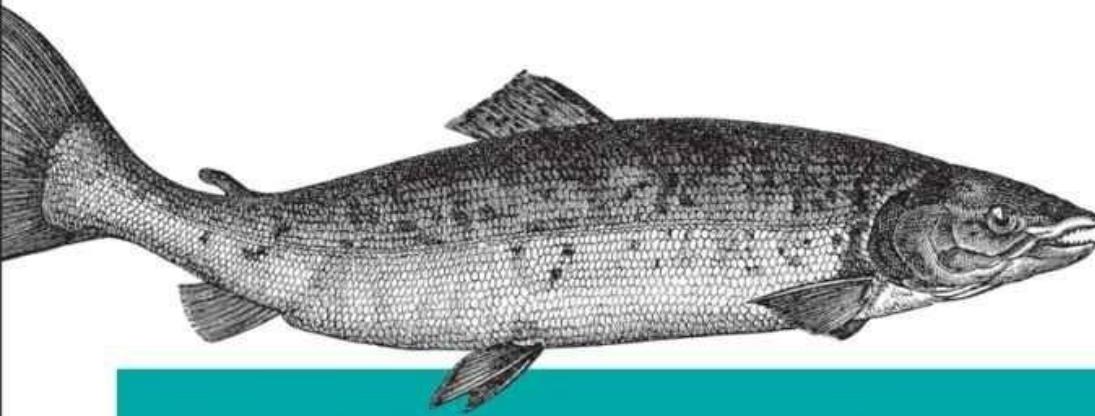
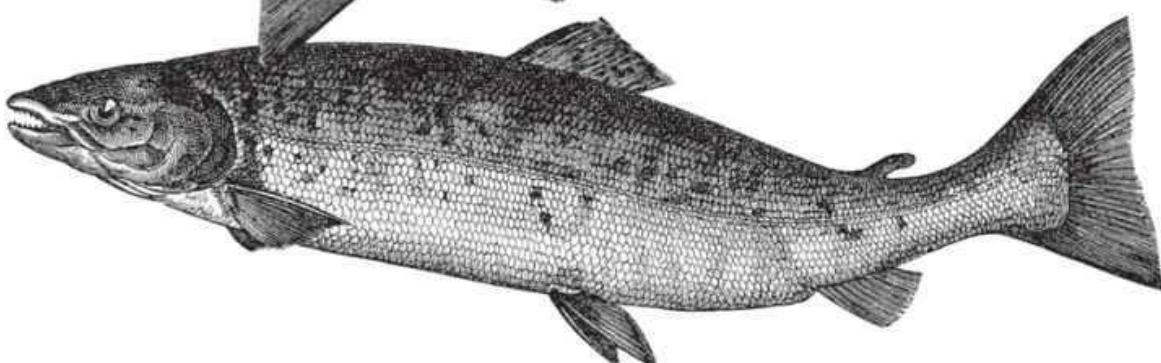
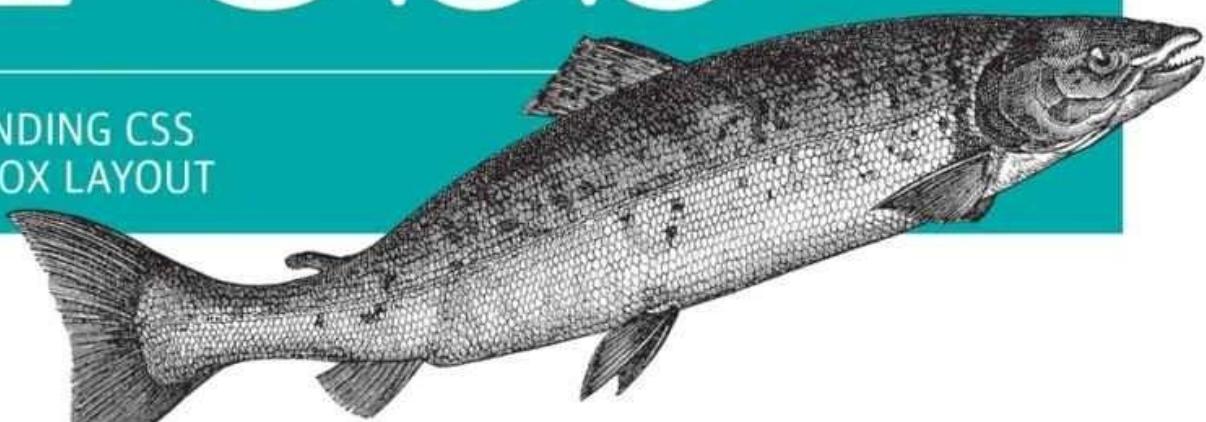


O'REILLY®



Flexbox in CSS

UNDERSTANDING CSS
FLEXIBLE BOX LAYOUT



Estelle Weyl

Flexbox em CSS

Noções básicas sobre o layout de caixa flexível CSS

Estelle Weyl •



Flexbox em CSS

por Estelle Weyl Copyright © 2017 Estelle Weyl. Todos os direitos reservados. Impresso nos Estados Unidos da América.

Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Os livros da O'Reilly podem ser comprados para uso educacional , comercial ou promocional de vendas. Edições online também estão disponíveis para a maioria dos títulos (<http://oreilly.com/safari>).

Para mais informações, entre em contato com nosso departamento de vendas corporativas/institucionais : 800- 998-9938 ou *corporate@oreilly.com*.

- Edição: Meg Foley
- Editor de Produção: Colleen
- Lobner Editor de Texto: Amanda
- Kersey Designer de Interiores:
- David Futato Designer de Capa:
- Randy Comer Ilustrador: Rebecca
- Demarest Junho de 2017:

Primeira Edição

Histórico de Revisões da Primeira Edição

- 2017-05-23: Primeiro Versão

O logotipo O'Reilly é uma marca registrada da O'Reilly Media, Inc. *Flexbox em CSS*, a imagem de capa do salmão e a imagem comercial relacionada são marcas comerciais da O'Reilly Media, Inc.

Embora o editor e o autor tenham usado esforços de boa-fé para garantir que as informações e instruções contidas neste trabalho sejam precisas, o editor e o autor se isentam de qualquer responsabilidade por erros ou omissões, incluindo, sem limitação, a responsabilidade por danos resultantes do uso de ou confiança neste trabalho. O uso das informações e instruções contidas neste trabalho é por sua própria responsabilidade. Se quaisquer exemplos de código ou outra tecnologia que este trabalho contém ou descreve estão sujeitos a licenças de código aberto ou aos direitos de propriedade intelectual de terceiros, é sua responsabilidade garantir que seu uso esteja em conformidade com tais licenças e / ou direitos.

978-1-491-98142-9

[LSI]

Este livro foi postado por AlenMiler no AvaxHome! <https://avxhm.se/blogs/AlenMiler>

Prefácio

Convenções Usadas neste Livro

As seguintes convenções tipográficas são usadas neste livro:

Itálico

Indica novos termos, URLs, endereços de e-mail, nomes de arquivos e extensões de arquivo.

Largura constante

Usado para listagens de programas, bem como dentro de parágrafos para se referir a elementos do programa, como nomes de variáveis ou funções, bancos de dados, tipos de dados, variáveis de ambiente, instruções e palavras-chave.

Largura constante em negrito

Mostra comandos ou outro texto que deve ser digitado literalmente pelo usuário.

Largura constante itálico

Mostra o texto que deve ser substituído por valores fornecidos pelo usuário ou por valores determinados pelo contexto.

NOTA

Esse elemento | significa uma nota geral, dica ou sugestão.

Usando exemplos de código

Sempre que você se deparar com um ícone parecido com , isso significa que há um exemplo de código associado. Exemplos ao vivo estão disponíveis em <http://standardista.com/flexbox/flexfiles>. Você pode clicar no  ícone enquanto lê este livro para ir diretamente para uma versão ao vivo do exemplo de código referenciado ou visitar o link para obter uma lista de todos os exemplos de código encontrados nestes capítulos.

Este livro está aqui para ajudá-lo a fazer o seu trabalho. Em geral, se o código de exemplo for oferecido com este livro, você poderá usá-lo em seus programas e documentação. Você não precisa entrar em contato conosco para obter permissão, a menos que esteja reproduzindo uma parte significativa do código. Por exemplo, escrever um programa que usa vários pedaços de código deste livro não requer permissão. Vender ou distribuir um CD-ROM de exemplos de livros O'Reilly requer permissão. Responder a uma pergunta citando este livro e quoting código de exemplo não requer permissão. Incorporar uma quantidade significativa de código de exemplo deste livro na documentação do seu produto requer permissão.

Apreciamos, mas não exigimos, atribuição. Uma atribuição geralmente inclui o título, o autor, o editor e o ISBN. Por exemplo: "*Caixas flexíveis em CSS* por Estelle Weyl (O'Reilly). Direitos autorais 2017 Estelle Weyl, 978-1-491-98142-9."

Se você acha que seu uso de exemplos de código está fora do uso justo ou da permissão dada acima, não hesite em contactar-nos pelo permissions@oreilly.com.

Safári O'Reilly

NOTA

O Safari (anteriormente Safari Books Online) é uma plataforma de treinamento e referência baseada em associação para empresas, governos, educadores e indivíduos.

Os membros têm acesso a milhares de livros, vídeos de treinamento, Caminhos de Aprendizagem, tutoriais interativos e listas de reprodução com curadoria de mais de 250 editoras, incluindo O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett e Course Technology, entre outras.

Para mais informações, visite <http://oreilly.com/safari>.

Como entrar em contato conosco

Por favor, envie comentários e perguntas sobre este livro para a editora:

- O'Reilly Media, Inc.
- 1005 Gravenstein Highway
- Sebastopol Norte, CA 95472
- 800-998-9938 (nos Estados Unidos ou
- Canadá) 707-829-0515 (internacional ou local)
- 707-829-0104 (fax)

Para comentar ou fazer perguntas técnicas sobre este livro, envie um e-mail para [*bookquestions@oreilly.com*](mailto:bookquestions@oreilly.com).

Para mais informações sobre nossos livros, cursos, conferências e notícias, consulte nosso site em [*http://www.oreilly.com*](http://www.oreilly.com).

Encontre-nos no Facebook: [*http://facebook.com/oreilly*](http://facebook.com/oreilly)

Siga-nos no Twitter: [*http://twitter.com/oreillymedia*](http://twitter.com/oreillymedia)

Veja-nos no YouTube: [*http://www.youtube.com/oreillymedia*](http://www.youtube.com/oreillymedia)

Sumário

Flexbox em CSS	2
Prefácio	5
Safári O'Reilly	7
Como entrar em contato conosco.....	8
Capítulo 1. Caixa flexível	11
O problema resolvido.....	11
Soluções simples.....	18
Aprendendo o Flexbox.....	19
Capítulo 2. Contêiner Flex.....	25
Propriedades do Flex Container.....	26
Dimensão cruzada da linha flexível.....	53
Contêiner Flex	55
Notas adicionais.....	81
Capítulo 3. Itens Flex.....	98
O que são itens Flex ?.....	99
Recursos do Flex Item	99
Propriedades específicas do item flexível.....	104
Fator de crescimento não nulo	110
Growing Proporcionalmente Baseado no Fator de Crescimento	110
Fator de crescimento com larguras diferentes	111
Fatores de crescimento e a propriedade flex.....	113
Proporcional com base na largura e no fator de encolhimento	122
No mundo real	123
Bases diferentes	125
Valores padrão	134
Unidades de Comprimento	136
Base Zero	142
A propriedade flex Shorthand.....	143
Rodapé pegajoso com flexão	156
Navegação com guias revisitada.....	163
Capítulo 4. Exemplos de Flexbox.....	166
Layout responsivo de duas colunas	167
Tela mais larga Layout	171
Página inicial da rede elétrica	173
Seções	180
Centralização vertical	185
Exemplo de flex embutido.....	186
Calendário.....	187

Grade Mágica.....	191
Desempenho	198
Bom para ir	198

Capítulo 1. Caixa flexível

O **CSS Flexible Box Module Level 1**, ou flexbox para abreviar, torna a tarefa antes difícil de dispor sua página, widget, aplicativo ou galeria quase simples. Com o flexbox, o layout é tão simples que você não precisará de uma estrutura CSS. Widgets, carrosséis, recursos responsivos – o que quer que seu designer sonhe – serão um aperto para o código. E, embora as bibliotecas de layout do flexbox já tenham aparecido, em vez de adicionar aveia-bl à sua marcação, leia este livro e saiba como, com algumas linhas de CSS, você pode criar quase qualquer recurso responsivo que seu site exige.

O problema resolvido

Por design, o flexbox é independente de direção. Isso é diferente dos layouts de bloco ou embutidos, que são definidos como tendenciosos vertical e horizontalmente, respectivamente. A web foi originalmente projetada para a criação de páginas em monitores.

O layout com viés vertical é insuficiente para aplicativos modernos que mudam de orientação, crescem e diminuem dependendo do agente do usuário e da direção do visor, e alteram os modos de gravação dependendo do idioma.

Layout na web tem sido um challenge para muitos. Durante anos, brincamos sobre os desafios da centralização vertical e do layout de várias colunas. Alguns layouts não eram motivo de riso, como garantir alturas iguais em uma grade de várias caixas lado a lado, com botões ou "mais" links fixados na parte inferior de cada caixa, com o conteúdo do botão bem centralizado verticalmente, como mostrado na [Figura 1-1](#), ou garantir caixas em uma galeria de conteúdo variada tinham a mesma altura, enquanto a linha superior de caixas da galeria estava bem alinhada com as caixas nas linhas subsequentes, como mostrado na [Figura 1-2](#).

O Flexbox torna todos esses desafios bastante simples.

Power Grid: Store or Homepage

Home

About

Blog

Careers

Contact Us



This is some awesome content that is on the page.

Go Somewhere



This is more content than the previous box, but less than the next.

Click Me



We have lots of content here to show that content can grow, and everything can be the same size if you use flexbox. Even if this has tons of text, it will line up with the other sections.

Do Something

Copyright © 2016

Figura 1-1. Layout da rede elétrica com flexbox, com botões alinhados na parte inferior



Gallery: Height & Row Grid

[Home](#)[About](#)[Blog](#)[Careers](#)[Contact Us](#)

Cough furball hide from vacuum cleaner or pooping rainbow while flying in a toasted bread costume in space give attitude.



Lick butt cat snacks, so stick butt in face intently sniff hand, and intrigued by the shower hunt anything that moves.



Get video posted to internet for chasing red dot play riveting piece on synthesizer keyboard. Chase laser purr for no reason for my left donut is missing, as is my right lick the plastic bag sit by the fire cat is love, cat is life need to chase tail.



Drink water out of the faucet hunt by meowing loudly at Sam next to human slave food dispenser flop over, but the dog smells bad yet plan steps for world domination or chase the pig around the house but need to chase tail.



Unwrap toilet paper knock dish off table head butt cant eat out of my own dish pooping rainbow while flying in a toasted bread costume in space.



Stare at ceiling hack up furballs.



Claws in your leg jump around on couch, meow constantly until given food. Pee in the shoe.



Knock dish off table head butt cant eat out of my own dish sleep nap so spit up on light gray carpet instead of adjacent linoleum hiss at vacuum cleaner for caticus cutieus.



Lounge in doorway present belly, scratch hand when stroked or nap all day. Knock over christmas tree use lap as chair chase imaginary bugs. Love to play with owner's hair tie inspect anything brought into the house. Paw at your fat belly rub face on owner so pee in the shoe. Refuse to leave cardboard box stare out the window, or eat a plant, kill a hand lick arm hair and fall over dead (not really but gets sympathy). Behind the couch pee in the shoe.



Put toy mouse in food bowl run out of litter box at full speed caticus cutieus all of a sudden cat goes crazy. Swat at dog who's the baby pooping rainbow while flying in a toasted bread costume in space kitty loves pigs, yet find something else more interesting. Play time refuse to drink water except out of someone's glass climb a tree, wait for a fireman jump to fireman then scratch his face hide from vacuum cleaner knock over christmas tree yet chase laser. The dog smells bad make muffins sleep on dog bed, force dog to sleep on floor asdflikjaertvikjsantvkjn (sits on keyboard).



Chirp at birds chase dog then run away paw at your fat belly mew. Give attitude cat slap dog in face. Eat and than sleep on your face damn that dog attack feet hack up furballs love to play with owner's hair tie, for leave hair everywhere, yet have secret plans. Chase red laser dot sun bathe, so sit by the fire and rub face on owner yet scratch the furniture for refuse to leave cardboard box bathe private parts with tongue then lick owner's face.

Figura 1-2. Galeria de seções de conteúdo variável alinhadas em colunas usando flexbox; as seções em cada linha tem altura igual

This is the header

this is the content

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Qui in voluptatum iusto sed nesciunt esse omnis nisi dolor. Esse voluptatem optio dolorem eum architecto quo error mollitia pariatur veniam nobis.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequuntur iure, nobis alias modi id minus nam, sunt, reiciendis numquam quasi veniam! Dicta saepe nisi voluptatum. Modi soluta saepe deserunt sit!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur possimus numquam, fugiat harum doloribus a dignissimos laboriosam, architecto porro magni aut, ipsa laborum deleniti eum doloremque, nam corporis odit accusamus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

This is a bit more content

This is the footer

Figura 1-3. Cabeçalho e rodapé adesivos no celular usando flexbox em vez de posição fixa



Figura 1-4. Widget com muitos componentes perfeitamente centralizados verticalmente



Além de realmente declarar uma altura, arriscar muitos espaços em branco ou conteúdo transbordante, não havia como tornar todas as colunas iguais em altura. Vários layouts de coluna foram criados flutuando cada coluna, com cada coluna sendo uma largura predeterminada e diferentes alturas dependentes do conteúdo da coluna. Embora você possa usar imagens de plano de fundo falsas com uma solução de layout de várias colunas ou o valor da tabela da propriedade display, o flexbox é uma maneira simples — e a maneira correta — tornar as colunas iguais em altura.

NOTA

Antes dos layouts flutuantes, era comum ver tabelas | usadas | para layout. As tabelas não devem ser usadas | para layout por muitos motivos, incluindo o fato de que o layout da tabela | não é semântico |, é difícil de atualizar se o layout for alterado, pode ser difícil de tornar acessível, aumenta o inchaço do código | e dificulta uma cópia de | texto. Dito isto, as tabelas são apropriadas | para dados tabulares.

O **layout do Santo Graal**, com um cabeçalho, três colunas e um rodapé, poderia ser resolvido de muitas maneiras, nenhuma delas simples, até que tivéssemos o flexbox. Geralmente, nós levamos flutuadores:

.HTML:

```
<cabeçalho>Cabeçalho</cabeçalho>
<principal>
  <nav>Links</nav>
  <aparte>Além do conteúdo</à parte>
  <artigo>Conteúdo do documento</artigo>
</principal>
<rodapé>Rodapé</rodapé>
```

.CSS:

```
principal {
  imagem de fundo: url(images/fakecolumns.gif);
  largura: 100%;
  flutuação: esquerda;
}
à parte, nav { float: esquerda; largura: 25%;
  transbordamento: oculto;
}
artigo {
```

```
flutuação: esquerda;  
largura: 50%;  
}
```

NOTA

Este código | aparece aqui por uma questão histórica - você não precisa mais fazer isso!

A saída é mostrada na [Figura 1-5](#).

A maioria dos projetos exige colunas de alturas iguais, mas adicionar diferentes cores de plano de fundo ou imagens para o lado, artigo e navegação na [Figura 1-5](#) realmente amplificaria que eles têm alturas diferentes. Para fornecer a aparência de colunas de altura igual, muitas vezes adicionamos um plano de fundo falso ao pai com base nas larguras de coluna declaradas em nosso CSS, como mostrado pela imagem de fundo cinza-branco-cinza na [Figura 1-5](#).

Para garantir que o pai fosse pelo menos tão alto quanto as colunas flutuantes, a maioria dos desenvolvedores adicionou uma *correção clara* como conteúdo gerado após a última coluna, embora fornecendo ao pai um largura flutuante de 100% foi uma solução igualmente viável.

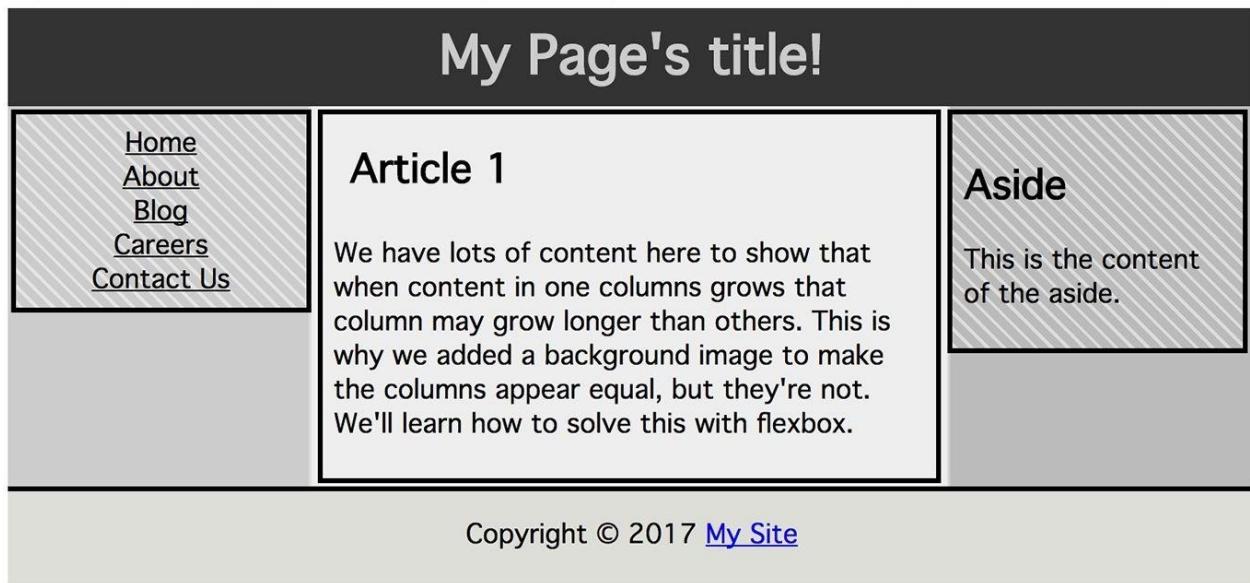


Figura 1-5. Layout do Santo Graal sem flexbox



Umclearfixé uma classe que pode ser adicionada ao seu CSS e, em seguida, a qualquer elemento para

certifique-se de que contém totalmente as suas crianças flutuadas. Ele funciona adicionando conteúdo gerado invisível que é exibido bloco ou tabela e, em seguida, limpo, limpando assim tudo acima dele. Exemplos comuns que você encontrará na natureza incluem:

```
.clearfix:after { content: ".\"; display: bloco;
  altura: 0; claro: ambos;
  visibilidade: oculta;
}
```

e

```
.clearfix:after { content: ""; display: tabela;
  claro: ambos;
}
```

Ao usar essa técnica, um bloco adicional ou descendente de célula de tabela anônima é inserido no contêiner como conteúdo gerado. Este descendente é limpo de quaisquer flutuadores na direção em linha ou bloco. Isso força o tamanho do bloco do recipiente que tem o .clearfix aplicado para incluir as alturas dos flutuadores (essas dimensões normalmente não são incluídas, pois os flutuadores são removidos do fluxo).

O método que usei em vez de adicionar uma classe .clearfix e conteúdo gerado foi aproveitar o fato de que, com CSS, todos os elementos flutuados devem ser pelo menos tão alto quanto seu descendente flutuante mais alto. Ao tornar o pai 100% largo e flutuá-lo, o pai seria pelo menos tão alto quanto seu descendente flutuante aninhado mais alto, enquanto estava sozinho em sua própria linha. Esse método flutuante de limpeza era suportado em navegadores antes que versões arcaicas do Internet Explorer começassem a oferecer suporte a conteúdo gerado:

```
principal {
  largura: 100%;
  flutuação: esquerda;
}
```

O layout anterior é, na verdade, mais feio do que o mostrado na [Figura 1-5](#). Eu adicionei preenchimento para torná-lo melhor. A alteração nas propriedades do modelo de caixa causou

a largura total deve ser maior que 100%, fazendo com que a última coluna caia. Isso é facilmente resolvido com o dimensionamento da caixa: borda-caixa; . Adicionar uma margem esquerda ou direita positiva também faria com que a última coluna caísse, sem uma solução rápida simples.

Entre margens em colapso e flutuações em queda, o método de layout antigo pode ser francamente confuso. Muitas pessoas começaram a usar grades YUI, Bootstrap, Foundation, 960 grid e outras bibliotecas de grade CSS para simplificar seu processo de desenvolvimento. Espero que sua conclusão seja que você não precisa de uma muleta de estrutura CSS.

Observe que o flexbox foi projetado para um tipo específico de layout, o de distribuição de conteúdo unidimensional. Embora você possa criar layouts semelhantes a grades (alinhamento bidimensional) com o flexbox, há uma especificação Grid, que com suporte aprimorado será a maneira correta de criar grades. Isso é discutido mais adiante em *Grid Layout in CSS* por Eric A. Meyer (O'Reilly).

Soluções simples

O Flexbox é uma maneira simples e poderosa de dispor aplicativos da Web ou seções de documentos, ditando como o espaço é distribuído, o conteúdo é alinhado e as exibições são visualmente ordenadas, permitindo a aparência de alongamento, encolhimento, reversão e até mesmo reorganizar a aparência do conteúdo sem alterar a marcação subjacente. O conteúdo agora pode ser facilmente disposto vertical ou horizontalmente, pode parecer ter a ordem reorganizada, pode ser disposto ao longo de um único eixo ou enrolado em várias lines, pode crescer naturalmente para abranger todos os espaço disponível, ou encolher para caber no espaço alocado, e muito mais.

Flexbox é uma maneira declarativa de calcular e distribuir espaço. Vários layouts de coluna são muito fáceis, mesmo que você não saiba quantas colunas seu conteúdo terá. O Flexbox permite que você tenha certeza de que seu layout não será interrompido quando você gerar dinamicamente mais conteúdo , quando o conteúdo for removido ou quando seus usuários esticarem ou reduzirem o navegador ou mudarem de retrato para paisagem modo.

Com o flexbox, reorganizar visualmente o conteúdo sem afetar a marcação subjacente é fácil. Com o flexbox, a aparência do conteúdo pode ser independente da ordem da fonte. Embora visualmente alterados, os properties flex p não devem afetar a ordem de como o conteúdo é lido pelos leitores de tela.

Os leitores de tela que seguem a ordem do código-fonte estão na especificação, mas o Firefox atualmente segue a ordem visual. Há uma discussão na comunidade de acessibilidade que este "bug" do Firefox pode ser o comportamento correto. Portanto, é possível que a especificação possa mudar.

E, o mais importante, com layouts flexíveis de módulo de caixa, os elementos podem ser feitos para se comportar de forma previsível para diferentes tamanhos de tela e diferentes dispositivos de exibição.

O Flexbox funciona bem para sites responsivos, pois o conteúdo pode aumentar e diminuir de tamanho quando o espaço fornecido é aumentado ou diminuído.

O Flexbox pode ser usado para mapear um documento inteiro por meio de layouts de bloco ou usado embutido para posicionar melhor o texto.

Aprendendo o Flexbox

Flexbox é uma relação pai e filho. O layout do Flexbox é ativado declarando a exibição: flex; ou display: inline-flex; em um elemento que então se torna um contêiner flex, organizando seus filhos dentro do espaço desde que um controlando seu layout. Os filhos deste recipiente flex tornam-se itens flex.

O Flexbox funciona em um sistema de grade de eixos. Com o flexbox, você adiciona valores de propriedade CSS a um *elemento de contêiner flex*, indicando como os filhos, os *itens flexíveis*, devem ser dispostos. As crianças podem ser dispostas da esquerda para a direita, da direita para a esquerda, de cima para baixo ou até mesmo de baixo para cima. Os itens flex podem ser dispostos lado a lado em uma única linha, ou permitidos, ou mesmo forçados, a serem enrolados em várias linhas com base nos valores da propriedade flex dos contêineres flex. Essas crianças podem ser exibidas visualmente conforme definido pela ordem de origem, invertidas ou reorganizadas para qualquer ordem de sua escolha.

Se os filhos do seu contêiner flex não preencherem toda a largura ou altura do recipiente, existem propriedades flexbox que ditam como lidar com o espaço extra, incluindo a preservação do espaço ou a distribuição entre eles. as crianças.

Quando o espaço é preservado, você pode agrupar as crianças à esquerda, à direita ou ao centro, ou pode espalhá-las , definindo como o espaço está espalhado entre ou ao redor das crianças.

Você pode fazer com que as crianças ocupem todo o espaço disponível distribuindo esse espaço extra entre um, alguns ou todos os itens flexíveis. Você pode ditar como as crianças crescem, distribuindo o espaço extra uniformemente, proporcionalmente ou por quantidades definidas. As crianças podem ser alinhadas em relação ao recipiente ou umas às outras, na parte inferior, superior ou central do recipiente, ou esticadas para encher o recipiente. Regardless da diferença no comprimento do conteúdo entre os contêineres irmãos, com o flexbox você pode fazer todos os irmãos do mesmo tamanho com uma única declaração CSS.

Se não houver espaço suficiente para conter todas as crianças, há propriedades de caixa

flexível que você pode usar para ditar como as crianças devem encolher para caber dentro de seu recipiente.

O Flexbox define um contexto de formatação junto com propriedades para controlar o layout. Quando você define um elemento a ser disposto como uma caixa flexível, ele apenas flexionará seus filhos imediatos, e não mais descendentes. No entanto, você também pode tornar essas caixas flexíveis descendentes, permitindo alguns layouts realmente complexos. Um elemento que tem um pai e um filho pode ser um contêiner flexível e um item flexível.

Elementos que não são flexionados e não estão absolutamente posicionados têm cálculos de layout tendenciosos para bloquear e inline direções de fluxo. O layout flexível , por outro lado, é tendencioso para as direções flexíveis. O valor flex-flow (consulte "The flex-flow Shorthand Property") determina como o conteúdo é mapeado para a parte superior, direita, inferior , esquerda, ao longo de um eixo horizontal ou vertical e por largura e altura.

Depois de definir um elemento como um contêiner flexível, seus filhos seguem as regras do flexbox para layout em vez das regras padrão de bloco , embutido e bloco embutido.

Dentro de um contêiner flex, os itens se alinham no "eixo principal". O eixo principal pode ser horizontal ou vertical para que você possa organizar os itens em colunas ou linhas. O eixo principal assume a direcionalidade definida através do modo de escrita: este conceito de eixo principal será discutido em profundidade mais adiante (ver "[Entendendo eixos](#)").

Nas próximas seções, abordaremos como criar um contêiner flexível usando a propriedade display e, em seguida, explicaremos as várias propriedades do contêiner flexível para distribuir e alinhar itens flexíveis dentro do contêiner flex. Depois de abordarmos as propriedades aplicadas ao contêiner flex, abordaremos as propriedades aplicadas diretamente aos itens flex . Aprenderemos como fazer com que os filhos dos contêineres flex diminuam e cresçam, e discutiremos as propriedades aplicadas a esses filhos que lhes permitem substituir a distribuição e o alinhamento definidos globalmente em todos os itens flexíveis pelo contêiner flex pai. Também incluímos vários casos de uso do flexbox.

A propriedade display

O primeiro passo é transformar um elemento em um contêiner flex. Isso é feito com dois novos valores para a propriedade de exibição bem conhecida.

