
Яндекс

Яндекс

CSS

Александр Нефедов

План

- Источники стилей и каскад
- Синтаксис
- Видимость элементов
- Нормальный поток и блоки
- Инлайновый контекст
- Flexbox, Grid
- Позиционирование
- Устаревшее
- Советы

Cascading Style Sheets

Каскадные таблицы стилей

> Язык для описания внешнего вида

Cascading Style Sheets

Браузеры развивались очень динамично и потребовался общий стандарт

Cascading Style Sheets

Нужен для того, чтобы разделить содержимое и внешний вид, позволяет переиспользовать стили

Cascading Style Sheets

CSS позволяет стилизовать контент под разные устройства

- Десктопы
- Телефоны
- Читалки
- Для печати
- Скринридеры
- Дисплеи Брайля

Cascading Style Sheets

CSS позволяет стилизовать контент под разные ситуации

- Точное устройство (мышь)
- Мобильные устройства
- Светлая / тёмная тема
- Контрестная тема
- Чёткий экран
- Медленный экран
- Уменьшенное число анимаций

Браузеры

- Chromium
- Safari
- Firefox
- Internet Explorer / Edge
- Opera (presto)

CSS

CSS обладает миллионом возможностей, которые можно сочетать в разных комбинациях. С их помощью можно делать удивительные вещи

Источники стилей, каскад и наследование

Стили браузера

```
section {
    display: block;
}
```

Итоговые стили

- Стили браузера
- Стили сайта
- Стили пользователя / расширений

Стили сайта

```
<link rel="stylesheet" href="some.css">
```

Стили сайта

```
<style>
    .header {
        font-size: 20px;
    }
</style>
```

Стили сайта

```
<span style="color: red">Hello world</span>
```

Часть свойств наследуется, а часть - нет

```
<!-- Имеет и цвет, и отступ -->
<span style="color: red; padding: 10px">
    <!-- Имеет только цвет -->
    <span>
        Hello world
    </span>
</span>
```

Синтаксис

Комментарии

```
/* Эти стили нужны, чтобы корректно отображался случай X */
.something {}
```

Селекторы

```
/* Класс */
.something {}

/* ID */
#something {}

/* HTML-тег */
something {}

/* HTML-атрибут */
[something="yes"] {}
```

Комбинаторы

```
/* Дочерний элемент */
.something .children {}

/* Прямой потомок */
.something > .children {}

/* Соседний элемент */
.something + .children {}

/* Имеет содержимое */
.something:not(.active) {}
```

Псевдоклассы

```
/* Навели мышь */
.something:hover {}

/* Невалидное поле формы */
.something:invalid {}
```

Псевдоэлементы

```
/* Дополнительный элемент перед содержимым */
.something::before {}

/* Подсказка в поле ввода */
.something::placeholder {}
```

Свойства

```
.something {
    width: 100px;  /* 1 */
    height: 100px; /* 2 */
    отступ: 10px;  /* 3 */
    padding: 10px; /* 4 */
}
```

Свойства

```
.something {
    width: 100px;  /* 1 */
    height: 100px; /* 2 */
    отступ: 10px;  /* 3 */
    padding: 10px; /* 4 */
}
```

Единицы измерения

- px
- %
- em
- rem
- vw, vh
- И ещё много других

Переменные

```
.something {
    --some-var: 100px;
}

.another-thing {
    width: var(--some-var);
}
```

calc

```
.something {
    width: calc(50% + 100px);
    height: calc(var(--some-var) + 100px);
}
```

Цвета

```
.something {
    background: black;              /* 1 */
    background: #000;               /* 2 */
    background: #000000;            /* 3 */
    background: #000000ff;          /* 4 */
    background: rgb(0, 0, 0);       /* 5 */
    background: rgba(0, 0, 0, 1);   /* 6 */
    background: hsl(0deg, 0%, 0%); /* 7 */
}
```

Сокращённые свойства

```
.something {
    margin: 10px;

    /* -> */

    margin-top: 10px;
    margin-right: 10px;
    margin-bottom: 10px;
    margin-left: 10px;
}
```

Сокращённые свойства

```
.something {
    margin: 10px;                   /* all */
    margin: 10px 20px;              /* top/bottom left/right */
    margin: 10px 20px 15px;         /* top left/right bottom */
    margin: 10px 20px 15px 25px;    /* top right bottom left */
}
```

Сокращённые свойства

Особенно внимательно с background и font!

Вендорные префиксы

```
.something {
    -webkit-user-select: none; /* Safari, Chromium <= 53 */
    -moz-user-select: none;    /* Firefox <= 68 */
    -ms-user-select: none;     /* IE 10-11 */
    user-select: none;
}
```

Директивы (правила, at-rule)

- @import
- @media
- @supports
- @keyframes
- @font-face
- И другие

@import

```
@import "some.css";
```

@font-face

```
@font-face {
    font-family: "YS Text";
    src: ...;
}
```

Видимость элементов

Скрытие элемента

```
/* Элемент не виден на экране и не занимает место */
/* С ним нельзя взаимодействовать */
.something {
    display: none;
}
```

Скрытие элемента

```
/* Элемент не виден на экране, но занимает место */
/* С ним нельзя взаимодействовать */
.something {
    visibility: hidden;
}
```

Скрытие элемента

```
/* Элемент не виден на экране, но занимает место */
/* С ним можно взаимодействовать */
.something {
    opacity: 0;
}
```

