привычного нам ltr (left-to-right). Flexbox адаптирован для этого. В нем есть понятие начала и конца, а не права и лева. Т.е. в браузерах с локалью rtl все элементы будут автоматически расположены в реверсном порядке.
6. Синтаксис CSS правил очень прост и осваивается довольно быстро.

Flexbox определяет набор CSS свойств для контейнера (flex-контейнер) и его дочерних элементов (flex-блоков). Первое, что нужно сделать – это указать контейнеру display:flex или display:inline-flex.

```
<div class="my-flex-container">
    <div class="my-flex-block">item1</div>
    <div class="my-flex-block">item2</div>
    <div class="my-flex-block">item3</div>
</div>
```

```
.my-flex-container{
        display: flex;
 }
```

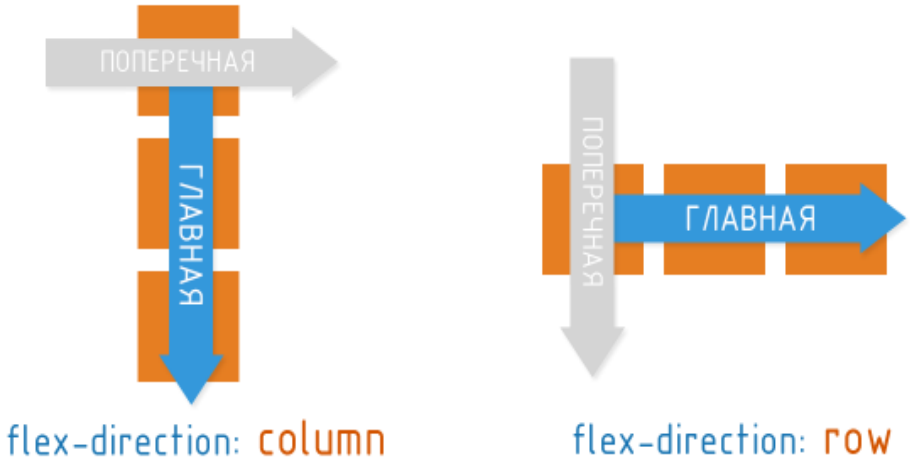
Главная и поперечная ось

Одним из основных понятий в flexbox являются оси.

Главной осью flex-контейнера является направление, в соответствии с которым располагаются все его дочерние элементы.

Поперечной осью называется направление, перпендикулярное главной оси.

flex-direction – направление главной оси



Доступные значения flex-direction:

row *(значение по умолчанию)* : слева направо (в rtl справа налево)

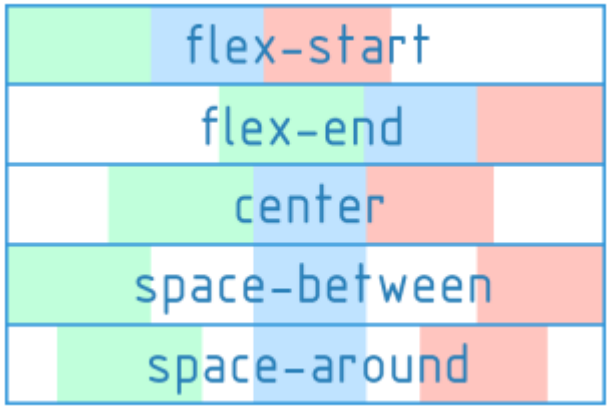
row-reverse: справа налево (в rtl слева направо)

column: сверху вниз

column-reverse: снизу вверх

(index_1)

justify-content – выравнивание по главной оси.



Доступные значения justify-content:

flex-start *(значение по умолчанию)* : блоки прижаты к началу главной оси

flex-end: блоки прижаты к концу главной оси

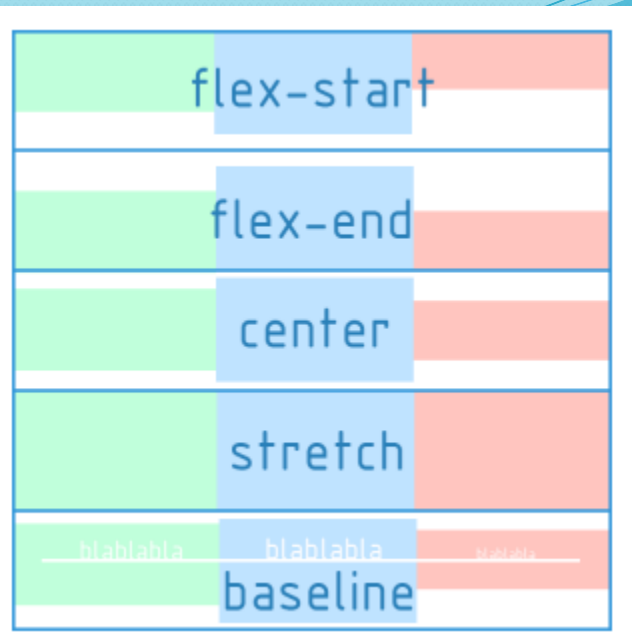
center: блоки располагаются в центре главной оси

space-between: первый блок располагается в начале главной оси, последний блок – в конце, все остальные блоки равномерно распределены в оставшемся пространстве.