EXPOSIÇÃO

Valores de layout do Flex : `flex` | `inline-flex`

Herdados:	Não
------------------	-----

```
/* CSS 1 */ display: em linha; display: bloco;  
exibição: lista-item; exibição: nenhuma;  
  
/* CSS 2.1 */ display: bloco em linha; display: tabela;  
display: tabela em linha;  
exibição: tabela-linha-grupo; exibição: tabela-cabeçalho-grupo;  
exibição: tabela-rodapé-grupo; exibição: tabela-linha; exibição:  
tabela-coluna-grupo; exibição: tabela-coluna; exibição: tabela-  
célula; exibição: tabela-legenda; exibição: herdar;  
  
/* Valores de exibição mais recentes */ exibição: inicial;  
exibição: unset;  
display: flex; display: inline-flex; exibição: grade;  
display: grade em linha; exibição: rubi; display: ruby-  
base; exibição: ruby-text;  
display: ruby-base-container;  
exibição: ruby-text-container;  
  
/* Valores experimentais */ exibição: run-in; exibição:  
conteúdo;  
exibição: inline-list-item;  
display: fluxo;  
exibição: fluxo-raiz;
```

NOTA

Atualmente, existem 30 valores para a propriedade `display` descritos nas várias especificações. Embora nem todos os valores de exibição mais recentes sejam totalmente suportados no momento da redação deste artigo, espera-se que eles sejam incluídos em todos os navegadores modernos.

Os valores de `run-in` e `compact` foram incluídos no CSS2, mas removidos no CSS2.1. `run-in` voltou para [o CSS Display Module Level 3](#), juntamente com fluxo, raiz de fluxo e conteúdo. O valor do item de lista embutida inclui a especificação [CSS Lists and Counters Module Level 3](#). Todos esses valores experimentais ainda estão sendo discutidos e não são totalmente suportados. Quando o valor de raiz de fluxo receber suporte, espere também obter suporte para valores separados por espaço, como `display: flow list-item block;`.

Dois dos valores mais recentes para a propriedade `display` foram adicionados na especificação CSS Flexible Box Layout Module Level 1: `flex` e `inline-flex`. O valor de `flex` transforma o elemento no qual ele é aplicado em uma caixa de contêiner `flex` em nível de bloco. Da mesma forma, o valor `inline-flex` transforma o elemento no qual ele é aplicado em um bloco `flex-container`, mas o contêiner `flex` é uma caixa de contêiner `flex` em nível inline.

A simples adição de qualquer um desses valores de propriedade de exibição em um elemento transforma o elemento em um contêiner flexível e os filhos do elemento em itens flexíveis. Por padrão, as crianças têm todas a mesma altura, mesmo que seu conteúdo produza elementos de alturas diferentes, como mostra a [Figura 1-6](#).

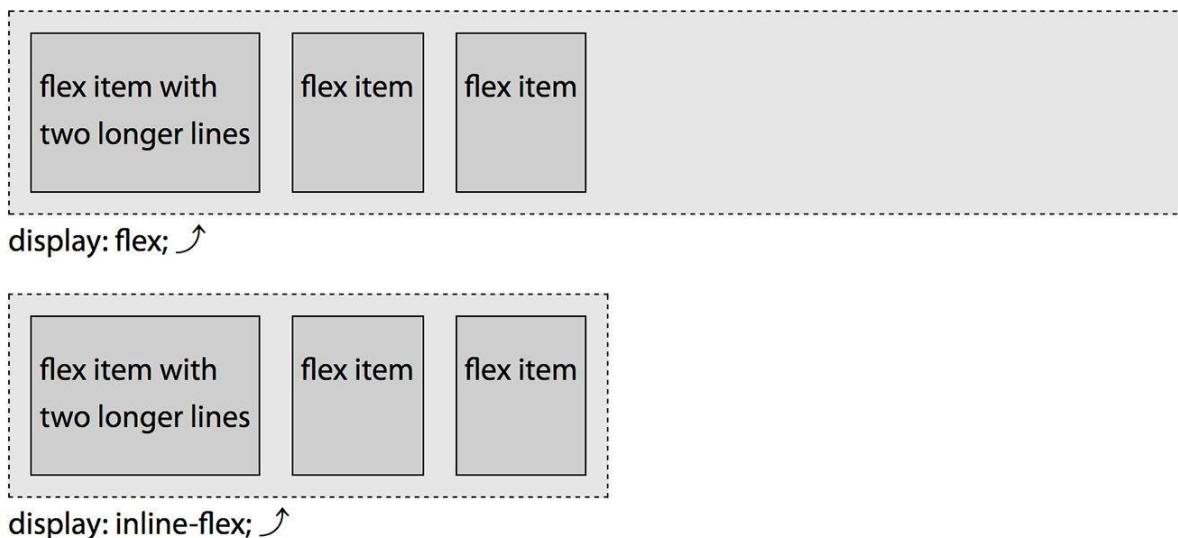


Figura 1-6. Adicionando `display: flex;` ou `display: inline-flex;` cria um contêiner flexível



NOTA

Pará pessoas familiarizadas com layouts baseados em flutuação, um aparência padrão que é criada simplesmente adicionando esses valores de exibição é semelhante a definir uma largura de contêiner para 100% e flutuá-la e todos os seus filhos à esquerda, ou usando o método `.clearfix`, mas melhor. Como crianças ainda se encaixam em uma única linha, mesmo que possam ter se enrolado se realmente flutuar. E, assim como SO elementos flutuados são pelo menos tão altos quanto seus filhos flutuantes mais altos, o recipiente será alto o suficiente para abranger seus filhos.

O valor `inline-flex` faz com que o contêiner flex se comporte como um elemento de nível inline. Ele será tão largo quanto necessário, conforme declarado, ou tão largo quanto uma coluna se a direção flexível estiver definida como coluna (definida a seguir). Como outros elementos de nível em linha, o contêiner flexível em linha fica junto em uma linha com outros elementos de nível em linha e é afetado pela altura da linha e pelo alinhamento vertical, o que cria espaço para os descendentes abaixo da caixa por inadimplência. O valor flexível da propriedade `display` se comporta como um elemento de bloco.

NOTA

Adicionamos preenchimento, margens, bordas ao contêiner flexível e itens para melhorar a aparência das figuras e melhorar a legibilidade das figuras. As propriedades do modelo de caixa afetam o layout flexível. Se não tivéssemos incluído essas propriedades, todos os itens flexíveis seriam agrupados contra o contêiner flex e uns contra os outros e seriam indistinguíveis uns dos outros. As explicações de ilustração não abordarão os efeitos das propriedades box-model até começarmos a cobrir alguns dos efeitos do layout do modelo de caixa em "The align-content Propriedade".

Se quisermos criar uma barra de navegação a partir de um grupo de links, é muito simples. Basta exibir: `flex;` :

```
nav {  
  display: flex;  
}  
  
<navegação>  
  <a href="/">Home</a>  
  <a href="/about">Sobre</a>
```

```

<a href="/blog">Blog</a>
<a href="/jobs">Carreiras</a>
<a href="/contact">Fale Conosco</a>
</navegação>

```

No código anterior, com sua propriedade de exibição definida como flex , o <nav> é transformado em um contêiner flex e seus links filho são todos itens flex. Esses links são caixas de nível flexível, semanticamente ainda links, mas agora flexionam itens em sua apresentação.

Eles não são caixas de nível embutido: em vez disso, eles participam do contexto de formatação flexível de seu contêiner. Portanto, o espaço em branco é ignorado:

```

nav {
  display: flex;
  borda-fundo: 1px sólido #ccc;
}
a {
  margem: 0 5px;
  preenchimento: 5px 15px;
  raio de borda: 3px 3px 0 0; cor de fundo: #ddaa00; texto-decoração: nenhuma; cor: #ffffff;
}
a:hover, a:focus, a:active { background-color: #ffcc22; cor: preto;
}

```

Com um pouco de CSS adicionado, temos uma barra de navegação com guias simples , como mostra a **Figura 1-7**.



Figura 1-7. Uma navegação simples com guias



Um contexto de formatação flexível é semelhante a um contexto de formatação de bloco , exceto que o layout flexível é usado em vez do layout de bloco: os flutuadores não se intrometem no contêiner flexível e as margens do contêiner flexível não colapsam com as margens do seu conteúdo.

Embora existam semelhanças, os contêineres flexíveis são diferentes dos contêineres de bloco. As propriedades Some CSS não se aplicam no contexto flex. As propriedades column-* , ::primeira linha e ::primeira letra não se aplicam quando se trata do flex

recipiente.

NOTA

Os pseudoelementos `::first-line` e `::first-letter` selecionam a primeira linha e a primeira letra dos elementos de nível de bloco, respectivamente.

Capítulo 2. Contêiner Flex

A primeira noção importante a ser entendida completamente é a de *contêiner flexível*, também conhecida como *caixa de contêiner*. O elemento no qual `display: flex` ou `display: inline-flex` é aplicado torna-se um contexto de formatação flex para os filhos da caixa que o contém, conhecido como *contêiner flex*. Depois de criarmos um contêiner flexível (adicionando um `display: flex` ou `display: inline-flex`), precisamos aprender a manipular o layout dos filhos do contêiner.

Os filhos dessa caixa de contêiner são *itens flexíveis*, sejam eles nós DOM, nós de texto ou conteúdo gerado. Filhos absolutamente posicionados de contêineres flex também são itens flex, mas são dimensionados e posicionados como se fossem o único item flex no contêiner flex.

Primeiro, aprenderemos tudo sobre as propriedades CSS que se aplicam ao contêiner flex, incluindo várias propriedades que afetam o layout dos itens flex. Os itens Flex em si são um conceito importante que você precisa tatear e serão abordados na íntegra no [Capítulo 3](#).

Propriedades do Flex Container

Os exemplos de propriedades de exibição na [Figura 1-6](#) mostram três itens flexíveis lado a lado, indo da esquerda para a direita, em uma linha. Com algumas declarações de valor de propriedade adicionais, poderíamos ter centralizado os itens, alinhá-los à parte inferior do contêiner, reorganizar sua ordem de aparência e colocá-los da esquerda para a direita ou de cima para baixo. Poderíamos até tê-los feito abranger algumas linhas.

Às vezes teremos um item flexível, às vezes teremos dezenas. Às vezes, saberemos quantos filhos um nó terá. Às vezes, o número de crianças não estará sob nosso controle. Podemos ter um número variado de itens em um contêiner de largura definida. Podemos saber o número de itens, mas não sabemos a largura do contêiner. Devemos ter CSS robusto que possa lidar com nossos layouts quando não sabemos quantos itens flexíveis teremos ou quão largo o contêiner flexível será (pense em responsivo). Existem várias propriedades fora dos novos valores de exibição que podemos adicionar ao contêiner flexível para fornecer controle sobre o layout que nos permitem criar responsivos layouts e widgets responsivos.

As propriedades display, flex-direction, flexwrap e flexflow afetam a ordenação e a orientação do contêiner flex. As propriedades justify-content, align-items e align-content podem ser aplicadas ao contêiner flex para afetar o alinhamento dos filhos do contêiner.

A propriedade de taquigrafia flexflow

A propriedade flexflow permite definir as direções dos eixos principal e cruzado e se os itens flex podem ser encapsulados em mais de uma linha, se necessário.

FLEXFLOW	
Valores	< >flex-direção <flexwrap>
Valor inicial :	linha nowrap
Aplica-se um:	Contêineres
Herdados:	Não
Porcentagens:	Não aplicável
Animável:	Não

A propriedade de taquigrafia flexflow define a flex-direction e o flexwrap propriedades para definir o invólucro do contêiner flexível e os eixos principal e cruzado.

O valor padrão de flex-direction é row. O valor padrão de flexwrap é nowrap. Contanto que a exibição esteja configurada para flex ou inline-flex, omitir flexflow, flex-direction e flexwrap é o mesmo que declarar qualquer um dos três seguintes, que todos significam a mesma coisa:

```
flexflow: linha; flexflow: nowrap; flexflow: linha  
nowrap;
```

Nos modos de gravação da esquerda para a direita, declarar qualquer um dos valores de propriedade recém-listados ou omitir completamente a propriedade flexflow criará um contêiner flexível com um eixo principal horizontal que não é encapsulado, como mostra a [Figura 2-1](#). Esse não é o visual que você está procurando. flexflow pode ajudar. Mas você pode estar se perguntando por que estamos introduzindo uma propriedade abreviada antes de entender as propriedades do componente.

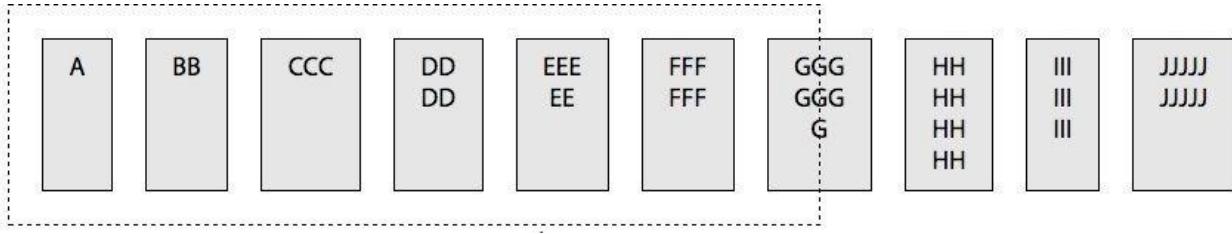


Figura 2-1. flexflow: linha;



Embora os autores da especificação incentivem o uso da taquigrafia flexflow, é necessário entender flexwrap e flex-direction, as duas propriedades que compõem essa taquigrafia. E, aprendendo sobre os valores que compõem a taquigrafia flexflow, aprenderemos a corrigir o layout desagradável mostrado na [Figura 2-1](#).

A propriedade flex-direction

Se você quiser que seu layout vá de cima para baixo, da esquerda para a direita, da direita para a esquerda ou até mesmo de baixo para cima, você pode usar a direção flexível para controlar o eixo principal ao longo do qual os itens flexíveis são dispostos.

FLEX-DIREÇÃO	
Valores	linha inverso de linha de coluna coluna-reverso
Valor inicial :	linha
Aplica-se um:	Contêineres
Herdados:	Não
Porcentagens:	Não aplicável
Animável:	Não

A propriedade flex-direction especifica como os itens flex são colocados no contêiner flex . Ele define o eixo principal de um contêiner flexível (veja "Entendendo eixos"), que é o eixo primário ao longo do qual os itens flexíveis são dispostos.

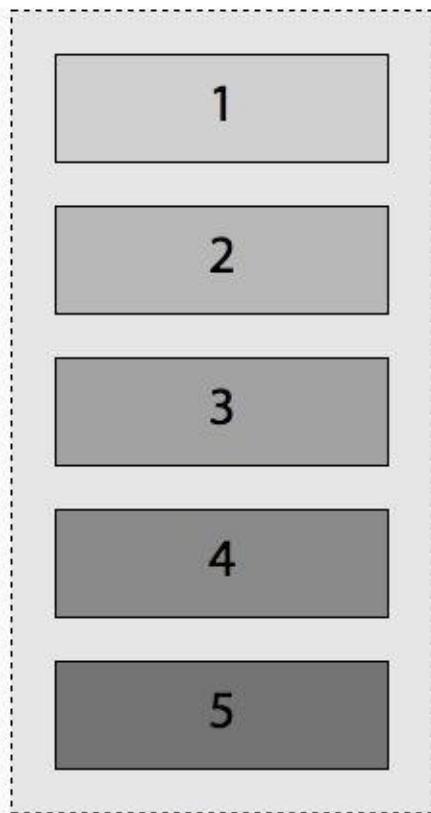
A [Figura 2-2](#) mostra os quatro valores de flex-direction, incluindo row, row-reverse, column e column-reverse em idiomas da esquerda para a direita. Observe que todas as propriedades flex discutidas aqui, como todas as propriedades CSS, aceitam os valores globais de inherit, initial e unset.



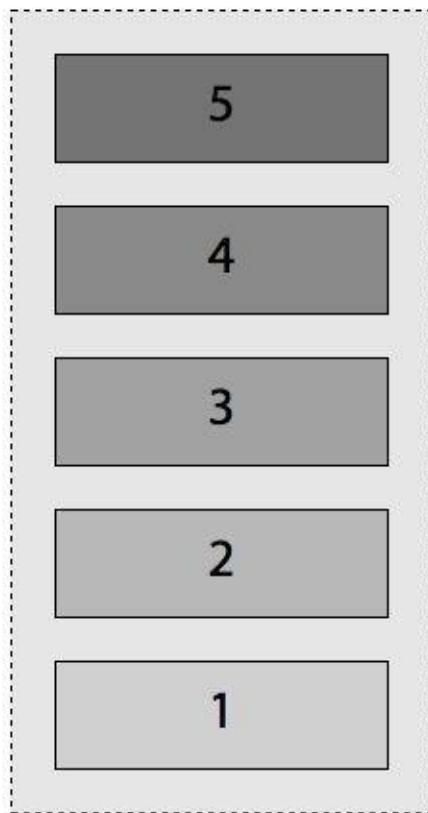
`flex-direction: row;` ↗



`flex-direction: row-reverse;` ↗



`flex-direction:`
`column;` ↗



`flex-direction:`
`column-reverse;` ↗

Figura 2-2. Os quatro valores da propriedade `flex-direction` quando o idioma está da esquerda para a direita

Especificamos idiomas da esquerda para a direita, porque a direção do eixo principal para a linha — a direção em que os itens flexíveis estão dispostos — é a direção do modo de gravação atual .

De preferência, deveríamos ter usado a propriedade taquigráfica `flexflow`. As duas colunas à direita na [Tabela 2-1](#) são equivalentes, com o valor `nowrap` sendo explicado na próxima seção.

Tabela 2-1. Os valores para equivalentes `flex-direction` e `flexflow`

flex-direção	flexflow de valor único	flexflow
remar	remar	linha nowrap
inverso de linha	inverso de linha	nowrap reverso de linha
coluna	coluna	coluna nowrap
coluna-reverso	coluna-reverso	coluna-reverso nowrap

IDIOMAS DA DIREITA PARA A ESQUERDA

Se você estiver criando Sites em inglês ou em outro idioma da esquerda para a direita (LTR), provavelmente desejará que SO itens flexíveis | sejam dispostos da esquerda | para um direita | e de cima | para baixo. A linha padrão ou de configuração | fará isso. Se você estiver escrevendo em árabe | ou em outro idioma da direita | para a esquerda, provavelmente deseja que SO itens flexíveis | sejam dispostos da direita | para um esquerda | (RTL) e de cima | para baixo. A linha padrão ou de configuração | fará isso por você também.

flex-direção: um linha colocará SO itens flexionar na mesma direção que a direção fazer texto |, também conhecida como *modo de gravação* /, seja uma linguagem RTL ou LTR. Enquanto um maioria dos Sites é apresentada em idiomas da esquerda para a direita , alguns sites estão em idiomas da direita para a esquerda e outros ainda são de cima para baixo. Com o flexbox, você pode incluir um único layout. Quando você altera o modo de gravação, o flexbox se encarrega de alterar um direção flexível | para você.

O modo de gravação | é set pelas propriedades | modo de escrita, direção e orientação texto ou pelo atributo Dir em HTML . Quando o modo de gravação está da direita para a esquerda — seja definido em HTML . com

Você pode inverter essa direção padrão com `flex-direction: row-reverse`.

Os itens flexíveis serão dispostos de cima para baixo quando a coluna flex-direction: estiver definida, e de baixo para cima se a flex-direction: column-reverse estiver definida, como mostra a [Figura 2-2](#). Observe se o valor de direção CSS for diferente do valor do atributo dir em um elemento, o valor da propriedade CSS terá precedência sobre o atributo HTML.

NOTA

Não usar flex-direção para alterar o layout para idiomas da direita para a esquerda. Em vez disso, usar o atributo Dir ou um propriedade | .CSS modo de escrita, que permite alternar entre horizontal e vertical para indicar um direção fazer idioma.

Nos modos de escrita horizontal, que inclui idiomas da esquerda para a direita e da direita para a esquerda, definindo flex-direction: row, flexflow: row, flexflow: row nowrap ou omitindo as propriedades longhand e shorthand para que o padrão seja row definirá todos os itens flexíveis horizontalmente, de um lado para o outro. Por padrão, todos eles serão alinhados horizontalmente, ao longo da linha do eixo principal, na ordem de origem. Nos idiomas da esquerda para a direita , eles serão alinhados da esquerda para a direita: o lado esquerdo é referido como *início principal* e o direito é o principal , para os pontos inicial e final do eixo principal. Nas linguagens da direita para a esquerda, a principal divisão é invertida: os itens flexíveis estão lado a lado da direita para a esquerda, com o eixo principal indo da esquerda para a direita, o lado direito sendo o principal início e o esquerdo sendo principal.

O valor inverter de linha é o mesmo que linha, exceto que os itens flexíveis são dispostos na direção oposta da direção do texto: os pontos inicial e final são invertidos. A direção do eixo principal do contêiner flexível é invertida pelo valor de linha inversa. Para idiomas da esquerda para a direita, como o inglês, o inverter de linha colocará todos os itens flexíveis de um lado para o outro da direita para a esquerda, horizontalmente, com o início principal à direita e o main-end agora no Esquerda. Eles são mostrados nos dois principais exemplos de imagem na [Figura 2-2](#).

Se a direção da página ou do contêiner flex tivesse sido invertida, como para hebraico ou árabe , com o atributo dir="rtl" no contêiner flex ou um ancestral no HTML, ou com a direção: rtl no contêiner flex ou o ancestral do contêiner no CSS, a direção do eixo principal invertido, e

portanto, os itens flex, seriam invertidos do padrão, indo da esquerda para a direita, como mostra a [Figura 2-3](#). Da mesma forma, a propriedade `writemode` afeta a direção na qual os itens flexíveis são atraídos para a página.

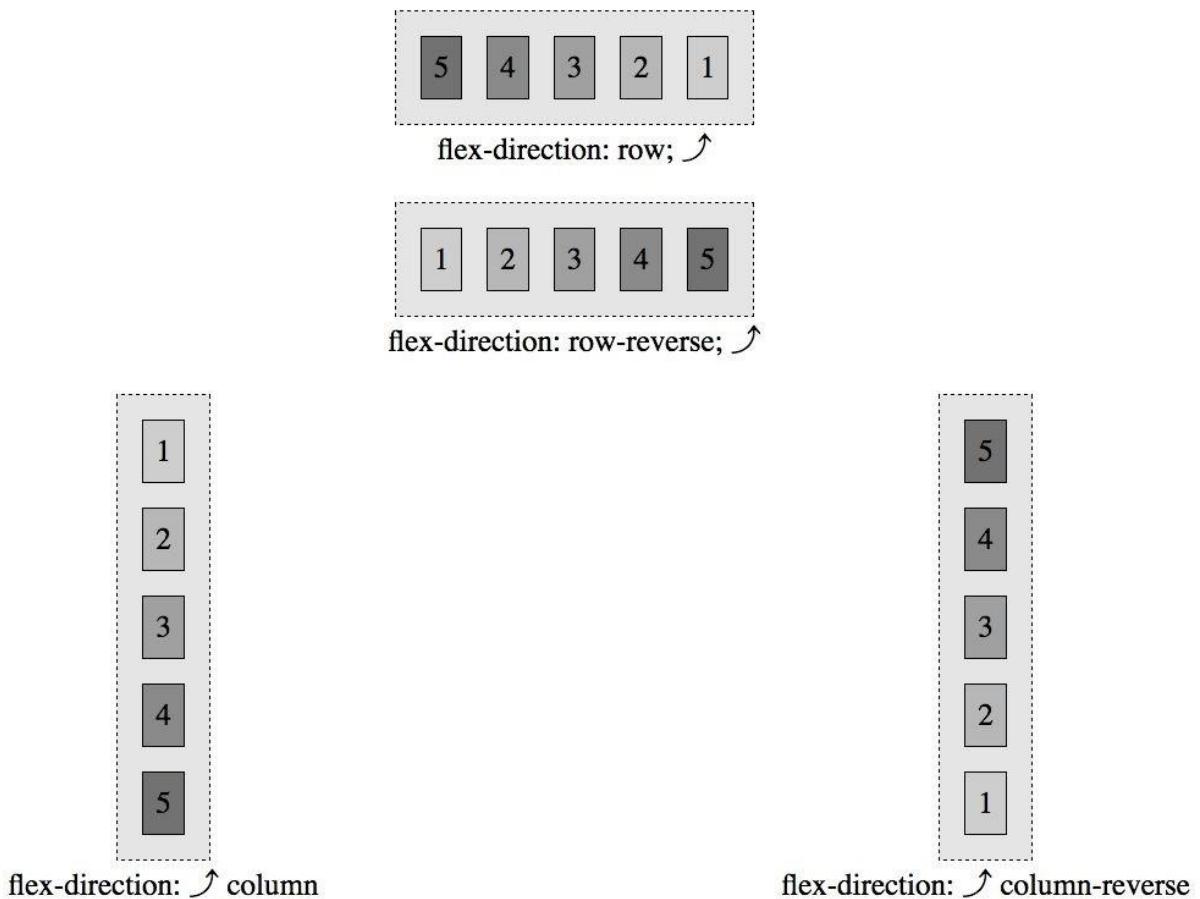


Figura 2-3. Os quatro valores da propriedade `flex-direction` quando a direção é da direita para a esquerda, demonstrados aqui

com visor: inline-flex

O valor da coluna irá dispor os itens flexíveis de cima para baixo. O valor da coluna define o eixo principal do contêiner flexível como a mesma orientação que o eixo do bloco do modo de gravação atual. Este é o eixo vertical nos modos de escrita horizontal e o eixo horizontal nos modos de escrita vertical. Basicamente, ele define o eixo principal do contêiner flexível como vertical na maioria dos casos.

Existem línguas escritas verticalmente, incluindo Bopomofo, hieróglifos egípcios, Hira gana, Katakana, Han, Hangul, Meroitic cursive e hieróglifos, Mongol, Ogham, Turco Antigo, Phags Pa, Yi, e às vezes

Japonês. Esses idiomas só são verticais quando um modo de gravação vertical é especificado. Se não for, então todas essas linguagens são horizontais. Se um modo de escrita vertical for especificado, todo o conteúdo será vertical, seja um dos idiomas escritos em inglês ou mesmo inglês.

NOTA

O modo de gravação controla a direção do fluxo do bloco: a direção na qual as linhas e os blocos são empilhados. O valor padrão, horizontal-tb, os empilha de cima para baixo. Os outros valores os empilham da direita para a esquerda ou da esquerda para a direita.

A direção controla a "direção de base" embutida, a direção na qual o conteúdo dentro de uma linha é ordenado. LTR, abreviação de "da esquerda para a direita", vai de "esquerda" nominal para "direita" nominal. RTL, abreviação de "da direita para a esquerda", é o oposto. Qual lado é nominalmente "esquerdo" para fins de direção é afetado pelo modo de escrita: se o modo de escrita é vertical, o lado "esquerdo" pode ser o topo!

A direção da base embutida é e deve ser sempre uma propriedade do conteúdo. Use o atributo *dir* em HTML. Não use a propriedade de direção CSS. O modo de escrita é uma preferência de layout.¹ Chinês, japonês e coreano podem ser escritos em qualquer orientação. Embora o inglês seja um idioma de cima para baixo, da esquerda para a direita, às vezes você verá e poderá até usar a escrita vertical para efeito estilístico.

Com a coluna, os itens flexíveis são exibidos na mesma ordem declarada no documento de origem, mas de cima para baixo em vez da esquerda para a direita, de modo que os itens flexíveis são dispostos um em cima do próximo em vez disso. de lado a lado:

```
nav {  
  display: flex;  
  flex-direção: coluna;  
  borda direita: 1px sólido #ccc;  
}  
a {  
  margem: 5px; preenchimento: 5px 15px; raio-  
  fronteira: 3px;  
  cor de fundo: #ccc; texto-decoração: nenhuma; cor:  
  preto;  
}  
a:hover, a:focus, a:active { background-color: #aaa; texto-  
  decoração: sublinhado;  
}
```

Usando uma marcação semelhante ao nosso exemplo de navegação anterior, simplesmente alterando algumas propriedades CSS, podemos criar uma boa navegação no estilo da barra lateral.

Para o novo layout da navegação, alteramos a direção flexível da linha padrão para a coluna, movemos a borda da parte inferior para a direita e alteramos as cores, o raio da borda e valores de margem, como visto na [Figura 2-4](#).



Figura 2-4. Alterar a direção flexível pode alterar completamente o layout do seu conteúdo

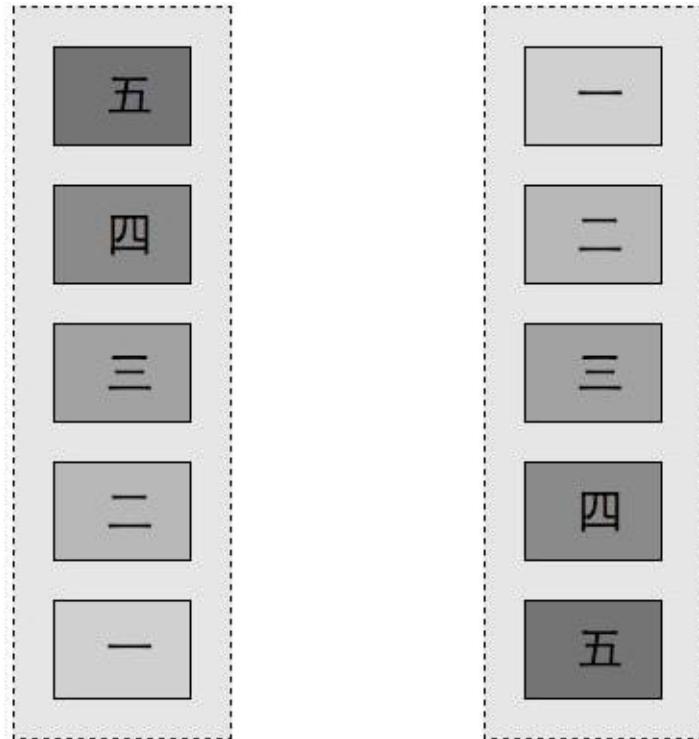


IDIOMAS DE CIMA PARA BAIXO

Pará idiomas | de cima para baixo |, como japonês |, ao escrever modo |: horizontal-tb é definido e suportado |, o eixo principal | é girado | 90 graus Não sentido horário | fazer padrão da esquerda | para um direita, então flex-direção: remar vai de cima | para baixo e flex-direção: a coluna prossegue | da direita | para um esquerda, como mostrado eme

[Figura 2-5](#).

Um propriedade | modo de escrita especifica a direção do fluxo fazer bloco |, definindo se as linhas de texto | estão dispostas horizontal ou verticalmente e a direção na qual os blocos progridem. Ele define a direção na qual os contêineres | em nível de bloco | são empilhados | e um direção na Qual o texto | e outros conteúdos em nível embutido flauta dentro de micrômetro contêiner de bloco |. É Apropriado usar o modo de escrita | para idiomas | de cima | para baixo e girar o texto. Não usar o modo de ~~gravacão para substituir um direcão simplesmente por motivos de layout. em vez disso usar um~~



`flex-direction: row-reverse;` ↗ `flex-direction: row;` ↗



`flex-direction: column;` ↗



`flex-direction: column-reverse;` ↗

Figura 2-5. Os quatro valores da propriedade flex-direction ao modo de gravação são horizontal-tb



O valor de coluna inversa é semelhante à coluna, exceto que o eixo principal é invertido, com o início principal na parte inferior e a extremidade principal no topo do eixo principal vertical, indo para cima, conforme disposto em o exemplo inferior direito na [Figura 2-3](#).

Os valores inversos apenas alteram a aparência. A ordem de fala e a ordem de tabulação permanecem as mesmas que a marcação subjacente.

O que aprendemos até agora é super poderoso e torna o layout uma brisa. Se incluirmos a navegação dentro de um documento completo, podemos ver como o layout pode ser simples com o flexbox.