Позиционирование

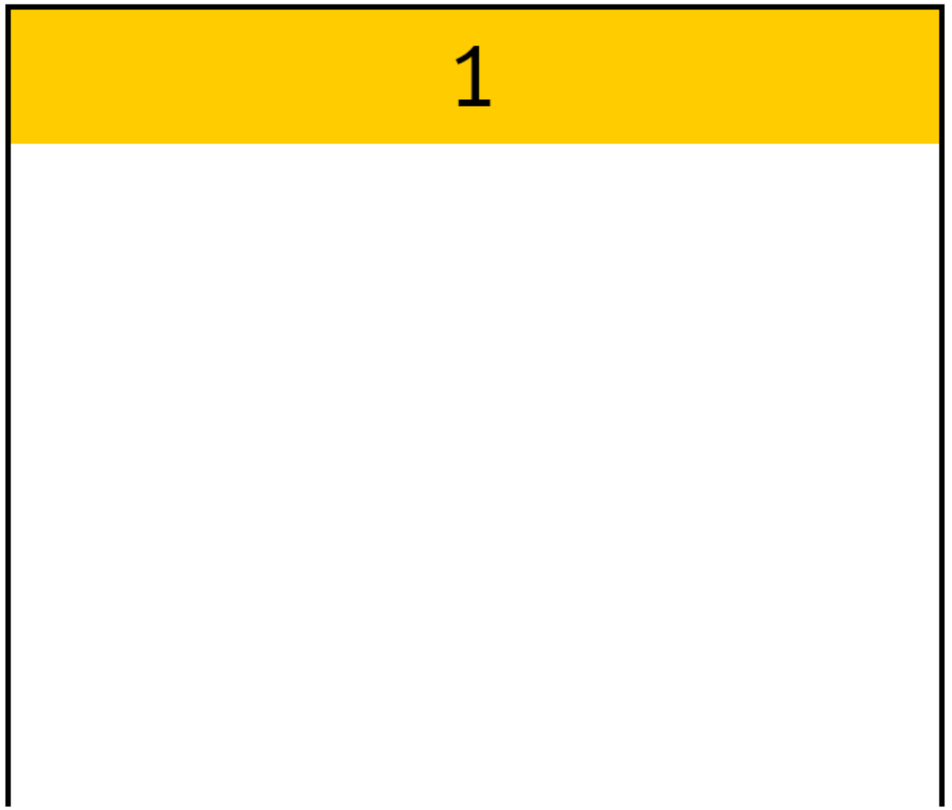
Поток

- Нормальный поток
- Поток флоатов
- Flexbox
- Сетка (grid)

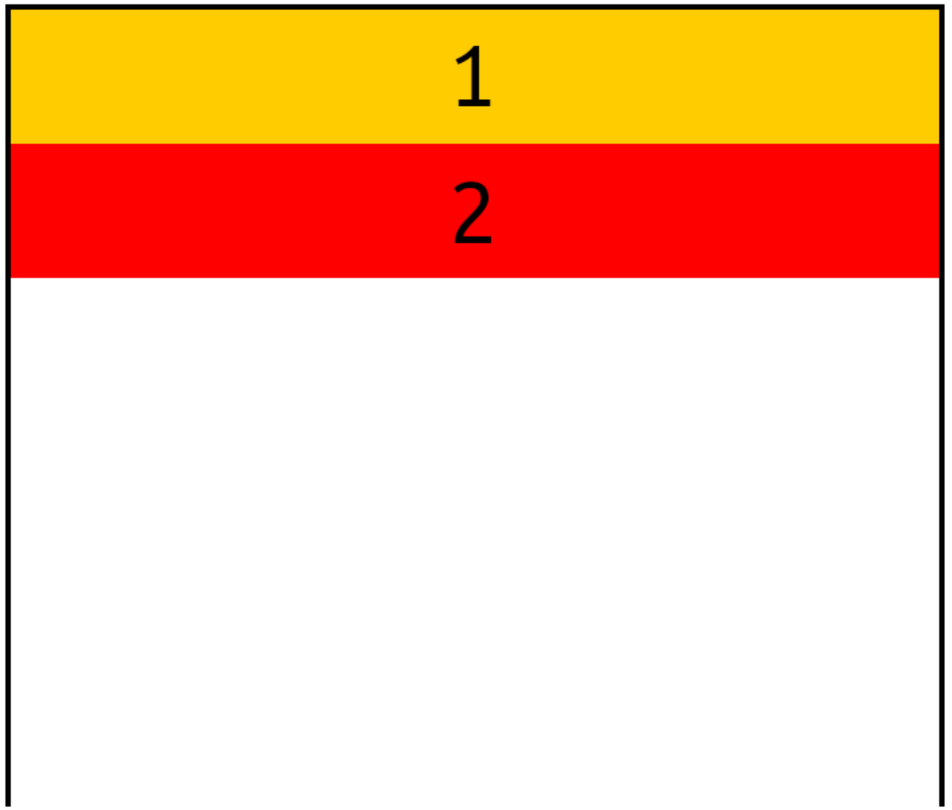
Нормальный поток

- Зависит от направления письма
- Опирается на содержащий элемент
- Располагает блочные элементы друг за другом внутри содержащего элемента

