Флекс-элементы и блочная модель

Ширина, высота, внутренние отступы и рамки для флекс-контейнеров и флекс-элементов работают как обычно: общий размер элементов складывается из этих компонентов. Это поведение также можно менять с помощью свойства `box-sizing`.

Есть и несколько важных отличий:

- флекс-элементы, в отличие от блочных элементов, не растягиваются на всю ширину контейнера по умолчанию;
- на флекс-элементы не действует свойство `float`.

Особенности свойства margin

Свойство `margin` таит много сюрпризов:

- внешние отступы не схлопываются, ни по горизонтали, ни по вертикали;
- внешние отступы не выпадают, ни из флекс-контейнера, ни из флекс-элементов;
- значение `auto` получило премию журнала Форбс в номинации «Самое влиятельное значение CSS-свойства внутри флекс-контейнера».

Всё дело в механизме распределения свободного места. Если внутри флекс-контейнера есть свободное место, то оно перераспределяется так:

1. находятся элементы, у которых есть внешние отступы со значением `auto`;
2. всё свободное место в соответствующих направлениях отдаётся таким отступам (то есть задаётся определённый размер отступа в пикселях);
3. если элементов с автоматическими отступами на одном направлении несколько, то место между ними перераспределяется поровну;
4. только после этого запускаются механизмы выравнивания.

Поэтому `margin: auto;` влияет на положение флекс-элементов на обеих осях, а также «ломает» механизмы выравнивания, ведь выравнивание происходит, когда есть свободное место. Но если всё свободное место «перетекло» в автоматический отступ, то и выравнивать нечего.

Эти особенности можно использовать во благо. Например, с помощью автоматических отступов вы можете управлять расположением элементов вдоль главной оси.
Закрыть

Выравнивание и внешние отступы

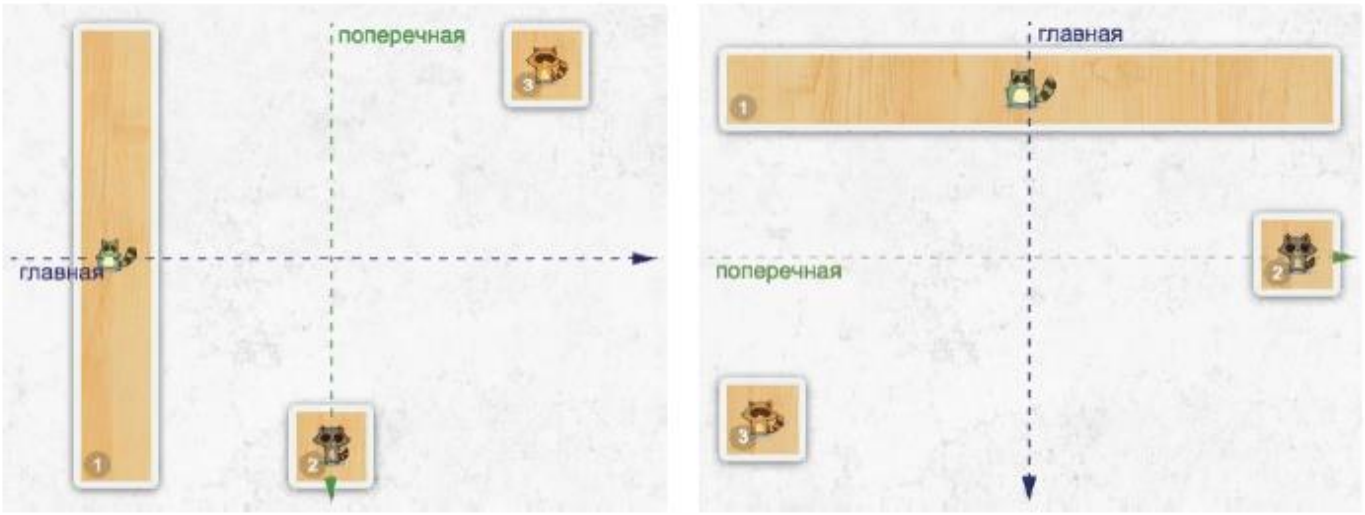
Автоматический `margin` влияет и на выравнивание флекс-элементов вдоль поперечной оси.

Если у флекс-элемента отступ сверху или снизу автоматический, то на него не влияют, ни `align-items`, ни `align-self`. Такой элемент прижмётся либо к верху контейнера, либо к низу.

А если задать автоматические отступы с противоположных сторон, то элемент разместится по центру флекс-контейнера, так как свободное место «впитается» отступами поровну.