Vamos expandir um pouco nosso exemplo HTML anterior e incluir a navegação como um componente dentro de uma home page:

```
<corpo>
  <cabeçalho>
    <h1>Título da minha página! </h1>
  </cabeçalho>
  <navegação>
    <a href="/">Home</a>
    <a href="/about">Sobre</a>
    <a href="/blog">Blog</a>
    <a href="/jobs">Carreiras</a>
    <a href="/contact">Fale Conosco</a>
  </navegação>
  <principal>
    <artigo>
      
      <p>Este é um conteúdo incrível que está na página. </p>
      <botão>Vá a algum lugar</botão>
    </artigo>
    <artigo>
      
      <p>Esta é mais conteúdo do que a caixa anterior, mas menos do que a próxima. </p>
      <botão>Clique em Mim</botão>
    </artigo>
    <artigo>
      
      <p>Temos muito conteúdo aqui para mostrar que O conteúdo pode crescer, e tudo pode ser do mesmo tamanho se você usar O flexbox. </p>
      <botão>Fazer Algo</botão>
    </artigo>
  </principal>
  <rodapé>Direitos autorais &#169; 2017</rodapé>
</corpo>
```

Simplesmente adicionando algumas linhas de CSS , temos uma página inicial bem definida, como

mostrado na **Figura 2-6**:

```
* {  
  contorno: 1px #ccc sólido;  
  margem: 10px;  
  preenchimento: 10px;  
}  
corpo, nav, main, artigo {  
  display: flex;  
}  
corpo, artigo {  
  flex-direção: coluna;  
}
```

Foram necessárias apenas duas declarações de propriedade/valor CSS para criar o layout básico de uma home page do site.

Obviamente, houve algum CSS adicional. Adicionamos borda, margem e preenchimento a todos os elementos para que você possa diferenciar os itens flexíveis por uma questão de aprendizado (eu não colocaria esse site menos atraente em producti). Caso contrário, tudo o que fizemos foi simplesmente declarar o corpo, a navegação, o principal e os artigos como contêineres flexíveis, fazendo com que todos os itens de navegação, links, principal, artigo, imagens, parágrafos e botões flexionem. Sim, os elementos podem ser tanto itens flexíveis quanto contêineres flexíveis, como vemos com a navegação, o principal e os artigos neste caso. O corpo e o artigos têm colunas definidas como suas direções flexíveis, e deixamos a navegação e o padrão principal para remar. Apenas duas linhas de CSS!

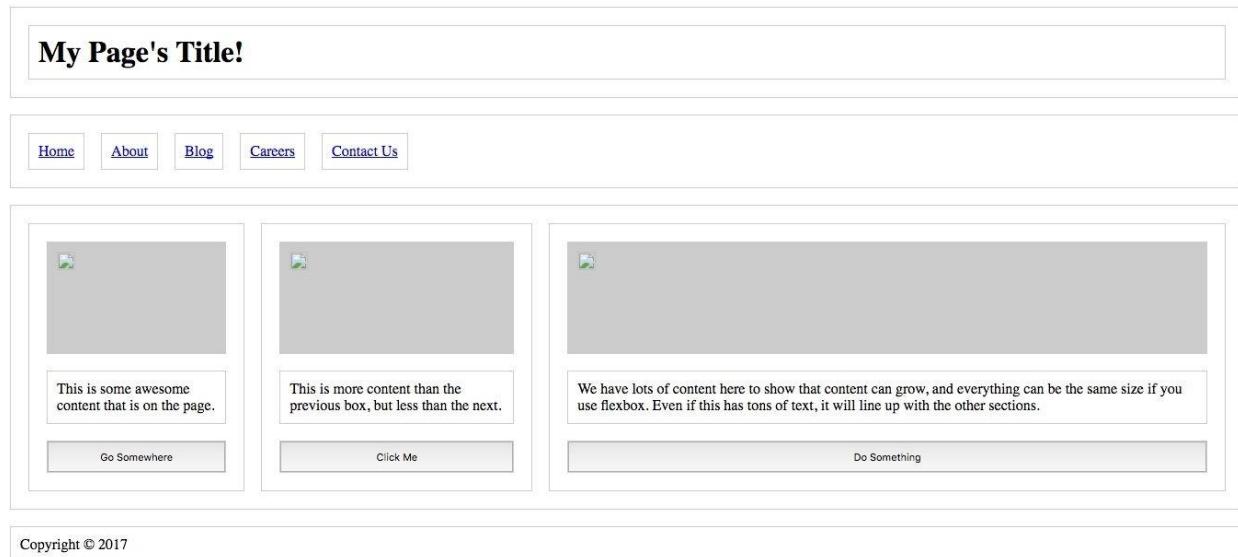


Figura 2-6. Layout da página inicial usando flex-direction: linha e coluna



Mas o que acontece quando a *dimensão principal* dos itens flexíveis (suas larguras combinadas para linha ou alturas combinadas para coluna) não se encaixam dentro do contêiner flex? Podemos fazê-los transbordar, como mostrado na [Figura 2-1](#), ou podemos permitir que eles se envolvam em linhas flexíveis adicionais.

A propriedade flexwrap

Até agora, os exemplos mostraram uma única linha ou coluna de itens flexíveis. Se as dimensões principais dos itens flexíveis não se encaixarem no eixo principal do contêiner flex, por padrão, os itens flexíveis não serão encapsulados. Em vez disso, os itens flex podem encolher se permitido fazê-lo através da propriedade flex do item flex (consulte "A propriedade flex") e/ou os itens flex podem estourar o recipiente delimitador caixa.

Você pode controlar esse comportamento. O flexwrap pode ser configurado no contêiner para permitir que os itens flex sejam encapsulados em várias linhas flex — linhas ou colunas de itens flex — em vez de ter itens flex transbordar o recipiente ou encolher à medida que permanecem em uma linha.

A propriedade flexwrap controla se o contêiner flex está limitado a ser um contêiner de linha única ou se pode se tornar multilinha, se necessário. Quando a propriedade flexwrap é definida para permitir várias linhas flexíveis, se o valor de wrap ou wrap-reverse é definido determina se quaisquer linhas adicionais aparecem antes ou depois da linha original de itens flexíveis.

FLEXWRAP	
Valores	nowrap de envoltório
Valor inicial :	nowrap
Aplica-se um:	Contêineres
Herdados:	Não
Porcentagens:	Não aplicável
Animável:	Não

Por padrão, não importa quantos itens flexíveis existam, todos os itens flexíveis são desenhados em uma única linha. Muitas vezes não é isso que queremos. É aí que o flexwrap entra em jogo. Os valores wrap e wrap-reverse permitem que os itens flex sejam encapsulados em linhas

flexíveis adicionais quando as restrições do contêiner flex pai forem Alcançado.

A Figura 2-7 demonstra os três valores da propriedade flexwrap quando o valor flex-direction está sendo padronizado como linha. Quando há duas ou mais linhas flexíveis , a segunda linha e as linhas flexíveis subsequentes são adicionadas na direção do eixo cruzado.

Se as linhas flexíveis adicionais são adicionadas acima ou abaixo, como no caso da Figura 2-7, ou à esquerda ou à direita da linha anterior é determinado pelo conjunto de seu wrap ou wrap-reverse, pelo valor flex-direction e pelo modo de gravação.

Geralmente para quebra automática, o eixo cruzado vai de cima para baixo para linha e linha reversa e a direção horizontal do idioma para coluna e coluna invertida. O valor de quebra automática de linha é semelhante ao de quebra em linha, exceto que linhas adicionais são adicionadas antes da linha atual e não depois dela.

Quando definido para wrap-reverse, a direção do eixo cruzado é invertida: as linhas subsequentes são desenhadas no topo no caso de flex-direction: linha e flex-direction: row-reverse e à esquerda da coluna anterior no caso de flex-direction: coluna e flex-direction: coluna-reverso . Da mesma forma, nos idiomas da direita para a esquerda, flexflow: row wrap-reverse e flexflow: row-reverse wrap-reverse, novas linhas também serão adicionadas na parte superior, mas para flexflow: column wrap-reverse e flexflow: column-reverse wrap-reverse novas linhas serão adicionadas à direita - o oposto da direção do idioma ou do modo de escrita, a direção do eixo cruzado invertido.

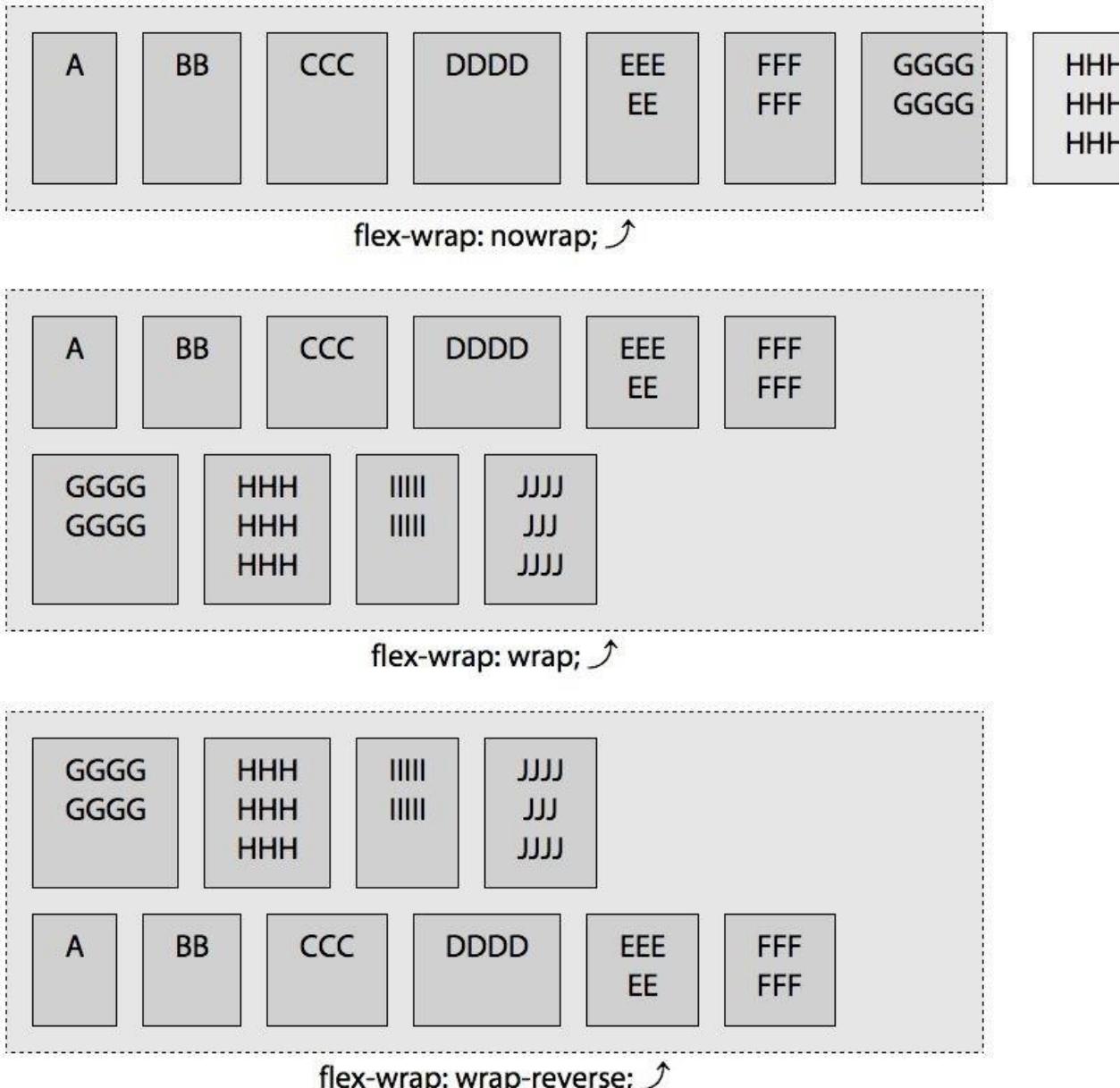


Figura 2-7. Os três valores da propriedade `flexwrap`



Você pode observar que, na [Figura 2-7](#), as novas linhas criadas nos exemplos `wrap` e `wrap-reverse` não têm a mesma altura da primeira linha. As linhas flexíveis são tão altas ou largas quanto o item flex mais alto ou mais largo dentro dessa linha. A dimensão principal da linha flex é a dimensão principal do contêiner flex, enquanto a dimensão cruzada da linha flex cresce para se adequar ao tamanho cruzado do item flex com o maior tamanho cruzado.

Embora os exemplos até agora sejam bastante simples, como você deve ter notado nos dois parágrafos anteriores, entender completamente todas as complexidades do flexbox é

não. A explicação anterior introduziu termos possivelmente confusos como eixo principal, início principal e extremidade principal e eixo cruzado, início cruzado e fim cruzado. Para entender completamente o flexbox e todas as propriedades do flexbox, é importante entender.

Para entender o layout flex, você precisa entender as direções físicas, eixos e tamanhos que afetam o contêiner flex e seus itens flex. Entender completamente o que a especificação significa quando usa esses termos tornará a compreensão completa do flexbox muito mais fácil.

Entendendo eixos

Os itens flexíveis são dispostos ao longo do eixo principal. Linhas flexíveis são adicionadas na direção do eixo cruzado. Os termos "principal" têm a ver com itens flexíveis. Os termos "cruzados" entram em jogo em contêineres flexíveis de várias linhas: quando wrap ou wrap-reverse é definido e os itens flexíveis realmente se envolvem em mais de uma linha.

Até introduzirmos o flexwrap, todos os exemplos tinham uma única linha de itens flex. Essa única linha de itens flexíveis envolvia o layout dos itens flexíveis ao longo do eixo principal, na direção principal, do início principal ao fim principal. Dependendo da propriedade flex-direction, esses itens flex foram dispostos lado a lado, de cima para baixo ou de baixo para cima, em uma linha ou coluna ao longo da direção do eixo principal.

A [Tabela 2-2](#) resume os termos "principal" e "cruzado". Ele lista as dimensões e direções do eixo principal e do eixo cruzado, juntamente com seu ponto inicial, pontos finais e direções para layouts de modo de gravação da esquerda para a direita. Além de descrever as dimensões e a direção do contêiner flex, esses termos podem ser usados para descrever a direção e a dimensão de itens flex individuais.

Tabela 2-2. Dimensões e direções dos eixos principal e cruzado, juntamente com seu ponto inicial, pontos finais e direções no layout da esquerda para a direita

Flexe direções nos modos de gravação da esquerda para a direita (LTR)				
	remar	inverso	de coluna	coluna-reverso
eixo principal	da esquerda para a direita	da para a esquerda	direita de cima para a baixo	de baixo para cima
dimensão principal	horizontal	horizontal	vertical	vertical
início principal	Esquerda	Direita	Início	fundo
extremidade principal	Direita	Esquerda	fundo	Início

tamanho principal	Largura	Largura	altura	altura
dimensão cruzada	vertical	vertical	horizontal	horizontal
início cruzado	Início	Início	Esquerda	Esquerda
cross-end	fundo	fundo	Direita	Direita
tamanho cruzado	altura	altura	Largura	Largura

No caso da linha flexflow: (Figura 2-1), a soma dos tamanhos principais da linha não inwrappable de itens flex foi maior do que o tamanho principal do contêiner pai flex. Em inglês mais simples, as larguras combinadas (e margens horizontais não colapsadas) dos itens flex eram mais largas do que a largura do contêiner flex.

Nos modos de gravação horizontal, para linha e inversão de linha, o tamanho principal refere-se às *larguras* dos itens flexíveis e do contêiner flex. No caso da coluna e do inverso da coluna, o tamanho principal é a altura.

A Figura 2-8 mostra os eixos e a direção de cada eixo para contêineres flexíveis com uma direção flexível de linha, linha reversa, coluna e coluna reversa para os modos de gravação LTR.

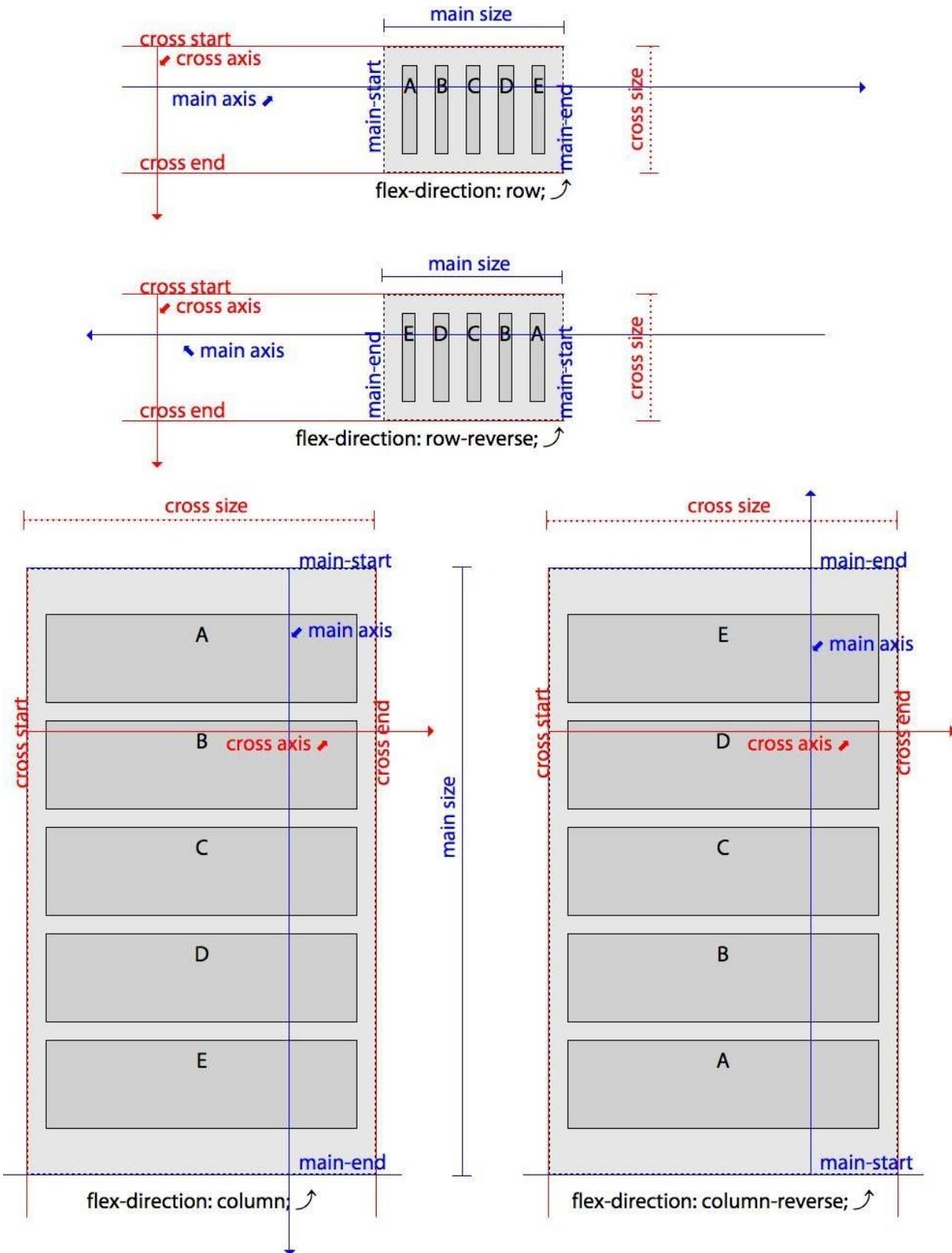


Figura 2-8. Eixos para linha, invertimento de linha, coluna e inverso de coluna em idiomas LTR

O eixo principal é o eixo primário ao longo do qual os itens flexíveis são dispostos, com os itens flexíveis sendo desenhados ao longo do eixo principal na direção do início principal ao fim principal.

Se todos os modos de escrita fossem da esquerda para a direita e de cima para baixo, como em inglês, a flex-direction seria menos complexa. Para as linguagens RTL e LTR, flex-direction: row e flex-direction: row-reverse, o eixo principal é horizontal. Para idiomas LTR, no caso de linha, o início principal está à esquerda e o fim principal está à direita. Eles são invertidos para inverter a linha, com o main-start mudando para a direita e o main-end agora à esquerda. Para a coluna, o eixo principal é vertical, com o início principal na parte superior e o eixo principal na parte inferior. Para coluna-reverso, o eixo principal também é vertical, mas com o início principal na parte inferior e o eixo principal está na parte superior.

Como mostrado na [Tabela 2-3](#), quando o modo de escrita estiver da direita para a esquerda, como em árabe e hebraico, se você tiver dir="rtl" definido em seu HTML, ou direção: rtl definido em seu CSS, a direção do texto irá da direita para a esquerda. Nesses casos, a direção da linha é invertida para ser da direita para a esquerda, com o início principal à direita e o fim principal à esquerda. Da mesma forma, o eixo principal para o inverso da linha irá da esquerda para a direita, com o início principal à direita. Para a coluna, o início principal ainda está na parte superior e o main-end está na parte inferior, assim como para os idiomas da esquerda para a direita, pois ambos são idiomas de cima para baixo.

Tabela 2-3. Dimensões e direções dos eixos principal e cruzado, juntamente com seus pontos de início, pontos finais e direções quando o modo de gravação está da direita para a esquerda

Direções flexíveis nos modos de gravação RTL

	remar	inverso	de coluna	coluna-reverso
dimensão principal	horizontal	horizontal	vertical	vertical
início principal	Direita	Esquerda	Início	fundo
extremidade principal	Esquerda	Direita	fundo	Início
tamanho principal	Largura	Largura	altura	altura
dimensão cruzada	vertical	vertical	horizontal	horizontal
início cruzado	Início	Início	Direita	Direita

cross-end	fundo	fundo	Esquerda	Esquerda
tamanho cruzado	altura	altura	Largura	Largura

Se o seu site tiver o modo de escrita: horizontal-tb definido, como na [Figura 2-5](#), o eixo principal do conteúdo para linha e linha reversa será vertical, enquanto coluna e coluna-reverso são horizontais. A propriedade writingmode está começando a receber suporte em navegadores, com suporte no Edge e Firefox, suporte prefixado no Chrome, Safari, Android e Opera e suporte para uma sintaxe mais antiga no Internet Explorer.

NOTA

Embora a propriedade de direção CSS | junto com unicode-bidi possa ser usada para controlar a direção fazer texto, não faça isso. É recomendável usar o atributo dir e a propriedade CSS em modo de gravação porque o atributo Dir do HTML diz respeito ao conteúdo .HTML e um propriedade | modo de escrita diz respeito ao layout.

É importante entender que as coisas se invertem quando a direção da escrita é invertida. Agora que você entende que, para tornar a explicação (e a compreensão) do layout flexível muito mais simples, faremos com que o resto das explicações e exemplos sejam todos baseados no modo de escrita da esquerda para a direita, mas incluiremos como o modo de escrita impacta as propriedades flexíveis e os recursos discutidos.

A forma como o layout flexível aparece na tela é determinada em parte pelas interações entre o flexflow — que inclui flex-direction e flexwrap — e o modo de gravação. Abordamos a direção em que os itens flex são adicionados a uma linha de itens flex, mas quando o fim da linha flex é atingido, como novas linhas flex são adicionadas?

Ao pensar em flex-direction, agora sabemos que os itens flex começarão a ser dispostos em todo o eixo principal do contêiner flex, a partir do início principal. As direções "cruzadas" entram em jogo quando se trata de adicionar linhas adicionais de itens flexíveis , conhecidas como *linhas* flexíveis. Quando a propriedade flexwrap é usada para permitir que o contêiner seja encapsulado se os itens flex não se encaixarem em uma linha, as direções *cruzadas* determinam a direção de linhas adicionais em contêineres flex de várias linhas.

Enquanto o layout dos itens flexíveis em cada linha flexível é feito na direção principal, indo do início principal ao fim principal, o envoltório para linhas adicionais é feito ao longo da direção transversal, do início cruzado ao fim cruzado.

O eixo transversal é perpendicular ao eixo principal. Como vemos na [Figura 2-9](#), quando temos linhas horizontais de itens flexíveis, o eixo cruzado é vertical. Linhas flexíveis são

adicionadas na direção do eixo cruzado. Nestes exemplos, com flexflow: row wrap e flexflow: row-reverse wrap definido em linguagens horizontal, novas linhas flex são adicionadas abaixo das linhas flex anteriores.

O tamanho cruzado é o oposto do tamanho principal, sendo altura para linha e linha inversa e largura para coluna e coluna reversa em ambos os idiomas RTL e LTR . As linhas Flex são preenchidas com itens e colocadas no recipiente, com linhas adicionadas começando no lado de partida cruzada do contêiner flex e indo em direção ao lado da extremidade cruzada.

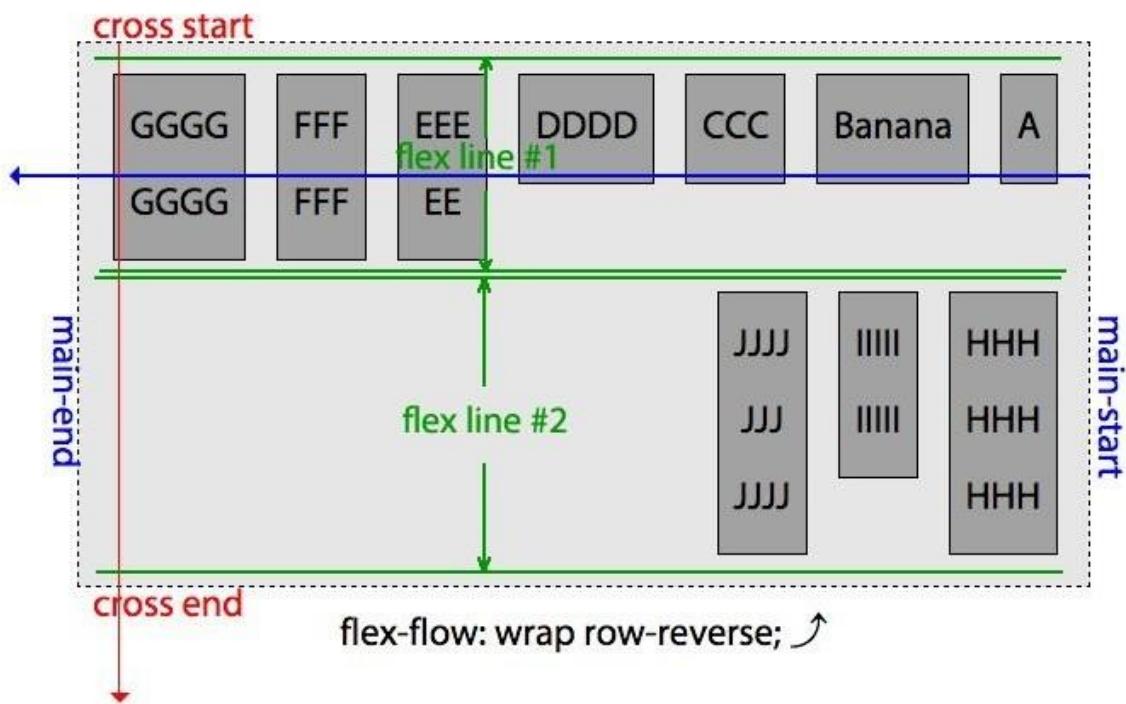
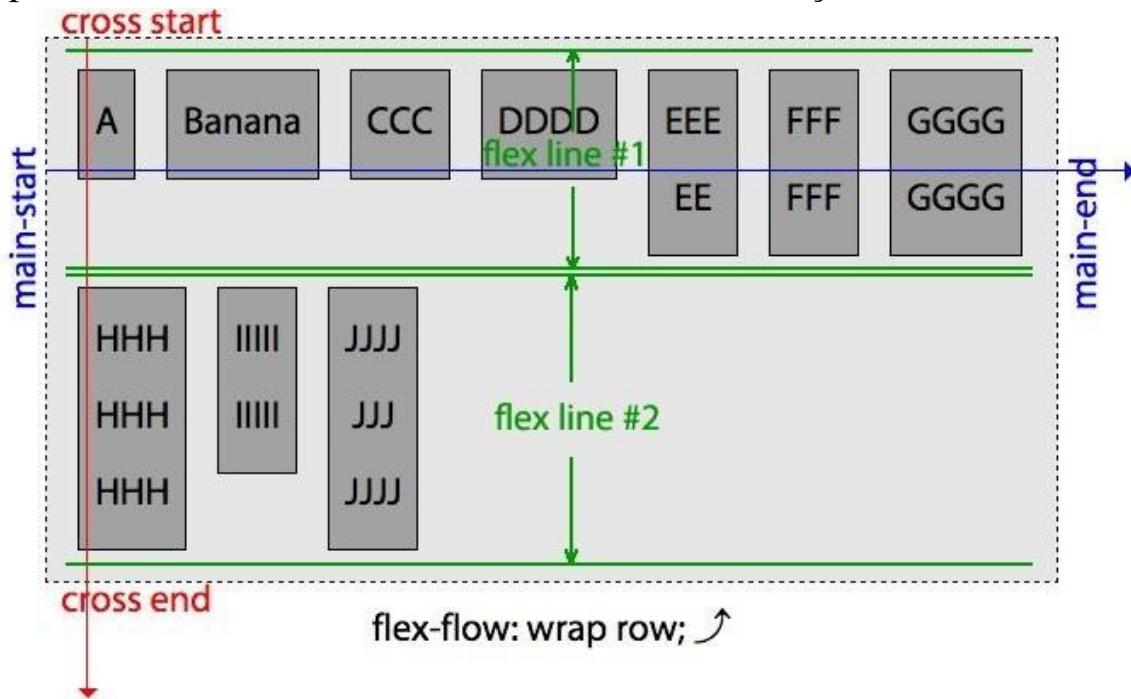


Figura 2-9. Flexibilizar linhas na linha e inverter a linha quando o flexwrap: o wrap está definido

O valor wrap-reverse inverte a direção do eixo cruzado. Normalmente para flex-direção da linha e linha-ré, o eixo cruzado vai de cima para cima para

abaixo, com o início cruzado na parte superior e o cross-end na parte inferior, como mostra a [Figura 2-9](#). Quando o flexwrap é definido para wrap-reverse, as direções cross-start e cross-end são trocadas, com o cross-start na parte inferior, cross-end na parte superior e o cross-axis indo de baixo para cima, como mostra a [Figura 2-7](#). Linhas flexíveis adicionais são adicionadas em cima ou acima da linha anterior.

Se a direção flexível estiver definida como coluna ou coluna invertida, por padrão, o eixo transversal vai da esquerda para a direita nos idiomas da esquerda para a direita, com novas linhas flexíveis sendo adicionadas à direita das linhas anteriores. Como mostrado na [Figura 2-10](#), quando o flexwrap é definido para wrap-reverse, o eixo cruzado é invertido, com o cross-start sendo à direita, o cross-end sendo à esquerda, o cross-axis indo da direita para a esquerda, com linhas flexíveis adicionais sendo adicionadas à esquerda de a linha previamente traçada.