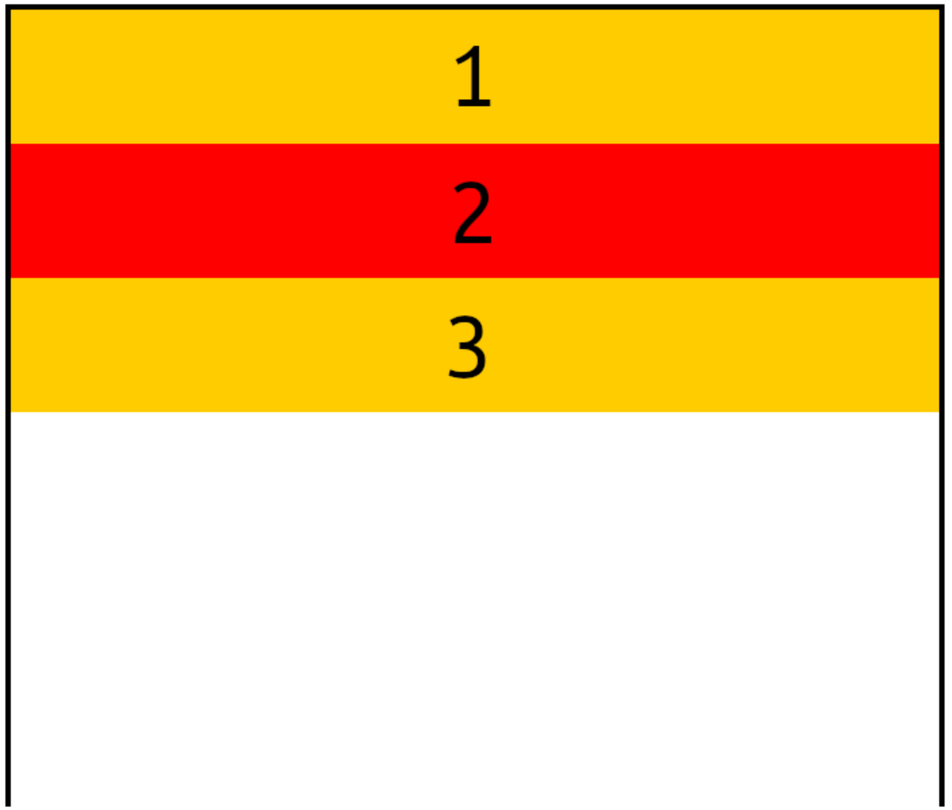
Содержащий элемент



Добавим содержимое



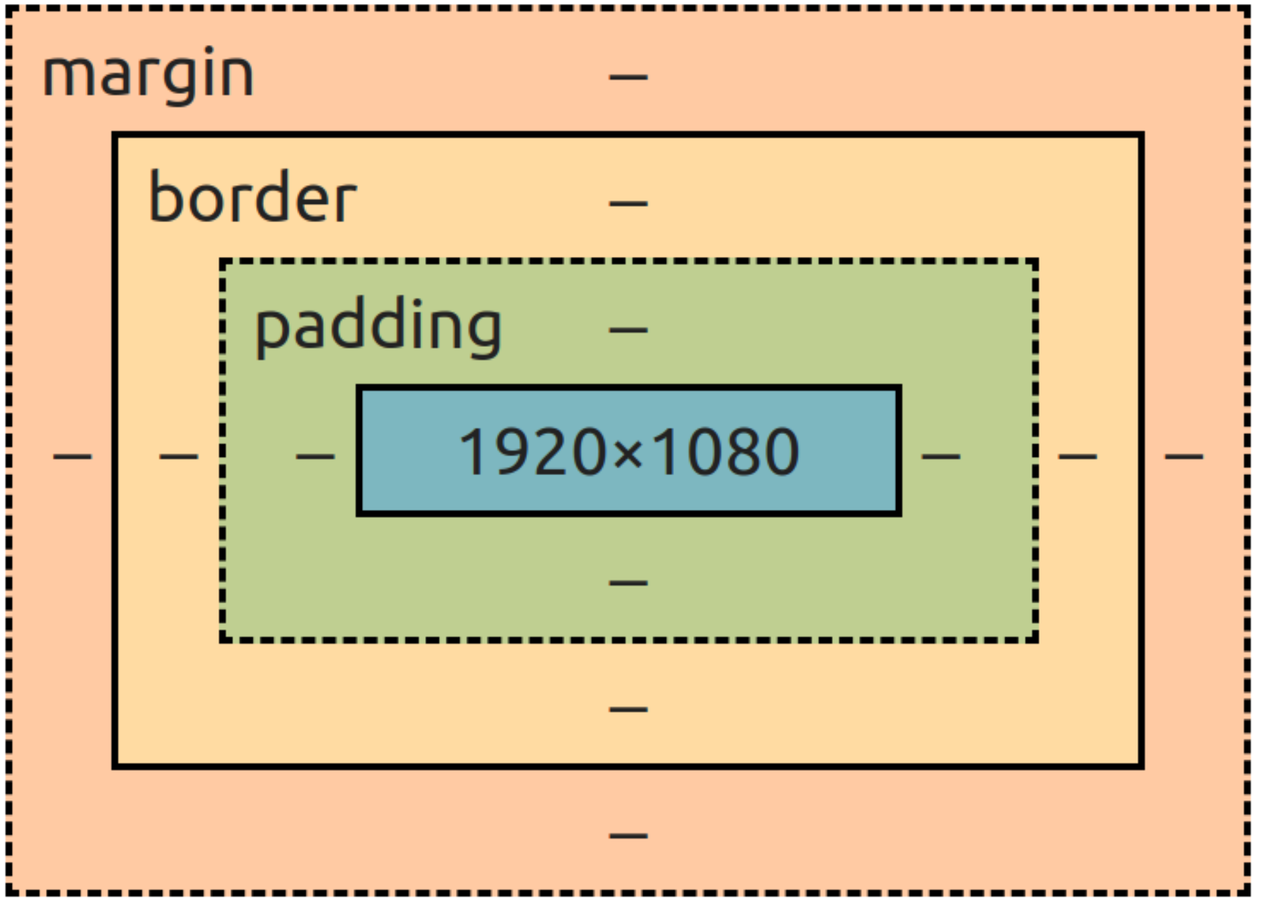
Добавим содержимое



Добавим содержимое

Нормальный поток

- Элементы занимают доступную им ширину
- Элементы занимают определённую высоту и следующие элементы сдвигаются вниз на это значение



Блочная модель