space-around: все блоки равномерно распределены вдоль главной оси, разделяя все свободное пространство поровну.

(index_2)

align-items – выравнивание по поперечной оси



Доступные значения align-items:

flex-start: блоки прижаты к началу поперечной оси

flex-end: блоки прижаты к концу поперечной оси

center: блоки располагаются в центре поперечной оси

baseline: блоки выровнены по их baseline

stretch *(значение по умолчанию)* : блоки растянуты, занимая все доступное место по поперечной оси, при этом все же учитываются min-width/max-width, если таковые заданы.

(index_3)

flex-wrap

По умолчанию flex-контейнер всегда будет располагать блоки внутри себя в одну линию. Однако, спецификацией также поддерживается многострочный режим. За многосторонность внутри flex-контейнера отвечает CSS свойство flex-wrap.

Доступные значения flex-wrap:

nowrap (значение по умолчанию) : блоки расположены в одну линию слева направо (в rtl справа налево)

wrap: блоки расположены в несколько горизонтальных рядов (если не помещаются в один ряд). Они следуют друг за другом слева направо (в rtl справа налево)

wrap-reverse: тоже что и wrap, но блоки располагаются в обратном порядке.

(index_4)

flex-flow – удобное сокращение для flex-direction + flex-wrap

flex-flow предоставляет возможность в одном свойстве описать направление главной и многострочность поперечной оси. По умолчанию flex-flow: row nowrap

flex-flow: <‘flex-direction’> || <‘flex-wrap’>

align-content

Определяет то, каким образом образовавшиеся ряды блоков будут выровнены по вертикали и как они поделят между собой все пространство flex-контейнера.

Важно: align-content работает только в многострочном режиме (т.е. в случае flex-wrap:wrap; или flex-wrap:wrap-reverse;)

Доступные значения align-content:

flex-start: ряды блоков прижаты к началу flex-контейнера.

flex-end: ряды блоков прижаты к концу flex-контейнера

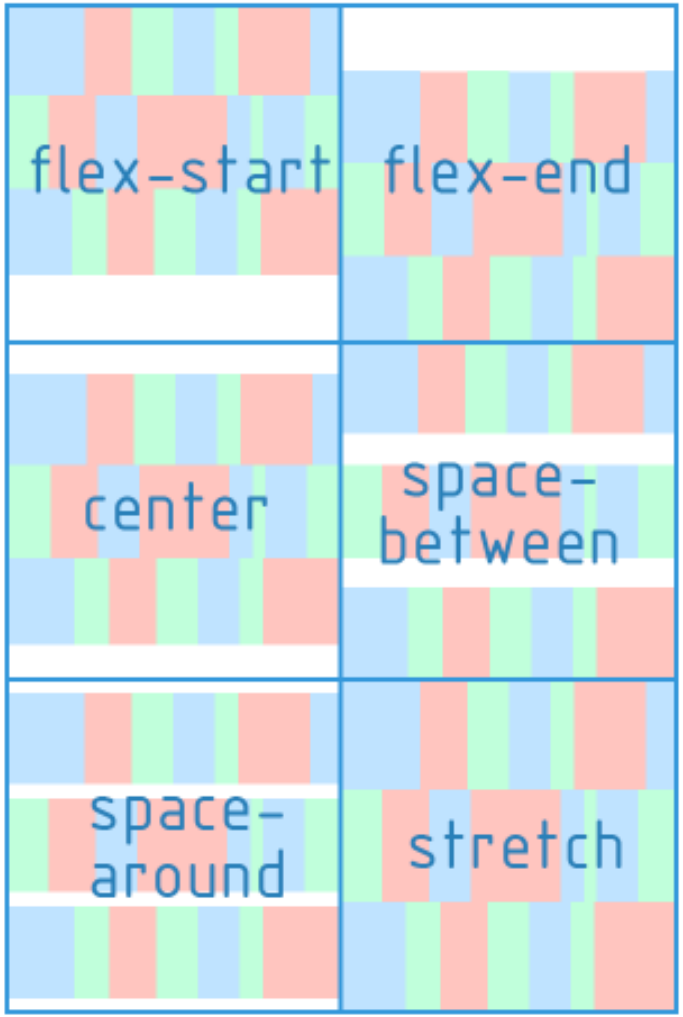
center: ряды блоков находятся в центре flex-контейнера

space-between: первый ряд блоков располагается в начале flex-контейнера, последний ряд блоков блок – в конце, все остальные ряды равномерно распределены в оставшемся пространстве.