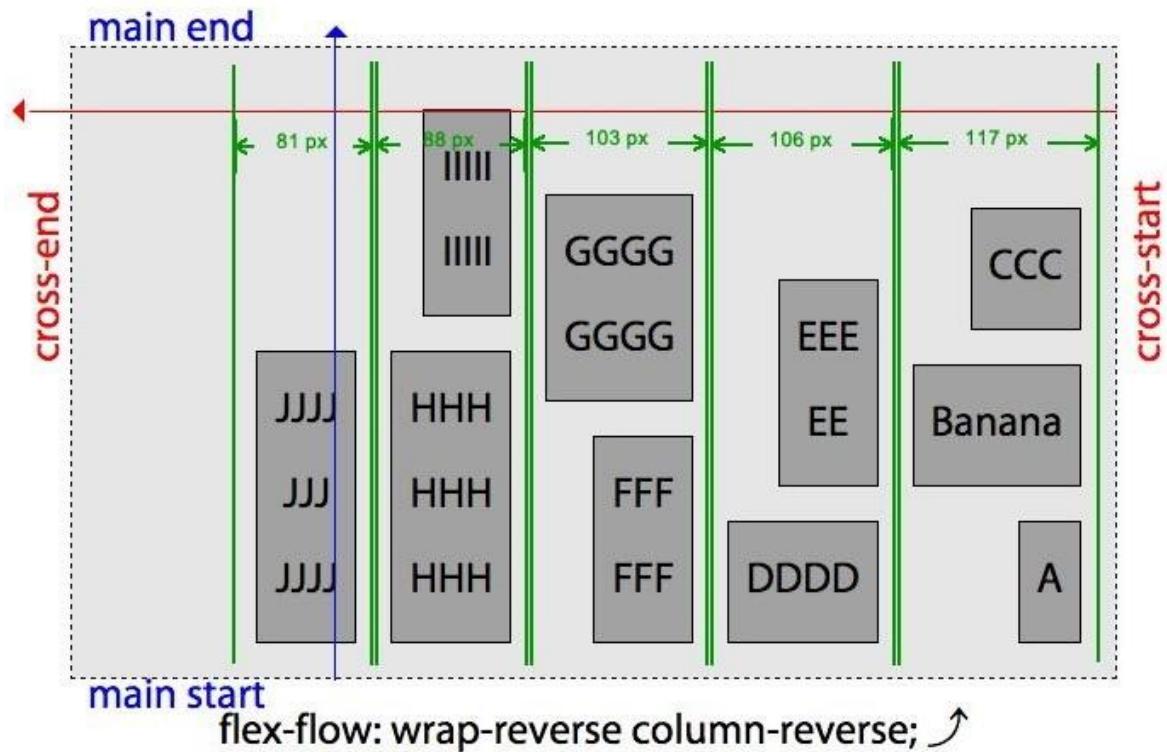
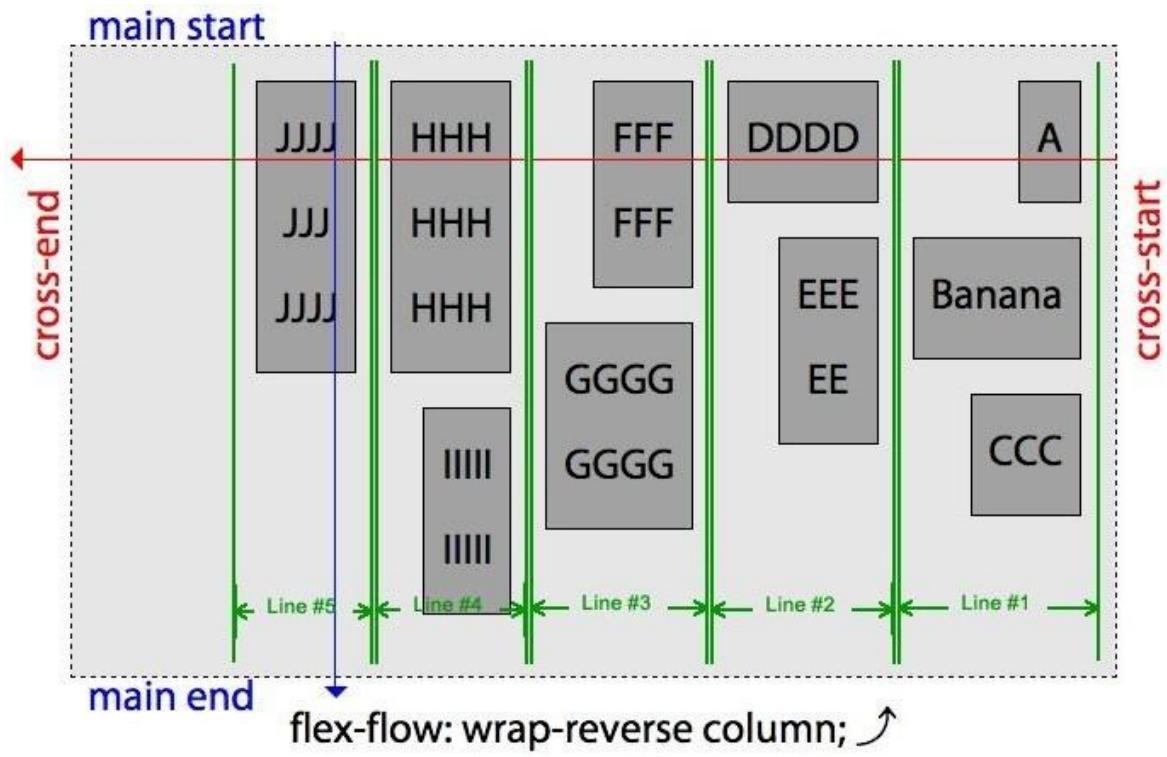


Figura 2-10. Linhas flexíveis na coluna e coluna-reverso quando `flexwrap: wrap-reverse` é definido

NOTA

Adicionamos align-items: flex-start (consulte "The align-items Property") e align-content: flex-start (consulte ["O alinhamento de conteúdo Propriedade"](#)) ao contêiner flexionar nas Figuras 2-9 e [2-10](#) para enunciar um altura e direções das linhas | flexíveis |. Essas propriedades | são abordadas | nas Suas presenças um seguir.

A propriedade flexwrap parecia bastante intuitiva quando foi descrita pela primeira vez. Acabou por ser um pouco mais complexo do que poderia ter parecido originalmente. Você pode nunca implementar o empacotamento reverso em uma linguagem da direita para a esquerda, mas este é um "Guia Definitivo". Agora que temos uma melhor compreensão das dimensões "cruzadas", vamos nos aprofundar na propriedade flexwrap.

flexwrap continuou

O valor padrão de nowrap impede o empacotamento, portanto, as direções cruzadas que acabamos de discutir não são relevantes quando não há chance de uma segunda linha flexível. Quando linhas adicionais são possíveis - quando flexwrap é definido para wrap ou wrap-reverse - essas linhas serão adicionadas na direção transversal, que é perpendicular ao eixo principal. A primeira linha começa no início cruzado com linhas adicionais sendo adicionadas no lado da extremidade cruzada.

O valor wrap-reverse inverte a direção do eixo cruzado. Normalmente, para flex-direção de linha e linha-ré, o eixo cruzado vai de cima para baixo, com o cross-start em cima e cross-end na parte inferior. Quando o flexwrap é wrap-reverse, as direções cross-start e cross-end são trocadas, com o cross-start na parte inferior, cross-end na parte superior e o cross-axis indo de baixo para cima. Linhas flexíveis adicionais são adicionadas sobre a linha anterior.

Você pode inverter a direção do eixo cruzado, adicionando novas linhas na parte superior ou à esquerda das linhas anteriores, incluindo flexwrap: wrap-reverse. Nas Figuras 2-9, 2-10 e [2-11](#), o [último exemplo em cada uma delas é o wrap-reverse](#). Você notará que a nova linha começa no início principal, mas é adicionada na direção inversa do eixo cruzado definido pela propriedade flex-direction.

Na [Figura 2-11](#), os mesmos valores de flexwrap são repetidos, mas com um valor de propriedade de coluna flex-direction: em vez de linha. Nesse caso, os itens flexíveis são dispostos ao longo do eixo vertical. Assim como no primeiro exemplo da [Figura 2-7](#), se

wrapping não é habilitado pela propriedade flexwrap, porque flexwrap: nowrap é explicitamente definido no contêiner ou se a propriedade for omitida e o padrão for nowrap, nenhuma nova linha flex será adicionada, mesmo que isso signifique que os itens flex sejam desenhados além da caixa delimitadora do contêiner flex.

Com coluna, assim como com linha, se os itens flex não se encaixarem na dimensão principal do contêiner flex, eles estourarão o contêiner flex, a menos que explicitamente forçados com largura mínima: 0 ou similar, caso em que encolherão para ajuste, embora os itens flexíveis não diminuam para menores do que suas bordas, preenchimento e margens combinadas.

Quando "flex-direção: coluna; flexwrap: envoltório; " ou "flexflow: envoltório de coluna; " é definido em um contêiner flex, os filhos do item flex são alinhados ao longo do eixo principal. Nos mods LTR, o primeiro item flexível é colocado no canto superior esquerdo, que é o main-start e o cross-start, respectivamente. Se houver espaço, o próximo item será colocado abaixo dele, ao longo do eixo principal. Se não houver espaço suficiente, o contêiner flex envolverá os itens flex em novas linhas. O próximo item flex será colocado em uma nova linha, que neste caso é uma linha vertical à direita da linha anterior, como pode ser observado no flexflow: column wrap exemplo, o exemplo superior direito na [Figura 2-1 1](#).

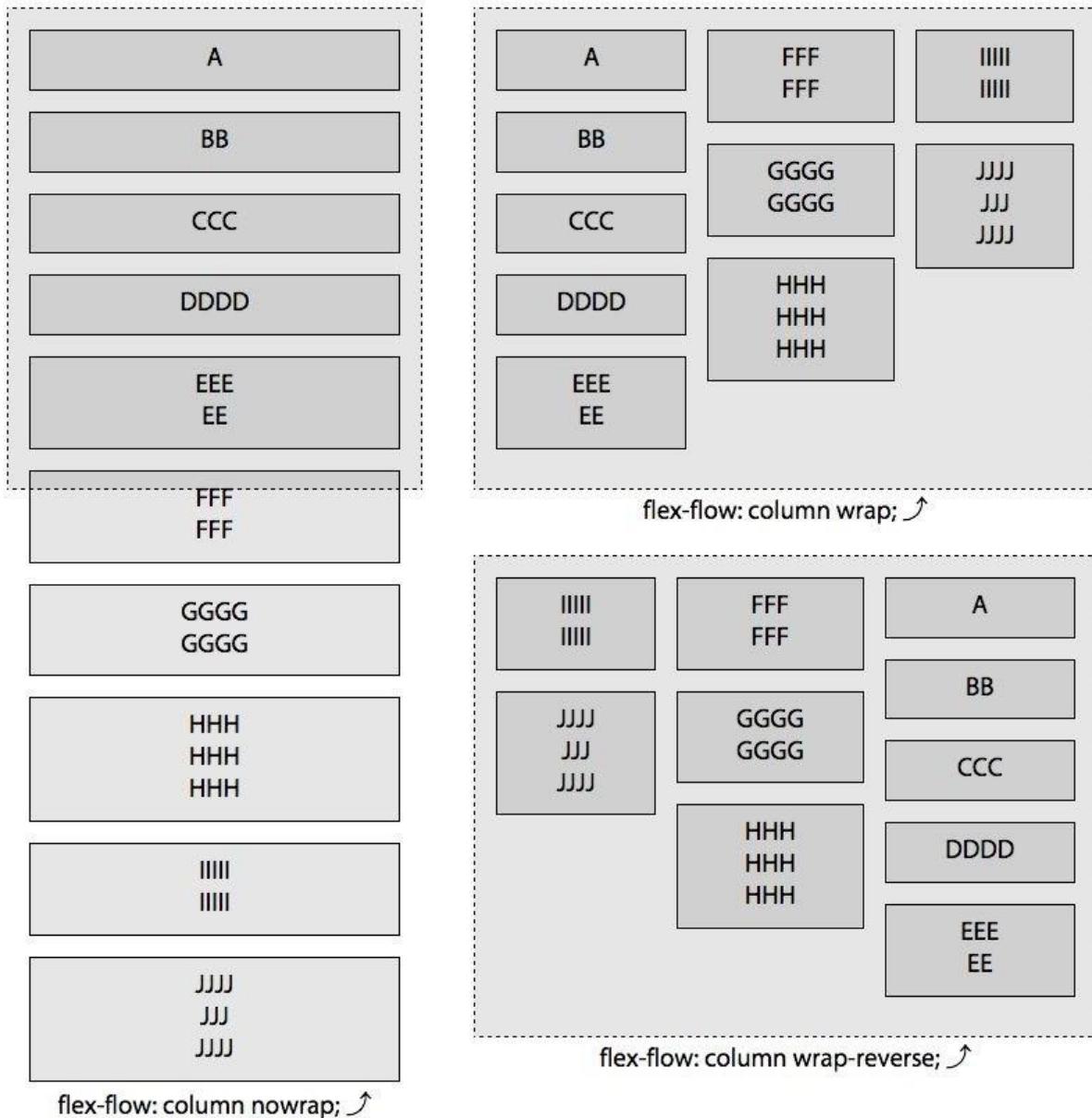


Figura 2-11. Os três valores da propriedade `flexwrap` com uma coluna `flex-direction`



Aqui, os itens flexíveis foram empacotados em 3 linhas. Os itens flexíveis estão se envolvendo em novas linhas como a altura: 440px foi definido no pai. A inclusão de uma altura força a criação de novas linhas flexíveis quando o próximo item flexível não se encaixa na linha flexível atual.

Como mostrado no exemplo inferior direito na [Figura 2-11](#), quando incluímos "flex-direction: column; flex-wrap: wrap-reverso;" ou "flexflow: coluna

envoltório-reverso; " e o eixo principal é o eixo vertical, os itens flexíveis são adicionados abaixo do item flex anterior, se puderem caber dentro do contêiner pai. Devido ao wrap-reverse, o eixo cruzado é invertido, o que significa que novas linhas são adicionadas na direção oposta do modo de gravação. Como mostrado neste exemplo e na [Figura 2-10](#), com o wrap-reverse, as colunas são dispostas da direita para a esquerda, em vez da esquerda para a direita. A primeira coluna de itens flex está no lado esquerdo do contêiner flex pai. Quaisquer colunas adicionais necessárias serão adicionadas à esquerda da coluna anterior. Novamente, estamos assumindo um modo de gravação padrão da esquerda para a direita para todos os exemplos.

Neste exemplo, os itens flexíveis têm alturas diferentes com base em seu conteúdo. Quando dividida em três linhas, a última linha não é preenchida. Como as linhas subsequentes estão sendo adicionadas à esquerda das linhas anteriores, o espaço vazio, se houver, estará no canto inferior esquerdo (supondo que o conteúdo justificado, descrito a seguir, seja definido ou padrão para flex-start).

Como você pode ver, flex-direction e flex-wrap têm grande impacto no seu layout e um no outro. Como geralmente é importante definir ambos se você for definir qualquer um deles, recebemos a propriedade flexflow , que é simplesmente uma abreviação de flex-direction e flex-wrap .

Dimensão cruzada da linha flexível

Semelhante aos exemplos de exibição na Figura 1-6, no exemplo de flexflow na [Figura 2-1](#), os itens flex cresceram para preencher a dimensão cruzada da linha flex em que estavam em. Como o padrão é nowrap, todos os itens flexíveis estarão em uma única linha. Com outras propriedades flex padronizadas para seus valores padrão, os itens flex são esticados enchendo o recipiente.

No exemplo flexwrap da [Figura 2-7](#), tivemos duas linhas flex. Todos os itens flex em uma única linha tinham a mesma altura, mas as linhas flex individuais não eram da mesma altura: em vez disso , todos os itens flex eram como alto como o item flex mais alto dentro dessa mesma linha flex.

Por padrão, todos os itens flexíveis parecerão ter a mesma altura. Controlamos isso nas Figuras 2-9 e [2-10](#), a fim de destacar a forma como as linhas flexíveis são desenhadas. Esse alongamento (ou substituição desse alongamento) é causado pelo valor padrão de alongamento da propriedade the justify-content, descrito na próxima seção.

Nesses exemplos, você notará que a segunda linha não é tão larga quanto o contêiner flex. Quando os itens flexíveis têm permissão para quebrar em várias linhas, por padrão, todos os itens em uma linha terão a mesma dimensão cruzada, o que significa que todos os itens em uma

linha flexível de linha e inverter linha terão a mesma altura e todos os itens em um a linha flexível da coluna ou a coluna-reversa terão a mesma largura.

A Figura 2-12 é um exemplo em que flexflow: column-reverse wrap-reverse é definido. Isso significa que o eixo principal é vertical, indo de baixo para cima com um eixo transversal horizontal indo da direita para a esquerda. Observe o espaço extra no canto superior esquerdo. Novos itens flexíveis são colocados na parte superior dos anteriores, com novas linhas sendo encapsuladas à esquerda da linha preenchida anteriormente. Por padrão, não importa os valores de flexflow, o espaço vazio, se houver algum, estará na direção de main-end e cross-end. Existem outras propriedades flexíveis que nos permitirão alterar isso.

Vejamos isso.

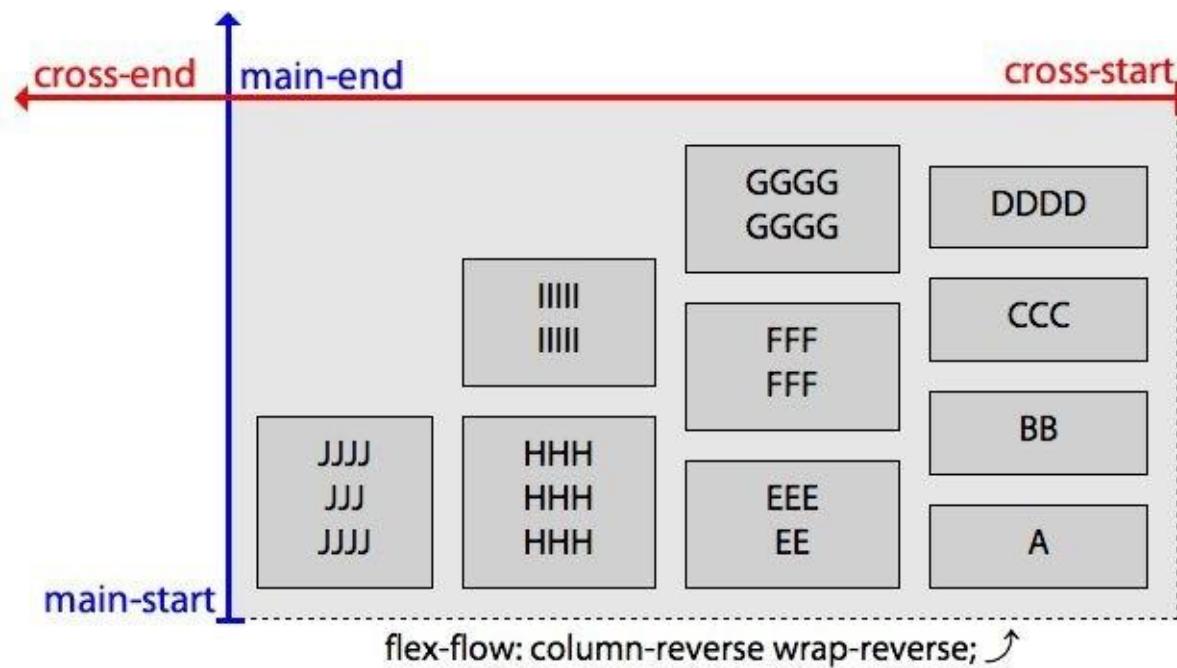


Figura 2-12. Não importa o valor do flexflow, o espaço vazio estará na direção da extremidade principal e da extremidade cruzada

Contêiner Flex

No [Capítulo 1](#), aprendemos a usar os valores de propriedade de exibição flex e inline-flex para instanciar um contêiner flex e transformar os filhos do contêiner em itens flex. Agora entendemos como a flex-direction define a direção dos itens flex dentro do contêiner flex, definindo as direções do eixo principal e do eixo cruzado e como o flexwrap pode ser usado para habilitar contêineres flexíveis de várias linhas e até mesmo inverter a direção do eixo cruzado. Também sabemos como usar o flexflow para definir tanto o flex-direction quanto o flexwrap.

Embora tenhamos aprendido a usar as propriedades flexflow para lidar com contêineres flex de várias linhas, não discutimos realmente o que acontece com o espaço extra quando os itens flexíveis não preenchem um linha ou coluna e o que acontece com o espaço extra quando nem todos os itens flexíveis têm a mesma dimensão cruzada.

Até agora, em nossos exemplos, quando os itens flex não encheram o contêiner flex, os itens flex foram agrupados em direção ao início principal no eixo principal. Podemos controlar isso. Os itens flexíveis podem ser nivelados contra a extremidade principal. Eles podem ser centrados. Podemos até espaçar os itens flexíveis uniformemente em todo o eixo principal.

A especificação de layout flex nos fornece propriedades de contêiner flexível para controlar a distribuição de espaço: além de exibição e fluxo flexível, as propriedades CSS Flexible Box Layout Module Nível 1 aplicadas ao flex os contêineres incluem as propriedades justify-content, aligncontent e alignitems.

A propriedade justify-content controla como os itens flexíveis em uma linha flexível são distribuídos ao longo do eixo principal. O aligncontent define como as linhas flexíveis são distribuídas ao longo do eixo transversal do contêiner flex. A propriedade alignitems define como os itens flex são distribuídos ao longo do eixo cruzado de cada uma dessas linhas flex.

As propriedades aplicadas a itens flexíveis individuais são discutidas no [Capítulo 3](#).

A propriedade justify-content

A propriedade justify-content nos permite definir como os itens flexíveis serão distribuídos ao longo do eixo principal do contêiner flex.

JUSTIFICAR-	
Valores	flex-start flex-end centro espaço entre espaço ao redor
Valor inicial :	flex-start
Aplica-se um:	Contêineres
Herdados:	Não
Porcentagens:	Não aplicável
Animável:	Não

A distribuição de elementos ao longo do eixo principal do contêiner é controlada com a propriedade justify-content. Se você se lembrar da [Figura 1-6](#), quando um elemento é convertido em um contêiner flex, os itens flex, por padrão, foram agrupados no início principal.

O conteúdo justificado define como o espaço é distribuído. Há cinco valores para a propriedade justify-content: flex-start, o valor padrão, e flex-end, center, space-between e space-around.

Como mostrado na [Figura 2-13](#), por padrão ou com o conteúdo justificado: flex-start explicitamente definido, os itens flexíveis são dispostos nivelados contra o main-start. Com o flex-end, os itens flexíveis são justificados para o main-end. agrupa os itens centralizados uns contra os outros centrados no meio da dimensão principal ao longo do eixo principal. O valor de espaço entre coloca o primeiro item flex em uma linha flexível nivelada com main-start e o último item flex em cada linha flex flush com main-end, e em seguida, coloca uma quantidade igual de espaço entre cada par de itens flexíveis adjacentes. space-around distribui uniformemente os itens flexíveis, como se houvesse margens não colapsantes de igual

tamanho em torno de cada item. Mais exemplos dos cinco valores de conteúdo justificado são demonstrados na [Figura 2-14](#).

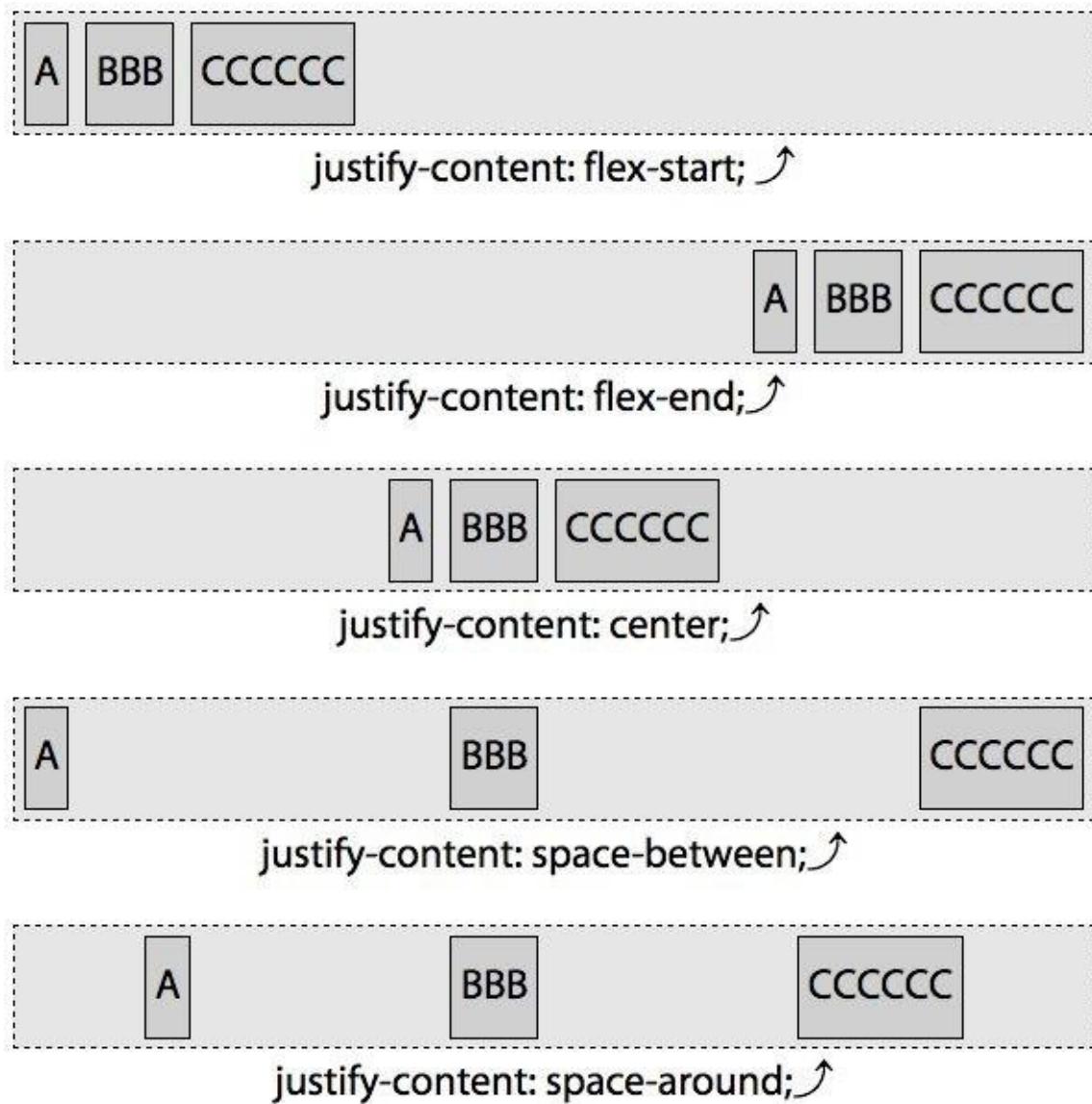


Figura 2-13. Os cinco valores da propriedade `justify-content`



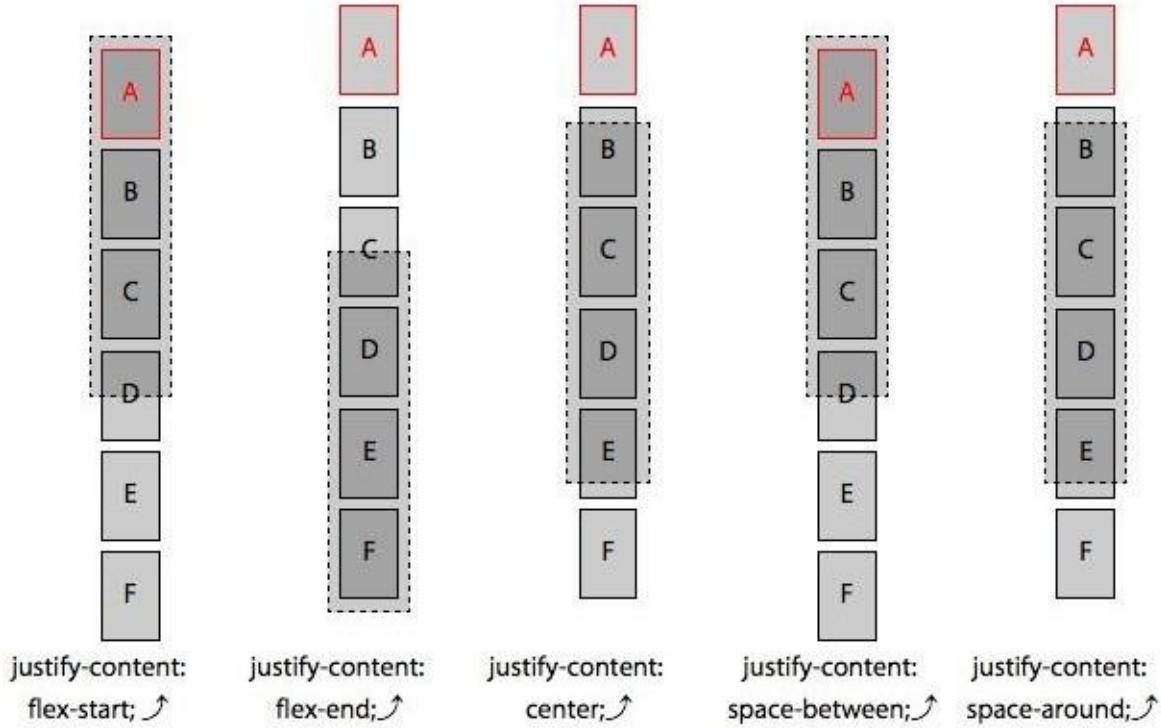


Figura 2-14. A direção de transbordamento em um contêiner flexível de linha única depende do valor da propriedade content 

À medida que a propriedade justify-content é aplicada ao contêiner flex, os itens flex serão distribuídos da mesma maneira, estejam eles em uma primeira linha flex preenchida ou em uma linha flexível subsequente parcialmente preenchida em um recipiente flexível de embalagem .

Esses são os conceitos básicos da propriedade justify-content. Mas, é claro, há mais na propriedade e em cada um dos valores. O que acontece se houver apenas um item flexível em uma linha? Se o modo de escrita estiver da direita para a esquerda? Se o flexflow: nowrap estiver definido e os itens flex transbordarem o contêiner flex?

Se nowrap estiver definido e os itens estourarem a linha, a propriedade justify-content ajudará a controlar a aparência do estouro de linha. A [Figura 2-14](#) ilustra o que acontece com os diferentes justify-content quando o flexflow: a coluna nowrap é definida e as alturas dos itens flexíveis são mais altas do que a dimensão principal do contêiner.

Vamos dar uma olhada nos cinco valores.

Setting `justify-content: flex-start` ([Figura 2-15](#)) define explicitamente o comportamento padrão de agrupar os itens flex em direção ao `main-start`, colocando o primeiro item flex de cada flex line flush contra o lado principal. Cada item flex subsequente é colocado nivelado com o lado principal do item flex anterior, até que o final da linha flex seja atingido se o `wrap` for definido. A localização do lado de partida principal depende da direção flexível e do modo de escrita, que é explicado em "[Entendendo eixos](#)".

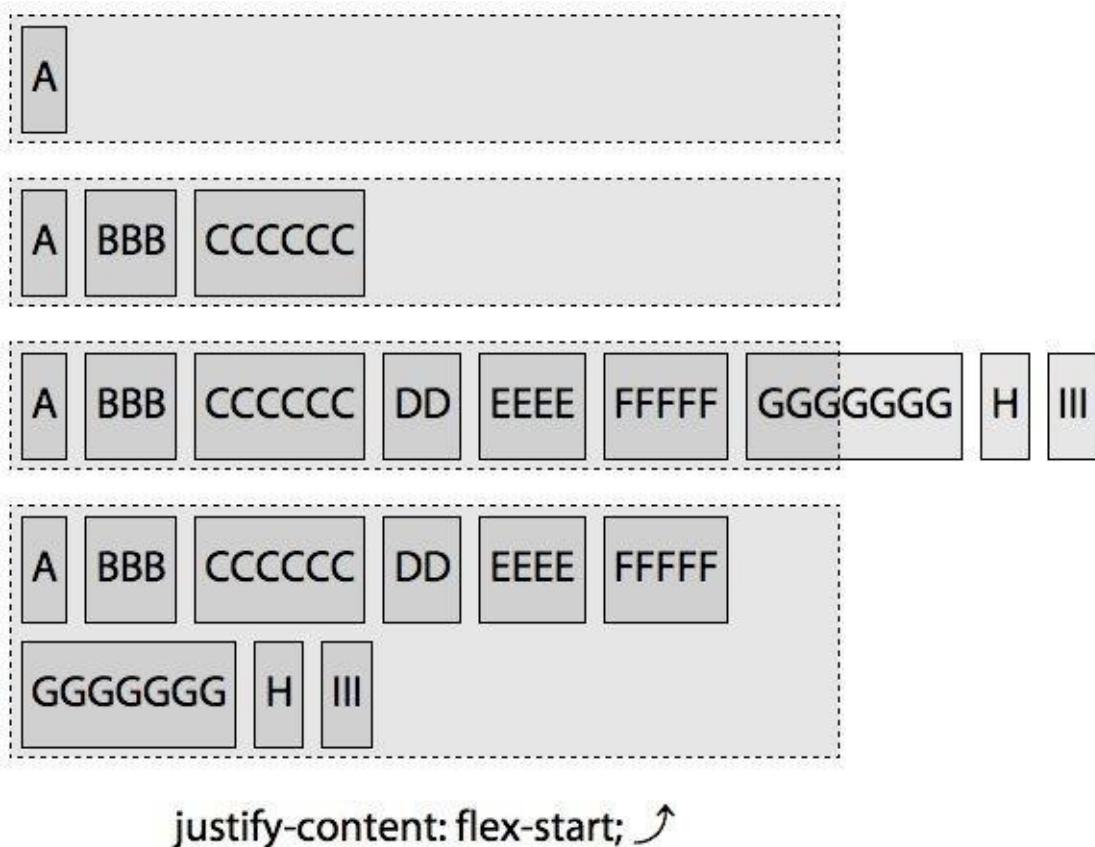


Figura 2-15. Impacto da configuração `justify-content: flex-start`



Se não houver espaço suficiente para todos os itens e `nowrap` for o padrão ou expressamente definido, os itens estourarão na borda principal, conforme mostrado no terceiro exemplo da [Figura 2-15](#).