Блочная модель

- width, height
- padding
- border
- margin

Расчёт размеров в нормальном потоке

```
ширина = ширина содержащего блока - margin-left - margin-right
```

Расчёт размеров в нормальном потоке

```
ширина = ширина содержащего блока - margin-left - margin-right
```

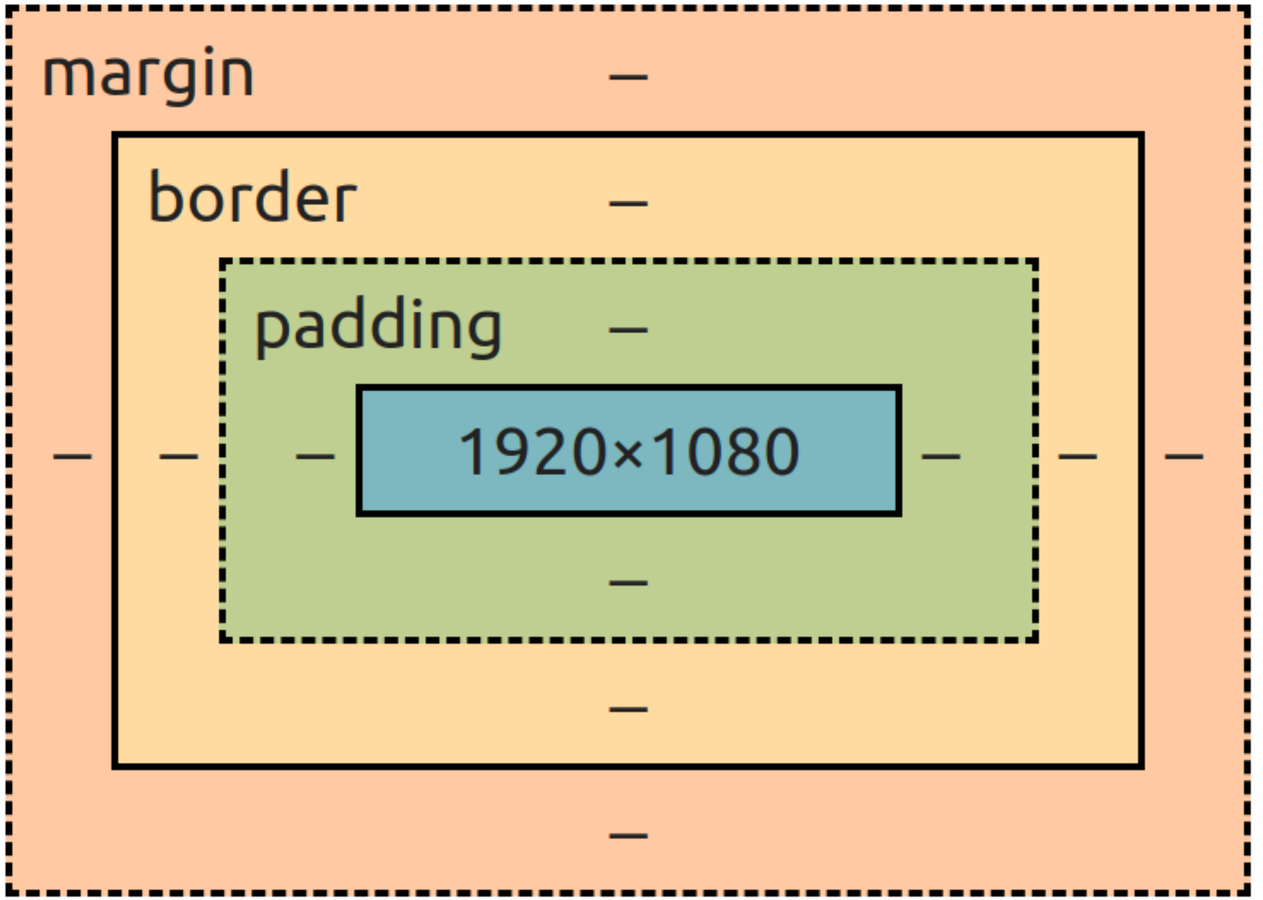
Что такое "ширина элемента"?

