21 Способ выровнять div по горизонтали

Во frontend-разработке неизбежное зло – это вёрстка. Насколько бы навороченными не были фреймворки для построения UI, рано или поздно возникнет необходимость в вёрстке. Да, и жизнь осложняется тем, что в CSS нет полифиллов, как таковых.

В этой заметке разберём один популярный вопрос на собеседовании. Встретить в Сети полный перечень всех способов, с учётом современных возможностей, непросто. Тем не менее, они так или иначе присутствуют на просторах.
Для всех нетривиальных способов даны ссылки, которые помогут в изучении той или иной технологии.
Поэтому данный материал будет полезен не только новичкам, но и тем, кто хочет пополнить собственную коллекцию приёмов.

Вопрос в студию

Есть два div-а:

```
<div class="parent">
  <div class="child">
  </div>
</div>
```

Назовите все способы выровнять внутренний `.child` по центру по горизонтали относительно `.parent`. Вот так

Ссылка на эту картинку в CodePen.
Число способов, которые предоставит соискатель, неплохо коррелирует с опытом в вёрстке.
Все приведённые приёмы доступны и здесь.

Вариант #01 - margin

Первое что приходит в голову (в пикселях):

```
.parent {
  width: 400px;
}
.child {
  width: 200px;
  margin-left: 100px;
}
```

Ссылка на CodePen.
Плюсы и минусы:

- Это просто
- Здесь необходимо, чтобы были заданы размеры div-ов
- Соответственно, необходимо заранее вычислить размеры div-ов и margin
- И, конечно, это никак не responsive

То же самое в процентах:

```
.parent {
  width: 400px;
```

```
}
.child {
  width: 50%;
  margin-left: 25%;
}
```

Ссылка на CodePen

Плюсы и минусы:

- По сути, то же самое что и первое
- Но уже лучше - вычислять абсолютные размеры `.child` div-а и margin мы заставляем браузер.
- Ну хоть какое-то responsive

Вариант #02 - padding и box-sizing

```
.parent {
  width: 400px;
  padding-left: 100px;
  box-sizing: border-box;
}
.child {
  width: 200px;
}
```

Ссылка на CodePen

С padding-ами не так просто - именно теми пикселями можно задать, только зная об одной особенности.

По умолчанию, у .parent стоит `box-sizing: content-box`, и это означает, что `width` задаёт размер контента без учёта padding. Т.е. размер `.parent` будет в данном случае `500px`. Подробнее про box-sizing и CSS Box Model.

Плюсы и минусы:

- В принципе, все предыдущие плюсы и минусы имеют место.
- Да, и `box-sizing` это CSS3

Вариант #03 - padding-и

Конечно, зная об этом, можно достаточно просто написать вот это:

```
.parent {
  width: 200px;
  padding-left: 100px;
  padding-right: 100px;
}
.child {
  width: 100%;
}
```

Ссылка на CodePen

Плюсы и минусы:

- В принципе, все предыдущие комментарии имеют здесь место
- А вот полный размер контейнера уже не так тривиально определить/задать

Вариант #04 - transparent border

```
.parent {
  width: 200px;
  border-left: 100px solid transparent;
  border-right: 100px solid transparent;
}
.child {
  width: 100%;
}
```

Ссылка на CodePen

Стоит отметить, что для правильного варианта обязательно задавать border-style (solid - один из вариантов, главное, что не дефолтный none), ну и, конечно, прозрачный цвет transparent.

Плюсы и минусы:

- Оригинально
- Но сложно для чтения и понимания.

Вариант #05 - margin auto

А вот это, первое придёт на ум чуть более опытному верстальщику

```
.parent {
  width: 400px;
}
.child {
  width: 200px;
  margin-left: auto;
  margin-right: auto;
}
```

Ссылка на CodePen

Да, это так аккуратно записали `margin: 0 auto` 🙂 Просто мы ничего не знаем о верхней границе, и не задаём 0.

Плюсы и минусы:

- Это очень просто и популярно
- margin считает сам браузер
- Нужно задавать явно размер div-а явно. Если этого не сделать, то auto будет равно 0.

