Css grid

part 2

grid template areas

С помощью свойства grid-template-areas (у контейнера) можно описать шаблон сетки задав имена областей с помощью свойства grid-area и с ссылаясь на них. Повторение названия области приводит к тому, что содержимое охватывает эти ячейки. ***Точка означает пустую ячейку.*** С помощью синтаксиса можно визуализировать структуру сетки.

grid template areas

Создадим разметку контейнера, в котором будут такие блоки: сайдбар, контент, сайдбар и футер.

После чего обозначим сетку.

```
.container {
    display: grid;
    grid-template-columns: 1fr 10fr 1fr;
    grid-template-rows: 150px 150px 100px;
    grid-template-areas:
      "sidebar-1  content   sidebar-2"
      "sidebar-1  content   sidebar-2"
      "footer     footer    footer"
  }
```

grid template areas

мы создали сетку 3 на 3. После чего объединим ячейки по именам в grit-template-areas. В этом свойстве каждое из значений - набор наименований (можно называть как угодно, главное, чтобы было понятно). В каждых кавычкам мы называем ячейки в колонках (сколько столбцов, столько названий. В то же время,сколько строк мы указали - столько и значений, объединенных в кавычки. Посмотрев в firefox developer tools теперь можно увидеть, что все ячейки имеют название. Все они сгруппированы по названиям.

grid-area

После группировки ячеек по названиям, каждому grid-item-у (дочернему элементу контейнера сетки) мы можем указать свою область (grid-area), в которой он должен разместиться. Значение - одно из имен, которые мы указали в grid-template-areas.

grid-area

Пример:

```
.footer {
    grid-area: footer;
  }
  .item1 {
    grid-area: sidebar-1;
  }
  .item2 {
    grid-area: content;
  }
  .item3 {
    grid-area: sidebar-2;
  }
```

grid-template-areas and media-queries

С помощью медиа запросов нам достаточно менять “шаблон” grid-template-areas и глобальный “каркас” будет меняться, в зависимости от расширения экрана.

grid template areas

Можно указывать grid-template-areas без явного указания grid-template-columns и grid-template-rows. Сетка все равно будет разбита ровно так, сколько мы укажем с помощью grid-template-areas.

+ в свойстве grid-item-а “grid-column\row” мы можем ссылаться на имя grid-area , чтобы спозиционировать элемент от **начала** до **конца** этой области, например, если grid-area называется footer, можно указать:

  grid-column: footer-start/footer-end

Наименование линий

Помимо line-start\line-end, которые есть изначально, мы можем указать свои названия линий. Помним, что линии и треки\ячейки - разные вещи.

Вот как дать наименование линиям:

grid-template-columns: [first-side-start] 150px [first-side-end content-start] 1fr [content end] 150px [second-side-end].

После чего в свойстве grid-item-а grid-column, вместо нумерованного указания grid-column-start\end, мы можем ссылаться на названия, которые мы дали линиям:

item2 {

grid-column: content-start;

}

Grid-auto-flow

В первой части рассматривалось свойство контейнера - grid-auto-flow, с помощью которого можно указать, строки или колонки будут создаваться, как implicit tracks (в случае, если это необходимо). Если при сложной раскладке у нас появляются “дыры” в сетке (например, галерея с картинками разных размеров), мы можем попытаться заполнить их со значением dense.Тогда сначала заполняются элементы, которые помещаются в строку\колонку, те items, которые не помещаются, переносятся на новый трек (на предыдущем треке появляется “дыра”, после чего свободное пространство заполняется элементами, следующими за “расширенным” item-ом, которые могут поместиться на свободное место.

Css grids и выравнивание. Items

justify-* - выравнивание по горизонтали

align-* выравнивание по вертикали

justify-items: start/end/center/stretch - выравнивание самих grid-item-ов

justify-items + align-items = place-items (на момент написания презентации - поддержка может быть неполной во всех главных браузерах)

Css grids и выравнивание. Content, self

justify\align-content: start\end\center\space-between\space-around - свойство, которое может пригодиться, если общая ширина\высота всех items в grid контейнере меньше, чем сам контейнер. Тогда всю совокупность (контент) мы можем выровнять внутри родительского элемента.

justify-self\align-self - свойства grid-item-ов, с помощью которых мы можем выравнивать их индивидуально.

Order

Так же, как и в flex-box, css grid поддерживает возможность указывать четкий порядок grid-items с помощью свойства order (хотя мы можем это сделать и с помощью других инструментов, которые есть в css grid)

По умолчанию, у всех элементов значение order - 0. Увеличение значения перемещает элемент к концу сетки.

***Внимание**! Если важна доступность, лучше не использовать это свойство, если оно меняет порядок разметки контента, так как, например скрин-ридеры могут неправильно прочитать содержимое*

Nested grid

Если есть необходимость, любой элемент, который является grid-item-ом (потомком grid-container-а), можно сделать одновременно и grid-container-ом. Делается это точно так же, как и обычно: display: grid.

Варианты использования - если item имеет сложную структуру, которая в свою очередь может быть разбита на сетку.

Так мы можем иметь еще больший контроль над позиционированием.

Css grid VS flex-box

Во многих источниках указывают, что flex-box создан для позиционирования по одной оси, сетка - сразу по 2-м осям.

Однако сетка так же хороша для выравнивания по одной из осей.

Изначальные плюсы flex-box - меньше концептов (проще), лучше поддержка у браузеров.

свойства flex-box (flex-grow, например) могут работать с css transition (по отличию от свойств сетки)

Только с помощью flex-box мы можем с легкостью сделать reverse дочерних элементов контейнера.

Если количество элементов во flex-container-е изменяется - совсем не критично. Когда нам нужно изменить конкретный элемент - мы привязываемся к селектору (тег, класс, id).

Css grid VS flex-box

Css grid, в свою очередь, имеет больше функционала и с помощью этого инструмента можно добиться более сложных результатов с меньшими усилиями. Не нужно так много медиа запросов, чтобы контролировать layout. Более декларативное позиционирование. Четкий контроль над каждым элементом.

Минус - иногда, разметка и сетка разбегаются и результат может быть неожиданным. Так, например, если из разметки исчезнет какой-то элемент, а сетка создавалась с учетом этого элемента , то раскладка может быть нарушена.