4 - Grid Layout Model

linkedin.com/in/barbaraeste/
github.com/babieste/

Introdução

O Grid Layout é um novo modelo de layout para CSS que tem habilidades poderosas para controlar o tamanho e posicionamento de boxes e seus conteúdos. Diferente do Flex Layout Model, que é orientado a um único eixo, o Grid Layout é otimizado para layouts bidimensionais: aqueles que o alinhamento do conteúdo é desejado em ambas as dimensões (vertical e horizontal).

Além disso, devido à sua capacidade de posicionar explicitamente itens na grade, o Grid Layout permite transformações dramáticas sem a alteração do markup. Ao combinar media queries com as propriedades que controlam o layout do contêiner e seus filhos, os autores podem adaptar seu layout às mudanças de tela, preservando uma estruturação semântica ideal de seu conteúdo.

Embora muitos layouts possam ser expressos com grid ou flexbox, cada um deles tem suas especialidades. Grid Layout impõe um alinhamento bidimensional, permite a sobreposição explícita de itens e possui recursos de abrangência mais poderosos. O Flexbox Layout se concentra na distribuição de espaço dentro de um eixo, usa um sistema de quebra de linha baseado no tamanho do conteúdo e depende da hierarquia de marcação subjacente para construir layouts mais complexos.

Conceitos e Terminologia

No grid layout, o conteúdo de um **grid container** é determinado posicionando-o e alinhando-o em um grid. O **grid** é um conjunto de linhas (**grid lines**) horizontais e verticais que se cruzam e dividem o espaço do contêiner em áreas (**grid areas**), nas quais os itens (**grid items**) podem ser colocados. Existem dois conjuntos de grid lines: um conjunto definindo as **colunas** que correm ao longo do block axis e um conjunto ortogonal que define as **linhas** ao longo do inline axis.