Configurando o conteúdo justificado: o `flex-end` coloca o último flex em uma linha nivelada contra a extremidade principal com cada item flexível anterior sendo colocado nivelado com o item subsequente. Nesse caso, se os itens não tiverem permissão para ser empacotados e se não houver o suficiente

para todos os itens, os itens irão estourar na borda principal-inicial, como mostrado no terceiro exemplo da [Figura 2-16](#). Qualquer espaço extra em uma linha flexível estará no lado *principal*.

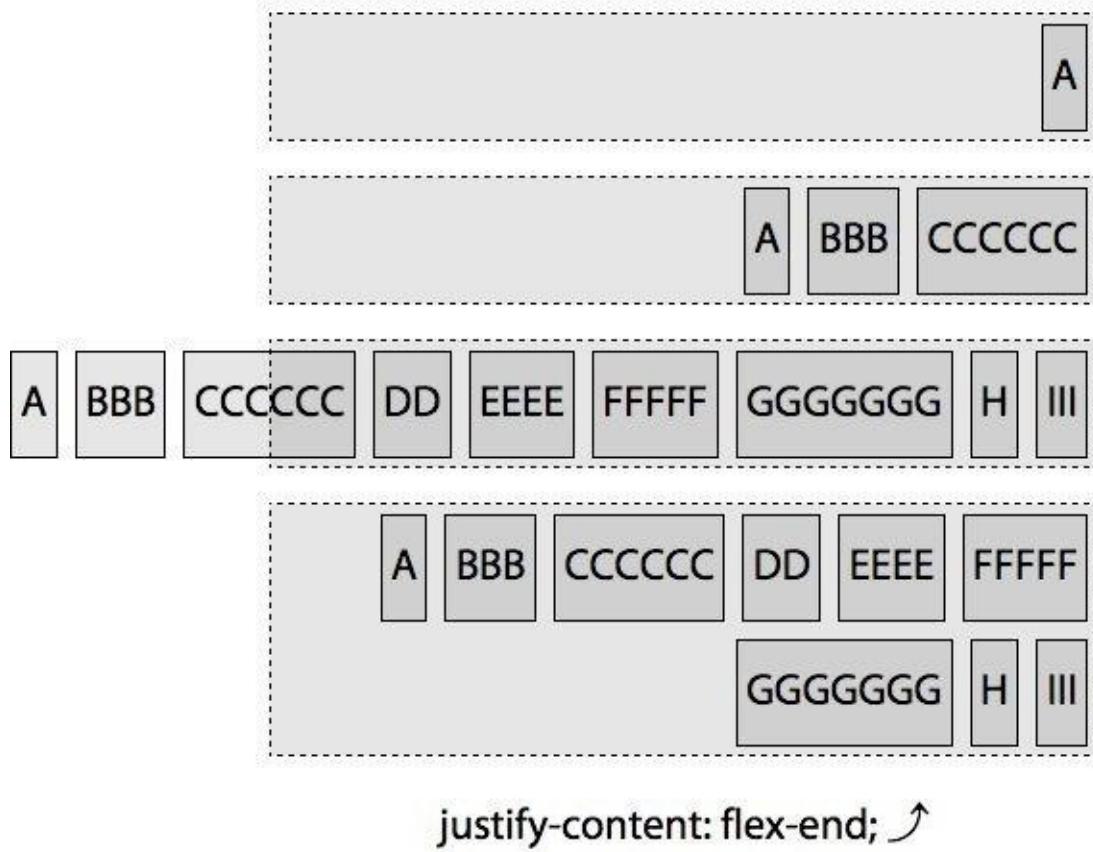


Figura 2-16. Impacto da configuração do conteúdo justificado: flex-end

Configurando o conteúdo justificado: o centro empacotará todos os itens juntos, nivelados uns contra os outros no centro de cada linha flexível, em vez de no início principal ou no final principal. Se não houver espaço suficiente para todos os itens e eles não tiverem permissão para ser empacotados, os itens estourarão uniformemente nas bordas principal inicial e principal, conforme mostrado no terceiro exemplo da [Figura 2-17](#). Se os itens flexíveis forem encapsulados em várias linhas, cada linha terá itens flexíveis centralizados, com espaço extra nas bordas principal inicial e principal.

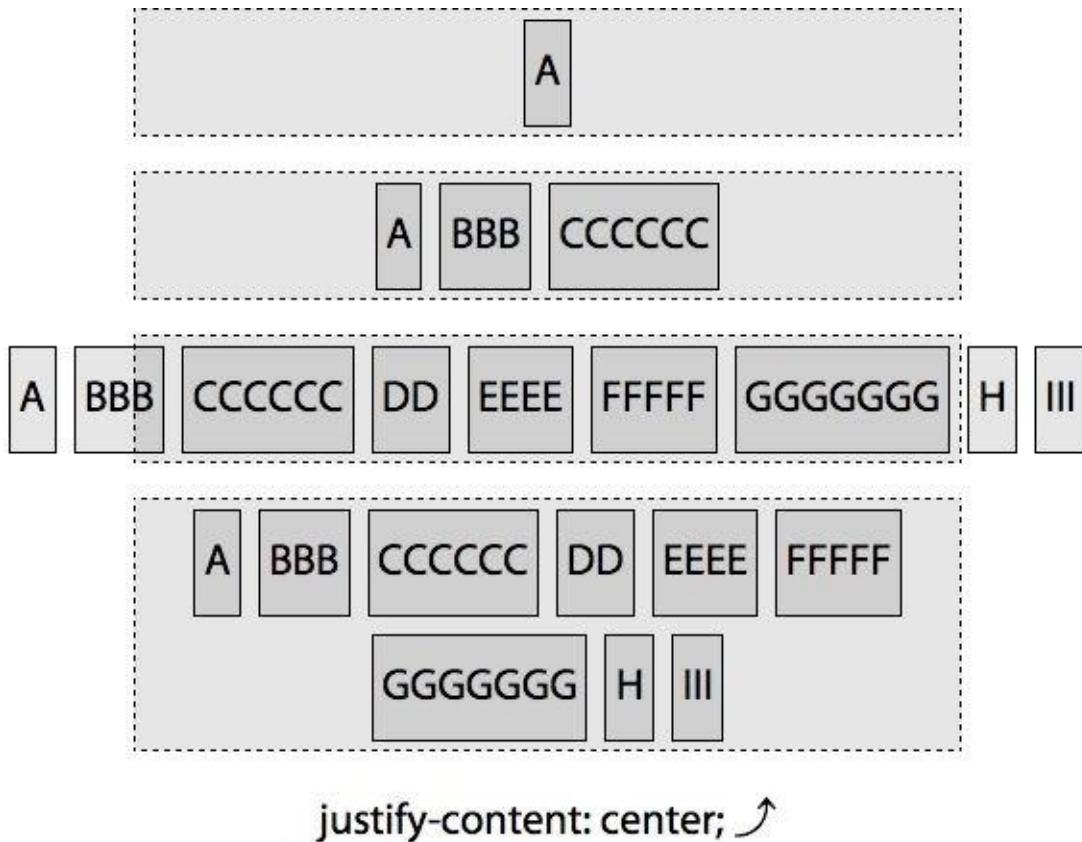


Figura 2-17. Impacto da configuração de conteúdo justificado: centro

Setting `justify-content: space-between` coloca o primeiro item flex nivelado com `main-start` e o último item flex na linha flush com `main-end` e, em seguida, coloca uma quantidade uniforme de espaço em torno de cada item flex, até que a linha flex seja filha de `d`. Em seguida, ele repete o processo com todos os itens flexíveis que são envolvidos em linhas flexíveis adicionais. Se houver três itens flexíveis, haverá a mesma quantidade de espaço entre o primeiro e o segundo itens que entre o segundo e o terceiro, mas não haverá espaço vazio extra entre a borda do contêiner e o primeiro item, e a borda oposta do contêiner e a borda externa do último item, conforme mostrado no segundo exemplo da [Figura 2-18](#). Com o espaço entre, o primeiro item é nivelado com o `main-start`, o que é importante lembrar quando você tem apenas um item flex ou quando seus itens flex estouraram o contêiner flex em um cenário `nowrap`. Isso significa que, se houver apenas um item flexível, ele será nivelado com o `main-start`, não centralizado, o que parece contra-intuitivo para muitos no início.

NOTA

Com o justificar-conteúdo: espaço-entre, o primeiro item flexível é colocado nivelado | .com o início-principal, portanto, se houver apenas micrômetro item ou se SO itens flexionar não tiverem permissão para envolver e transbordando | o contêiner flex, um aparência será um mesma fazer flex-start — um comparação pode ser vista na [Figura 2-14](#) — que pode Sor menos do que intuitiva.

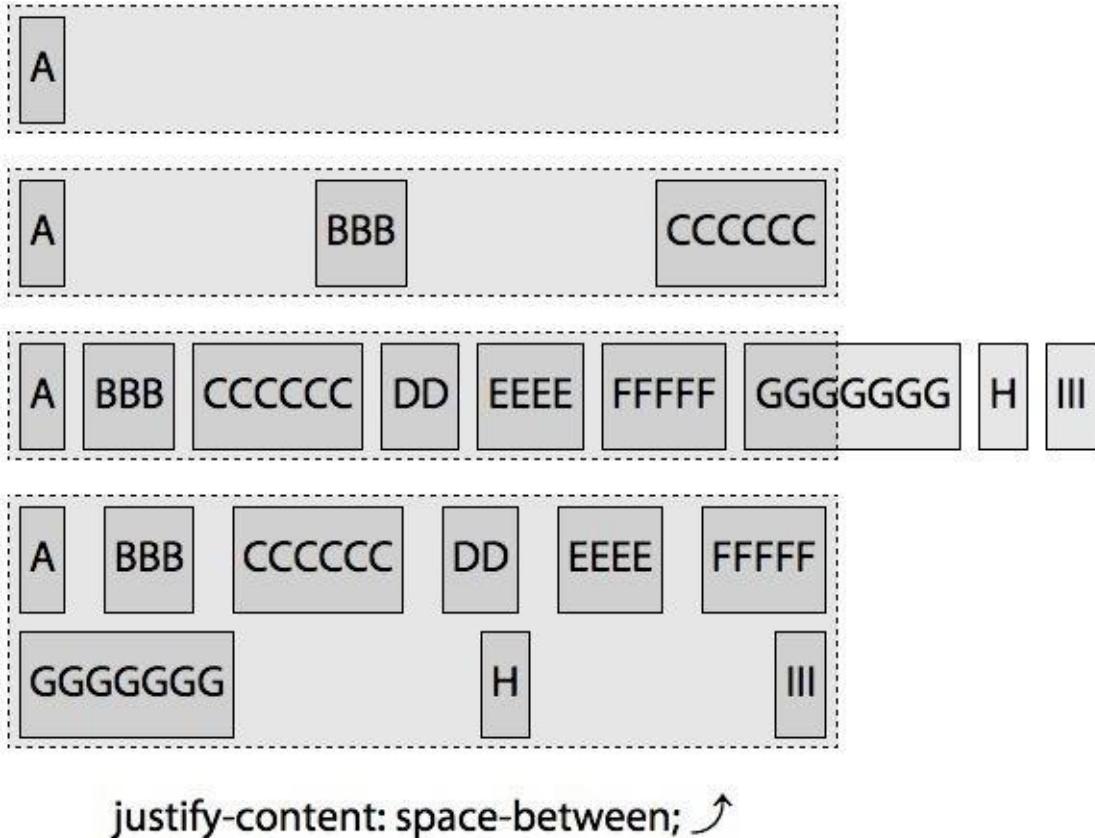


Figura 2-18. Impacto da configuração justificar-conteúdo: espaço-entre

Com o conteúdo justificado: o espaço entre quaisquer dois itens em uma linha flexível será igual, mas não necessariamente o mesmo entre linhas flexíveis. Quando configurado para permitir o empacotamento, na última linha flex, o primeiro item flex dessa última linha é nivelado contra o main-start, o último, se houver dois ou mais nessa linha, será contra o main-end , com espaço igual entre os pares adjacentes de itens flex. Como mostrado no último exemplo da [Figura 2-18](#), A e G, os primeiros itens em cada linha flexível são nivelados em relação ao main-start. F e I, os últimos itens em cada linha, são nivelados contra a extremidade principal. Os itens flexíveis são distribuídos uniformemente com o espaçamento entre

quaisquer dois itens adjacentes sendo os mesmos em cada uma das linhas, mas o espaço entre os itens flexíveis na primeira linha é mais estreito do que o espaço entre os itens flexíveis na segunda linha.

Definindo o conteúdo justificado: o espaço-redor distribui uniformemente o espaço extra na linha em torno de cada um dos itens flexíveis, como se houvesse margens não colapsantes de tamanho igual em torno de cada elemento nos lados da dimensão principal. Portanto, haverá duas vezes mais espaço entre o primeiro e o segundo item do que há entre o início principal e o primeiro item e o final principal e o último item, como mostrado na [Figura 2-19](#).

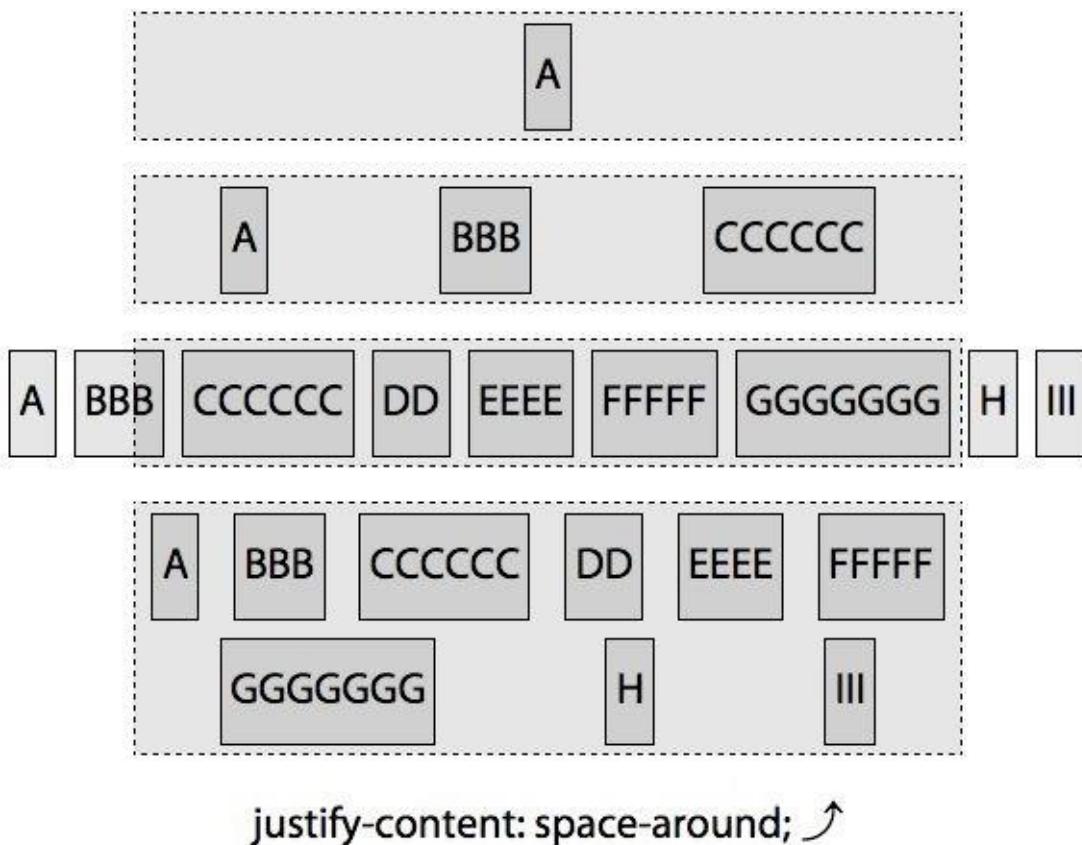


Figura 2-19. Impacto da configuração de conteúdo justificado: espaço ao redor

Se o `nowrap` estiver definido e não houver espaço suficiente na direção principal do contêiner flexível para todos os itens flex, os itens flex transbordarão igualmente em ambos os lados, semelhante a centro de configuração, conforme mostrado no terceiro exemplo na [Figura 2-14](#).

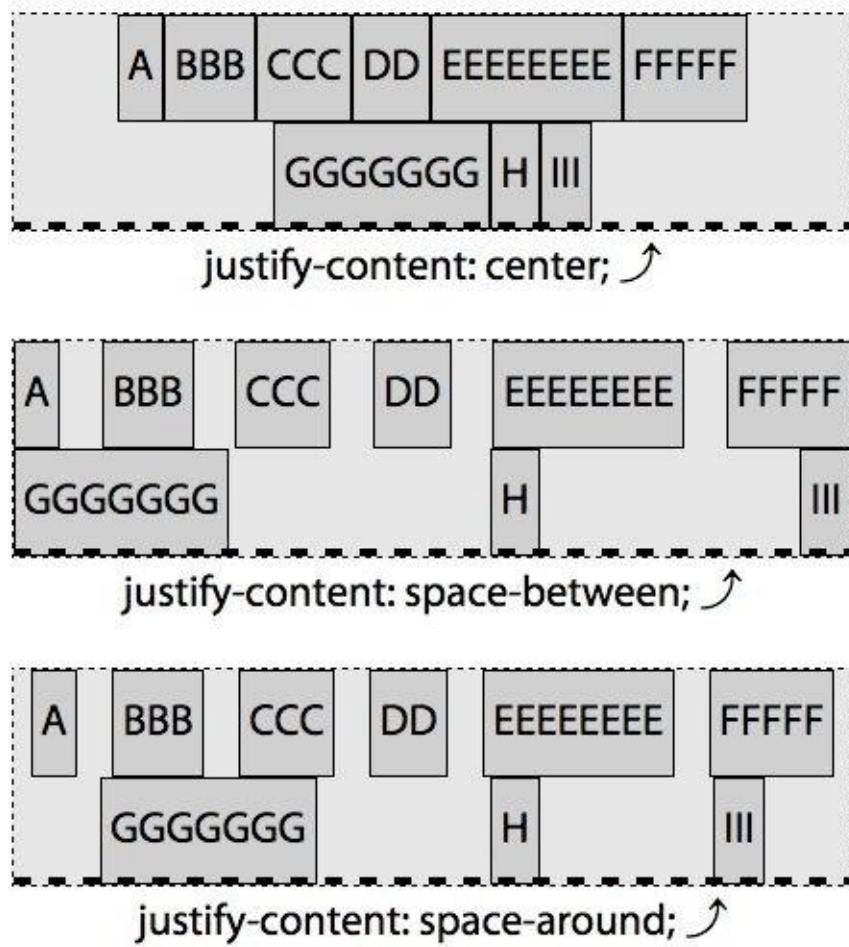
Se os itens flexíveis forem encapsulados em várias linhas, o espaço ao redor de cada item flexível será baseado no espaço disponível em cada linha flexível. Enquanto o espaço em torno de cada um

em uma linha flexível com ser o mesmo, pode diferir entre linhas, como mostrado nos últimos exemplos na [Figura 2-19](#). Os espaços entre A e B e entre G e H têm o dobro da largura dos espaços entre a aresta principal e A e a aresta e G.

Com a margem adicionada aos itens flexíveis para tornar os exemplos menos horríveis, isso pode ser difícil de ver. Comparar exemplos sem margem de centro, espaço ao redor e espaço entre pode ser mais útil.

[A Figura 2-20](#) demonstra a diferença entre os conceitos de espaçamento de centro, espaço ao redor e espaço entre. Quando definidos para o centro, os itens flexíveis são agrupados uns contra os outros na entrada c da dimensão principal.

Com o espaço entre, você notará o primeiro e o último item flexível em ambas as linhas flexíveis, além do início principal e do fim principal, respectivamente. O espaço entre cada um dos itens flex na primeira linha flex é de 24 px: os 120 px de espaço livre são divididos em 5 intervalos iguais colocados entre os 6 itens flex. O espaço entre cada item flex na segunda linha flex é de 150 px: os 300 px de espaço livre são divididos em dois e colocados



entre os três itens flex.

Figura 2-20. Comparando centro, espaço entre e espaço ao redor

Com o espaço ao redor, os itens flexíveis não necessariamente se aproximam das bordas: se

houver algum espaço extra, ele é colocado em torno de cada item flexível. Em todos os três exemplos da [Figura 2-20](#), há 120 px de espaço livre e 6 itens na primeira linha e 300 px de espaço livre e 3 itens flexíveis na segunda linha flex. Na primeira linha do exemplo space-around, os 120 px de espaço livre são divididos entre os dois lados dos seis itens, colocando 10 px à esquerda e à direita de cada item. Dessa forma, há 10 px de espaço entre *o início principal* e o primeiro item e o *último item* e *o fim principal*, e o dobro disso, ou 20 px, entre cada item flexível adjacente. Na segunda linha flex, os 300 px de espaço livre são divididos entre os dois lados dos 3 itens flex, colocando 50 px nas bordas externas e duas vezes isso, ou 100 px, entre itens adjacentes.

Também pode ajudar a comparar a borda de transbordamento e as bordas dos cinco justificam-propriedades de conteúdo com um eixo principal diferente. Na [Figura 2-21](#), todos os contêineres flexíveis têm o seguinte CSS:

```
contêiner {  
  display: inline-flex;  
  flexflow: sem enrolamento de coluna-reverso;  
  altura: 200px;  
}
```

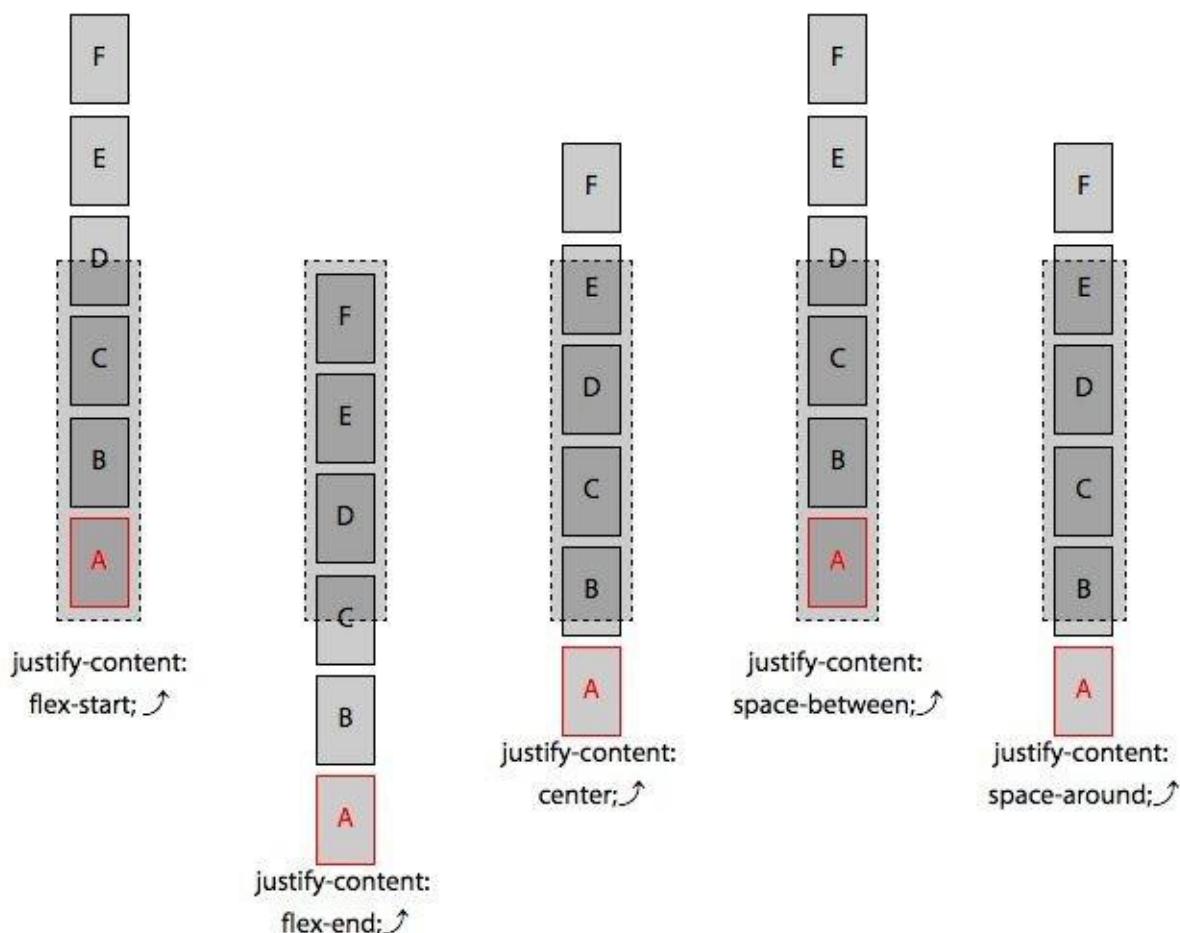


Figura 2-21. Os cinco valores da propriedade justify-content quando flex-direction: column-reverse é definido e os itens flex transbordam o contêiner flex

O layout quando há espaço livre negativo agora deve ser mais intuitivo:

flex-start e space-between overflow na extremidade principal.

flex-end transborda no início principal.

transbordamento do centro e do espaço em ambas as extremidades.

Com o justify-content: space-between, o primeiro item flexível é colocado nivelado com o main-start. Se os itens flexíveis não se encaixarem em uma linha dentro do contêiner flex, eles transbordarão no lado principal, conforme mostrado na [Figura 2-21](#). Com o conteúdo justificado: espaçado, se houver apenas um item flexível em uma linha, ou mais itens do que uma linha flexível sem encapsulamento pode conter, ele será centralizado. Se houver vários itens, os espaços externos terão metade do tamanho dos espaços entre os itens adjacentes.

Em nossos exemplos, também impedimos que os itens flexíveis encolhessem, um recurso padrão que ainda não abordamos. A altura : 200px limitou a altura do recipiente, garantindo que os itens flexíveis pudessem transbordar seu recipiente. O visor: a declaração inline-flex fornecia um contêiner flexível que era tão largo quanto seu conteúdo. Se tivéssemos incluído wrap em vez de nowrap, o recipiente teria dobrado de largura, permitindo duas colunas ou linhas flexíveis.

Exemplos de justify-content

Aproveitando o valor padrão na [Figura 1-7](#), criando uma barra de navegação alinhada à direita a esquerda da barra d

`Home` `About` `Blog` `Careers` `Contact Us`

`justify-content: flex-start;` ↑

`Home` `About` `Blog` `Careers` `Contact Us`

`justify-content: flex-end;` ↑

`איש קשר` `קרירה` `בלוג` `עלינו` `דף הבית`

`justify-content: flex-start;` ↑

`איש קשר` `קרירה` `בלוג` `עלינו` `דף הבית`

`justify-content: flex-end;` ↑

Figura 2-22. Navegação alinhada à direita e à esquerda em idiomas LTR e RTL usando conteúdo justificado



Para modos de escrita da direita para a esquerda, não precisamos alterar o CSS. Como mostrado na [Figura 2-21](#), e como discutido no [Capítulo 1](#), os ems flexíveis são agrupados em direção ao início principal. Em inglês, o main-start está à esquerda. Para o hebraico, o main-start está à direita.

Simplesmente adicionando uma única linha ao nosso CSS, alteramos a aparência do nosso exemplo de navegação, tornando a versão em inglês nivelada para a direita e a tradução em hebraico para a esquerda:

```
nav {  
  display: flex;  
  justify-content: flex-end;  
  border-bottom: 1px solid #ccc;  
}
```

Poderíamos ter centrado essa navegação, como mostrado na [Figura 2-23](#):

```
nav {  
  display: flex;  
  justify-content: center;  
  border-bottom: 1px solid #ccc;  
}
```



Figura 2-23. Alterando o layout com um par de valores de propriedade



A propriedade alignitems

Enquanto o justify-content define como os itens flex são alinhados ao longo do eixo principal do contêiner flex , a propriedade alignitems define como os itens flex são alinhados ao longo do eixo cruzado de sua linha flex.

ALIGNITEMS	
Valores	flex-start flex-end centro Basal esticar
Valor inicial :	alongamento
Aplica-se um:	Contêineres
Herdados:	Não
Porcentagens:	Não aplicável
Animável:	Não

Com a propriedade alignitems, você pode alinhar itens flexíveis ao início, ao fim ou ao centro do eixo cruzado de sua linha flexível. Definido no elemento container, alignitems é semelhante ao justify-content, mas na direção perpendicular, definindo o alinhamento entre eixos para todos os itens flex, incluindo itens flex anônimos, dentro do flex container (aprenderemos a substituir esse valor para itens flex individuais quando cobrirmos align-self ("O align-self Propriedade").

Com os itens de alinhamento, você pode definir todos os itens para que seu eixo cruzado seja nivelado contra o início *cruzado* ou a extremidade cruzada de sua linha flexível ou esticado nivelado para ambos. Ou você pode centralizar todos os itens flex no meio da linha flex. Esticado por todo o eixo cruzado é o padrão e é o que vimos na maioria dos exemplos até agora. Há cinco valores, incluindo flex-start, flex-end, center, baseline e o trecho padrão, conforme mostrado na **Figura 2-24**.