1. width задает ширину без учёта padding и border
2. width задает итоговую ширину

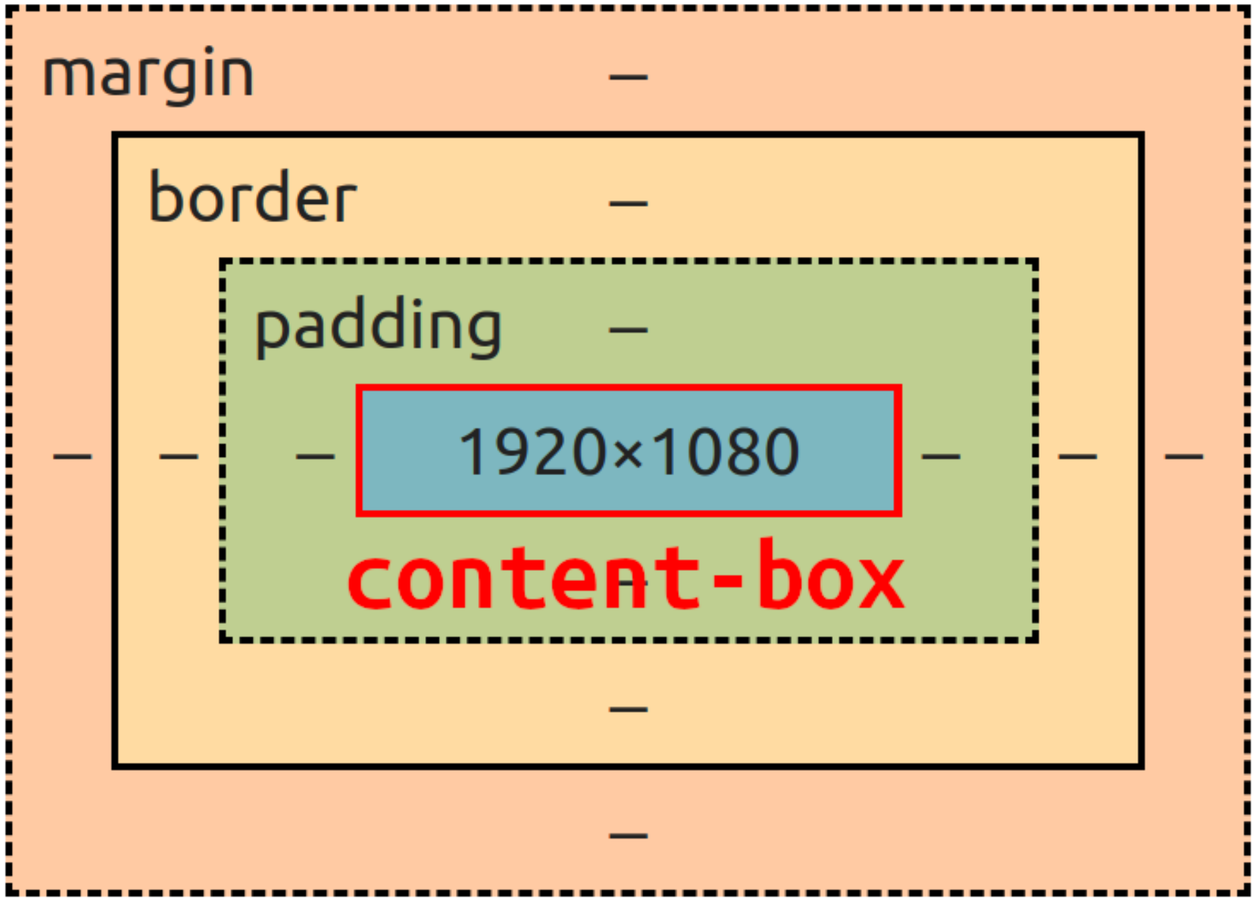
Что такое "ширина элемента"?

1. width задает ширину без учёта padding и border content-box
2. width задает итоговую ширину border-box

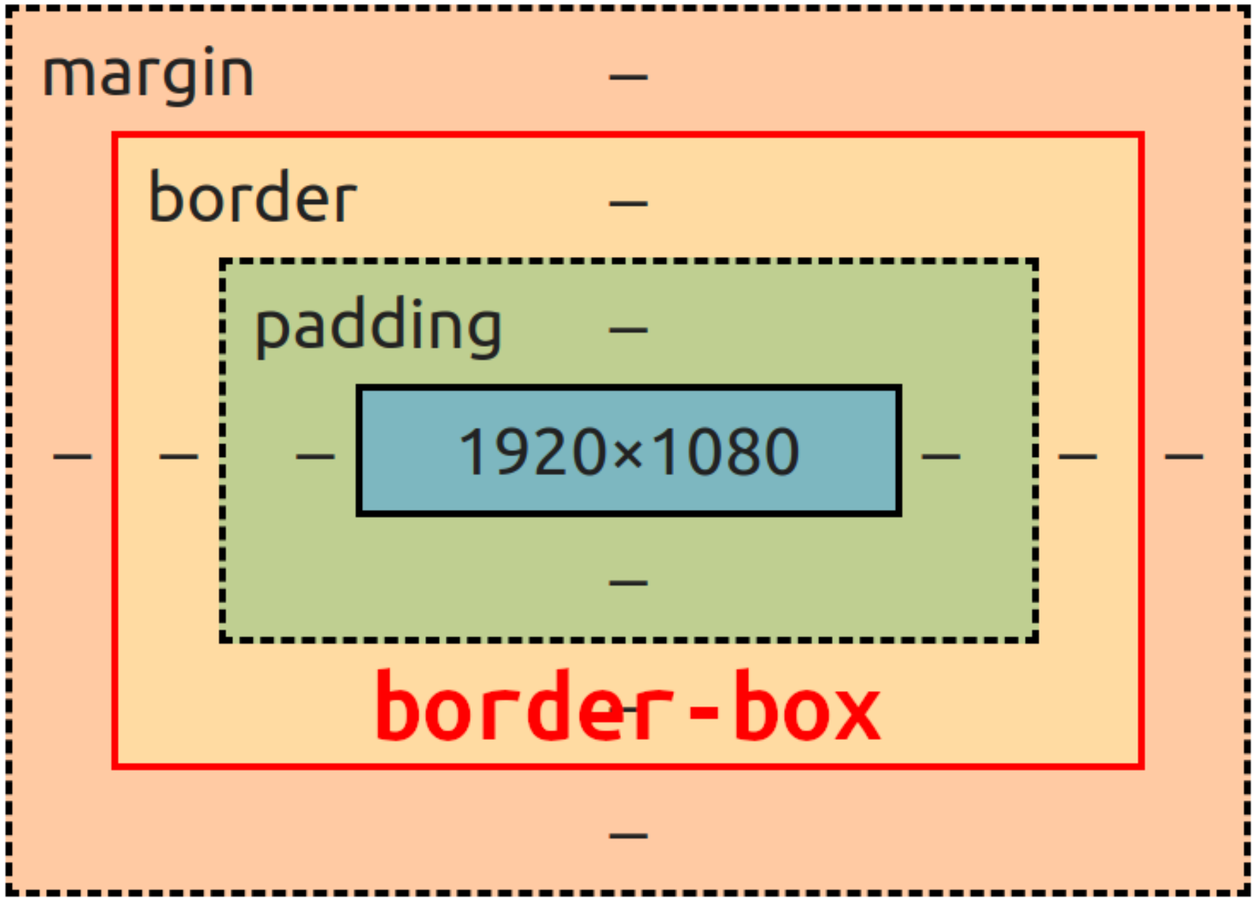
box-sizing позволяет задавать режим, в котором работают свойства width и height