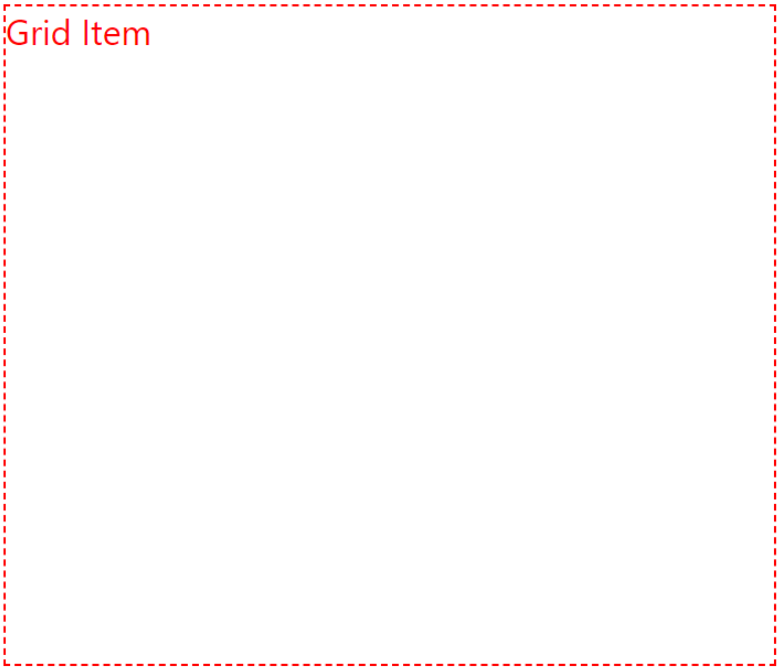
Grid Lines

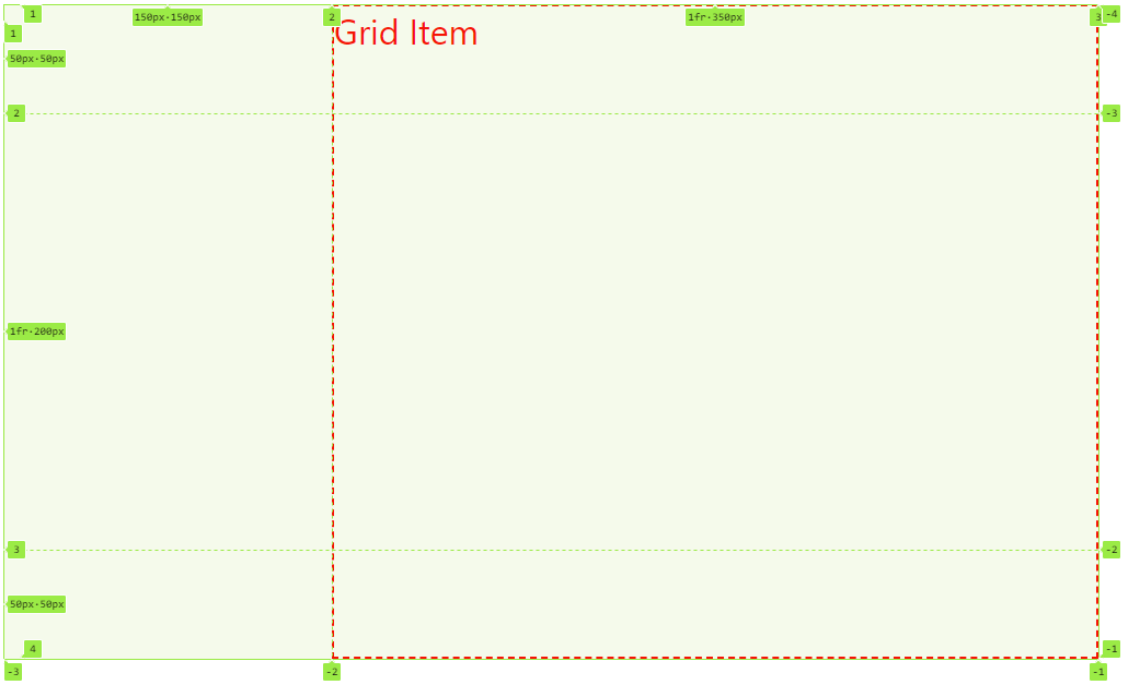
Grid lines são as linhas horizontais e verticais dividindo o grid. Uma grid line existe em ambos os lados de uma coluna ou linha. Estas podem ser referenciadas por um índice numérico ou por um nome especificado pelo autor. Um grid item referencia as grid lines para determinar sua posição no grid, usando as propriedades de posicionamento de grid.

Os seguintes exemplos criam três **column grid lines** e quatro **row grid lines**. O primeiro exemplo demonstra como um autor posicionaria um grid item usando os índices das linhas.

```html
<html>
  <head>
    <style>
      .grid {
        display: grid;
        grid-template-columns: 150px 1fr;
        grid-template-rows: 50px 1fr 50px;
        height: 300px;
        width: 500px;
      }

      .item {
        border: 1px dashed red;
        color: red;
        grid-column: 2;
        grid-row-end: 4;
        grid-row-start: 1;
      }
    </style>
  </head>
  <body>
    <div class="grid">
      <div class="item">Grid Item</div>
    </div>
  </body>
</html>
```