space-around: ряды блоков равномерно распределены в от начала до конца flex-контейнера, разделяя все свободное пространство поровну.

stretch (значение по умолчанию): Ряды блоков растянуты, дабы занять все имеющееся пространство.

(index_5)



CSS правила для дочерних элементов flex-контейнера

flex-basis – базовый размер отдельно взятого flex-блока

Задает изначальный размер по главной оси для flex-блока до того, как к нему будут применены преобразования, основанные на других flex-факторах. Может быть задан в любых единицах измерения длинны (px, em, %, ...) или auto(по умолчанию). Если задан как auto – за основу берутся размеры блока (width, height), которые, в свою очередь, могут зависеть от размера контента, если не указаны явно.

flex-grow – “жадность” отдельно взятого flex-блока

Определяет то, на сколько отдельный flex-блок может быть больше соседних элементов, если это необходимо. flex-grow принимает безразмерное значение (по умолчанию 0)

Пример 1:

Если все flex-блоки внутри flex-контейнера имеют flex-grow:1, то они будут одинакового размера

Если один из них имеет flex-grow:2, то он будет в 2 раза больше, чем все остальные

Пример 2:

Если все flex-блоки внутри flex-контейнера имеют flex-grow:3, то они будут одинакового размера

Если один из них имеет flex-grow:12, то он будет в 4 раза больше, чем все остальные

Т.е абсолютное значение flex-grow не определяет точную ширину. Оно определяет его степень “жадности” по отношению к другим flex-блокам того же уровня.

flex-shrink – фактор “сжимаемости” отдельно взятого flex-блока

Определяет, насколько flex-блок будет уменьшаться относительно соседних эдементов внутри flex-контейнера в случае недостатка свободного места. По умолчанию равен 1

(index_6)

flex – короткая запись для свойств flex-grow, flex-shrink и flex-basis

flex: none | [<'flex-grow'> <'flex-shrink'>? || <'flex-basis'>]

```
т.е. ... */
.my-flex-block{
    flex-grow:12;
    flex-shrink:3;
    flex basis: 30em;
}
/* это то же самое, что ... */
.my-flex-block{
        flex: 12 3 30em;
}
```

align-self – выравнивание отдельно взятого flex-блока по поперечной оси.

Делает возможным переопределять свойство flex-контейнера align-items для отдельного flex-блока.

Доступные значения align-self (те же 5 вариантов, что и для align-items)

flex-start: flex-блок прижат к началу поперечной оси

flex-end: flex-блок прижат к концу поперечной оси

center: flex-блок располагаются в центре поперечной оси

baseline: flex-блок выравнен по baseline

stretch (значение по умолчанию) : flex-блок растянут, чтобы занять все доступное место по поперечной оси, при этом учитываются min-width/max-width, если таковые заданы.

(index_7)

order – порядок следования отдельно взятого flex-блока внутри flex-контейнера.

По умолчанию все блоки будут следовать друг за другом в порядке, заданном в html. Однако этот порядок можно изменить с помощью свойства order. Оно задается целым числом и по умолчанию равно 0.

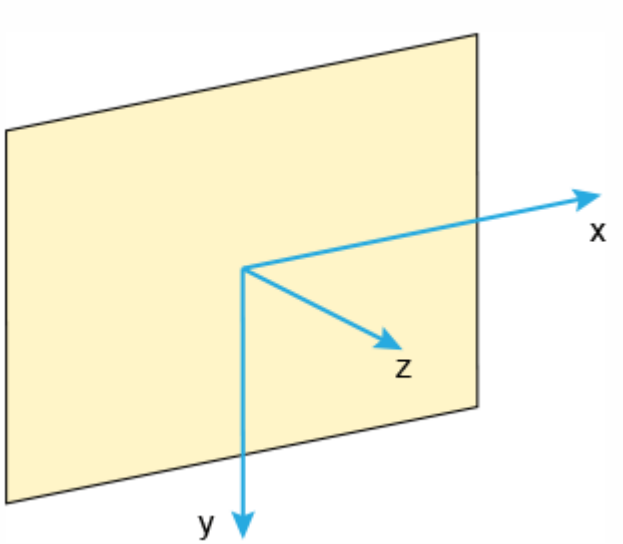
Значение order не задает абсолютную позицию элемента в последовательности. Оно определяет вес позиции элемента.