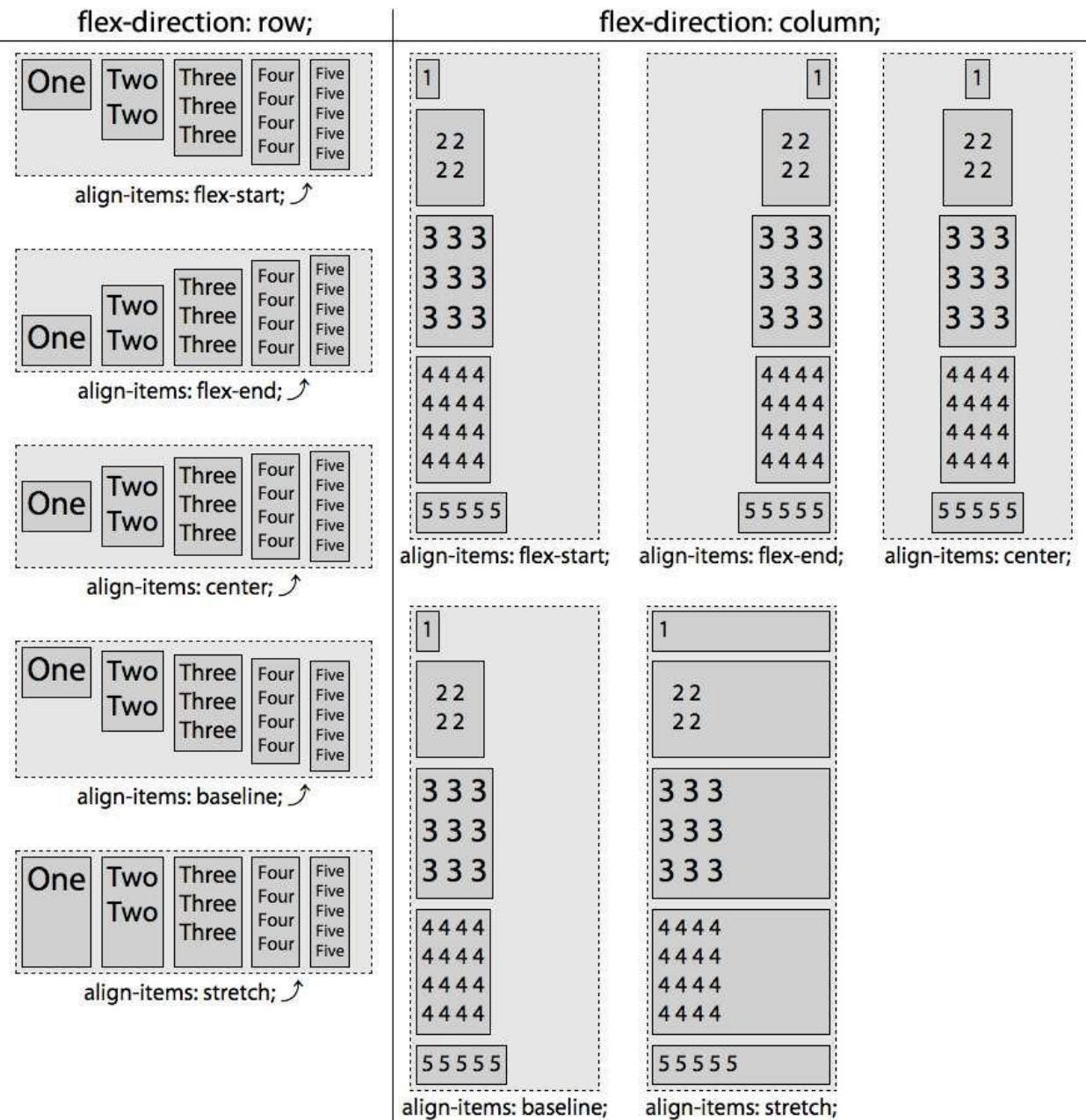


Figura 2-24. Os cinco valores da propriedade align-items quando você tem uma única linha de itens flexíveis e coluna única de itens flexíveis



NOTA

Enquanto align-items definir o alinhamento para todos os itens flexíveis | dentro de micrômetro contêiner, o alinhar-se

permite substituir o alinhamento para itens flexíveis | indivíduos.

Na [Figura 2-24](#), você observará como os itens flexíveis abraçam o lado de partida cruzada ou extremidade cruzada do contêiner flexível, são centralizados ou se esticam para abraçar ambos, com a possível exceção da linha de base.

A ideia geral é que o flex-start coloca os itens flex no cross-start , o flex-end os coloca na borda cross-end, centraliza-os no eixo cruzado, o alongamento padrão se estende o item flex do cross-start ao cross-end, e a linha de base é semelhante ao flex-start, mas na verdade alinha as linhas de base dos itens flex e, em seguida, empurra o grupo de itens flex para cross-start.

Com a linha de base, as linhas de base dos itens flexíveis estão alinhadas: o item flex that tem a maior distância entre sua linha de base e seu lado de partida cruzada será nivelado contra a borda de partida cruzada da linha. Essa é a ideia geral – e explica muito bem os contêineres flexíveis sem embrulho – mas há mais do que isso.

Nas figuras de itens de alinhamento de várias linhas a seguir, o seguinte código foi incluído:

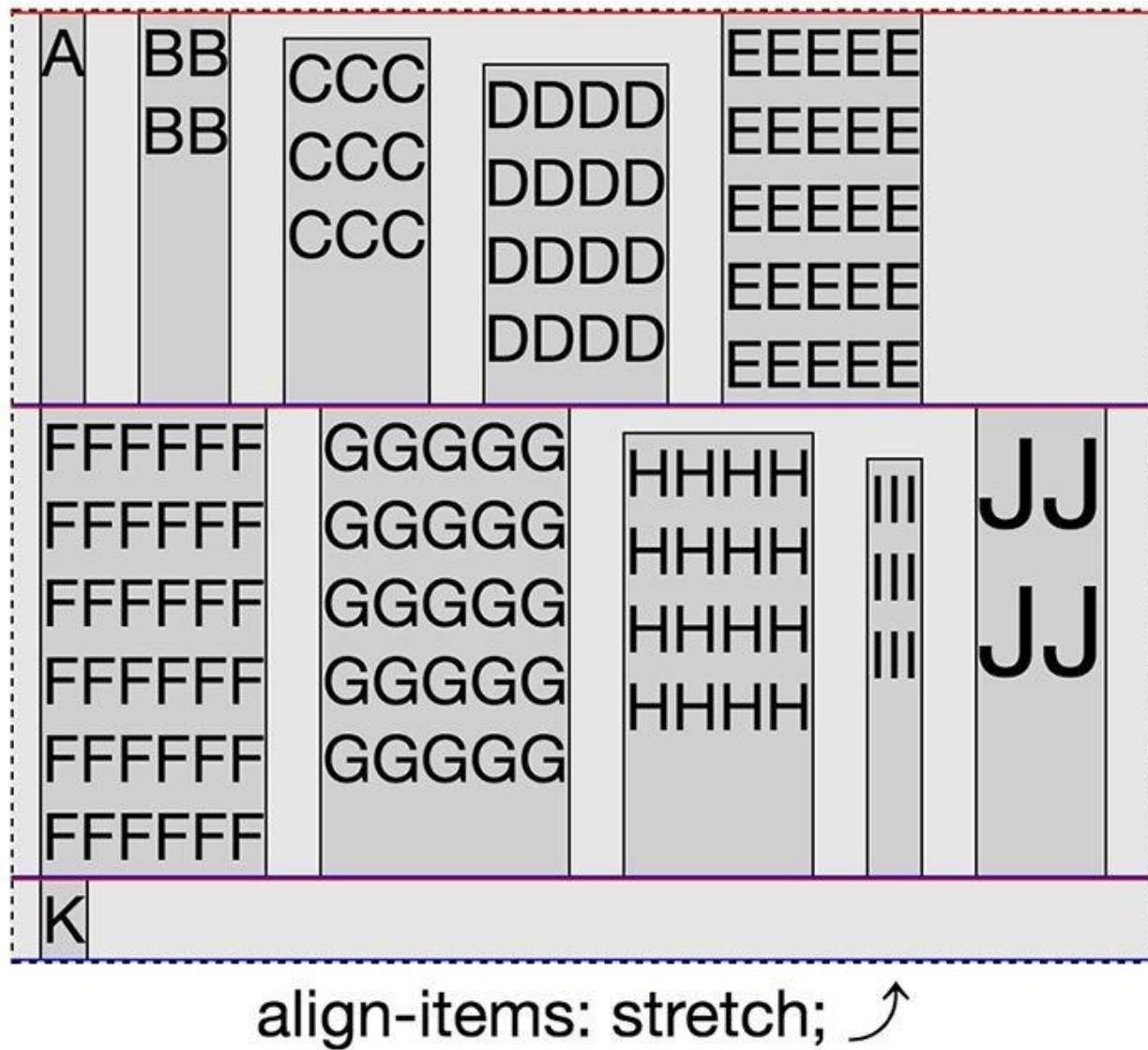
```
flex-container { display: inline-flex; flexflow: envoltório  
  de linha; borda: 1px tracejado; preenchimento: 10px;  
}  
flex-item {  
  borda: 1px sólido;  
  margem: 0 10px;  
}  
.C, .H {  
  margem-topo: 10px;  
}  
.D, .Eu {  
  margem-topo: 20px;  
}  
.J {  
  tamanho da fonte: 3rem;  
}
```

Para cada linha flexível nas Figuras 2-25, 2-27, 2-28, 2-29 e [2-30](#), a linha vermelha é cross-start e a azul é [cross-end](#) . As linhas aparecem roxas quando uma nova linha flexível fica ao lado da linha flexível anterior. C, H, D e eu temos valores diferentes para as margens superior e inferior. Adicionamos um pouco de margem aos lados de todos os itens flexíveis para tornar os números mais legíveis, o que não afeta o impacto da propriedade align-items. J tem o tamanho da fonte aumentado, aumentando a altura da linha.

Isso entrará em jogo quando discutirmos o valor da linha de base.

O padrão é align-items: stretch, como mostra a [Figura 2-25](#).

alignitems: esticar



align-items: stretch; ↗

Figura 2-25. alignitems: esticar

O padrão e o explicitamente definido alignitems: stretch estica todos os itens flexíveis extensíveis em uma linha para serem tão altos ou largos quanto o item flex mais alto ou mais largo da linha. O que significa "extensível"? Embora, por padrão, todos os itens flexíveis se estendam para ocupar 100% do tamanho cruzado, se a altura mínima, a largura mínima, a altura máxima, a largura máxima, a largura ou a altura forem definidas, essas propriedades terão precedência.

Se definido ou padrão para esticar, o início cruzado dos itens flex será nivelado com o cross-start da linha flex e o cross-end dos itens flex será nivelado com o cross-end da linha flex. O item flexível com o maior tamanho cruzado permanecerá seu tamanho padrão, e os outros itens flexíveis se esticarão, crescendo para o tamanho do maior item flexível em essa mesma linha flex. O tamanho cruzado dos itens flexíveis inclui as margens, conforme demonstrado pelos itens C, D, H e I.

NOTA

As margens na direção transversal têm um impacto. Na [Figura 2-26](#), adicionamos 30 px de | margem à borda | de partida cruzada | de C fazer item flexionar e 40 Px à borda | transversal fazer D. Você notará que Um, B e E estão nivelados | contra ambas as bordas cruzadas, mas o C e o D aparecem empurrados para | dentro. Isto é devido | às margens: suas margens | externas são niveladas | contra o cross-start e cross-end.

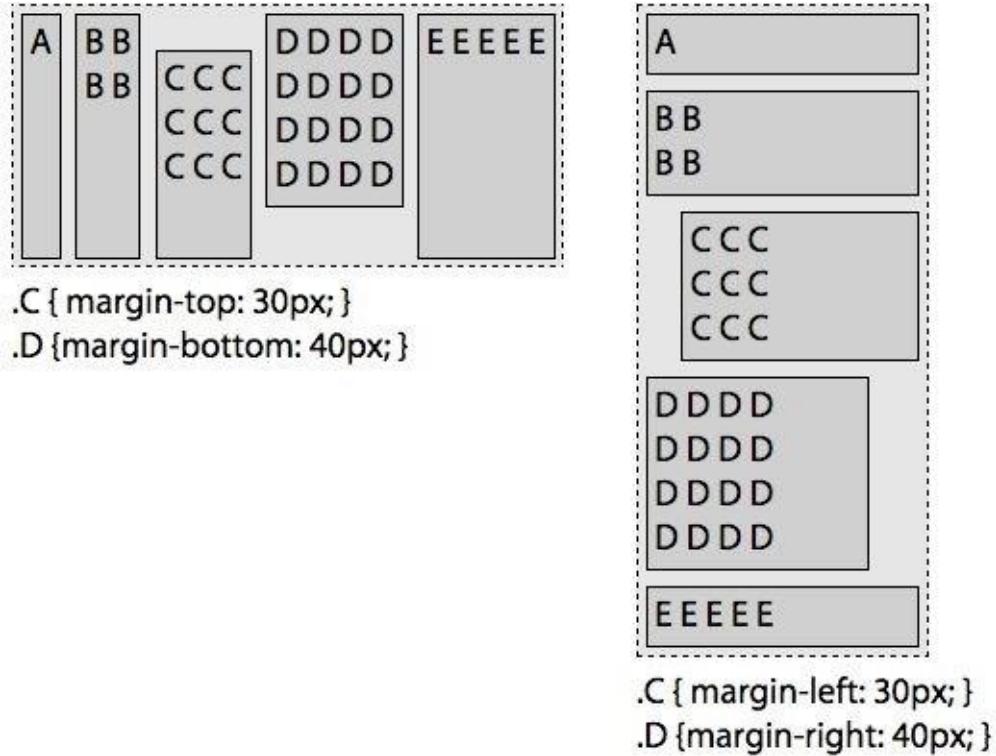


Figura 2-26. Efeito de margens entre eixos na propriedade align-items



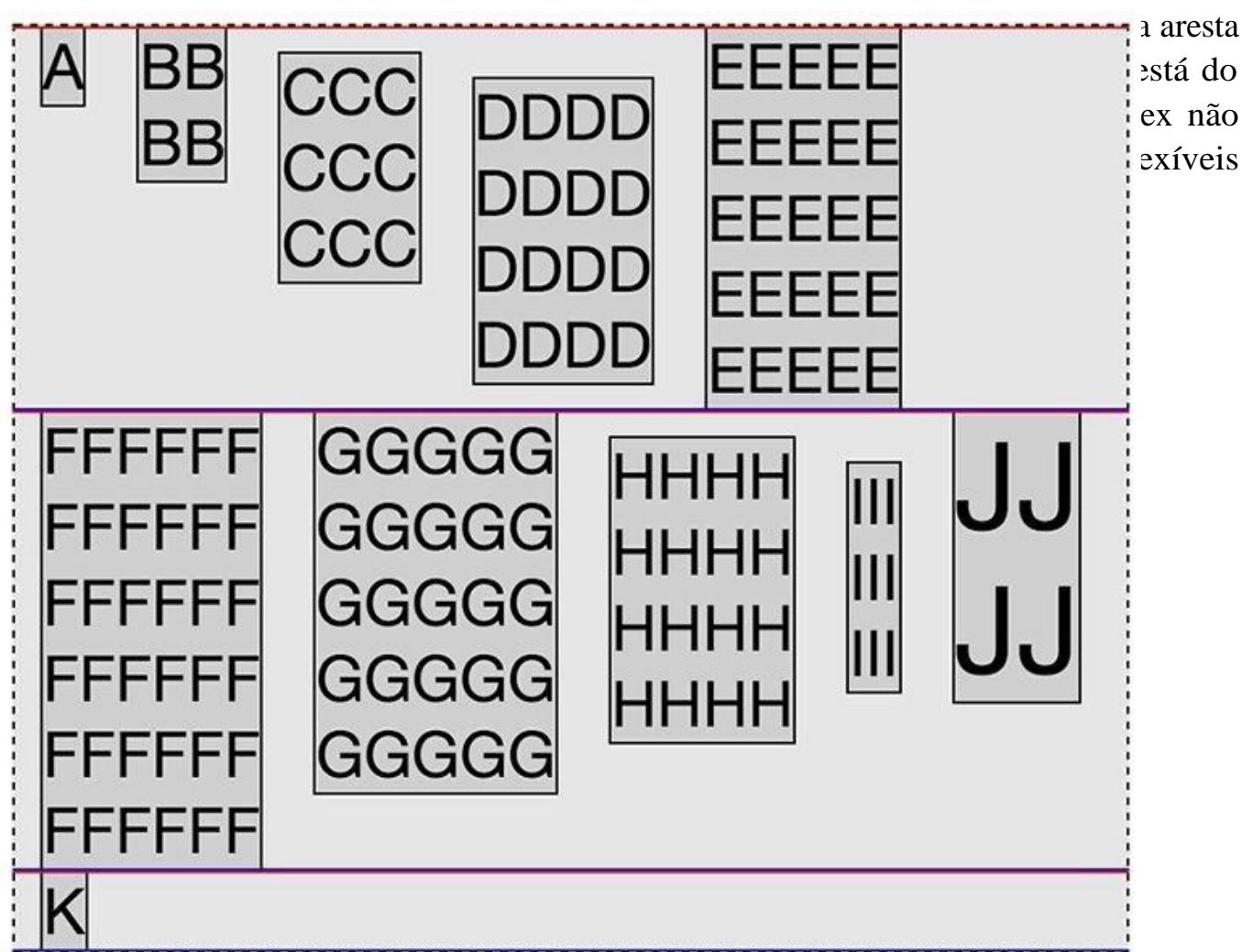
O tamanho do item flex esticado inclui as margens nos lados de partida cruzada e extremidade cruzada: é a borda externa da margem dos itens flexíveis que será nivelada

com cross-start e cross-end. Esta é a razão pela qual C, D, H e I podem parecer menores do que os outros itens flexíveis em suas linhas flexíveis. Não são. A borda externa das margens superior e inferior é nivelada com o início cruzado e a extremidade cruzada das linhas flexíveis que ocupam. Essas linhas flexíveis são, por sua vez, tão altas quanto o item mais alto da linha (ou tão largas quanto o item mais largo quando a dimensão transversal é horizontal).

As linhas flexíveis são tão altas quanto precisam ser para conter seus contêineres flexíveis. Nas cinco figuras de alignitems, a altura da linha flex contendo apenas K é muito menor do que a linha que contém E, que é menor do que a linha que contém F. K tem apenas uma linha de texto e nenhuma margem, enquanto E tem cinco linhas de texto. A segunda linha, que inclui F com seis linhas de texto, tornando-a ainda mais alta do que a primeira linha.

align-items: flex-start

O valor
de arran-
lado de
aparecimen-
to C, D, E



align-items: flex-start ↑

Figura 2-27. *align-items: flex-start*

align-items: flex-end

Definindo itens de alinhamento: o flex-end alinhará a borda cruzada de todos os itens flexíveis ao longo da borda transversal da linha em que estão, conforme mostrado na [Figura 2-28](#). Nesses exemplos, nenhum dos itens flexíveis tem uma margem inferior maior que 0 px, portanto, ao contrário de nossos outros exemplos, este exemplo não parece irregular.

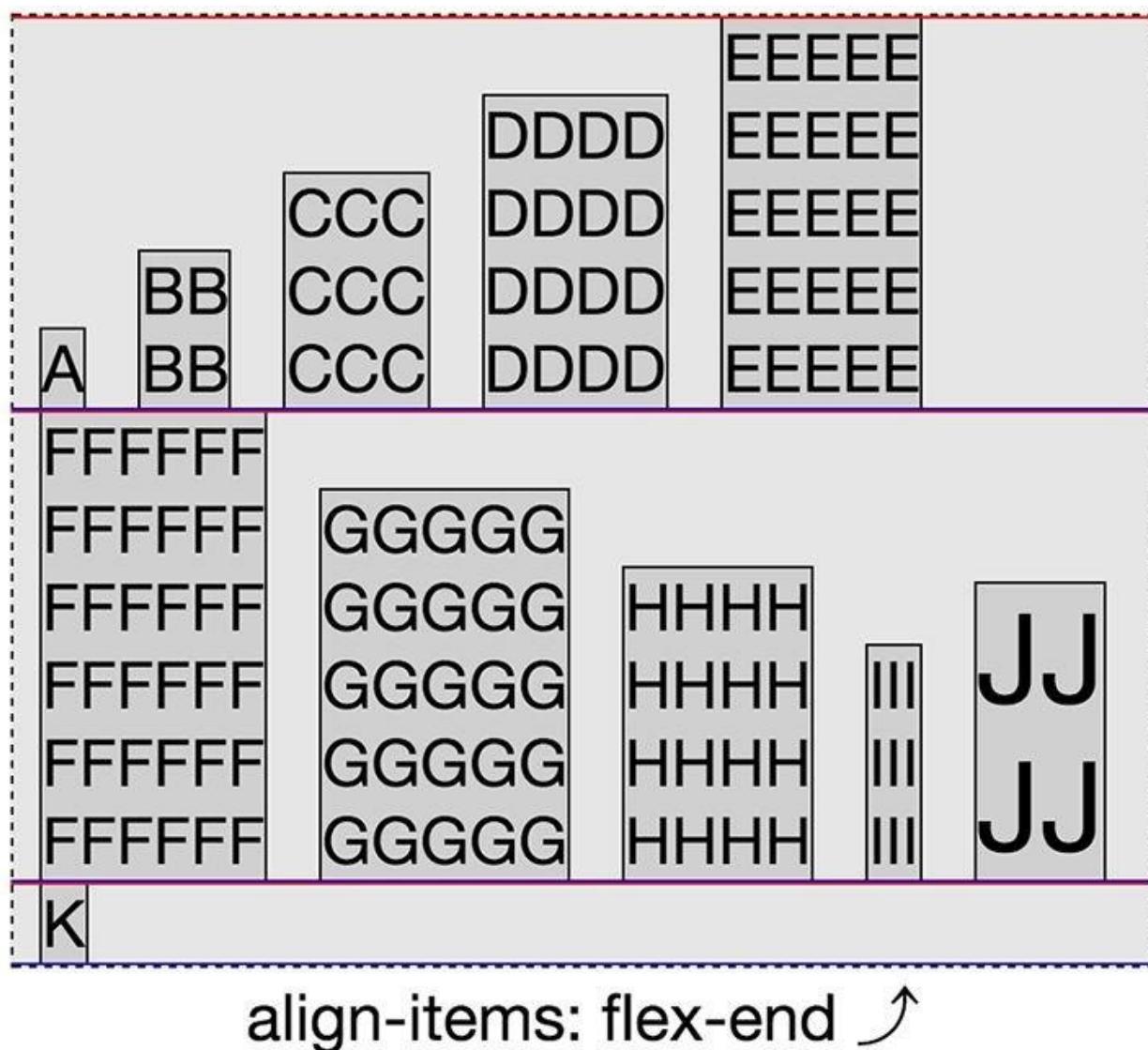
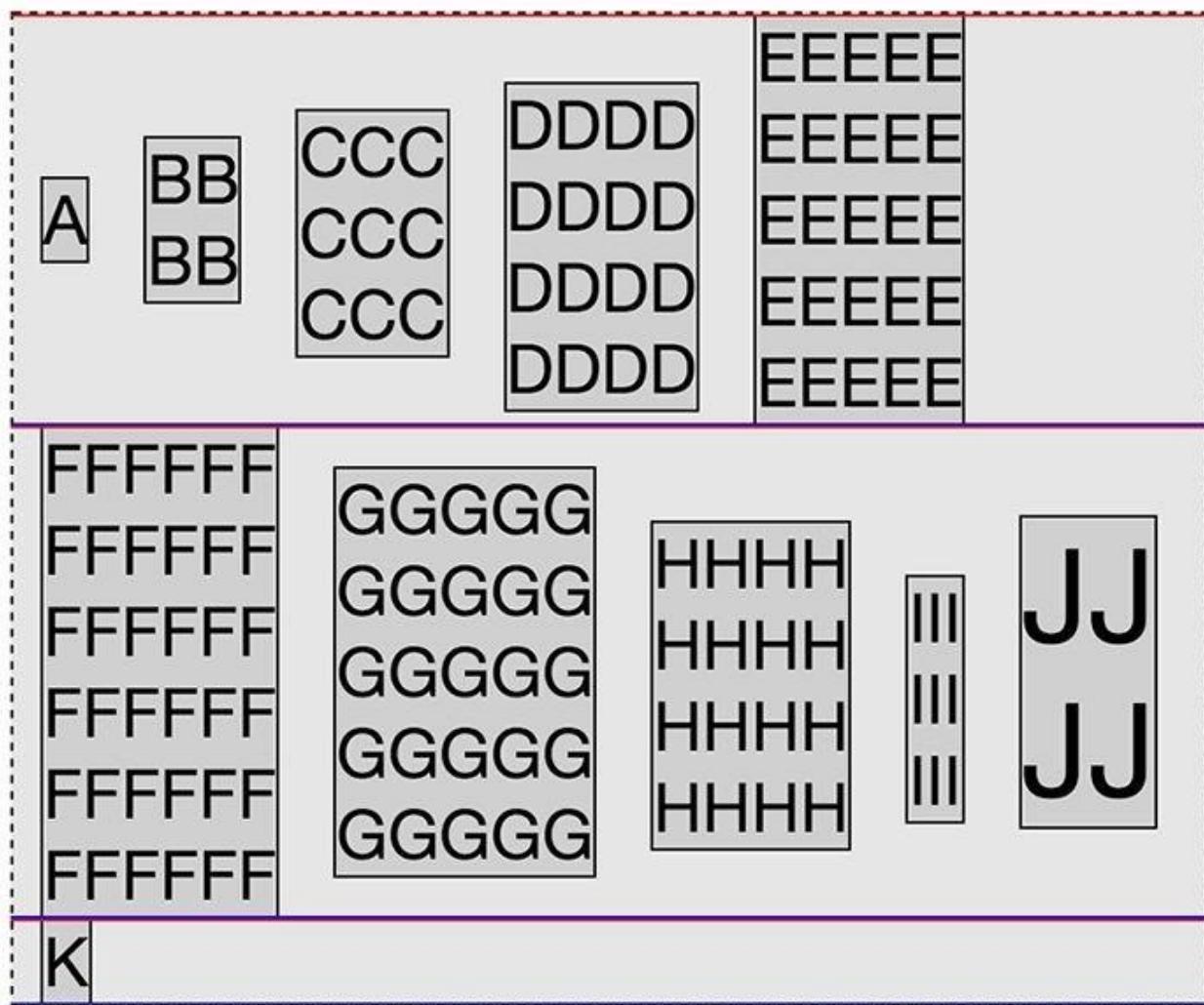


Figura 2-28. *align-items: flex-end*

align-items: centro

Conforme mostrado na [Figura 2-29](#), a configuração align-items: center centralizará o tamanho cruzado dos itens flexíveis ao longo do ponto médio do eixo cruzado da linha. O centro é o ponto médio entre as bordas externas da margem, lembrando que as margens do item flexível não colapsam. Como as margens de borda cruzada para C, D, H e I não são simétricas, os itens flexíveis não aparecem centralizados ao longo do eixo cruzado, mesmo que sejam. Nos idiomas LTR e RTL, no caso de flex-direction: row e row-reverse, o ponto médio é o ponto médio da margem superior, borda superior, preenchimento superior, conteúdo ou altura, preenchimento inferior, borda inferior e margem inferior. Para flex-direction: coluna e coluna-reverso, o ponto médio é o ponto médio da margem esquerda, borda esquerda, preenchimento esquerdo, conteúdo ou largura, preenchimento direito, borda direita e direito margem.



align-items: center; ↑

Figura 2-29. *alignitems: flex-center*

NOTA

Os itens Flex podem transbordar seu contêiner flex pai no eixo principal se o nowrap estiver definido e o tamanho principal estiver restrito. Da mesma forma, se o tamanho cruzado do contêiner flexível for restrito, o conteúdo poderá transbordar a borda de partida cruzada e/ou extremidade cruzada do contêiner flexível. A direção do estouro não é determinada pela propriedade alignitems, mas pela propriedade aligncontent, discutida a seguir. O alignitems alinha os itens flex dentro da linha flex e não afeta diretamente a direção de estouro dos itens flex dentro do contêiner.

align-items: linha de base

O valor da linha de base pode ser um pouco mais confuso. Com a linha de base, os itens flexíveis em cada linha são todos alinhados em suas linhas de base, que é basicamente a parte inferior da primeira linha de texto, se houver alguma. O item flex em cada linha flex com o maior distância entre sua linha de base e sua borda de margem de início cruzado é colocado nivelado contra a borda de partida cruzada da linha, com todas as linhas de base de todos os outros itens flex alinhadas com a linha de base de aquele item flexível.

Em vez de alinhar o início cruzado de cada item flexível nivelado contra o início cruzado de cada linha, os itens flexíveis são alinhados para que suas linhas de base se alinhem. Em muitas implementações, a linha de base será parecida com flex-start, mas será diferente do valor flex-start se os itens flex tiverem margens, preenchimento ou borda diferentes no lado de início cruzado, ou se as primeiras linhas de conteúdo dos itens flexíveis não tiverem todas as mesmas alturas de linha.

Você notará que A, B, C, D e E parecem todos alinhados no topo. O que você pode ter perdido é que eles não estão nivelados ao topo – eles não estão nivelados contra a linha vermelha. D tem uma margem superior de 20 px. A borda externa da margem superior de D é nivelada contra o início cruzado da linha flex, que é nivelada com a parte superior do contêiner flex. Como observou revily, a distância entre a linha de partida cruzada e a linha de base é determinada pelo item na linha que tem a maior distância entre sua margem externa em seu lado de partida cruzada e sua linha de base. No exemplo de linha de base da Figura 2-30, os itens com a maior distância em suas próprias linhas flexíveis são D, J e K. As margens externas desses itens no lado de partida cruzada são colocadas niveladas contra a borda de partida cruzada de suas respectivas linhas, e os outros itens nas mesmas linhas flexíveis têm sua linha de base alinhada com D, J e a linha de base de K.

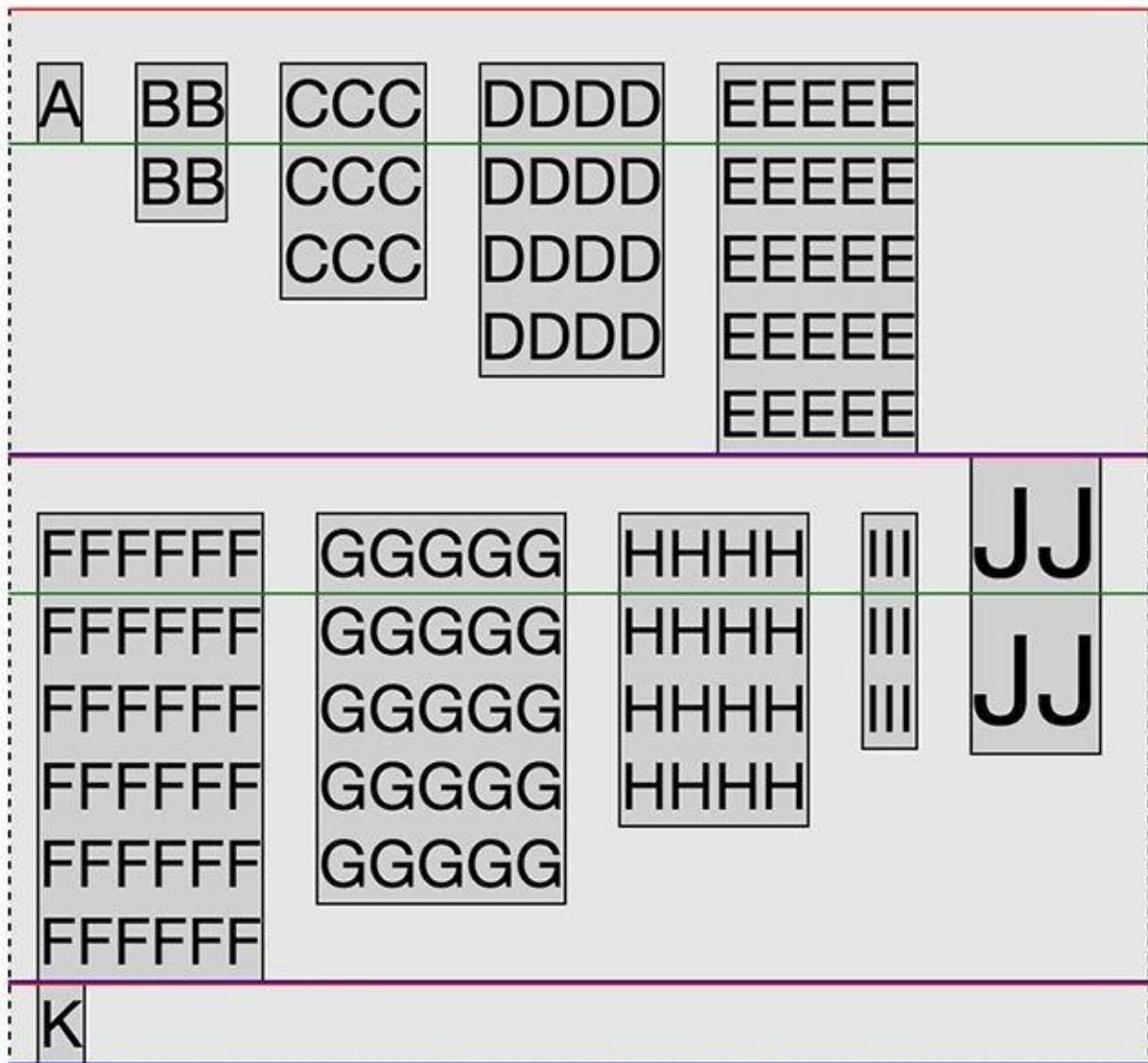


Figura 2-30. align-items: linha de base



Como A, B, C, D e E têm a mesma altura de linha, e D tem a margem superior mais alta, eles estão alinhados com a linha de base de D e a margem superior de D é nivelada contra a borda de partida cruzada. Quando se trata da primeira linha , porque todos eles têm a mesma altura de linha, borda e preenchimento, parece que eles estão alinhados como flex-start, mas eles são, na verdade, um pouco mais baixos, acomodando-se à margem superior de D. A linha verde denota onde está a linha de base.