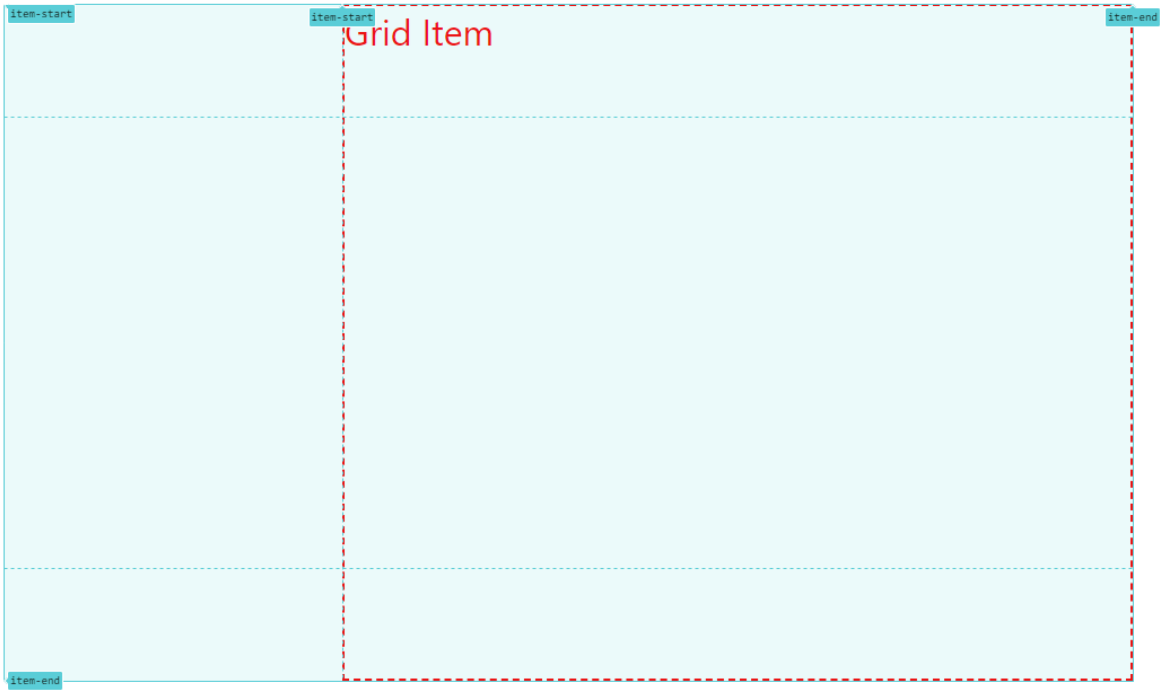




Neste segundo exemplo, o autor utilizou nomes para identificar as grid lines:

```
<html>
  <head>
    <style>
      .grid {
        display: grid;
        grid-template-columns: 150px [item-start] 1fr [item-end];
        grid-template-rows: [item-start] 50px 1fr 50px [item-end];
        height: 300px;
        width: 500px;
      }

      .item {
        border: 1px dashed red;
        color: red;
        grid-column: item-start / item-end;
        grid-row: item-start / item-end;
      }
    </style>
  </head>
  <body>
    <div class="grid">
      <div class="item">Grid Item</div>
    </div>
  </body>
</html>
```



Grid Tracks e Grid Cells

Grid track é um termo genérico para uma **grid column** ou uma **grid row**. Em outras palavras, é o espaço entre grid lines adjacentes. Cada grid track recebe uma **função de dimensionamento**, que controla quão alta ou larga a coluna ou linha pode crescer, e consequentemente quão distante são as grid lines que as cercam.

Uma **grid cell** é a intersecção de uma grid row com uma grid column. É a menor unidade da grid que pode ser referenciada ao posicionar grid items.

Grid Area

Uma **grid area** é o espaço lógico utilizado para posicionar um ou mais grid items. Uma grid area consiste de uma ou mais grid cells adjacentes. É limitada por quadro grid lines, uma em cada lado da grid area, e participa no tamanho das grid tracks que cruza. Uma grid area pode ser nomeada utilizando a propriedade '**grid-template-areas**', ou referenciada implicitamente pelas grid lines que a delimitam.

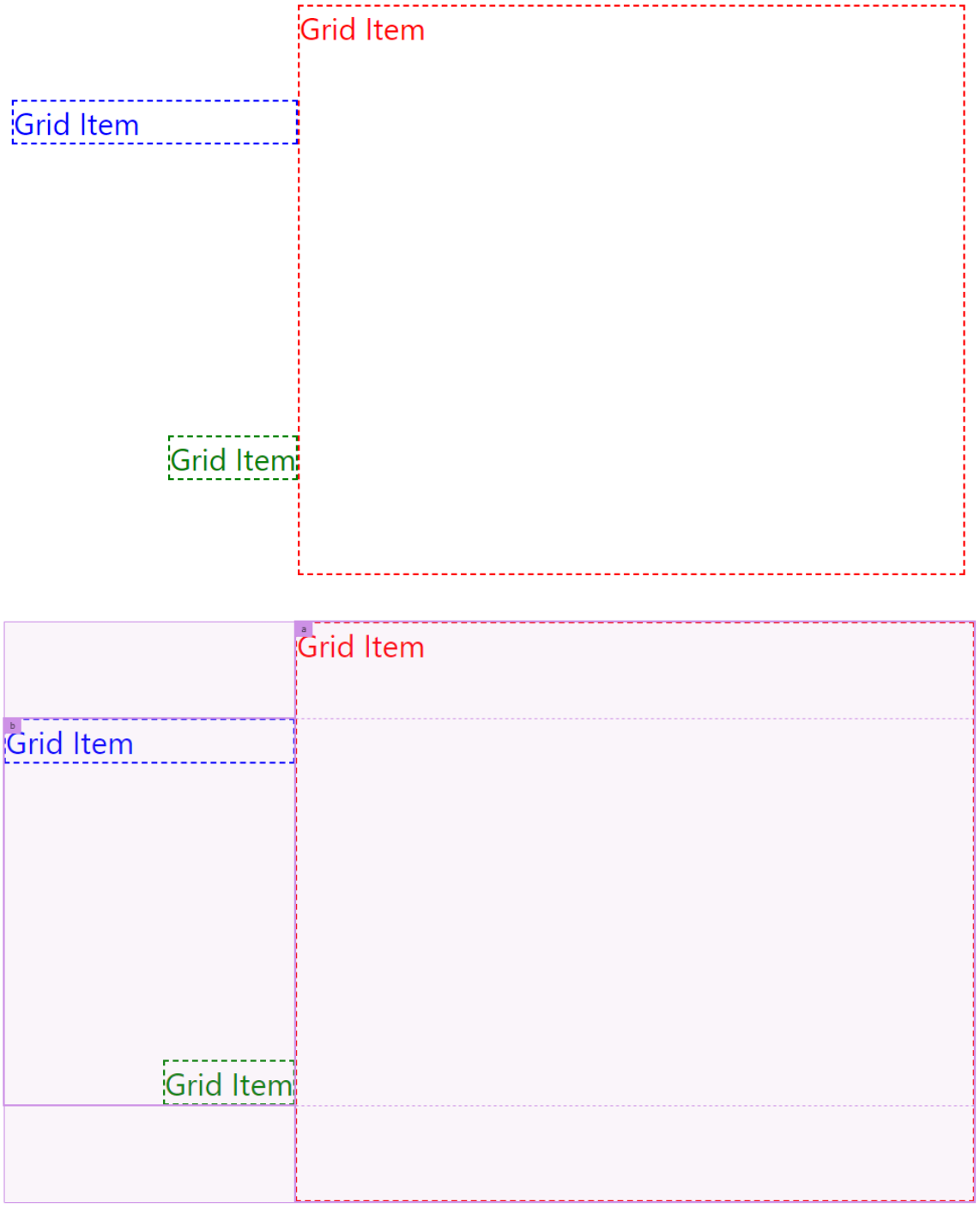
No seguinte exemplo, são definidas duas grid areas **a** e **b**. O grid items que foram posicionados em b são alinhados em coordenadas diferentes da grid area. Por padrão, grid items são esticados para conter na grid area, e ficaram sobrepostos caso não fossem alinhados.

```
  <head>
    <style>
      .grid {
        display: grid;
        grid-template-areas:
          ". a"
          "b a"
          ". a";
        grid-template-columns: 150px 1fr;
        grid-template-rows: 50px 1fr 50px;
        height: 300px;
        width: 500px;
      }

      .item-1 {
        border: 1px dashed red;
        color: red;
        grid-area: a;
      }

      .item-2 {
        align-self: start;
        border: 1px dashed blue;
        color: blue;
        grid-area: b;
      }

      .item-3 {
        align-self: end;
        border: 1px dashed green;
        color: green;
        grid-area: b;
        justify-self: end;
      }
    </style>
  </head>
  <body>
    <div class="grid">
      <div class="item-1">Grid Item</div>
      <div class="item-2">Grid Item</div>
      <div class="item-3">Grid Item</div>
    </div>
  </body>
</html>
```