Вариант #06 - С помощью CSS-препроцессоров

Если имеются CSS-препроцессоры (скажем, Less), то вычисления соответствующих границ можно переложить на них. Не забудем про эту возможность:

```
@parent-width: 400px;
@child-width: 200px;
@child-margin: (@parent-width - @child-width) / 2;

.parent {
  width: @parent-width;
}
```

```
.child {
  width: @child-width;
  margin-left: @child-margin;
}
```

Ссылка на CodePen
Это Less, а ещё есть SASS, SCSS, PostCSS и другие.
Плюсы и минусы:

- В итоговой CSS это будут всё те же пиксели 🙂

Вариант #07 - CSS-переменные и calc

Мало кто знает, но и в CSS есть экспериментальные переменные, а ещё и оператор calc

```
:root {
  --parent-width: 400px;
  --child-width: 200px;
}

.parent {
  width: var(--parent-width);
}
.child {
  width: var(--child-width);
  margin-left: calc((var(--parent-width) - var(--child-width))
/ 2);
}
```

Ссылка на CodePen
Про CSS-переменные можно прочитать здесь, а про calc - здесь
Ну и, конечно, стоит ознакомиться с поддержкой - здесь и здесь, соответственно.

Вариант #08 - position: relative

```
.parent {
  width: 400px;
}
.child {
  position: relative;
  width: 200px;
  left: 100px;
}
```

Ссылка на CodePen

Здесь, в принципе, всё понятно. Для тех, кто впервые видит position.

Вариант #09 - position: absolute

А вот про это стоит спросить отдельно.

```
.parent {
  position: relative;
  width: 400px;
}
.child {
  position: absolute;
  left: 100px;
  right: 100px;
}
```

Ссылка на CodePen

Этот вариант достаточно хорошо коррелирует с опытом вёрстки. Предлагаем читателям самостоятельно разобраться, почему необходимо указать `position: relative` у parent-а.

Плюсы и минусы:

- Да, здесь не задаётся ширина у `.child`.

Вариант #10 - translateX(-50%)

```
.parent {
  width: 400px;
}
.child {
  position: relative;
  width: 200px;
  left: 50%;
  transform: translateX(-50%)
}
```

Ссылка на CodePen

Подробнее с возможностями transform можно ознакомиться здесь.

Плюсы и минусы:

- Не самый очевидный трюк 🙂
- Хотя довольно встречается на просторах Сети.

Вариант #11 - эклектика

На просторах Сети можно встретить и такого монстра:

```
.parent {
  position: relative;
  width: 400px;
}
.child {
  position: absolute;
  width: 200px;
  left: 50%;
  margin-left: -100px;
}
```

Ссылка на CodePen.

Предлагаем читателям самостоятельно разобраться, что здесь происходит, и почему это не самый лучший пример 🙂 Да, здесь `100px = ширина .child / 2`

Вариант #12 - inline-block + text-align

Вот этот вариант коррелирует со знаниями о блочных и inline элементах

```
.parent {
  width: 400px;
  text-align: center;
}
.child {
  display: inline-block;
  width: 200px;
}
```

Ссылка на CodePen
В этом примере блочный элемент div мы сделали строчным, а дальше он выравнивается как обычный текст.
Подробнее о блочных и строчных элементах.

Вариант #13 - display: table

Любители верстать всё в таблицах могут применить свои навыки и для div-ов:

```
.parent {
  display: table;
  width: 400px;
  border-collapse: separate;
  border-spacing: 100px 0;
}
```

Ссылка на CodePen
Оставим читателям удовольствие разобраться самостоятельно с этим примером 🙂

Вариант #14 - extra div + float

Если допустить появление дополнительного тега, т.е.:

```
<div class="parent">
```

```
  <div class="extra">
  </div>
  <div class="child">
  </div>
</div>
```

То можно расположить элементы с помощью float:

```
.parent {
  width: 400px;
}
.extra {
  width: 100px;
  float: left;
}
.child {
  width: 200px;
  float: left;
}
```

Ссылка на CodePen.

Подробнее о float можно прочитать есть прекрасная статья.

Плюсы и минусы:

- Это нечестно добавить один элемент
- Да и разметку следует держать чистой
- Но это очень просто 🙂

Вариант #15 - псевдоэлементы + float

Как ни странно, но "дополнительные" теги уже есть. Называются они псевдоэлементами. Их можно использовать с тем же успехом:

```
.parent {
  width: 400px;
}
.parent:before {
```

```
  content: '';
  width: 100px;
  float: left;
}
.child {
  width: 200px;
  float: left;
}
```

Ссылка на CodePen.

У каждого элемента есть условные два внутренних - `:before` и `:after` псевдоэлементы. Это очень мощный инструмент в вёрстке, позволяющий держать разметку чистой.