Направление главной оси и внешние отступы

Будет ли результат таким, как на картинке снизу, если повернуть главную ось?



Нет, не будет!

Дело в том, что «старые нефлексовые» свойства, такие как отступы или размеры, ничего не знают про направление осей. Они «мыслят по-старому», понятиями «верх» и «низ», «право» и «лево».

Поэтому когда главная ось направлена слева направо, горизонтальные отступы перемещают флекс-элементы вдоль *главной* оси. Но если направить главную ось сверху вниз, то те же отступы начнут работать вдоль *поперечной* оси.

То же относится и к вертикальным отступам.

Базовый размер флекс-элемента, flex-basis

На примере отступов видно, что «старые» свойства внутри флекс-контейнера ведут себя достаточно глупо. Ширина и высота тоже не умеют реагировать на поворот главной оси. Поэтому ввели понятия *главный размер* или *main size* и *поперечный размер* или *cross size*.

Если главная ось направлена горизонтально, то главный размер — это ширина, свойство `width`, а поперечный размер — это высота, свойство `height`. Если главная ось направлена вертикально, то всё наоборот.

А хотелось бы иметь «умное» свойство для задания размера флекс-элементов, которое знает про главную ось и «поворачивается» вместе с ней.

И такое свойство есть — это `flex-basis`. Оно задаёт *базовый размер* флекс-элемента или размер вдоль главной оси.

Если `flex-basis` не задан или его значение равно `auto`, то базовый размер берётся из `width` или `height`.

Свойство `flex-basis` принимает те же значения, что и `width`/`height`:

```
flex-basis: 100px; /* базовый размер 100 пикселей */

flex-basis: 50%;   /* базовый размер 50% контейнера */
```

Свойство `flex-basis` «сильнее» свойств `width` и `height`, и если у флекс-элемента заданы все три свойства, то `flex-basis` переопределит либо ширину, либо высоту в зависимости от направления главной оси.

Коэффициент растягивания элементов, flex-grow

На самом деле, базовый размер — это не просто размер элемента вдоль главной оси, это *начальный* или *исходный* размер вдоль главной оси.

Почему так важны эти *начальный* или *исходный*?

И снова всё дело в механизме перераспределения свободного места во флексбоксе.

Если внутри флекс-контейнера по главной оси остаётся свободное место, то мы можем попросить флекс-элемент, чтобы он увеличился и занял это место. Это делается с помощью свойства `flex-grow`, которое можно назвать «коэффициентом флекс-жадности» флекс-элемента.

Свойство `flex-grow` принимает неотрицательные числовые значения, его значение по умолчанию — `0`.

Если значение `flex-grow` равно нулю, то флекс-элемент «не претендует» на оставшееся свободное место во флекс-контейнере и не будет увеличиваться, чтобы занять это место.

Если значение `flex-grow` больше нуля, то флекс-элемент будет увеличиваться, «захватывая» оставшееся свободное место.

Получается, что базовый размер — это исходный размер флекс-элементов до применения `flex-grow`.

Пропорциональное растягивание элементов

Если сразу у нескольких флекс-элементов значение `flex-grow` больше нуля, то они будут делить свободное место между собой.

Свободное место будет добавляться флекс-элементам пропорционально значениям их «коэффициента жадности». Например, если во флекс-контейнере есть два элемента:

```
.element-1 { flex-grow: 1; }

.element-2 { flex-grow: 2; }
```

То второму элементу достанется в два раза больше свободного места, чем первому. Если изменить значения коэффициентов у этих элементов на такие:

```
.element-1 { flex-grow: 50; }

.element-2 { flex-grow: 100; }
```

То ничего не изменится, так как отношение коэффициентов не изменилось: 100 в два раза больше 50. То есть важно не само значение коэффициента, а его соотношение с коэффициентами остальных элементов.

Расчёт итогового размера с flex-grow

1 шаг. Рассчитываем свободное место во флекс-контейнере:

```
Свободное место = Ширина контейнера - Сумма базовых размеров элементов
```

2 шаг. Считаем размер минимальной доли свободного места:

```
Доля свободного места = Свободное место / Сумма flex-grow всех элементов
```

3 шаг. Базовый размер каждого флекс-элемента увеличиваем на размер минимальной доли свободного места, умноженной на значение `flex-grow` этого элемента:

```
Итоговый размер = Базовый размер + (Доля свободного места * flex-grow)
```

Для верхнего блока с енотами хочется задать коэффициенты `1` и `2`. Но нужные размеры блоков получаются с коэффициентами `1` и `3`. Давайте посчитаем:

```
Свободное место = 300px - (50px * 2) = 200px

Доля свободного места = 200px / (1 + 3) = 50px

Итоговый размер зелёного енота = 50px + (50px * 1) = 100px

Итоговый размер коричневого енота = 50px + (50px * 3) = 200px
```

Но если задать флекс-элементам нулевой базовый размер, свободное место будет занимать всю ширину флекс-контейнера, и коэффициенты жадности будут другими.