A grid area de um grid item cria o **containing block** em que este é posicionado. Grid items posicionados na mesma grid area não afetam diretamente o layout um dos outros. Indiretamente, entretanto, um grid item ocupando uma grid track que possui uma função de dimensionamento intrínseca pode afetar o tamanho dessa grid track, o que por sua vez pode afetar a posição ou tamanho de outro grid item.

Grid Containers

nome: 'display'
novos valores: 'grid' | 'inline-grid'

Um **grid container** estabelece um **grid formatting context** independente para seu conteúdo. O conteúdo de um grid container é disposto em um grid, com grid lines delimitando os containing blocks dos grid items.

Um grid container é dimensionado utilizando as regras do contexto de formatação que participa:

1. Como um **block-level box** em um block formatting context, é dimensionado como um block box que estabelece um contexto de formatação, com o tamanho inline sendo ‘auto’;
2. Como um **inline-level box** em um inline formatting context, é dimensionado como um inline-level box atômico.

Grid Items

Em termos gerais, os **grid items** de um **grid container** são os boxes que representam seu conteúdo. Grid items estabelecem um contexto de formatação independente para seu conteúdo. Contudo, são em si **grid-level boxes**, e não block-level boxes: eles participam do contexto de formatação do grid.

Um grid item é dimensionado com relação ao containing block definido pela sua grid area. Cálculos para tamanhos automáticos de grid items variam de acordo com os seus valores para as propriedades de self-alignment (‘justify-self’/’align-self’):

alinhamento	comportamento do grid item
'normal'	preenche a grid area
'stretch'	preenche a grid area
'start'/'center'/etc	'fit-content'

Margens e padding definidos em porcentagem são resolvidos para valores absolutos baseando-se no inline size do containing block, assim como ocorre nos block boxes.

Definindo o Grid

O grid explícito

As propriedades '**grid-template-rows**', '**grid-template-columns**' e '**grid-template-areas**' definem, juntas, o **grid explícito** de um grid container, especificando as suas **grid tracks explícitas**. O grid final poderá ser maior por conta dos items posicionados fora do grid explícito. Neste caso, **grid tracks implícitas** serão criadas, dimensionadas pelas propriedades '**grid-auto-rows**' e '**grid-auto-columns**'.

Índices numéricos nas propriedades de posicionamento no grid contam das bordas do grid explícito. Índices positivos contam a partir do lado "start" (iniciando-se em 1), enquanto índices negativos contam a partir do lado "end" (iniciando-se em -1).

nome	'grid-template-columns' | 'grid-template-rows'
valor	none | <track-list> | <auto-track-list>
valor inicial	none
aplica-se à	grid containers

Estas propriedades especificam, **como uma lista de tracks separadas por espaços**, os nomes das linhas e as funções de dimensionamento das grid tracks. A propriedade '**grid-template-columns**' especifica a lista de tracks para as colunas e a propriedade '**grid-template-rows**' especifica a lista de tracks para as linhas.