Em todos os exemplos, aumentamos o tamanho da fonte de J para 3rem para criar um item flexível que tivesse uma linha de base diferente de todos os outros itens flexíveis. Somente quando align-items: baseline é definido é que isso afeta o alinhamento do item flex dentro da linha flex, como mostrado na [Figura 2-30](#). Quando align-items: baseline é definido, as linhas de base de todos os itens na segunda linha flex são alinhadas com a linha de base de J, pois J é o item flex com o maior espaço entre a margem superior externa e a parte inferior do first linha de texto. Novamente, a linha verde denota a localização aproximada da linha de base.

Notas adicionais

A propriedade align-items é definida no contêiner flex e afeta todos os itens flex dentro desse contêiner flex. Se você quiser alterar o alinhamento de um ou mais itens flexíveis, mas não todos, poderá incluir a propriedade align-self nos itens flexíveis que deseja alinhar de forma diferente. O align-self usa os mesmos valores que align-items e é discutido no [Capítulo 3](#).

Não é possível substituir o alinhamento para itens flex anônimos (filhos de nó de texto não vazio de contêineres flex): seu align-self sempre corresponde ao valor de align-items de seu contêiner flex pai .

Nos exemplos de align-items, o tamanho cruzado do contêiner flexível era tão alto quanto necessário . Nenhuma altura foi declarada no contêiner, por isso o padrão era altura: auto. Por causa disso, o contêiner flex cresceu para se adequar ao conteúdo. Você deve ter notado que os contêineres flex de exemplo tinham a mesma altura e as alturas da linha flexível eram as mesmas em todos os exemplos.

Se o tamanho da cruz, neste caso a altura, tivesse sido definido para um tamanho específico, poderia ter havido espaço extra na extremidade cruzada ou não ter espaço suficiente para caber no conteúdo.

O Flexbox nos permite controlar o alinhamento das linhas flexíveis com a propriedade align-content. A propriedade align-content é a última propriedade em que precisamos nos concentrar que se aplica ao contêiner flex (versus os itens flex). A propriedade align-content afeta apenas o alinhamento de linha flexível em contêineres flexíveis de várias linhas.

A propriedade aligncontent

A propriedade aligncontent alinha as linhas de um contêiner flex dentro de um contêiner flex que tem espaço extra na direção do eixo cruzado e determina qual direção terá estouro quando houver não há espaço suficiente para caber nas linhas flexíveis.

ALIGNCONTENT

Valores | flex-start | flex-end centro | espaço entre | | espacais esticar |

Valor inicial : alongamento

Aplica-se a: Multiline flex containers

Herdados: Não

Porcentagens: Não aplicável

Animável: Não

A propriedade aligncontent nos permite ditar como qualquer espaço extra de direção cruzada em um contêiner flexível é distribuído entre e ao redor de linhas flexíveis. Isso é diferente da propriedade alignitems discutida anteriormente, que dita o posicionamento do item flexível dentro de cada linha flexível .

A propriedade aligncontent determina como as linhas flexíveis são alinhadas dentro de um contêiner flex de várias linhas. Quando há espaço extra na direção do eixo cruzado, você pode determinar se as linhas são agrupadas no início cruzado, na extremidade cruzada, centralizadas ou esticadas para ocupar todo o espaço disponível ou distribuídas pelo contêiner flexível com o espaço extra distribuído entre ou em torno das linhas flexíveis.

Pense no alinhamento do conteúdo como semelhante à forma como o conteúdo justificado alinha itens individuais ao longo do eixo principal do contêiner flex, mas para linhas flexíveis no eixo cruzado do contêiner. Essa propriedade só se aplica a contêineres flexíveis de várias linhas, não tendo efeito sobre contêineres flexíveis de linha única e sem embrulho.

Figure 2-31 demonstra os seis valores possíveis da propriedade aligncontent.

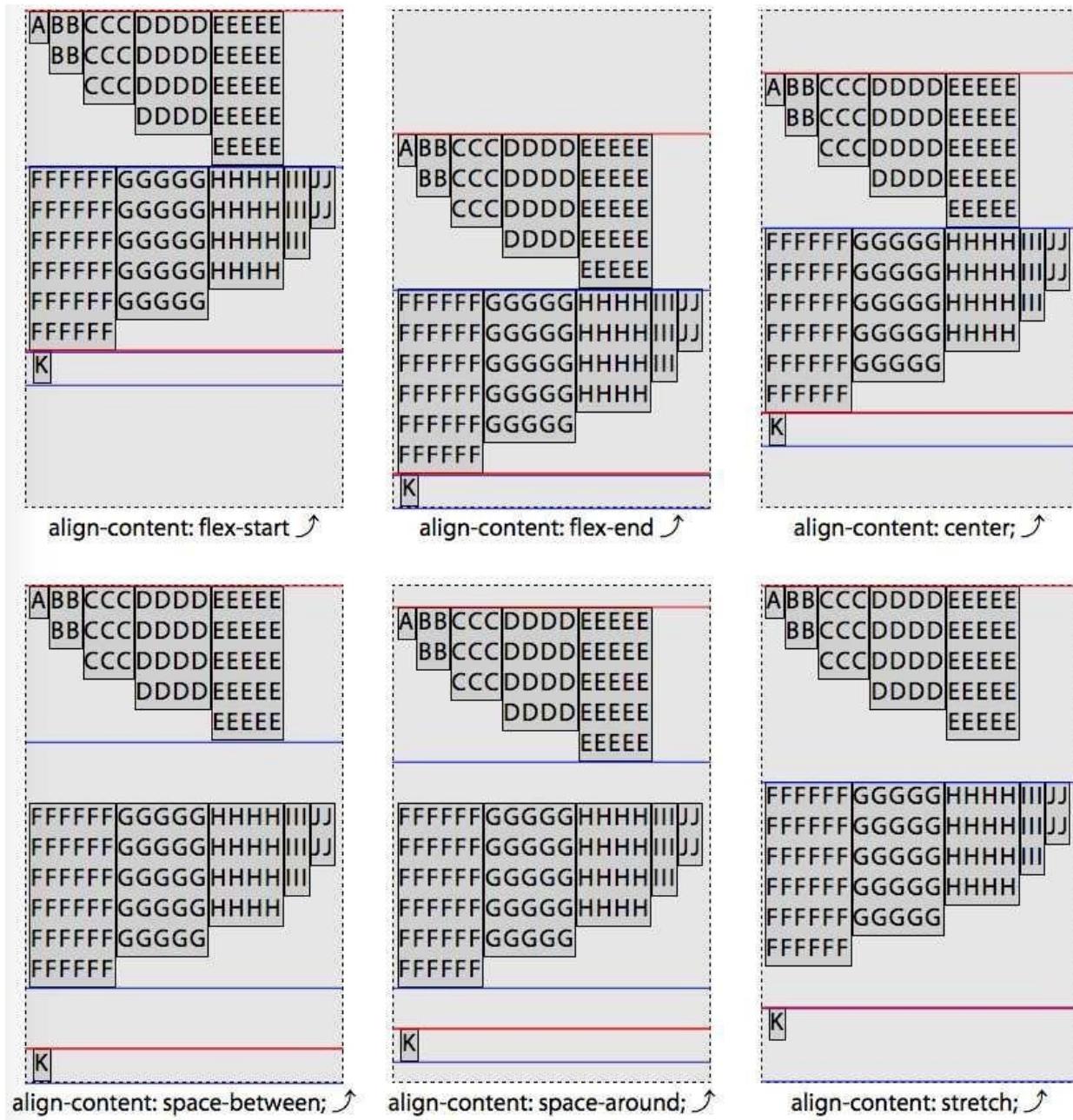


Figura 2-31. Valores da propriedade aligncontent do contêiner flexível



Usamos o CSS a seguir como base para os seis exemplos anteriores, sem margens nos itens flexíveis:

```
flex-container {
  display: flex;
```

```
flexflow: envoltório de linha; itens de alinhamento: flex-
start; borda: 1px tracejado; altura: 480px;
}
```

Distribuição de espaço extra

Na [Figura 2-31](#), cada contêiner flex tem três linhas flex. Com uma altura de 480 px, o contêiner flex é mais alto do que as alturas combinadas padrão das 3 linhas flex. Os itens mais altos em cada linha — E, F e K — são 150 px, 180 px e 30 px, respectivamente, para um total combinado de 360 px. Cada contêiner flexível tem 120 px extras de espaço livre na direção de tamanho cruzado. O lado de partida cruzada de cada linha flexível é denotado com uma linha vermelha, o lado da extremidade transversal com uma linha azul; eles podem parecer roxos quando se sobrepõem. Com cinco dos valores de alignitems, o espaço livre é distribuído fora das linhas flexíveis, como é mais aparente na [Figura 2-32](#). Com o alongamento, o espaço extra é distribuído uniformemente entre todas as linhas flexíveis, aumentando seu tamanho cruzado.

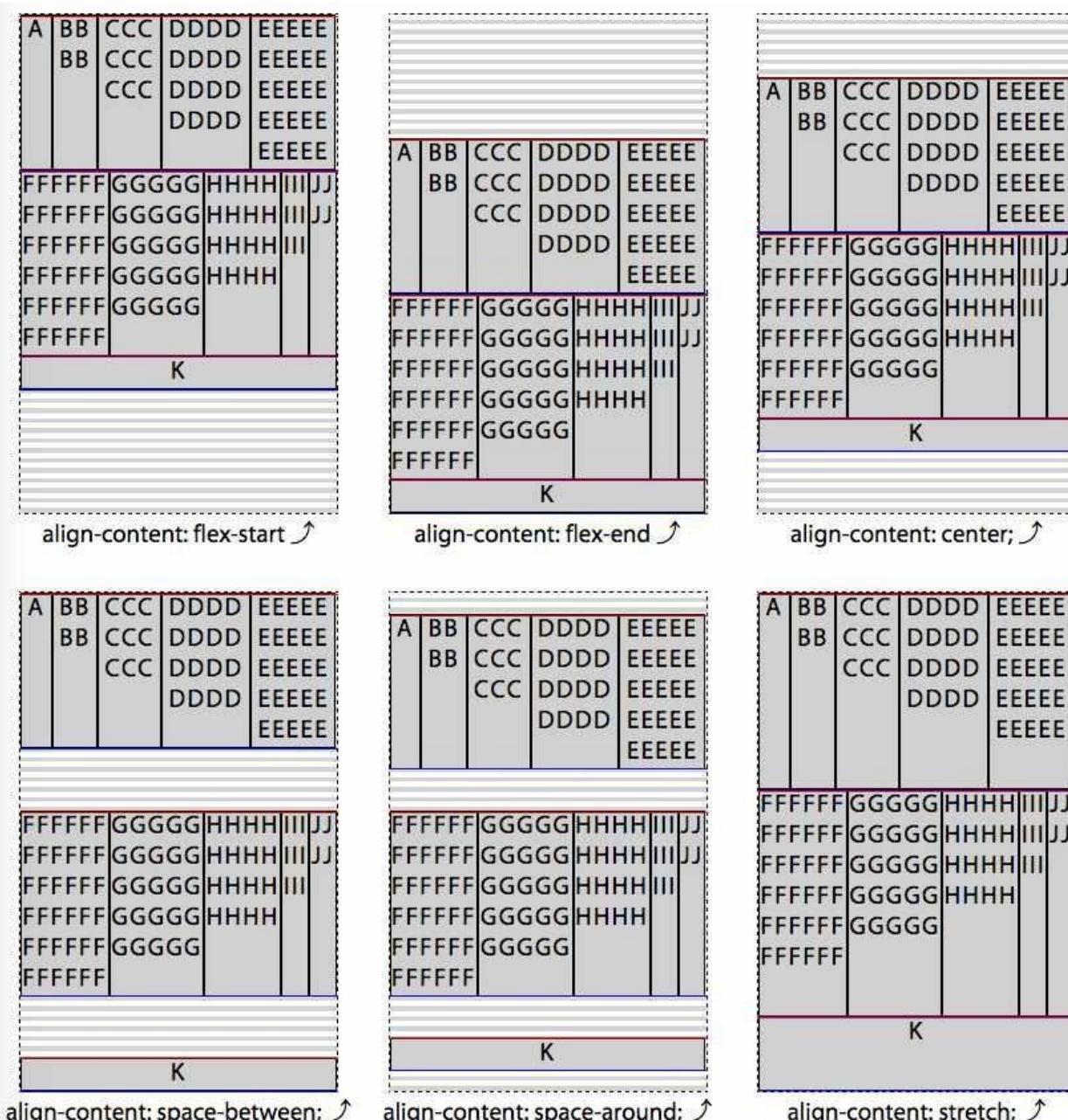


Figura 2-32. Distribuição de espaço extra para os diferentes valores de aligncontent



A Figura 2-32 reitera os seis valores possíveis da propriedade aligncontent , com alignitems: stretch e flex: 1 definido para permitir que os itens flex cresçam para ocupar suas linhas inteiras para causar o impacto dos valores aligncontent mais aparente (e, geralmente, para ser menos de uma ofensa):

```

flex-container { display: flex; flexflow: envoltório de
  linha; itens de alinhamento: alongamento; borda:
  1px tracejado; altura: 480px;
}
flex-items {
  flex: 1;
}

```

Assim como no exemplo anterior, com uma altura de 480 px, e com linhas flexíveis de 150 px, 180 px e 30 px de altura, temos 120 px de espaço livre ao longo do direção cruzada distribuída de forma diferente, dependendo do valor da propriedade aligncontent:

$$480 - (150 + 180 + 30) = 120$$

Como mostrado nos primeiros exemplos nas Figuras 2-31 e [2-32](#), com flex-start o 120 px está no lado da extremidade cruzada. Com o flex-end, os 120 px extras de espaço disponível estão no lado de partida cruzada. Com o centro, as linhas são centralizadas, com 60 px de espaço extra nos lados de partida cruzada e extremidade cruzada, como mostrado no exemplo superior direito de ambas as figuras. Com espaço entre, há 60 px entre pares adjacentes de linhas flexíveis, como mostrado no examples inferior esquerdo. Com o espaço ao redor, o espaço é distribuído uniformemente em torno de cada linha: os 120 px são distribuídos uniformemente, colocando 20 px de espaço não colapsado nos lados de partida cruzada e extremidade cruzada de cada linha flexível, de modo que há 20 px de espaço extra nos lados de partida cruzada e extremidade cruzada de o recipiente flexível e 40 px de espaço entre as linhas flexíveis adjacentes.

O valor de alongamento é diferente: com o alongamento, as linhas se esticam com o espaço extra distribuído uniformemente entre as linhas flexíveis em vez de entre elas. Neste caso, 40 px foram adicionados a cada uma das linhas flexíveis. Você observará no sexto exemplo nas Figuras 2-31 e [2-32](#), que não há área dentro do contêiner que não seja ocupada por uma linha flexível. Stretch é o valor padrão, pois você deseja preencher todo o espaço disponível.

Se não houver espaço suficiente para todas as linhas, elas transbordarão no cross-start, cross-end ou ambos, dependendo do valor da propriedade aligncontent, conforme mostrado na [Figura 2-33](#).

A [Figura 2-33](#) mostra os valores de alinhamento de conteúdo de tamanho quando as linhas flexíveis estouram seu contêiner flex pai. A única diferença no CSS entre isso e [A Figura 2-31](#) é a altura do contêiner flex. Definimos a altura: 240px para criar contêineres flexíveis não altos o suficiente para abranger todos os itens flex filhos:

```

flex-container { display: flex; flexflow: envoltório de linha;
  itens de alinhamento: flex-start; borda: 1px tracejado;
}

```

```
altura: 240px;  
}
```

Se as linhas flexíveis transbordarem o contêiner flex, o flex-start, o espaço-entre, e o alongamento transbordarem o lado da extremidade transversal, estiquem e centralizem o transbordamento uniformemente dos lados da extremidade transversal e do cross-start, e apenas flex-end transborda apenas no lado de partida cruzada.

ABB	CCC	DDDD	EEEEE	
BB	CCC	DDDD	EEEEE	
CCC	DDDD	EEEEE		
DDD	EEEEE			
	EEEEE			
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	
FFFFFF	GGGGG	HHHH		
FFFFFF	GGGGG			
K				

ABB	CCC	DDDD	EEEEE	
BB	CCC	DDDD	EEEEE	
CCC	DDDD	EEEEE		
DDD	EEEEE			
	EEEEE			
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	
FFFFFF	GGGGG	HHHH		
FFFFFF	GGGGG			
K				

ABB	CCC	DDDD	EEEEE	
BB	CCC	DDDD	EEEEE	
CCC	DDDD	EEEEE		
DDD	EEEEE			
	EEEEE			
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	
FFFFFF	GGGGG	HHHH		
FFFFFF	GGGGG			
K				

ABB	CCC	DDDD	EEEEE	
BB	CCC	DDDD	EEEEE	
CCC	DDDD	EEEEE		
DDD	EEEEE			
	EEEEE			
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	
FFFFFF	GGGGG	HHHH		
FFFFFF	GGGGG			
K				

ABB	CCC	DDDD	EEEEE	
BB	CCC	DDDD	EEEEE	
CCC	DDDD	EEEEE		
DDD	EEEEE			
	EEEEE			
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	
FFFFFF	GGGGG	HHHH		
FFFFFF	GGGGG			
K				

ABB	CCC	DDDD	EEEEE	
BB	CCC	DDDD	EEEEE	
CCC	DDDD	EEEEE		
DDD	EEEEE			
	EEEEE			
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	JJ
FFFFFF	GGGGG	HHHH	III	
FFFFFF	GGGGG	HHHH		
FFFFFF	GGGGG			
K				

Figura 2-33. Aparência da propriedade aligncontent quando as linhas estão transbordando o contêiner

aligncontent: flex-start

Com o aligncontent: flex-start, a borda de partida cruzada da primeira linha de itens flex será nivelada contra cross-start, com cada linha subsequente colocada nivelada contra a linha anterior, como mostrado na Figura 2-34.

ABBBBB	CCCC	DDDD	EEEEE	
BBBB	CCCC	DDDD	EEEEE	
CCC	DDDD	EEEEE		
DDDD	EEEEE			
EEEEE				

FFFFFF	GGGGGG	HHHH	IIIIJJ	
FFFFFF	GGGGGG	HHHH	IIIIJJ	
FFFFFF	GGGGGG	HHHH	IIII	
FFFFFF	GGGGGG	HHHH		
FFFFFF	GGGGG			
FFFFFF				

flex-flow: row wrap; ↗
 align-content: flex-start;

EEEEE	DDDD	CCC	BBA	
EEEEE	DDDD	CCC	BB	
EEEEE	DDDD	CCC		
EEEEE	DDDD			
EEEEE				

JJ	IIII	HHHH	GGGGGG	FFFFFF
JJ	IIII	HHHH	GGGGGG	FFFFFF
IIII	HHHH	GGGGGG	FFFFFF	
HHHH	GGGGGG	FFFFFF		
GGGGGG	FFFFFF			
FFFFFF				

flex-flow: row wrap-reverse; ↗
 align-content: flex-start;

K				
FFFFFF				
FFFFFF	GGGGGG			
FFFFFF	GGGGGG	HHHH		
FFFFFF	GGGGGG	HHHH	IIII	
FFFFFF	GGGGGG	HHHH	IIIIJJ	
FFFFFF	GGGGGG	HHHH	IIIIJJ	
	EEEEE			
	DDDD			
CCC	DDDD	EEEEE		
BBB	CCC	DDDD	EEEEE	
ABBB	CCC	DDDD	EEEEE	

flex-flow: row wrap-reverse; ↗
 align-content: flex-start;

K				
FFFFFF				
	GGGGGG			
	GGGGGG	HHHH		
	GGGGGG	HHHH	IIII	
	GGGGGG	HHHH	IIII	JJ
	GGGGGG	HHHH	JJ	
	EEEEE			
	DDDD			
IIII	DDDD	CCC		
IIII	DDDD	CCC	BB	
IIII	DDDD	CCC	BB	A

flex-flow: row-reverse wrap-reverse; ↗
 align-content: flex-start;

Figura 2-34. Lembre-se, o wrap-reverse inverte a direção cruzada; com flex-start, o excesso de espaço ou linhas transbordantes, se houver, está sempre na extremidade transversal

Em outras palavras, todas as linhas flexíveis serão agrupadas na borda de partida cruzada do contêiner flex. Observe que todo o espaço extra (os 120 px extras neste caso) está no cross-end. A lembrança da direção cruzada é invertida com flex-direction: wrap-reverse (consulte "The flex-direction Property"), como mostra a [Figura 2-15](#). Se não houver espaço suficiente para todas as linhas, elas transbordarão no cruzamento.

alinharccontent: flex-end

Com o aligncontent: flex-end, a borda cross-end da última linha de itens flex é colocada nivelada com a borda cross-end do contêiner flex, e cada linha anterior é colocada nivelar com a linha subsequente, deixando todo o espaço em branco extra no início cruzado. Em outras palavras, todas as linhas flexíveis serão agrupadas contra a borda transversal do contêiner flex. Se o conteúdo estourar o contêiner flexível , ele o fará no lado de partida cruzada.

aligncontent: centro

Declarando aligncontent: o centro agrupa as linhas flexíveis juntas, de modo que elas são niveladas umas contra as outras, assim como são agrupadas com flex-start e flex-end, mas no centro do recipiente flex.

o centro, o flex-start e o flex-end têm a extremidade transversal de cada linha sendo nivelada contra o cross-start da linha subsequente. Em vez de todas as linhas serem agrupadas no cross-start do contêiner como no flex-start ou no cross-end como no flex-end, com o aligncontent: no centro, o grupo de linhas está centralizado entre os do contêiner flex cross-start e cross-end, com quantidades iguais de espaço vazio — 60 px de cada lado neste caso — entre a aresta de arranque cruzado do contentor flex e a borda de partida cruzada da primeira linha flex e entre a borda transversal do contêiner flex e a borda cruzada da última linha flex, conforme mostrado no exemplo superior direito em 2-31 e [2-32](#). Se os itens flex transbordarem o contêiner flex, as linhas transbordarão igualmente em ambas as direções passando pelas bordas de início cruzado e extremidade cruzada do contêiner, conforme mostrado no exemplo de centro-esquerda na [Figura 2-33](#) .

aligncontent: espaço entre

Quando o aligncontent: space-between é definido, as linhas flexíveis são distribuídas uniformemente no contêiner flex. A *distribuição uniforme* é baseada no espaço disponível, não no tamanho das linhas; o espaço extra é dividido igualmente entre as linhas, não proporcionalmente.

Se nos lembrarmos de voltar para justificar-conteúdo: espaço-entre, o primeiro item flexível é nivelado contra o main-start. O segundo item flexível, se houver um, é nivelado contra a extremidade principal. O resto dos itens flex, se houver, são espalhados com quantidades iguais de espaço livre distribuído entre os itens flex. Isso é semelhante a como o alinhamento de conteúdo: espaço entre funciona. Se houver mais de uma linha flexível, a primeira linha será nivelada contra a partida cruzada do contêiner, a última linha será nivelada contra a extremidade cruzada do contêiner e o espaço extra disponível será distribuído uniformemente entre as linhas adicionais, se houver. O espaço extra é distribuído uniformemente, não proporcionalmente. O espaço entre quaisquer duas linhas flexíveis dentro do contêiner flex é igual, mesmo que os tamanhos cruzados das várias linhas flexíveis difiram.

NOTA

Somente contêineres flexíveis com várias linhas podem ter espaço livre no eixo cruzado para que as linhas sejam alinhadas |. Se houver apenas uma linha, um propriedade | alinhamento de conteúdo não afetará um distribuição fazer conteúdo. Em contêineres flexionar .com uma única linha de itens flexionar, um linha solitária se estende | para preencher todo o espaço disponível.

A linha do meio, se houver um número ímpar de linhas, não é necessariamente centralizada, pois as linhas não têm necessariamente todas dimensões cruzadas equivalentes. Em vez disso, o espaçamento entre quaisquer duas linhas adjacentes é o mesmo: há a mesma quantidade de espaço entre a primeira e a segunda linha que há entre qualquer outra flexão adjacente, como mostrado no exemplo inferior esquerdo na [Figura 2-32](#). Neste caso, temos 120 px no total de espaço livre, que é dividido igualmente, com metade, ou 60 px, entre a primeira e a segunda flex lines, e 60 px entre a segunda e a terceira flex linhas:

$$120\text{px} / 2 = 60\text{px}$$

Se não houver espaço suficiente para abranger todas as linhas flexíveis, o espaço livre é

negativo e alinhamento de conteúdo: a aparência do espaço entre é idêntica à flex-start, com o estouro no lado da extremidade transversal.

aligncontent: espaço ao redor

O valor space-around distribui as linhas dentro de um contêiner flex de várias linhas uniformemente, como se todas as linhas flex tivessem margens iguais e não colapsantes em ambas as lados cruzado inicial e cruzado . Porque há uma distribuição igual do extra espaço disponível em torno de cada linha, o espaço entre as bordas do recipiente e a primeira e a última linha flexível é metade do tamanho da distância entre quaisquer duas flex Linhas. A distribuição do espaço extra é mostrada na [Figura 2-35](#).

A	B	C	D	E
B	C	D	D	E
C	C	D	D	E
	D	D	D	E
			E	E
<hr/>				
F	G	H	I	J
F	G	H	I	J
F	G	H	I	
F	G	H		
F	G			
F				
<hr/>				
K				

align-content: space-around; ↑

Figura 2-35. Distribuição do espaço livre ao alinhar conteúdo: o espaço ao redor é definido

Neste caso, temos 120 px de espaço livre distribuídos para 3 linhas flexíveis, com metade de cada linha flexível sendo espaço extra sendo adicionado à borda de partida cruzada e metade sendo adicionada à borda da extremidade cruzada:

$$(120\text{px} / 3 \text{ linhas}) / 2 \text{ lados} = 20\text{px por lado}$$

Neste exemplo, há 20 px nos lados externos das linhas flex, com o dobro disso entre quaisquer duas linhas flex adjacentes. Há 20 px entre a borda de partida cruzada e a primeira linha, 20 px entre a borda transversal e a última linha, e 40 px (duas vezes 20 px) entre a primeira e a segunda linhas flex e a segunda e terceira linhas flex.

Se não houver espaço suficiente para as linhas flexíveis e elas transbordarem o contêiner flex, o espaço livre será negativo e o conteúdo de alinhamento: o valor space-around aparecerá idêntico 1 ao centro, com o estouro uniformemente distribuídos entre os lados de partida cruzada e de extremidade cruzada.

alinhcontent: alongamento

Omitir a propriedade aligncontent ou defini-la explicitamente como o valor padrão aligncontent : stretch faz com que as linhas se estendam para ocupar todo o espaço extra disponível.

Se você der uma olhada de perto na [Figura 2-32](#), notará que o alongamento e o espaço entre eles são, na verdade, bastante diferentes. No espaço-méio, o espaço extra é distribuído *entre* as linhas. No trecho, o espaço extra é dividido no número de linhas e adicionado *às* linhas. As linhas originais tinham 150 px, 180 px e 30 px de altura, respectivamente. Quando definido para esticar, os 120 px extras são adicionados às linhas em quantidades iguais. Como temos três linhas flexíveis e 120 px de espaço extra, cada linha flex cresce 40 px, dando-nos linhas flexíveis que são 190 px, 220 px e 70 px, respectivamente, como mostrado na [Figura 2-36](#).

A	B	C	D	E
BB	CCC	DDDD	EEEE	
	BB	CCC	DDDD	EEEE
	CCC	DDDD	EEEE	
		DDDD	EEEE	
			EEEE	
				EEEE
FFFFF	GGGGG	HHHH	III	JJ
FFFFF	GGGGG	HHHH	III	JJ
FFFFF	GGGGG	HHHH	III	
FFFFF	GGGGG	HHHH		
FFFFF	GGGGG			
FFFFF				
K				

align-content: stretch; ↗

Figura 2-36. Quando definido para esticar, o espaço extra é distribuído igualmente, não proporcionalmente, para cada linha

Esse espaço extra, neste exemplo, aparece na extremidade cruzada de cada linha individual, uma vez que incluímos itens de alinhamento: flex-start. Se tivéssemos usado itens de alinhamento: flex-end, o espaço extra em cada linha teria sido aparente no flex-start das linhas individuais.

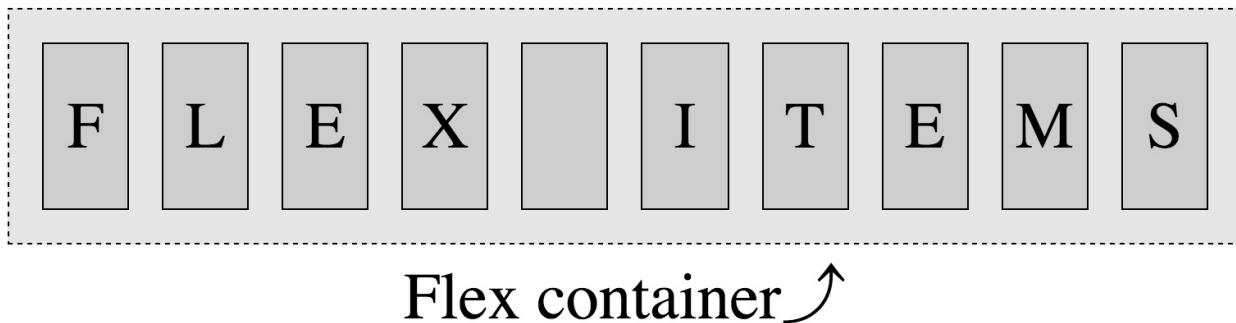
Se não houver espaço suficiente para caber em todas as linhas flexíveis, as linhas se comportarão como se estivessem definidas para flex-start, com a primeira linha flexível nivelada contra a borda de partida cruzada de o recipiente flex.

Estamos dando uma olhada nas propriedades do contêiner flex. É hora de dar uma olhada nas propriedades diretamente aplicadas aos itens flexíveis.

¹ O layout horizontal está incorreto para o mongol. Por causa dessa exceção, não é realmente uma "preferência".

Capítulo 3. Itens Flex

Nos capítulos anteriores, aprendemos a dispor globalmente todos os itens flexíveis dentro de um contêiner flex adicionando valores de propriedade flexbox a esse contêiner. A especificação de layout de caixa flexível fornece várias propriedades adicionais aplicáveis diretamente aos itens flexíveis. Com essas propriedades específicas do item flex, podemos controlar com mais precisão o layout dos itens flexíveis não anônimos dos contêineres flexíveis individuais.



*Figura 3-1. Itens com display: flex; tornam-se contêineres flexíveis e não absolutamente posicionados
crianças se tornam itens flexíveis* 

Agora que temos uma boa compreensão das propriedades aplicáveis ao contêiner flex, é hora de se concentrar nas propriedades aplicáveis aos itens flex.

