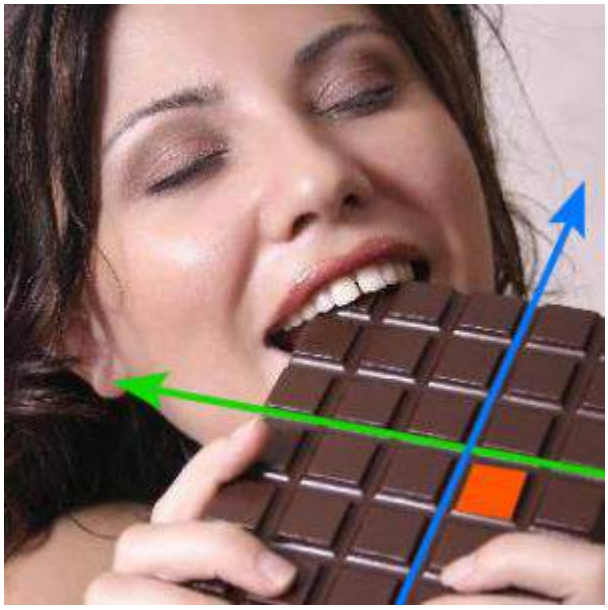
Что такое Flexbox? Описание всех css свойств, основные принципы, преимущества и недостатки.

28



Flexbox по праву можно назвать удачной попыткой решения огромного спектра проблем при построении лейаутов в css. Но прежде чем перейти к его описанию, давайте выясним, что же не так со способами верстки, которыми мы пользуемся сейчас?

Любой верстальщик знает несколько путей выровнять что-либо по вертикали или сделать 3-х колоночный макет с резиновой средней колонкой. Но давайте признаем, что все эти способы довольно странные, похожи на хак, подходят не во всех случаях, сложны для восприятия и не работают при несоблюдении определенных магических условий, которые сложились исторически.

Случилось так потому, что html и css развивались эволюционно. В начале веб-страницы были похожи на однопоточные текстовые документы, чуть позже разбиение страницы на блоки делали таблицами, затем стало модным верстать float-ами, а после официальной смерти ie6 добавились еще и приемы с inline-block. В итоге мы получили в наследство гремучую смесь всех этих приемов, используемую для построения лейаутов 99,9% всех существующих веб-страниц.

Спецификация CSS Flexible Box Layout Module (в народе Flexbox) призвана кардинально изменить ситуацию в лучшую сторону при решении огромного количества задач. Flexbox позволяет контролировать размер, порядок и выравнивание элементов по нескольким осям, распределение свободного места между элементами и многое другое.

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.

Поддержка браузерами

Поддержка браузерами пока неполная (2014). Виноват в этом в основном Internet explorer, который поддерживает спецификацию 2011 года только начиная с 10 версии, . Не смотря на это, я бы порекомендовал обратить внимание на обширность поддержки всеми остальными мобильными и десктопными браузерами! Тут все прекрасно. Если Вам нужна мобильная версия сайта или web-based приложение, то его уже можно (а, возможно, и нужно) делать, используя все преимущества flexbox!

Немного истории

- 2008 – CSS Working Group обсуждает предложение “Flexible Box Model” на основе XUL (XML User Interface Language – язык разметки в приложениях Mozilla) and XAML (Extensible Application Markup Language – язык разметки в приложениях Microsoft).
- 2009 – опубликован черновик “Flexible Box Layout Module”. Chrome и Safari добавляет частичную поддержку, пока Mozilla начинает поддерживать XUL-подобный синтаксис, известный как “Flexbox 2009”.
- 2011 – Tab Atkins берется за развитие Flexbox и публикует 2 черновика. В этих черновиках синтаксис изменен значительно. Chrome, Opera и IE 10 внедряют поддержку этого синтаксиса. Он известен под названием “flexbox 2011”
- 2012 – Синтаксис снова немного изменен и уточнен. Спецификация переходит в статус Candidate Recommendation и известна под названием “flexbox 2012”. Opera внедряет беспрефиксную поддержку, Chrome поддерживает текущую спецификацию с префиксами, а Mozilla без них, IE10 добавляет поддержку устаревающего “flexbox 2011” синтаксиса.
- 2014 – все новые браузеры поддерживают последнюю спецификацию (включая IE 11)

Мы будем рассматривать все примеры на основе новой спецификации. Если вам нужна поддержка старых Chrome, FF и IE10, лучше использовать autoprefixer от Андрея Ситника, который автоматически добавит css правила и вендорные префиксы для устаревших спецификаций.

Начинаем погружение

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`.

