**Flexbox**

Модуль (Flexible Box) направлен на обеспечение более эффективного способа размещения, выравнивания и распределения пространства между элементами в контейнере, даже если их размер неизвестен и/или динамичен.

Основная идея гибкого макета заключается в том, чтобы дать контейнеру возможность изменять ширину/высоту (и порядок) своих элементов, чтобы наилучшим образом заполнить доступное пространство (в основном для размещения всех видов устройств отображения и размеров экрана). Гибкий контейнер расширяет элементы, чтобы заполнить доступное свободное пространство, или сжимает их, чтобы предотвратить переполнение.

Самое главное, что макет флексбокса не зависит от направления, в отличие от обычных макетов (блок, который основан на вертикали, и встроенный, который основан на горизонтали). Хотя они хорошо работают для страниц, им не хватает гибкости для поддержки больших или сложных приложений (особенно когда речь идет об изменении ориентации, изменении размера, растяжении, сжатии и т. Д.).

**Заметка:** Макет flexbox наиболее подходит для компонентов приложения и мелкомасштабных макетов, в то время как макет [Grid](https://css-tricks.com/snippets/css/complete-guide-grid/) предназначен для макетов большего масштаба.

**Свойства контейнера Parent(flex)**

1. **Display(дисплей)**

Это определяет гибкий контейнер; встроенный или в виде блока в зависимости от заданного значения. Это обеспечивает гибкий контекст для всех его непосредственных «детей».

.container {

display: flex; */\* or inline-flex \*/*

}

Обратите внимание, что столбцы CSS не влияют на контейнер flex.

#### flex-direction (гибкое направление)

Это устанавливает главную ось, тем самым определяя направление размещения гибких элементов в гибком контейнере. Flexbox — это (помимо дополнительной упаковки) однонаправленная концепция макета. Думайте о гибких элементах, в основном расположенных либо в горизонтальных рядах, либо в вертикальных столбцах.

.container {

flex-direction: row | row-reverse | column | column-reverse;

}

* ***row*** (по умолчанию): слева направо; справа налево
* ***row-reverse:*** справа налево; слева направо
* ***column:*** то же самое, но сверху внизrow
* ***column-reverse:*** то же самое, но снизу вверх row-reverse

#### flex-wrap (флекс-обертывание)

По умолчанию все элементы flex будут пытаться поместиться на одной линии. Это можно изменить и разрешить перенос элементов по мере необходимости с помощью этого свойства.

.container {

flex-wrap: nowrap | wrap | wrap-reverse;

}

* ***nowrap*** (по умолчанию): все элементы flex будут находиться на одной строке
* ***wrap:*** гибкие элементы будут переноситься на несколько линий, сверху вниз.
* ***wrap-reverse:*** гибкие элементы будут переноситься на несколько линий снизу вверх.

#### flex-flow (гибкий поток)

Это сокращение для свойств, которые вместе определяют основную и поперечные оси гибкого контейнера. Значение по умолчанию — .flex-directionflex-wraprow nowrap

.container {

flex-flow: column wrap;

}

#### justify-content (обоснование-содержание)

Это определяет выравнивание вдоль главной оси. Это помогает распределить дополнительное свободное пространство, оставшееся, когда все гибкие элементы на линии являются негибкими или гибкими, но достигли своего максимального размера. Он также осуществляет некоторый контроль над выравниванием элементов, когда они переполняют линию.

.container {

justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly | start | end | left | right ... + safe | unsafe;

}

* ***flex-start*** (по умолчанию): элементы располагаются к началу гибкого направления.
* ***flex-end:*** предметы располагаются ближе к концу гибкого направления.
* ***start:*** предметы располагаются к началу направления
* ***end:*** предметы располагаются ближе к концу направления
* ***left:*** элементы располагаются по левому краю контейнера, если это не имеет смысла, то он ведет себя как .flex-directionstart
* ***right:*** элементы располагаются по правому краю контейнера, если это не имеет смысла, то он ведет себя как .flex-directionend
* ***center:*** элементы центрируются вдоль линии
* ***space-between:*** позиции равномерно распределены по линии; первый элемент находится в начальной строке, последний элемент в конечной строке
* ***space-around:*** элементы равномерно распределены в линию с равным пространством вокруг них. Обратите внимание, что визуально пространства не равны, так как все элементы имеют равное пространство с обеих сторон. Первый элемент будет иметь одну единицу пространства против края контейнера, но две единицы пространства между следующим элементом, поскольку этот следующий элемент имеет свой собственный интервал, который применяется.
* ***space-evenly:*** элементы распределены таким образом, чтобы расстояние между любыми двумя элементами (и пространство до краев) было равным.