Funções de dimensionamento

A lista de tracks é constituída de uma série de **funções de dimensionamento de tracks** e nomes para as linhas. Cada função de dimensionamento de track pode ser especificada:

1. Como um tamanho;
2. Como uma porcentagem relativa ao tamanho do grid container;
3. Como uma medição do conteúdo ocupando a coluna ou linha;
4. Como uma fração do espaço disponível no grid;
5. Como um intervalo especificado pela notação 'minmax()', que por sua vez pode combinar qualquer um dos mecanismos citados acima para especificar funções de tamanho mínimo e máximo para a célula.

Dada a seguinte declaração de '**grid-template-columns**':

```
grid-template-columns: 100px 1fr max-content minmax(min-content, 1fr);
```

Cinco grid lines são criadas:

1. No lado que se inicia o grid container;
2. A 100px a partir do lado que se inicia o grid container;
3. A uma distância da linha anterior equivalente à metade do espaço disponível (a largura do grid container subtraindo-se a largura das grid tracks não-flexíveis);
4. A uma distância da linha anterior equivalente ao tamanho máximo de qualquer um dos grid items que pertencem a coluna em questão;
5. A uma distância da linha anterior que é pelo menos tão larga quanto o menor conteúdo da coluna em questão, mas não mais larga do que a outra metade do espaço disponível do grid container.

Se os tamanhos não-flexíveis (100px, max-content e min-content) somam a um valor maior do que a largura do grid container, os tamanhos flexíveis resultarão em 0.

Alguns outros exemplos de definições válidas para as grid tracks:

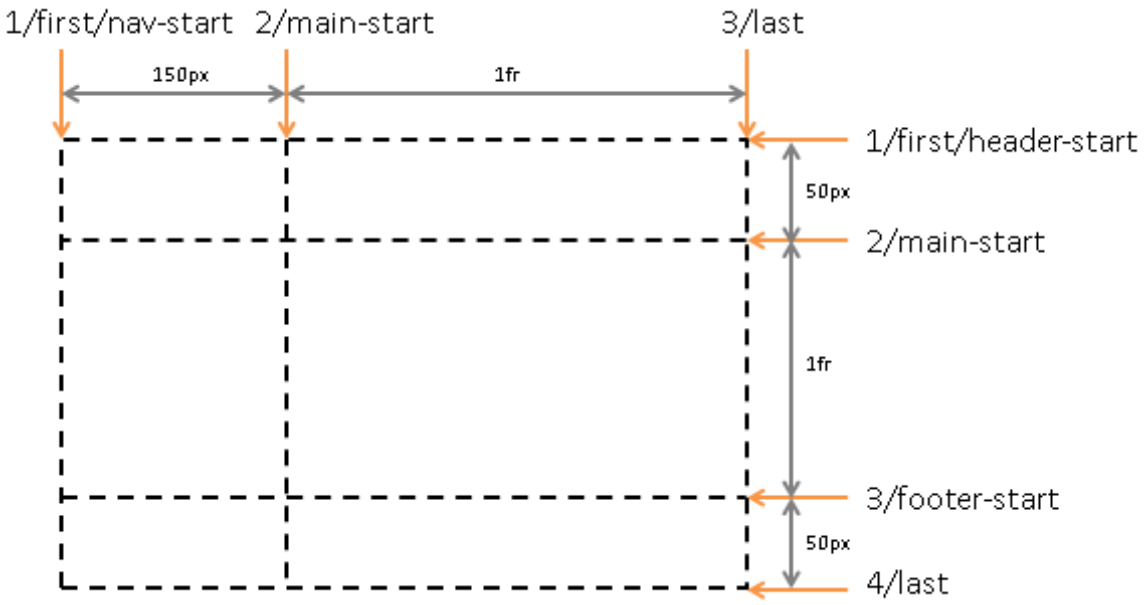
1. grid-template-rows: 1fr minmax(min-content, 1fr);
2. grid-template-rows: 10px repeat(2, 1fr auto minmax(30%, 1fr));
3. grid-template-rows: calc(4em - 5px);

Nomeando Grid Lines

Enquanto as grid lines podem ser referenciadas através de seus índices numéricos, **nomes para as linhas** tornam as propriedades de posicionamento mais fáceis de compreender e manter. Nomes para as linhas podem ser **definidos explicitamente** com as propriedades '**grid-template-columns**' e '**grid-template-rows**', ou **definidas implicitamente** com a propriedade '**grid-template-areas**'.