По сути дела, разметка теперь выглядит такой:

```
<div class="parent">
  <div class="parent:before">
  </div>
  <div class="child">
  </div>
</div>
```

Чтобы этот элемент отобразился, необходимо задать ему `content`.

Подробнее про псевдоэлементы.

Плюсы и минусы:

- Экономия разметки!
- Но это float 🙂 Сейчас уже почти полностью вытеснено flexbox-ами.

Вариант #16 - псевдоэлементы + text-align

С учётом всего перечисленного, этот вариант очевиден:

```
.parent {
  width: 400px;
  /* text-align: left; */
```

```
  font-size: 0;
}
.parent:before {
  content: '';
  display: inline-block;
  width: 100px;
}
.child {
  display: inline-block;
  width: 200px;
}
```

Ссылка на CodePen.
И предлагаем читателям самостоятельно разобраться, почему здесь присутствует `font-size: 0`

Вариант #17 - flexbox

```
.parent {
  width: 400px;
  display: flex;
  justify-content: center;
}
.child {
  width: 200px;
}
```

Ссылка на CodePen
Этот вариант предлагают опытные верстальщики.
Про flexbox-ы написано масса статей. Для начинающих, предлагаем ознакомиться вот с этим.
Вариант с `display: -webkit-box` можно считать эквивалентным.
Плюсы и минусы:

- Ничего не нужно задавать у внутреннего элемента

- Ну и если есть другие элементы - одно удовольствие их выравнивать flex-ом
- Конечно, сейчас это менее актуально, но помним про поддержку.

Вариант #18 - flex-direction: column

Как дополнение к предыдущему, раскрывает Ваши возможности с flexbox-ами:

```
.parent {
  width: 400px;
  display: flex;
  flex-direction: column;
  align-items: center;
}
.child {
  width: 200px;
}
```

Ссылка на CodePen.

Плюсы и минусы:

- Нужен, собственно, когда у Вас `flex-direction: column` 🙂

Вариант #19 - flex + space-between

С flex-ами можно творить чудеса, например очень ровно выровнять набор элементов.

```
.parent {
  width: 400px;
  display: flex;
  justify-content: space-between;
}
.parent:before, .parent:after {
  content: '';
}
.child {
```

```
  width: 200px;
}
```

Ссылка на CodePen
Можно считать, что варианты с `space-around`, `margin: auto` эквивалентны данному.

Вариант #20 - CSS Grid

CSS grid - это современный и самый мощный, в то же время простой и гибкий инструмент для организации сеток и выравнивания.

```
.parent {
  width: 400px;
  display: grid;
  grid-template-columns: 1fr 200px 1fr;
}
.child {
  grid-column: 2 / 3
}
```

Ссылка на CodePen
Здесь организуется сетка из трёх колонок (с автоматическим разбиением первой и последней). Сам .child располагается между 2ой и третьей линий сетки (т.е. во второй колонке).
Это далеко не единственный способ выровнять внутренний див по центру с помощью CSS Grid.
Рекомендуем читателям ознакомиться.
Плюсы и минусы:

- Современно
- Со временем станет менее актуально, но всё же.

Вариант #21 - JS

С помощью JS можно сделать то же самое 🙂 Это, безусловно считается моветоном, но знать про это нужно:

```
const updateChildWidth = () => {
  const parent = getComputedStyle(document.getElementsByClassN
ame('parent')[0]);
  const parentWidth = +parent.width.replace('px', '');
  const margin = 100;
  const childWidth = parentWidth - 2 * margin;
  console.log(childWidth);
  const child = document.getElementsByClassName('child')[0];
  child.style.width = `${childWidth}px`;
  child.style.marginLeft = `${margin}px`;
}
updateChildWidth();

window.onresize = updateChildWidth;
```

Ссылка на CodePen

Да, обратите внимание на последнюю строчку. Вычислить один раз размер div - недостаточно. Некоторая категория пользователей (к которым и относится автор) изменяет размер окна после открытия. Поэтому обновление размеров - тоже немаловажная часть.

Плюсы и минусы:

- Делать с помощью JS, то что можно сделать с помощью CSS - это моветон
- Может быть полезно в совсем сложных случаях.

В принципе и всё🙂 Надеюсь, подобный перечень способов натолкнёт на более подробное изучение некоторых возможностей CSS.

Если есть какие-то добавления - ждём Ваших комментариев 🙂