(index_8)

margin: auto по вертикали. Мечты сбываются!

Flexbox можно любить хотя бы за то, что привычное всем выравнивание по горизонтали через margin:auto здесь работает и для вертикали!

Трансформации



Любая трансформация происходит относительно центральной точки элемента, её положение можно задать с помощью свойства transform-origin. Координатные оси показаны на рисунке; оси X и Y находятся в плоскости экрана, а ось Z ему перпендикулярна.

translate()

Сдвигает элемент на заданное значение по горизонтали и вертикали в плоскости экрана.

transform: translate(tx[, ty])

Здесь:

tx — значение смещения по оси X в пикселях, процентах или других единицах CSS;

ty — смещение по оси Y. Если значение ty не указано, то оно считается равным tx: translate(2) соответствует translate(2, 2).

translate3d()

Сдвигает элемент на заданное значение в трёхмерном пространстве.

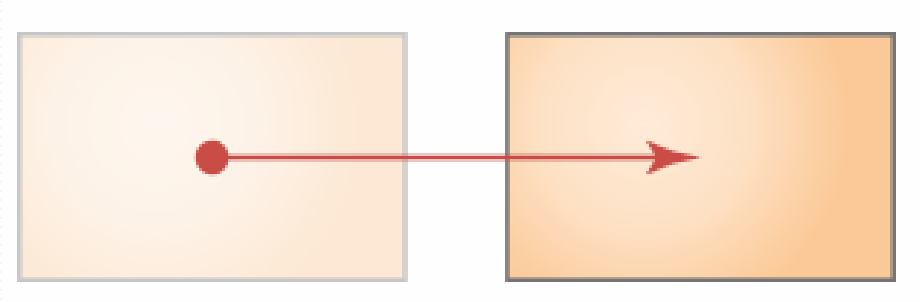
transform: translate3d(tx, ty, tz)

Здесь: tx — смещение по оси X; ty — смещение по оси Y; tz — смещение по оси Z. tz не может указываться в процентах, в этом случае функция игнорируется.

translateX()

Сдвигает элемент по горизонтали на указанное значение. Положительное значение сдвигает вправо, отрицательное влево.

transform: translateX(tx)

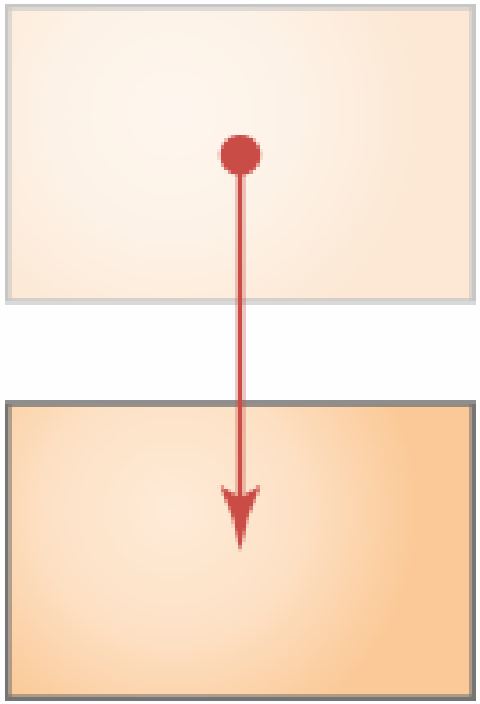


translateY()

Сдвигает элемент по вертикали на указанное значение. Положительное значение сдвигает вниз, отрицательное вверх.

transform: translateY(ty)

translateY(ty) является синонимом translate(0, ty).



translateZ()

Сдвигает элемент по оси Z в трёхмерном пространстве. Положительное значение сдвигает вперёд, отрицательное назад.

transform: translateZ(tz)

translateZ(tz) является синонимом translate3d(0, 0, tz).

scale()

Задаёт масштаб элемента по горизонтали и вертикали.

transform: scale(sx[, sy]);

Здесь:

sx — изменение масштаба по оси X;

sy — изменение масштаба по оси Y. Значение больше 1 увеличивает масштаб элемента, меньше 1, наоборот, его уменьшает. Если задано только одно значение, то масштабирование будет происходить пропорционально в обе стороны.

scale3d()

Масштабирует элемент в трёхмерном пространстве.

transform: scale3d(sx, sy, sz)

scaleX()

Масштабирует элемент по горизонтали.

transform: scaleX(sx);