Использовать `flex-basis: 0` и `flex-grow` для точного управления относительными размерами не стоит. Лучше использовать базовый размер в процентах.

Тонкость. На размер оставшегося свободного места влияет не только `flex-basis`, но и рамки, и отступы. Если `flex-basis` явно задано нулевое значение, то `min-width` на размер свободного места влиять не будет, так как ограничения размеров к флекс-элементам применяются уже после перераспределения свободного места.
Закрыть

Коэффициент сжатия элементов, flex-shrink

Если сумма базовых размеров флекс-элементов больше, чем размер флекс-контейнера, то возникает *отрицательное пространство*.

Механизм перераспределения работает не только для свободного места, но и для отрицательного пространства. Флекс-элементы умеют распределять отрицательное пространство между собой и сжиматься.

За уменьшение флекс-элементов отвечает свойство `flex-shrink`, которое можно назвать «коэффициентом сжатия».

Свойство `flex-shrink` принимает неотрицательные числовые значения, его значение по умолчанию — `1`.

Если значение `flex-shrink` больше нуля, то флекс-элемент будет уменьшаться, «впитывая» часть отрицательного пространства, если оно существует.

Если значение `flex-shrink` равно нулю, то флекс-элемент уменьшаться не будет.

Флекс-элементы стараются быть максимально «гибкими» и не выпадать из своего контейнера, поэтому у `flex-shrink` значение по умолчанию равно `1`. Но если задавать нулевые значения для коэффициента сжатия, то выпадения элементов добиться можно.

Пропорциональное сжатие элементов

Свойство `flex-shrink` очень похоже на свойство `flex-grow`. Оно задаёт пропорции, в которых флекс-элементы «впитывают» отрицательное пространство.

Чем больше значение коэффициента сжатия у элемента, тем больше отрицательного пространства он «впитает» и тем сильнее сожмётся.

Когда базовые размеры флекс-элементов *одинаковы*, пропорции сжатия элементов считаются так же, как пропорции увеличения. Если базовые размеры флекс-элементов отличаются, то механизм усложняется. Подробно мы разберём его в следующем задании.

Расчёт итогового размера с flex-shrink

Ниже описан механизм расчёта размеров элементов, когда места в контейнере не хватает:

1 шаг. Рассчитываем отрицательное пространство (ОП) во флекс-контейнере:

```
ОП = Ширина контейнера - Сумма базовых размеров элементов
```

2 шаг. Находим сумму произведений базовых размеров (СПБР) элементов на их коэффициенты сжатия:

```
СПБР = (Базовый размер₁ * flex-shrink₁) + (Базовый размер₂ * flex-shrink₂) + … + (Базовый размерₙ * flex-shrinkₙ)
```

3 шаг. Для каждого элемента считаем «нормированный коэффициент сжатия» (НКС), для чего произведение базового размера элемента на его коэффициент сжатия делим на СПБР:

```
НКС = (Базовый размер * flex-shrink) / СПБР
```

4 шаг. Базовый размер элемента уменьшаем на часть ОП пропорциональную НКС элемента. ОП для расчёта берём по модулю, то есть отбрасывая минус:

```
Итоговый размер = Базовый размер - (НКС * ОП)
```

Получается, что доля отрицательного пространства, которую «впитает» элемент, зависит от двух факторов:

- соотношения коэффициента сжатия элемента с коэффициентами других элементов,
- соотношения базового размера элемента с базовыми размерами других элементов.

flex-shrink и min-width

Давайте рассчитаем размеры элементов и убедимся в правильности описанного алгоритма.

```
Отрицательное пространство = 200px - 100px - 300px = -200px


Сумма произведений размеров на коэффициенты = (1 * 100px) + (1 * 300px) = 400px


Нормированный коэффициент 1 элемента = (1 * 100px) / 400px = 0.25
Нормированный коэффициент 2 элемента = (1 * 300px) / 400px = 0.75


Итоговый размер 1 элемента = 100px - (200px * 0.25) = 50px
Итоговый размер 2 элемента = 300px - (200px * 0.75) = 150px
```

Есть несколько тонкостей, касающихся сжатия флекс-элементов:

- элементы сжимаются в пределах своих базовых размеров, внутренние отступы и рамки не сжимаются;
- «ограничительные» свойства, такие как `min-width`, применяются к элементам после этапа перераспределения свободного места или отрицательного пространства.