HTML

```
<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>
```

CSS

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

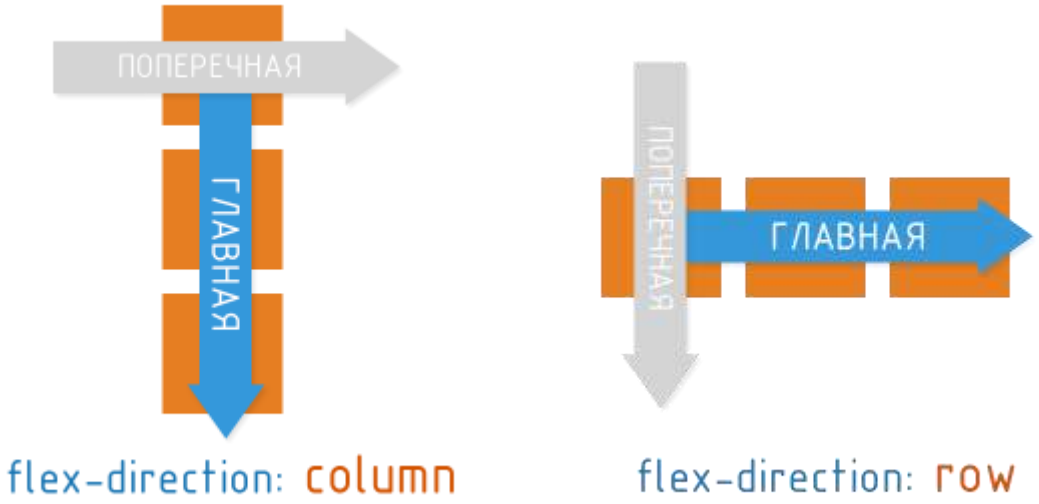
Основные свойства flex-контейнера. Главная и поперечная ось.

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.

Главная ось в ltr локали по умолчанию располагается слева направо. Поперечная – сверху вниз. Направление главной оси flex-контейнера можно задавать, используя базовое css свойство `flex-direction`.

`flex-direction` – направление главной оси



Доступные значения flex-direction:

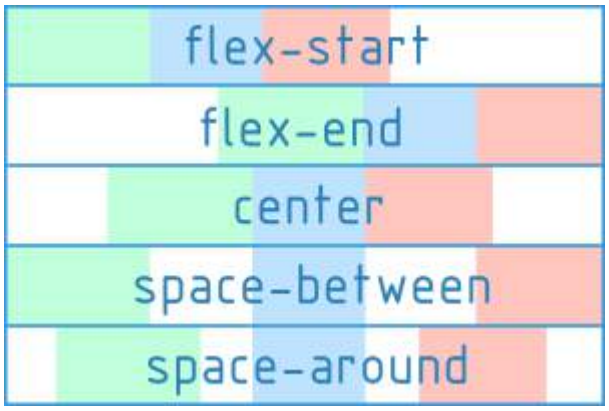
- `row` *(значение по умолчанию)* : слева направо (в rtl справа налево)
- `row-reverse`: справа налево (в rtl слева направо)
- `column`: сверху вниз
- `column-reverse`: снизу вверх

`justify-content` – выравнивание по главной оси.

Css свойство `justify-content` определяет то, как будут выровнены элементы вдоль 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.



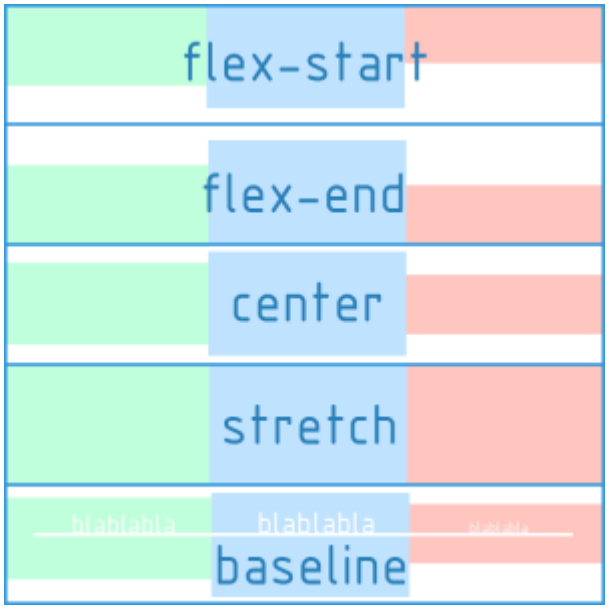
`align-items` – выравнивание по поперечной оси.

Css свойство `align-items` определяет то, как будут выровнены элементы вдоль 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.



CSS свойства `flex-direction`, `justify-content`, `align-items` должны применяться непосредственно к flex-контейнеру, а не к его дочерним элементам.

Демо основных свойств flex-контейнера

Оси и выравнивания по ним – это основы flex. Расслабьтесь, покликайте по демке и используйте ее, если нужно будет освежить в памяти.

открыть в новом окне

Многострочная организация блоков внутри flex-контейнера.

`flex-wrap`

Все примеры, которые мы приводили выше, были построены с учетом однострочного (одностолбцового) расположения блоков. Надо сказать, что по умолчанию flex-контейнер всегда будет располагать блоки внутри себя в одну линию. Однако, спецификацией также поддерживается многострочный режим. За многострочность 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`: то-же что и wrap, но блоки располагаются в обратном порядке.