Por exemplo, o seguinte código fornece nomes significativos para todas as linhas do grid. Note que algumas linhas possuem mais de um nome:

```
.grid {
  display: grid;
  grid-template-columns: [first nav-start] 150px [main-start] 1fr [last];
  grid-template-rows: [first header-start] 50px [main-start] 1fr [footer-start] 50px [last];
}
```



Repetindo linhas e colunas: a notação ‘repeat()’

A notação ‘**repeat()**’ representa um fragmento repetido da lista de tracks, permitindo um intervalo maior de linhas ou colunas que exibem um padrão recorrente a ser escrito de forma mais compacta.

O exemplo seguinte mostra duas maneiras equivalentes de escrever a mesma definição para o grid:

```
.grid {
  display: grid;
  grid-template-columns: 10px [col-start] 250px [col-end]
                         10px [col-start] 250px [col-end]
                         10px [col-start] 250px [col-end]
                         10px [col-start] 250px [col-end] 10px;
  /* equivalente à definição acima, porém mais fácil de escrevera */
  grid-template-columns: repeat(4, 10px [col-start] 250px [col-end]) 10px;
}
```

A sintaxe da notação ‘**repeat()**’ é:

repeat([1, ∞] | auto-fill | auto-fit, <track-list>)

O primeiro argumento especifica o número de repetições. O segundo argumento é a lista de tracks, que será repetida esse número de vezes. No entanto, existem restrições:

1. A notação não pode ser aninhada;
2. Repetições automáticas (‘**auto-fill**’ ou ‘**auto-fit**’) não podem ser combinadas com tamanhos flexíveis.

Quando ‘**auto-fill**’ é determinado como o número de repetições, **se o grid container possui um tamanho definitivo ou um tamanho máximo no eixo em questão, então o número de repetições será o maior número inteiro que não causa o grid a transbordar o content box do container**; se qualquer número de repetições causará o transbordo, então será uma repetição apenas. **Caso contrário, se o grid container possuir um tamanho mínimo definitivo, então o número de repetições será o menor número inteiro positivo que atenda a esse requisito mínimo**. Por fim, a lista de tracks será repetida apenas uma vez.

O valor ‘**auto-fit**’ funciona de maneira equivalente à ‘**auto-fill**’, com exceção de que depois de posicionados, qualquer célula vazia é colapsada.

Tamanhos flexíveis: a unidade ‘fr’

Um **tamanho flexível** é uma dimensão com a unidade ‘**fr**’, que representa uma **fração** do espaço restante disponível no grid container. Grid tracks dimensionadas com ‘**fr**’ são chamadas de **tracks flexíveis**, uma vez que eles ajustam conforme o tamanho disponível de maneira similar aos flex items.

A distribuição do espaço restante ocorre depois que todos os grid tracks não-flexíveis são calculados. O tamanho total dessas linhas ou colunas é subtraído do espaço disponível, produzindo o espaço restante, que é então dividido entre as linhas e colunas de tamanho flexível em proporção ao seu flex factor.

Valores flexíveis não são comprimentos (e não são compatíveis com comprimentos, como são as porcentagens), e por isso não podem ser representados ou combinados com outras unidades nas expressões ‘calc()’.

Áreas nomeadas: a propriedade ‘grid-template-areas’

nome	'grid-template-areas'
valor	none | <string>+
valor inicial	none
aplica-se à	grid containers

A propriedade ‘**grid-template-areas**’ especifica **grid areas nomeadas**, que não estão associadas a nenhum grid item específico, mas podem ser referenciadas nas propriedades de posicionamento de grid. A sintaxe de ‘**grid-template-areas**’ também provê uma visualização da estrutura do grid, fazendo o layout geral do grid container mais fácil de se entender.

Uma **linha** é criada para **cada string** definida na propriedade, e uma **coluna** é criada para cada **célula** da string. Todas as strings devem ter a mesma quantidade de células, e pelo menos uma célula, para que a declaração seja válida.

No exemplo seguinte, a propriedade '**grid-template-areas**' é utilizada para criar um layout onde áreas são definidas para um cabeçalho (header), navegação (nav), rodapé (footer) e conteúdo principal (main).

```css
.grid {
  display: grid;
  grid-template-areas: "header header"
                       "nav    main"
                       "footer footer";
}

.grid-header {
  grid-area: header;
}

.grid-nav {
  grid-area: nav;
}

.grid-main {
  grid-area: main;
}

.grid-footer {
  grid-area: footer;
}
```

Nomes implícitos

A propriedade '**grid-template-areas**' gera **nomes** para as **grid lines** que são **atribuídos implicitamente** a partir das grid areas nomeadas. Para cada área nomeada foo, quatro nomes de linha implícitos são criados: dois denominados 'foo-start', atribuídos às linhas row-start e column-start da grid area, e dois denominados 'foo-end', atribuídos às linhas row-end e column-end da grid area.