Блочная модель



Блочная модель



Блочная модель

content-box

```
ширина + border-left + padding-left + border-right + padding-right =
ширина содержащего блока - margin-left - margin-right
```

Отступы и границы

- padding может быть равен 0 или быть положительным числом
- border может быть равен 0 или быть положительным числом
- margin может быть 0, положительным числом, отрицательным числом или auto

Расчёт размеров

`ширина` + border-left + padding-left + border-right + padding-right =
ширина содержащего блока - `margin-left` - `margin-right`

Расчёт размеров

`ширина` + 0 + 0 + 0 + 0 = 300 - `margin-left` - `margin-right`

Ширина и margin равны auto

`ширина` = 300 - `margin-left` - `margin-right`

Ширина и margin равны auto

`300` = 300 - `0` - `0`

Ширина задана явно, а margin равны auto

`200 = 300 - margin-left - margin-right`

Ширина задана явно, а margin равны auto

```
200 = 300 - 50 - 50
```

Ширина - известна, margin-left равен auto, margin-right равен 0

```
200 = 300 - 100 - 0
```

margin и padding могут быть заданы в процентах

```
.something {
    margin: 10%;
    padding: 10%;
}
```

margin и padding могут быть заданы в процентах

И вертикальные, и горизонтальные margin/padding считаются от ширины содержащего элемента!

margin "вываливаются"

margin влияет на расположение соседних элементов, однако в нормальном потоке родительский элемент считает свои размеры без учёта margin дочерних элементов. Тем самым margin дочерних элементов могут "вываливаться" за пределы родителя

Этого не происходит, если с соответствующей стороны у родителя есть padding или border

Схлопывание margin

Вертикальные margin у соседних элементов могут быть "объединены" в один отступ. Его размер будет равен наибольшему из двух отступов

Чаще всего этого проще избежать с помощью flexbox

Строчный (инлайновый) контекст

Строчный контекст

Самым простым примером строчного контекста является параграф текста. Элементы располагаются внутри строки, пока хватает места, затем происходит перенос

Строчный контекст

В строчном контексте часть свойств не имеют смысла и не работают:

- width, height
- margin-top, margin-bottom

Строчный контекст

Элементами строки могут быть не только текст, но также замещающие элементы (например, <img>) и блочные элементы

`display: inline-block` позволяет расположить элемент внутри строки, однако сами размеры можно задать с помощью блочных свойств

Расположение элементов внутри строки

text-align позволяет задать горизонтальное расположение элементов внутри строки

Расположение элементов внутри строки

line-height позволяет задать высоту прямоугольника для элемента внутри одной строки

Расположение элементов внутри строки

line-height часто используют для выравнивания элементов по вертикали. Однако это не всегда удачный выбор (например, если текст займёт две строки, то вся идея нарушится). Кроме того, для line-height нельзя задать значение 100%.