#### align-items (выравнивание элементов)

Это определяет поведение по умолчанию для того, как гибкие элементы расположены вдоль **поперечной оси** на текущей линии. Думайте об этом как о версии для поперечной оси (перпендикулярной главной оси).justify-content

.container {

align-items: stretch | flex-start | flex-end | center | baseline | first baseline | last baseline | start | end | self-start | self-end + ... safe | unsafe;

}

* ***stretch*** (по умолчанию): растянуть для заполнения контейнера (по-прежнему соблюдайте минимальную ширину/максимальную ширину)
* ***flex-start / start / self-start:*** предметы размещаются в начале поперечной оси.
* ***flex-end / end / self-end:*** предметы размещаются в конце поперечной оси.
* ***center:*** элементы центрированы по поперечной оси
* ***baseline:*** элементы выравниваются, например, их базовые показатели выравниваются

#### align-content (выравнивание содержимого)

Это выравнивает линии гибкого контейнера внутри, когда в поперечной оси есть дополнительное пространство, подобно тому, как выравниваются отдельные элементы внутри главной оси.

**Заметка:** Это свойство вступает в силу только для многострочных гибких контейнеров, где установлено значение либо или). Однострочный гибкий контейнер (т.е. где задано значение по умолчанию) не будет отражать .flex-wrapwrapwrap-reverseflex-wrapno-wrapalign-content

.container {

align-content: flex-start | flex-end | center | space-between | space-around | space-evenly | stretch | start | end | baseline | first baseline | last baseline + ... safe | unsafe;

}

* ***normal*** (по умолчанию): элементы располагаются в положение по умолчанию, как если бы значение не было задано.
* ***flex-start / start:*** предметы, расположены в начале контейнера.
* ***flex-end / end:*** предметы, расположены в конце контейнера.
* ***center:*** элементы, центрированные в контейнере
* ***space-between:*** равномерно распределенные предметы; Первая строка находится в начале контейнера, а последняя — в конце
* ***space-around:*** элементы равномерно распределены с равным пространством вокруг каждой строки
* ***space-evenly:*** предметы равномерно распределены с равным пространством вокруг них
* ***stretch:*** линии растягиваются, чтобы занять оставшееся пространство

#### gap, row-gap, column-gap (зазор, зазор-строка, столбец-зазор)

[Свойство gap](https://css-tricks.com/almanac/properties/g/gap/) явно управляет пространством между элементами flex. Применяется этот интервал *только между элементами*, а не по внешним краям.

.container {

display: flex;

...

gap: 10px;

gap: 10px 20px; */\* row-gap column gap \*/*

row-gap: 10px;

column-gap: 20px;

}

## Свойства для дочерних элементов (элементы flex)

#### Order (порядок)

По умолчанию элементы flex размещаются в исходном порядке. Однако свойство управляет порядком, в котором они отображаются в контейнере

.item {

order: 5; */\* default is 0 \*/*

}

Элементы с одинаковым порядком возвращаются к исходному порядку.***order***

#### flex-grow (гибкое выращивание)

Это определяет способность гибкого элемента расти в случае необходимости. Он принимает значение, которое служит пропорцией. Он определяет, какое количество доступного пространства внутри гибкого контейнера должен занимать элемент.

Если для всех элементов задано значение, оставшееся пространство в контейнере будет распределено поровну между всеми дочерними элементами. Если один из «детей» имеет значение, то он займет в два раза больше места.

.item {

flex-grow: 4; */\* default 0 \*/*

}

Отрицательные числа недопустимы.

#### flex-shrink (гибкая усадка)

Это определяет способность гибкого элемента сжиматься при необходимости.

.item {

flex-shrink: 3; */\* default 1 \*/*

}

Отрицательные числа недопустимы.

#### flex-basis (гибкая основа)

Это сокращение комбинированное. Второй и третий параметры являются необязательными. Значение по умолчанию — но если вы установите его с одним числовым значением, например , это изменит значение на 0%, то это похоже на настройку .flex-grow,flex-shrinkflex-basisflex-shrinkflex-basis0 1 autoflex: 5;flex-basisflex-grow: 5; flex-shrink: 1; flex-basis: 0%;

.item {

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

}

#### align-self(выравнивание-самонастройка)

Это позволяет переопределить выравнивание по умолчанию (или выравнивание, указанное параметром) для отдельных элементов

.item {

align-self: auto | flex-start | flex-end | center | baseline | stretch;

}