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

По сути,`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'>`

CSS

```
/* т.е. ... */

.my-flex-block{
  flex-direcrion:column;
  flex-wrap: wrap;
}

/* это то же самое, что ... */

.my-flex-block{
  flex-flow: column wrap;
}
```

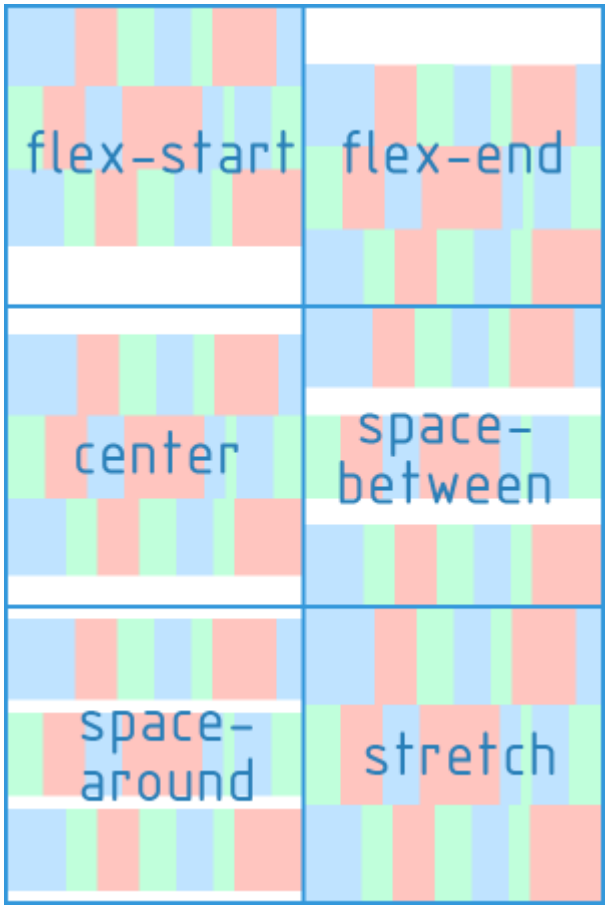
`align-content`

Существует также свойство `align-content`, которое 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.



CSS свойства `flex-wrap` и `align-content` должны применяться непосредственно к flex-контейнеру, а не к его дочерним элементам.

Демо свойств многострочности в flex

открыть в новом окне

CSS правила для дочерних элементов flex-контейнера (flex-блоков)

`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 указанны 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`.

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

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

CSS

```
/* 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;
}
```

Демо для flex-grow, flex-shrink и flex-basis

открыть в новом окне

`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.

`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.

HTML

```
<div class="my-flex-container">
  <div class="my-flex-block" style="order: 5" >item1</div>
  <div class="my-flex-block" style="order: 10">item2</div>
  <div class="my-flex-block" style="order: 5" >item3</div>
  <div class="my-flex-block" style="order: 5" >item4</div>
  <div class="my-flex-block" style="order: 0" >item5</div>
</div>
```

В данном случае, блоки будут следовать один за другим вдоль главной оси в следующем порядке: item5, item1, item3, item4, item2

Демо для align-self и order

открыть в новом окне

`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!

```
.my-flex-container {
  display: flex;
  height: 300px; /* Или что угодно */
}

.my-flex-block {
  width: 100px;  /* Или что угодно */
  height: 100px; /* Или что угодно */
  margin: auto;  /* Магия! Блок отцентрирован по вертикали и горизонтали! */
}
```

Вещи, которые следует помнить

1. Не следует использовать flexbox там, где в этом нет необходимости.
2. Определение регионов и изменение порядка контента во многих случаях все-таки полезно делать зависимым от структуры страницы. Продумывайте это.
3. Разберитесь в flexbox и знайте его основы. Так намного легче достичь ожидаемого результата.
4. Не забывайте про margin-ы. Они учитываются при установке выравнивания по осям. Также важно помнить, что margin-ы в flexbox не “коллапсятся”, как это происходит в обычном потоке.
5. Значение float у flex-блоков не учитывается и не имеет значения. Это, наверно, как-то можно использовать для graceful degradation при переходе на flexbox.
6. flexbox очень хорошо подходит для верстки веб-компонентов и отдельных частей веб-страниц, но показал себя не с лучшей стороны при верстке базовых макетов (расположение article, header, footer, navbar и т.п.). Это все еще спорный момент, но эта статья довольно убедительно показывает недостатки xanthir.com/blog/b4580

В заключение

Я думаю, что flexbox, конечно же, не вытеснит все остальные способы верстки, но, безусловно, в ближайшее время займет достойную нишу при решении огромного количества задач. И уж точно, пробовать работать с ним нужно уже сейчас. Одна из следующих статей будет посвящена конкретным примерам работы с flex-версткой. Подписывайтесь на новости ;)