Расположение элементов внутри строки

vertical-align позволяет задать вертикальное расположение прямоугольника для элемента внутри строки. vertical-align не работает в блочном контексте!

baseline

top

Расположение элементов внутри строки

vertical-align может принимать значение в пикселях. А значение middle - на самом деле, не геометрическая середина

Часто инлайновые элементы располагают внутри блочных, и сверху образуется дополнительный отступ. Его можно убрать с помощью vertical-align: top

Пробелы между тегами

В случае обычного написания html часто бывает так, что в разметку встраиваются пробелы и переносы. В этом случае они могут начать занимать место, что часто бывает нежелательным

```
<button>
    Профиль
</button>
<button>
    Выход
</button>
```

Как вариант, можно изменить строчный контекст на flexbox.

flexbox

flexbox

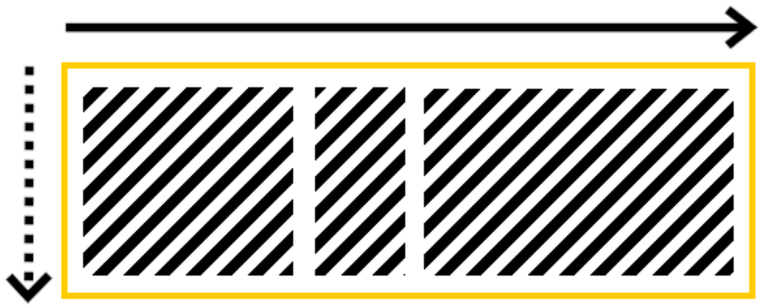
flexbox позволяет разместить элементы в строку или в столбец, имеет возможности по выравниванию, переносам, а также настраиваемые правила по поведению элементов в случае нехватки или избытка места

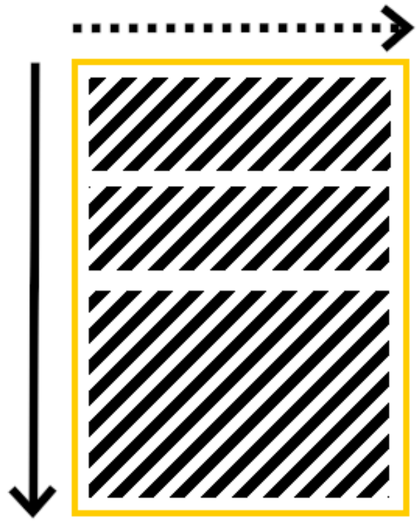
Во флексбоксах не вываливаются margin, не работает margin collapse, а пробельные символы между тегами не влияют на раскладку.

flexbox может быть использован в большинстве случаев, которые встречаются в жизни

flexbox

Для flex-раскладки очень важны две оси: основная и дополнительная, они перпендикулярны друг другу. Основная может быть как горизонтальной, так и вертикальной.





flexbox

```
.something {
    display: flex;
    flex-direction: column; /* row */
}
```

flexbox

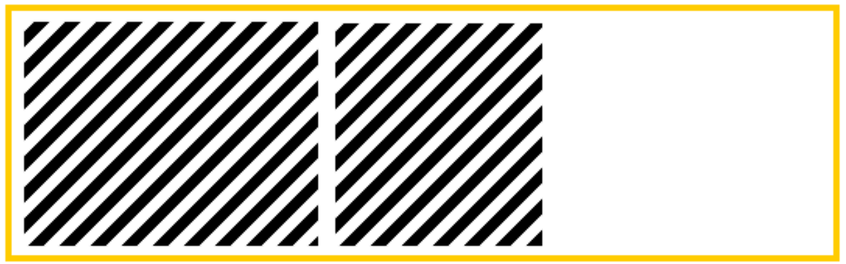
По умолчанию, дочерние элементы располагаются в одной строке или колонке, однако это можно изменить.

```
.something {
    display: flex;
    flex-wrap: wrap;
}
```

В целом, разбиение элементов на строки напоминает строчный контекст, так что дальше будем рассматривать расположение элементов внутри одной строки / колонки.

Размеры элементов вдоль основной оси

Каждый элемент имеет базовый размер, минимальный размер, а также факторы увеличения и уменьшения. Если сумма базовых размеров меньше, чем доступное расстояние, то элементы, имеющие фактор увеличения, увеличиваются. И наоборот: если места не хватает, то элементы, имеющие фактор уменьшения, уменьшаются.



Размеры элементов вдоль основной оси

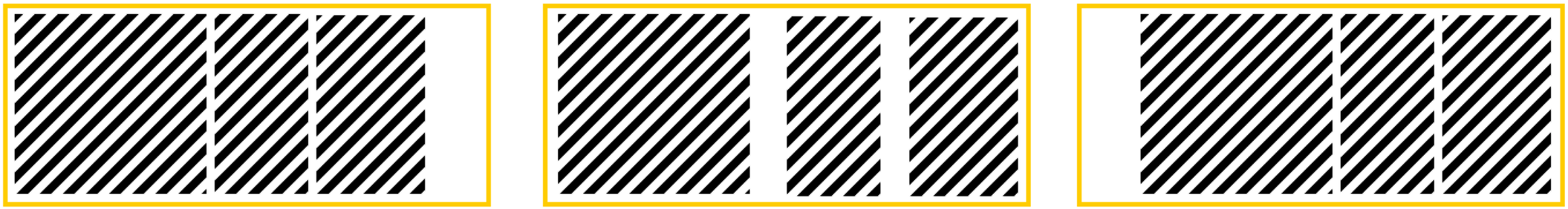
Базовый размер - flex-basis, а факторы настраиваются через flex-grow / flex-shrink. Всё вместе может быть представлено одним сокращённым свойством flex.

Важный момент - элементы по умолчанию имеют минимальный размер, который равен размеру содержимого. Это может препятствовать желаемому уменьшению элемента в некоторых случаях.

Расположение элементов вдоль основной оси

Если после размещения элементов осталось пустое место, то есть несколько возможностей.