Значение -1 отражает элемент по горизонтали

scaleY()

Масштабирует элемент по вертикали.

transform: scaleY(sy);

Значение -1 отражает элемент по вертикали

scaleZ()

Масштабирует элемент по оси Z.

transform: scaleZ(sz);

scaleZ(sz) является синонимом scale3d(1, 1, sz).

rotate()

Поворачивает элемент на заданный угол относительно точки трансформации, задаваемой свойством transform-origin.

transform: rotate(α)

Здесь: α — угол поворота. Положительное значение поворачивает элемент по часовой стрелке, отрицательное против.

rotate3d()

Поворачивает элемент в трёхмерном пространстве без искажений. Вращающийся элемент имеет три степени свободы — оси X, Y и Z, относительно которых происходит поворот. Они задаются с помощью вектора [x, y, z] с учётом точки вращения.

transform: rotate3d(x, y, z, α)

Здесь:

x — целое число описывающее координату X вектора оси вращения;

y — целое число описывающее Y-координату вектора оси вращения;

z — целое число описывающее координату Z вектора оси вращения;

α — угол поворота. Положительное значение угла поворачивает элемент по часовой стрелке, отрицательное против.

rotateX()

Поворачивает элемент вокруг оси X на заданный угол α.

transform: rotateX(α)

rotateX(α) является синонимом rotate3D(1, 0, 0, α).

rotateY()

Поворачивает элемент вокруг оси Y на заданный угол α.

transform: rotateY(α)

rotateY(α) является синонимом rotate3D(0, 1, 0, α).

rotateZ()

Поворачивает элемент вокруг оси Z на заданный угол α.

transform: rotateZ(α)

rotateZ(α) является синонимом rotate3D(0, 0, 1, α).

skew()

Наклоняет элемент на заданное значение по горизонтали и вертикали в плоскости экрана.

transform: skew(ax[, ay])

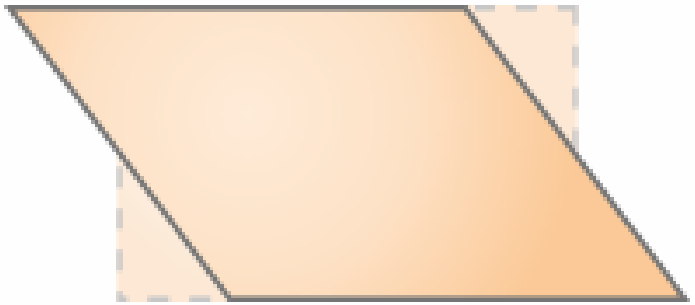
Здесь:

ax — угол наклона по оси X;

ay — угол наклона по оси Y. Если значение ay не указано, то оно считается равным ax: skew(30deg) соответствует skew(30deg, 30deg).

skewX()

Наклоняет элемент на заданный угол по горизонтали. Положительное значение наклоняет влево, отрицательное вправо.



transform: skewX(α)

Здесь:

α — угол наклона по оси X. В примере показано создание наклонного меню с помощью функции skewX().

Переходы

Переходы в CSS позволяют плавно перейти от одного состояния элемента к другому. Работает это так, что отдельные свойства анимируются от начального состояния до конечного.

Вы можете определить:

transition-property: какие свойства анимируются;
transition-duration: как долго длится анимация;
transition-timing-function: как вычисляются промежуточные состояния;
transition-delay: анимация начинается через некоторое время.

Вы можете установить каждое свойство CSS по отдельности или использовать сокращённую версию: transition. В этом случае, только transition-duration является обязательным.

Имейте в виду, что переход представляет собой специфический вид анимации, где есть только начальное и конечное состояние.

(index_9)

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?

Свойства анимации

- ***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.

@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.

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, 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, 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.

animation-delay

Как и с задержкой перехода, 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.

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.

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.

animation-fill-mode

- ***none*** Если установлена задержка анимации - то в течении времени задержки элемент будет оставаться на месте, а потом скачком перейдет к 0% кадру. После окончания анимации элемент не останется в том состоянии, где остановился, а перепрыгнет в начальное состояние.
- ***backwards*** Заставляет элемент двигаться после загрузки страницы к 0% кадру, даже если установлена задержка animation-delay, и оставаться там, пока не начнется анимация. После окончания анимации элемент не останется в том состоянии, где остановился, а перепрыгнет в начальное состояние.
- ***forwards*** Указывает браузеру, что после окончания анимации элемент останется в том состоянии, где остановился.
- ***both*** Включает в себя backwards и forwards - после загрузки страницы элемент установится к 0% кадру, а после окончания анимации элемент останется там, где отановился.

(index_10)