Como uma grid area nomeada é referenciada pela grid lines nomeadas implícitas que produz, adicionando explicitamente nomes do mesmo formato para as grid lines (foo-start/foo-end) cria uma grid area nomeada. Tais áreas nomeadas implicitamente não aparecem como valores de '**grid-template-areas**', mas podem ser referenciadas nas propriedades de posicionamento de grid.

O grid implícito

As propriedades '**grid-template-rows**', '**grid-template-columns**' e '**grid-template-areas**' definem um número fixo de grid tracks que formam o **grid explícito**. Quando grid items são posicionados fora destes limites, o grid container gera **grid tracks implícitos** adicionando **grid lines implícitas** ao grid. Essas linhas, junto com o grid explícito, formam o **grid implícito**.

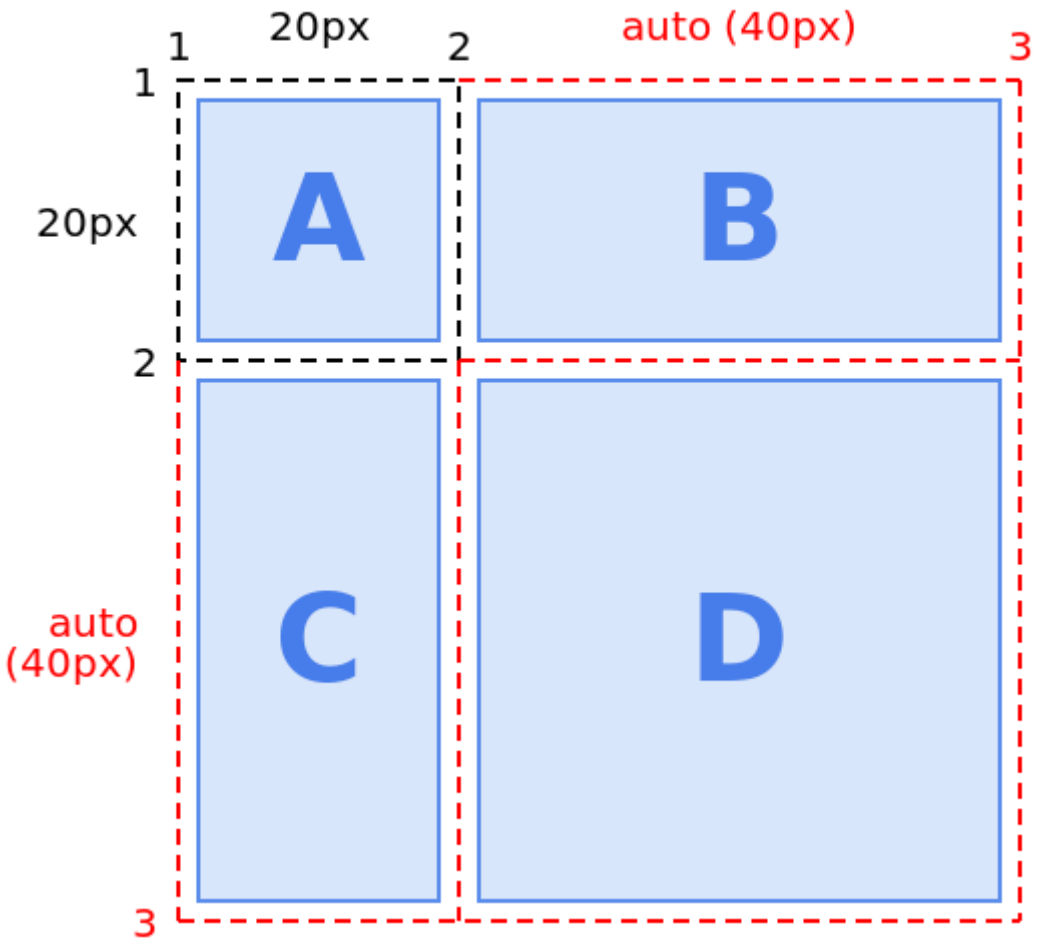
As propriedades '**grid-auto-rows**' e '**grid-auto-columns**' dimensionam essas grid tracks implícitas. A propriedade '**grid-auto-flow**' define o auto-posicionamento dos grid items sem definir um posicionamento explícito.

nome	'grid-auto-columns' | 'grid-auto-rows'
valor	<track-size>+
valor inicial	auto
aplica-se à	grid containers