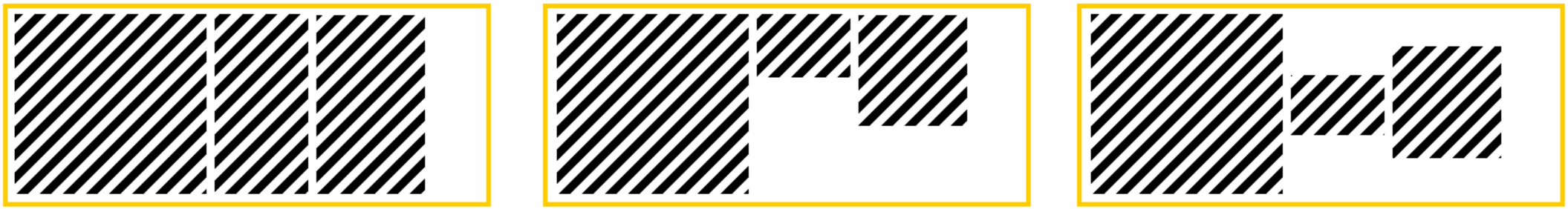
margin со значением auto вдоль основной оси позволяет заполнить всё свободное место. Если такого отступа нет, то в дело вступает свойство justify-content.



Размещение вдоль побочной оси

По умолчанию, элементы занимают всё пространство вдоль побочной оси. Однако это поведение можно настраивать с помощью свойства align-items (и align-self у отдельных элементов).

У align-items есть несколько вариантов размещения: занять всё пространство, расположить по центру или около одного из краёв и другие.



Размеры и размещение отдельных строк

Также можно влиять на размещение самих строк. Для этого можно использовать свойство align-content.

Итого: justify-content влияет вдоль основной оси, align-items и align-content влияют вдоль побочной оси.

grid

grid

Сетка (grid) обладает богатыми возможностями расположения элементов: расположение по (собственно) сетке, правила работы столбцов и колонок, их размеров и выравнивания, объединения ячеек, и многими другими.

Подробно рассмотреть сетку сейчас мы не успеем, однако это хороший и практичный способ сделать более сложное расположение элементов, чем через flexbox.

Материалы

- MDN
- Grid Garden

Дополнительное позиционирование

Дополнительное позиционирование

Свойство position позволяет дополнительно влиять на расположение элементов.

position: relative в сочетании со свойствами top / right / bottom / left позволяет задавать смещение элемента от той позиции, что он бы занял изначально.

position: absolute

Абсолютное позиционирование позволяет расположить элемент над другими. При этом в качестве опорного элемента используется первый найденный родительский элемент, имеющий position, не равный значению static. В этом случае свойства top / right / bottom / left задают позицию элемента.

position: relative часто используется без top / right /bottom /left, чтобы задать опорный элемент.

position: fixed

Очень похоже на абсолютное позиционирование, только позиционирование идёт относительно вьюпорта браузера, а не конкретного элемента.

Порядок наложения элементов

Если подходить просто, то свойство z-index позволяет задать порядок наложения элементов. Чем число больше - тем "позднее" будет нарисован слой, "позже" других слоёв.

Однако полные правила гораздо сложнее и могут быть найдены по словам Stacking context (контекст наложения). Вот хорошая статья на эту тему: https://doka.guide/css/stacking-context/

Устаревшие раскладки

Таблицы (не путать с grid)

Таблицы использовались раньше, когда требовалось расположить элементы в колонке, строчке или по таблице. В зависимости от задачи сейчас таблицы можно заменить на flexbox или grid.

Таблицы всё ещё имеют смысл в случае вывода табличных данных и не стоит их все поголовно менять на сетки. Таким образом, таблицы прошли путь от инструмента представления данных до раскладки, а затем вернулись к истокам :)

Флоаты (float)

Первоначально давали возможность расположить в параграфе текста какой-то элемент (например, картинку), чтобы текст "обтекал" этот элемент. Затем флоаты стали использовать для расположения элементов. Сейчас такие раскладки можно делать с помощью flexbox, а флоатам оставить исходную задачу: обтекание текста.

Советы

Разбираться в непонятных моментах

Если вы видите, что что-то работает неожиданным образом - это повод разобраться. Если вы не знаете, как свёрстан тот или иной сайт - всегда можно изучить :)

Писать CSS в едином стиле с небольшой вложенностью

Чем меньше вложенность, тем меньше вероятность, что для переопределения вам потребуется эту вложенность повторить или "усилить" селектор. Это может очень быстро выйти за рамки понятного.

Разбивать код на файлы и независимые компоненты

Бывает, что CSS пишут в одном файле и главное, чтобы работало. Но чаще всего в этом коде может разобраться только его автор, а через некоторое время не может и он сам. Поэтому любой код, который будет жить хоть какое-то время, желательно сразу писать читаемым.

Придумать общие правила для z-index

Часто z-index пишут по принципу "у меня новый компонент, он точно должен быть выше всех". Это приводит к значениям, растущим на каждом шаге на порядок и больше, всё приходит в хаотическое состояние. Гораздо логичнее сделать единое место со всеми значениями и иметь их все перед глазами. Например, таким местом может являться объявление переменных.

Невозможные условия возможны

"Точно рабочее свойство" может не поддерживаться в каких-то браузерах. А точно помещающееся слово перестанет помещаться при локализации на какой-то язык. А у какого-то пользователя в браузере включён зум. Подумайте о таких ситуациях наперёд.

Материалы

Материалы

- MDN
- Дока
- Can I Use
- Web Dev

Контакты

Александр Нефедов

 4eb0da@yandex-team.ru

 4eb0da