И эти тонкости могут приводить к неожиданным эффектам, когда элементы выпадают из флекс-контейнера.

Сокращённое свойство flex

С помощью сокращённого свойства `flex` можно одновременно задать коэффициенты растягивания, сжатия и базовый размер флекс-элемента.

Свойство `flex` состоит из трёх компонентов, которые пишутся через пробел в следующем порядке: `flex-grow`, `flex-shrink` и `flex-basis`. В примере ниже два правила аналогичны:

```
.flex-item {
  flex: 1 2 300px;
}

.flex-item {
  flex-grow: 1;
  flex-shrink: 2;
  flex-basis: 300px;
}
```

Ещё у свойства `flex` есть особые значения: `initial`, `auto`, `none`. Также второй и третий компоненты необязательны. Ниже показаны различные значения свойства и их расшифровки.

```
flex: initial; -> flex: 0 1 auto;
flex: auto;  -> flex: 1 1 auto;
flex: none;  -> flex: 0 0 auto;
flex: 1 0;   -> flex: 1 0 0%;
flex: 1;     -> flex: 1 1 0%;
```

В некоторых браузерах неполные или особенные значения свойства `flex` интерпретируются с ошибками. Поэтому лучше задавать все три компоненты в значении этого свойства.

Многострочный флекс-контейнер и flex-shrink

Во всех примерах, рассмотренных раньше, флекс-контейнер был однострочным, ведь перенос флекс-элементов на новую строку по умолчанию запрещён — работает `flex-wrap: nowrap;`.

А как будут растягиваться и сжиматься элементы в многострочном контейнере, с `flex-wrap: wrap;`?

В таком контейнере свойство `flex-shrink` будет работать как обычно, но необходимость в нём будет возникать намного реже. Ведь при нехватке места в строке флекс-элементы будут переноситься на новую строку.

Но если появятся флекс-элементы, базовый размер которых больше размера флекс-контейнера, то такие элементы будут сжиматься и занимать целую строку. Наверное, это единственный случай, когда `flex-shrink` делает что-то полезное в многострочном контейнере.

Закрыть

Многострочный флекс-контейнер и flex-grow

В отличие от `flex-shrink`, свойство `flex-grow` в многострочном флекс-контейнере срабатывает намного чаще и пользы приносит намного больше.

В каждой строке такого контейнера может быть свободное место и механизм перераспределения этого места работает построчно.

Поэтому возможность «растянуть» флекс-элементы, чтобы строки заполнялись по ширине полностью, будет возникать достаточно часто.

Закрыть

flex-basis: 100% и flex-wrap

Познакомимся с интересным эффектом, который возникает при использовании базовых размеров в процентах.

Если задать базовый размер флекс-элемента `100%` и при этом включить перенос элементов на новую строку, то элементы расположатся столбцом, хотя главная ось контейнера будет по-прежнему направлена слева направо.

Закрыть

- Теория

Поля ввода с динамической шириной

Ещё один случай, когда может пригодиться флексбокс — поля ввода с динамической шириной. Требования к ним такие:

- На одной строке с полем могут находиться другие элементы: кнопки, ссылки, подписи.
- Размер дополнительных элементов не определён, он зависит от их содержимого.
- При этом поле должно растягиваться на всё оставшееся в родительском контейнере место.

- И изменять ширину при изменении размеров контейнера.

  Решить эту задачу можно только при помощи флексбокса. Превратим контейнер поля ввода во флекс-контейнер, все элементы внутри него превратятся во флекс-элементы, базовый размер которых будет зависеть от их содержания — `flex-basis: auto;`. И останется только задать ненулевой коэффициент растягивания полям ввода.

  В широком контейнере всё будет работать отлично. Проблемы могут появиться в слишком узких контейнерах: по умолчанию поля ввода не будут сжиматься после определённой ширины, что приведёт к выпаданию текста из остальных элементов.

  Чтобы справиться с этими проблемами, надо задать всем элементам кроме полей ввода нулевой коэффициент сжатия, а самим полям ввода явно прописать минимальную ширину.