No exemplo seguinte, um grid 2x2 com uma grid cell explícita de 20px x 20px e três grid cells implícitas de 40px x 40px são geradas para conter os grid items adicionais:

```
<html>
  <head>
    <style>
      .grid {
        display: grid;
        grid-template-columns: 20px;
        grid-template-rows: 20px;
        grid-auto-columns: 40px;
        grid-auto-rows: 40px;
      }

      .A { grid-column: 1; grid-row: 1; }
      .B { grid-column: 2; grid-row: 1; }
      .C { grid-column: 1; grid-row: 2; }
      .D { grid-column: 2; grid-row: 2; }
    </style>
  </head>
  <body>
    <div class="grid">
      <div class="A">A</div>
      <div class="B">B</div>
      <div class="C">C</div>
      <div class="D">D</div>
    </div>
  </body>
</html>
```



Posicionamento automático: a propriedade ‘grid-auto-flow’

nome	'grid-auto-flow'
valor	[row | column] || dense
valor inicial	row
aplica-se à	grid containers

Grid items que não são posicionados explicitamente são posicionados automaticamente em um espaço não ocupado do grid container pelo algoritmo de auto-posicionamento. A propriedade ‘**grid-auto-flow**’ controla como esse algoritmo funciona, definindo como itens posicionados automaticamente são colocados no fluxo do grid.

1. ‘**row’:** O algoritmo posiciona os itens preenchendo uma linha por vez, adicionando mais linhas se necessário;
2. ‘**column’:** O algoritmo posiciona os itens preenchendo cada coluna por vez, adicionando mais colunas se necessário;
3. ‘**dense’:** Se especificado, o algoritmo usa um empacotamento “denso”, que tenta preencher os buracos no grid mais cedo se itens menores aparecerem. Isso pode fazer com que os itens pareçam fora de ordem. Se omitido, um empacotamento “esparso” é usado, onde o algoritmo apenas move “para frente” na grid na hora de colocar os itens. Isso garante que os itens apareçam “em ordem”, mesmo que isso deixe lacunas que poderiam ter sido preenchidas por itens posteriores.

Posicionando grid items

Todos os grid items são associados com uma grid area, um set retangular de grid cells adjacentes que o grid item ocupa. Essa área define o containing block do grid item, onde as propriedades de self-placement (‘**justify-self**’ e ‘**align-self**’) determinam sua posição exata.

A **localização** da grid area de um grid item é determinada pelo seu **posicionamento**, que consiste de uma **grid position** e uma **grid span**:

1. **Grid position:** A localização do grid em cada eixo. A posição pode ser definitiva (especificada explicitamente) ou automática (definida pelo auto-posicionamento)

2. **Grid span:** Quantas grid tracks o grid item ocupa em cada eixo.

As propriedades de posicionamento da grid - ‘**grid-row-start**’, ‘**grid-row-end**’, ‘**grid-column-start**’, ‘**grid-column-end**’ e suas abreviações ‘**grid-row**’, ‘**grid-column**’ e ‘**grid-area**’ - permitem que o autor especifique o posicionamento de um grid item.

Alguns exemplos de uso das propriedades de posicionamento:

```
.one {
  /* Posiciona o item na área nomeada "main" */
  grid-area: main
}

.two {
  /* Alinha a borda inicial da linha com a borda
  inicial da área nomeada "main" */
  grid-row-start: main;
}

.three {
  grid-row: 2; /* Posiciona o item na segunda linha */
  grid-column: 3; /* Posiciona o item na terceira coluna */
  /* Equivalente a grid-area: 2 / 3*/
}

.four {
  /* Termina na sétima linha, expandindo 5 linhas para cima
  iniciando-se na segunda linha */
  grid-row: span 5 / 7;
}

.five {
  /* Item auto-posicionado, ocupando duas linhas e três colunas */
  grid-area: span 2 / span 3;
}
```

Alinhamento e Espaçamento

Depois que os grid tracks são posicionados e suas dimensões calculadas, os grid items podem ser alinhados em suas grid areas.

As propriedades '**margin**' podem ser utilizadas para alinhar os itens de maneira similar como é feito no block layout. Além disso, grid items respeitam as propriedades de alinhamento de box ('**justify-content**', '**justify-self**', '**justify-items**', '**align-content**', '**align-self**' e '**align-items**').

Por padrão, grid items esticam para preencher suas grid areas. Contudo, se as propriedades '**justify-self**' ou '**align-self**' possuem um valor diferente de '**stretch**', estes serão dimensionados automaticamente para encaixar seu conteúdo.

As propriedades '**row-gap**', '**column-gap**' (e sua abreviação '**gap**') definem o **gutter** entre linhas e colunas. O efeito dessas propriedades é como se as grid lines adquirissem espessura. Para fins de cálculo de dimensionamento, cada gutter é tratado como uma grid track vazia e do tamanho fixo especificado.