O que são itens Flex ?

Criamos contêineres flexíveis simplesmente adicionando um display: flex ou display: inline-flex a um elemento que tem nós filho. Os filhos desses contêineres flexíveis são chamados de *itens* flexíveis, sejam eles nós DOM, nós de texto não vazios ou conteúdo gerado.

Quando se trata de filhos de nó de texto de contêineres flex, se o nó de texto não estiver vazio (contendo conteúdo diferente de espaço em branco), ele será encapsulado em um item flexível anônimo, comportando-se como seus irmãos de item flexível. Embora esses itens flex anônimos herdem todas as propriedades flex definidas pelo contêiner flex, assim como seus irmãos de nó DOM, eles não são diretamente direcionáveis com CSS. Portanto, não podemos definir diretamente nenhuma das propriedades específicas do item flexível neles.

O conteúdo gerado pode ser direcionado diretamente; portanto, todas as propriedades discutidas neste capítulo se aplicam igualmente ao conteúdo gerado e aos nós do elemento.

Os nós de texto somente de espaço em branco dentro de um contêiner flexível são ignorados, como se seus

A propriedade display foi definida como none, como mostra o exemplo de código a seguir:

```
nav ul {  
  display: flex;  
}  
  
<navegação>  
<ul>  
  <li><a href="#1">Link 1</a></li>  
  <li><a href="#2">Link 2</a></li>  
  <li><a href="#3">Link 3</a></li>  
  <li><a href="#4">Link 4</a></li>  
  <li><a href="#5">Link 5</a></li>  
</ul>  
</navegação>
```

No código anterior, com sua propriedade de exibição definida como flex, a lista não ordenada é o contêiner flex e seus itens de lista filho são todos itens flex. Esses itens de lista, sendo itens flexíveis, são caixas de nível flexível, semanticamente ainda listam itens, mas não itens de lista em sua apresentação. Eles também não são caixas de nível de bloco. Em vez disso, eles participam do contexto de formatação flexível de seu contêiner. O espaço em branco é ignorado. Os links, que são descendentes dos itens flex, não são afetados pela inclusão de flex display no pai de seus pais.

Recursos do Flex Item

As margens dos itens flexíveis não colapsam. As propriedades float e clear não têm efeito sobre

itens flexíveis e não tiram um item flexível do fluxo.

Além disso, o alinhamento vertical não tem efeito sobre um item flexível. No entanto, o carro alegórico ainda pode afetar a geração de caixa influenciando o valor calculado da propriedade de exibição, como mostra o exemplo de código a seguir:

```
à parte {  
  display: flex;  
}  
img {  
  flutuação: esquerda;  
}  
  
<à parte>  
  <! -- este é um comentário -->  
  <h1>Cabeçalho</h1>  
  
    
  Algum texto  
</à parte>
```

Neste exemplo, o lado é o contêiner flex. Os nós de texto somente de comentário e espaço em branco são ignorados. O nó de texto que contém "algum texto" é quebrado em um item flexível anônimo. O cabeçalho, a imagem e o nó de texto que contêm "algum texto" são todos itens flexíveis. Como a imagem é um item flexível, o float é ignorado. Mesmo que as imagens e os nós de texto sejam nós de nível embutido, sendo itens flexíveis, desde que não estejam absolutamente posicionados, eles são bloqueados:

```
à parte {  
  display: flex;  
}  
  
<à parte>  
  <! -- um comentário -->  
  <h1>Cabeçalho</h1>  
  
    
  Algum texto <a href="foo.html">com um link</a> e mais texto  
</à parte>
```

No último exemplo, a marcação é semelhante ao código no segundo exemplo, com a adição de um link dentro do nó de texto não vazio. Neste caso, estamos criando cinco itens flexíveis. Os nós de texto somente de comentário e espaço em branco são ignorados. O cabeçalho, a imagem, o nó de texto before o link , o link e o nó de texto após o link são todos os itens flexíveis. Os nós de texto que contêm "algum texto" e " e mais texto" são quebrados em itens flexíveis anônimos. Como esses dois filhos de nó de texto do <à parte> não são contíguos, eles são encapsulados em itens flexíveis anônimos separados. O cabeçalho, a imagem e o link, sendo nós DOM, podem ser direcionados diretamente com CSS. Os contêineres flexíveis anônimos não são diretamente direcionáveis.

Posicionamento absoluto

Enquanto um valor de `float: left` ou `float: right` on the child of a flex container não flutua o item – como o filho é um item flex e o float é ignorado – definindo a posição: absoluto é uma história diferente. Os filhos absolutamente posicionados de contêineres flexíveis, assim como qualquer outro elemento absolutamente posicionado, são retirados do fluxo do documento.

Eles não são convertidos em itens flexíveis. Eles não estão no fluxo de documentos. Eles não participam do layout flex. No entanto, eles podem ser afetados pelas propriedades definidas no contêiner flex , assim como um filho pode ser afetado por um elemento pai que não é um contêiner flex. Além de herdar propriedades herdáveis como fariam se o pai não fosse um contêiner flexível, as propriedades do pai afetam a origem do posicionamento.

O filho absolutamente posicionado de um contêiner flexível é afetado pelo valor de conteúdo justificado do contêiner flexível pai e seu próprio valor `alignself`, se houver um. Por exemplo, se você definir `align-content: center`; na criança absolutamente posicionada, ela será, por padrão, centralizada no eixo cruzado do pai do contêiner flexível. A propriedade `order` pode não afetar onde o filho do contêiner flexível posicionado de forma absoluta é desenhado, mas afeta a ordem de quando ele é desenhado em relação a seus irmãos.

largura mínima

Na [Figura 3-2](#), você observará que a linha definida como o padrão nowrap estoura o contêiner flex. Isso ocorre porque, quando se trata de itens flexíveis, o valor implícito de min-width é automático, em vez de 0. Originalmente na especificação, se os itens não se encaixassem nesse único eixo principal, eles encolheriam. No entanto, a especificação de largura mínima foi alterada. Na especificação CSS 2.1, o valor padrão para min-width é 0. Agora, para itens flexíveis, o tamanho mínimo implícito é auto, não 0.

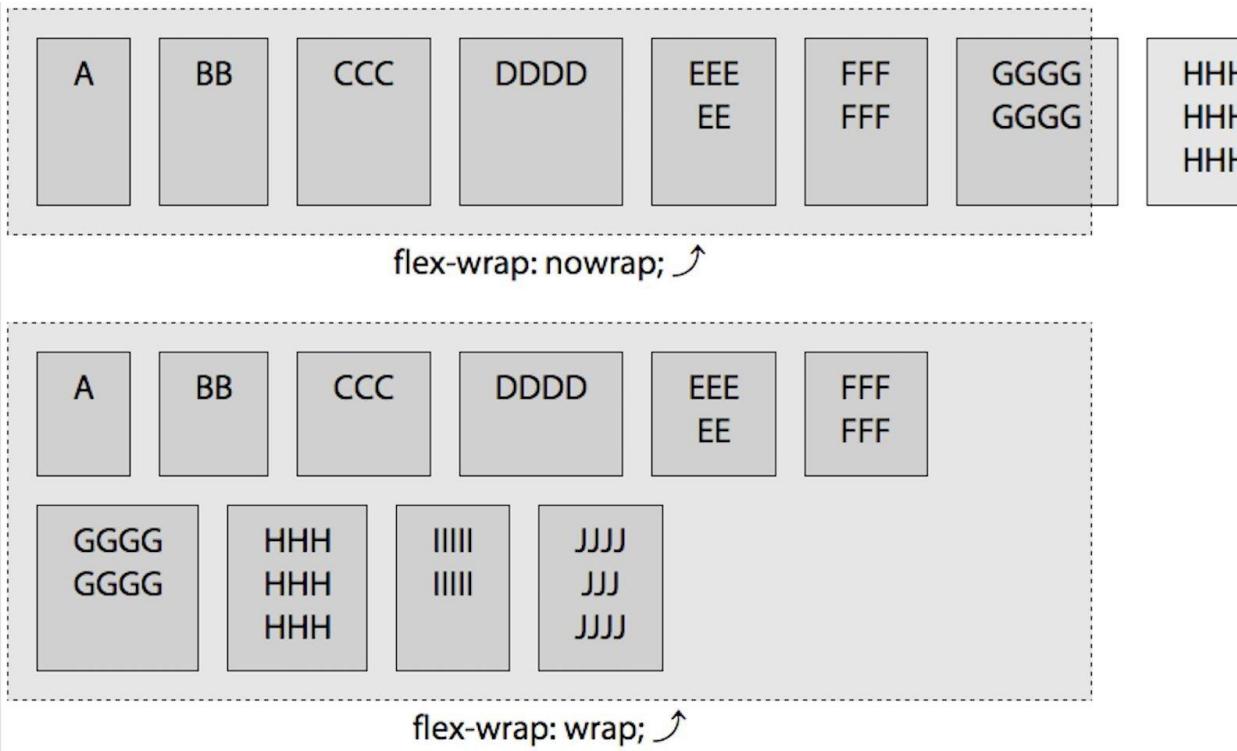


Figura 3-2. Flexibilizar itens que transbordam seu contêiner quando a largura mínima tiver como padrão automático, a menos que a encapsulação seja

permitido 

Se você definir a largura mínima para uma largura mais estreita que o valor calculado de auto — por exemplo, se declararmos largura mínima: 0; — os itens flexíveis no exemplo nowrap diminuirão para serem mais estreitos do que os itens flex em contêineres que não têm permissão para empacotar, como mostra a [Figura 3-3](#). Isso é o que o Safari 9 exibe por padrão, pois não atualizou a alteração implícita de largura mínima. Abordaremos a largura do item flexível em profundidade de reate e em ["The flex Property"](#).

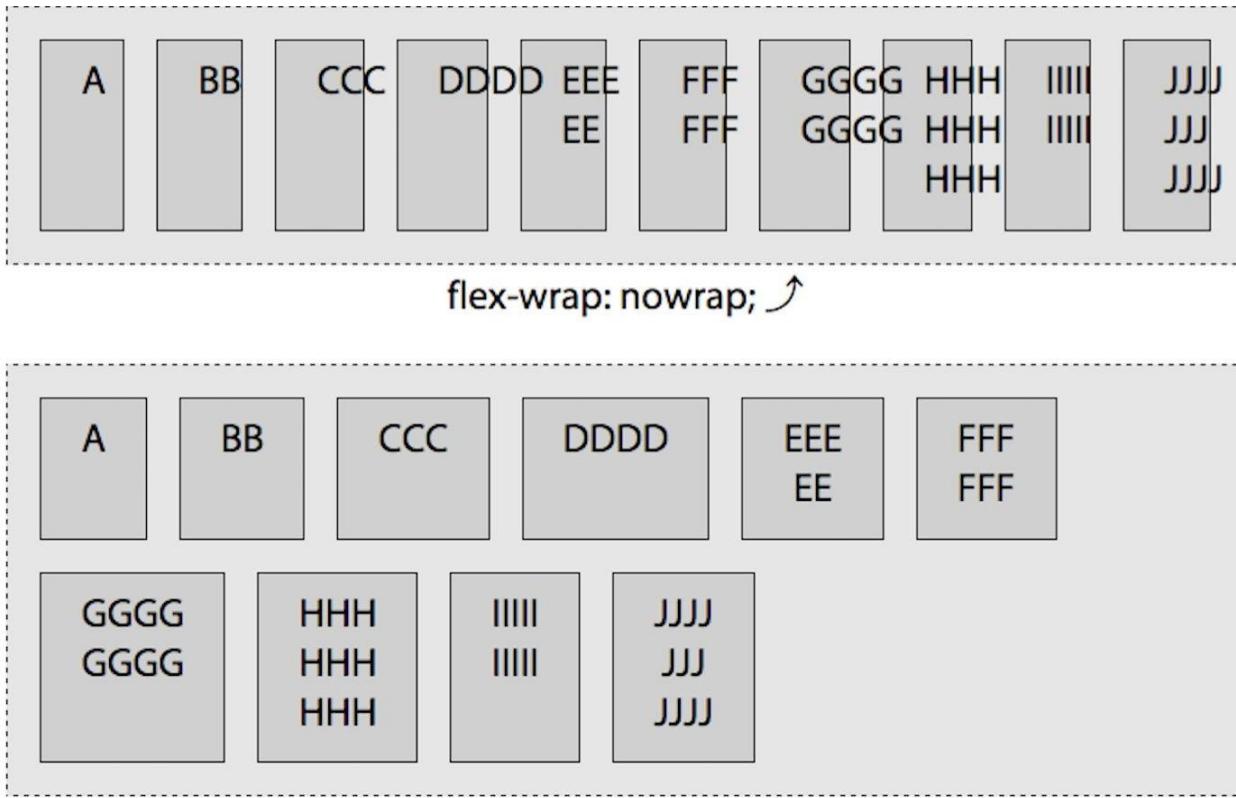


Figura 3-3. Os itens flexíveis em contêineres sem encapsulamento diminuirão se a largura mínima for explicitamente definida como 0, que é o padrão no Safari 9 

Propriedades específicas do item flexível

Embora o alinhamento, a ordem e a flexibilidade dos itens flexíveis sejam, até certo ponto, controláveis por meio de propriedades definidas em seu contêiner flex pai, existem várias propriedades que podem ser aplicadas a itens flexíveis individuais para um controle mais granular.

A propriedade `flex` shorthand, juntamente com suas propriedades componentes de `flex-grow`, `flex-shrink` e `flex-basis`, controlam a flexibilidade dos itens flex. O `align-self` ajuda a controlar o alinhamento de um item flexível. A propriedade `order` fornece um controle mais granular da ordenação visual de itens individuais ou grupos de flex. Todas essas propriedades são discutidas nas seções a seguir.

O aspecto definidor do layout flex é a capacidade de tornar os itens flex "flex": alterando sua largura ou altura para preencher o espaço disponível na dimensão principal. Um contêiner flexível distribui espaço livre para seus itens proporcionalmente ao seu fator de crescimento flexível ou os encolhe para evitar o estouro proporcional ao fator de encolhimento flexível.

Definir a propriedade de taquigrafia `flex` em um item flex ou definir as propriedades individuais

que compõem a taquigrafia permite que os desenvolvedores definam os fatores de crescimento e encolhimento. Se houver excesso de espaço, você pode dizer aos itens flexíveis para crescer para preencher esse espaço. Ou não. Se não houver espaço suficiente para caber todos os itens flexíveis dentro do contêiner flexível em seus tamanhos definidos ou padrão, você poderá dizer aos itens flexíveis que diminuam proporcionalmente para caber no espaço. Ou não. Tudo isso é feito com a propriedade `flex`, que é uma propriedade abreviada para `flexgrow`, `flexshrink` e `flexbasis`. Embora essas três subpropriedades possam ser usadas separadamente, é altamente recomendável sempre usar a taquigrafia `flex`.

Apresentaremos brevemente a propriedade de taquigrafia flexível e, em seguida, mergulharemos mais profundamente nos três valores de mão longa que a compõem. É importante entender completamente o que cada componente está fazendo para que você possa usar efetivamente a propriedade de taquigrafia `flex`.

A propriedade `flex`

A taquigrafia `flex`, uma propriedade `flex item`, é composta pelas propriedades `flexgrow`, `flexshrink` e `flexbasis`, que definem o fator de crescimento `flex`, o fator `flex shrink` e a base flexível, respectivamente.

FLEX	
Valores	<code><flexgrow> <flexshrink>? <flexbasis></code>] <code>nenhum</code> <code>Automático</code>
Valor inicial	<code>flex-grow: 1;</code> <code>flex-shrink: 1;</code> <code>flex-base: 0%;</code>
Aplica-se um:	Itens Flex (filhos de destinatários)
Herdados:	Não
Porcentagens:	Válido apenas para o valor <code>flexbasis</code> , em relação ao tamanho principal

A propriedade `flex` especifica os componentes de um comprimento flexível: o "comprimento" do item `flex` é o comprimento do item `flex` ao longo do eixo principal (consulte "[Entendendo eixos](#)"). Quando uma caixa é um item `flex`, `flex` é consultado em vez da propriedade `main-size` (altura ou largura) do item `flex` para determinar o tamanho da caixa. Os "componentes" da propriedade `flex` incluem o fator de crescimento `flex`, o fator `flex shrink` e a base `flex`. Se o destino de um seletor não for um item `flex`, a aplicação da propriedade `flex` a ele não terá efeito.

A base flexível determina como os fatores de crescimento e encolhimento da flexão são implementados. Como o próprio nome sugere, o componente flexbasis da taquigrafia flex é a base sobre a qual o item flex determina o quanto ele pode crescer para preencher o espaço disponível ou quanto ele deve encolher para caber todos os itens flexíveis quando não há espaço suficiente. É o tamanho inicial de cada item flexível e pode ser restrito a esse tamanho específico especificando 0 para os fatores de crescimento e encolhimento:

```
.flexItem {  
  flex: 0 0 200px;  
}
```

No trecho de código anterior, o item flex terá um tamanho principal de exatamente 200 px, já que a base flexível é de 200 px e não é permitido crescer nem encolher.

É importante entender os três componentes que compõem a propriedade taquigráfica flex, que é a propriedade que você deve empregar. A razão pela qual é aconselhável usar o flex em vez de suas três subpropriedades individuais é porque a especificação (e, portanto, o navegador) redefine quaisquer valores flexíveis ausentes para padrões sensíveis, que nem sempre são os padrões de propriedade individual.

Para garantir que você se adapte totalmente à flexão, vamos mergulhar profundamente em seus três componentes.

A ordem de flexão é importante, com o primeiro número de flutuação sendo o fator de crescimento.

A propriedade flexgrow

A propriedade flexgrow define se um item flex pode crescer quando há espaço disponível e, se é permitido crescer e há espaço disponível, quanto crescerá proporcionalmente em relação ao crescimento de outros irmãos de itens flexíveis.

FLEXGROW

Valores <número>

Valor inicial: flex-grow: 0; 1

quando omitido como parte da

Aplica-se um: Itens Flex (filhos de destinatários)

Herdados: Não

Animável: Sim

O valor flexgrow e da parte flexgrow da taquigrafia flex é sempre um número. Números negativos não são válidos. Os valores flutuantes, desde que sejam maiores que 0, são válidos.

O valor especifica o fator de crescimento, que determina o quanto o item flex crescerá em relação ao restante dos irmãos do item flex à medida que o espaço livre do contêiner flex estiver Distribuído. Se houver algum espaço disponível dentro do contêiner flex, o espaço será distribuído proporcionalmente entre as crianças com um fator de crescimento positivo diferente de zero com base nos vários valores desses fatores de crescimento.

Por exemplo , com o contêiner flexível horizontal de 750px de largura com três itens flexíveis todos definidos como largura: 100px , como nos quatro exemplos da [Figura 3-4](#), você pode ditar nenhum, um, dois ou todos os três itens flexíveis para crescer para preencher os 450px extras de espaço disponível.

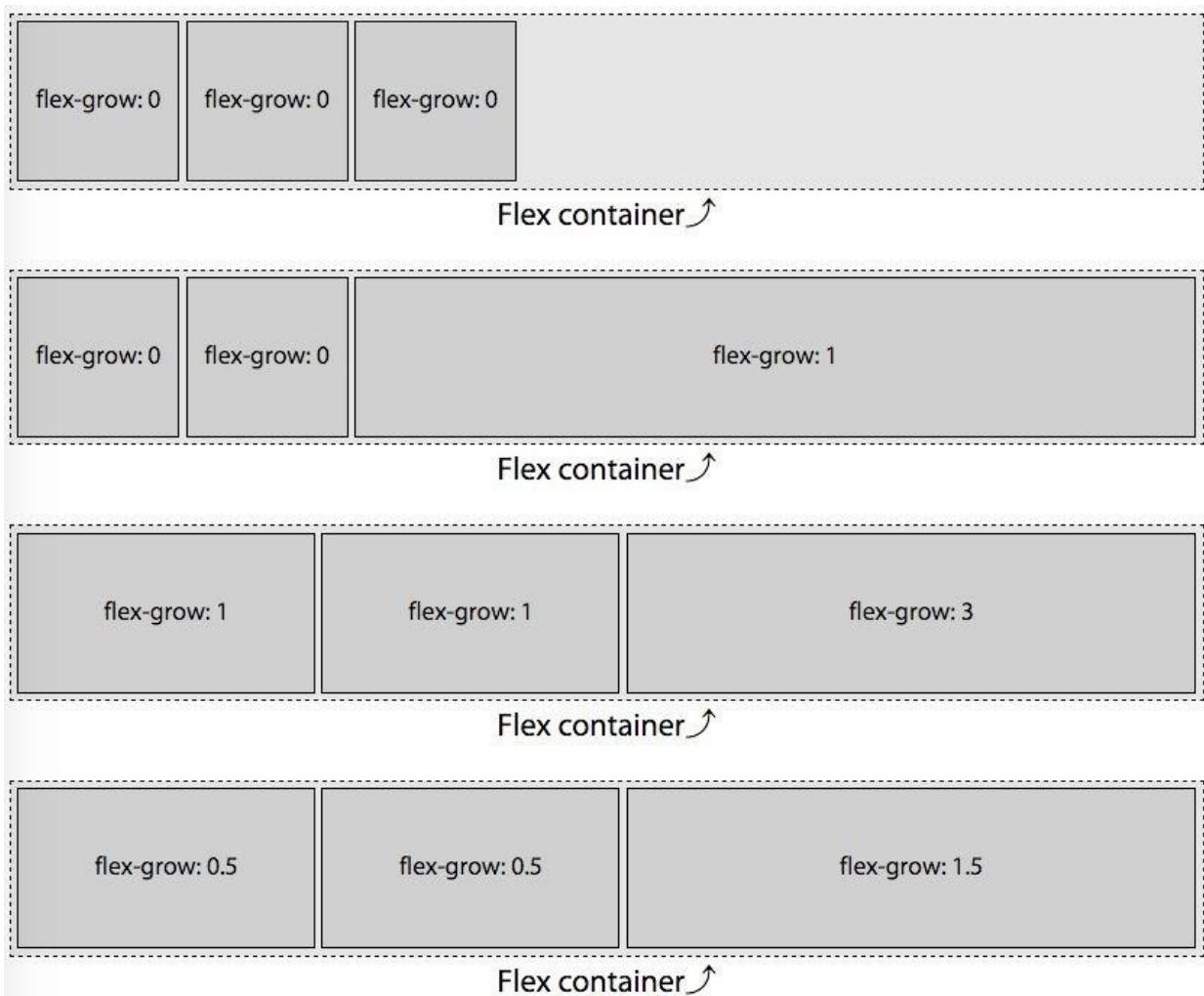


Figura 3-4. Com um fator de crescimento de 0, o item flex não crescerá; qualquer valor positivo permitirá que o item crescer proporcionalmente ao valor 

Conforme observado em "["min-width"](#)", se nenhuma largura ou base flexível for definida, a base flex terá como padrão automático, o que significa que cada base de item flexível é a largura de seu conteúdo não encapsulado. [auto](#) é um valor especial: o padrão é conteúdo, a menos que o item tenha uma largura definida nele, momento em que o flexbasis se torna essa largura. O valor [auto](#) é discutido em "["auto"](#)". Se não tivéssemos definido a largura, neste cenário de exemplo, com nosso tamanho de fonte pequeno, teríamos mais de 450 px de espaço distribuível ao longo do eixo principal.

NOTA

SO exemplos nesta seção | descrevem SO conceitos básicos fazer fator de crescimento. O tamanho principal | de micrômetro item flex é impactado | pelo espaço | disponível, o fator de crescimento de todos os itens flexionar , bem como um base flexível do item. Ainda temos que cobrir o flexbasis. Revisitaremos os fatores de crescimento e encolhimento e esses exemplos quando aprendermos sobre [uma base flexível](#) |.

Observar que um quantidade de espaço | distribuível | depende da base. Na [Figura 3-4](#), como um largura dos itens flexíveis foi ajustada para 100 px cada e nenhuma base foi definida, temos 450 px de espaço distribuível. Quando uma largura diferente ou nenhuma largura é declarada, ou se uma base é definida, o espaço | distribuível | é diferente, como

Qualquer valor positivo maior que 0 para o fator de crescimento significa que o item flexível pode crescer para preencher uma parte ou todo o espaço disponível. Você pode dizer aos itens para não crescerem com um fator de crescimento de 0, como demonstrado no primeiro exemplo da Figura 3-

4. Você pode misturar e combinar: cultivando alguns itens flexíveis e não outros, como demonstrado no segundo exemplo.

Nos 4 exemplos da [Figura 3-4](#), como o contêiner flex tem 750 px de largura e cada um dos 3 itens flex tem uma largura: 100px declarado, há 450 itens extras pixels a serem distribuídos entre os itens flexíveis que podem crescer:

$$750\text{px} - (3 * 100\text{px}) = 450\text{px}$$

Temos 450 px de espaço vazio na direção final principal, pois os itens de alinhamento têm como padrão o flex-start. O primeiro exemplo não tem fatores de crescimento definidos, portanto, nenhum item flexível cresceu.

Fator de crescimento não nulo

No segundo exemplo da [Figura3-4](#), os dois primeiros itens flexíveis, com um fator de crescimento definido como 0, não crescerão. Apenas o terceiro item flex tem um valor maior que 0, com flexgrow: 1 conjunto, o que significa que o único elemento com um valor de fator de crescimento positivo ocupará todo o espaço extra disponível.

Os dois primeiros itens flexíveis com fator de crescimento flexível de 0 permanecerão em 100px de largura, mesmo que o conteúdo não se encaixe. Apenas o terceiro item flexível pode crescer e, portanto, deve crescer, ocupando os 450 pixels extras para se tornar 550 px de largura.

O fator de crescimento precisa ser um valor flutuante positivo e não nulo para que o item flexível cresça. Neste exemplo, o fator de crescimento foi 1, mas se tivesse sido 0,000001 ou 10000000, o layout seria o mesmo.

Growing Proporcionalmente Baseado no Fator de Crescimento

Se todos os itens forem autorizados a crescer, o excesso de espaço é distribuído proporcionalmente com base nos fatores de crescimento flexíveis. No terceiro exemplo, com dois itens flex com um fator de crescimento de 1 e um item flex com um fator de crescimento de 3, temos um total de cinco fatores de crescimento:

$$(2 \times 1) + (1 \times 3) = 5$$

Com 5 fatores de crescimento, e um total de 450 px precisando ser distribuído, cada fator de crescimento vale 90 px:

$$450\text{px} / 5 = 90\text{px.}$$

Antes de serem autorizados a crescer com base no fator de crescimento do item flexível individual, os itens flexíveis tinham cada um 100 px de largura, conforme estabelecido pela propriedade width. Com cada fator de crescimento sendo de 90 px, temos 2 itens flex com uma largura de 190 px cada e o último item flex tem uma largura de 370 px:

$$100\text{px} + (1 \times 90\text{px}) = 190\text{px} \quad 100\text{px} + (3 \times 90\text{px}) = 370\text{px}$$

No último exemplo, temos um total de 2,5 fatores de crescimento:

$$(2 \times 0,5) + (1 \times 1,5) = 2,5$$

Com 2,5 fatores de crescimento e um total de 450 px precisando ser preenchido, cada fator de

crescimento vale 180 px:

$$450\text{px} / 2,5 = 180\text{px}$$

Novamente, o tamanho padrão do item flex era de 100 px, com a base flexível como padrão para auto, levando-nos a ter exatamente o mesmo layout do segundo exemplo. Isso demonstra como a distribuição de espaço extra é proporcional. Novamente vemos 2 itens flex com uma largura de 190 px e o último item flex com uma largura de 370 px:

$$100\text{px} + (0,5 \times 180\text{px}) = 190\text{px} \quad 100\text{px} + (1,5 \times 180\text{px}) = 370\text{px}$$

Se tivéssemos declarado fatores de crescimento de 0,1, 0,1 e 0,3, respectivamente, ou 25, 25, 75, o layout teria sido idêntico.

Fator de crescimento com larguras diferentes

Como mencionado, o espaço disponível é distribuído proporcionalmente entre os itens flexíveis em cada linha flexível com base nos fatores de crescimento flex, sem levar em conta as larguras originais subjacentes.

Na [Figura 3-5](#), no segundo exemplo, temos itens flexíveis com 100 px, 250 px e 100 px de largura, com fatores de crescimento de 1, 1 e 3, respectivamente, em um com 750 px de largura. Isso significa que temos 300 px de espaço extra para distribuir entre um total de 5 fatores de crescimento. Cada fator de crescimento é, portanto, 60 px, o que significa que o primeiro e o segundo itens flexíveis, com um fator de crescimento de 1, crescerão cada um em 60 px e o último crescerá 180 px à medida que o fator de crescimento for definido como 3.

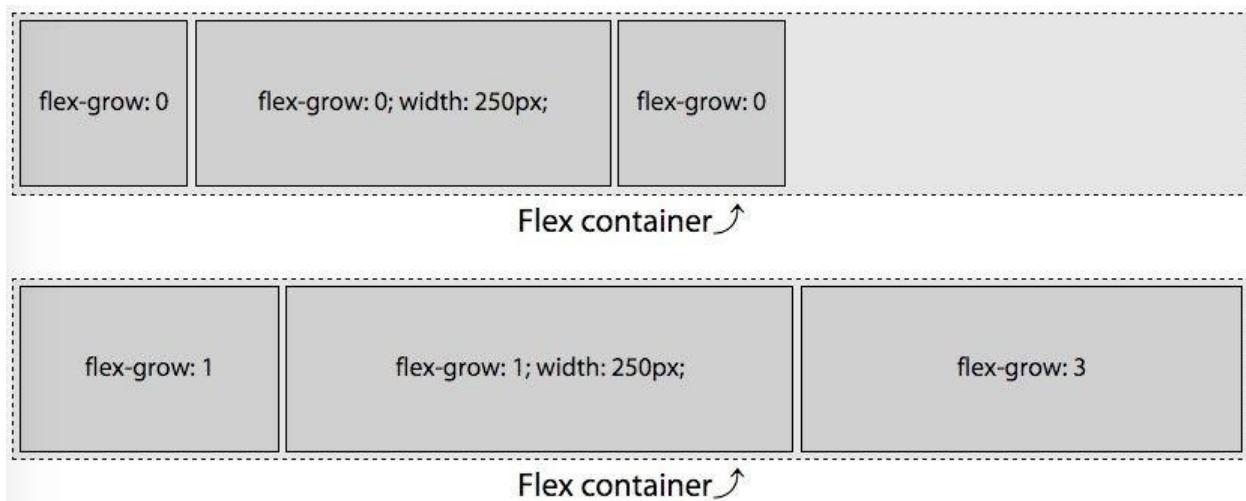


Figura 3-5. O espaço disponível é distribuído uniformemente para cada fator de crescimento; qualquer valor positivo permitirá que o item cresça proporcionalmente ao valor 

O espaço disponível, os fatores de crescimento e a largura de cada fator de crescimento são:

Espaço disponível: $750px - (100px + 250px + 100px) = 300px$ Fatores de crescimento: $1 + 1 + 3 = 5$
Largura de cada fator de crescimento: $300px / 5 = 60px$

Quando flexionada, a largura com base em sua largura original e fatores de crescimento tornam-se:

$item1 = 100px + (1 * 60px) = 160px$ $item2 = 250px + (1 * 60px) = 310px$

$item3 = 100px + (3 * 60px) = 280px$

$item1 + item2 + item3 = 160px + 310px + 280px = 750px$

Fatores de crescimento e a propriedade flex

A propriedade flex leva até três valores — o fator de crescimento, o fator de encolhimento e a base. O primeiro valor flutuante positivo não nulo, se houver, é o fator de crescimento. Quando o fator de crescimento é omitido na taquigrafia, o padrão é 1.

Caso contrário , se nem flex nem flexgrow estiverem incluídos, o padrão será 0.

No segundo exemplo da [Figura 3-4](#), porque declaramos um valor para flexgrow apenas, a base flexível foi ajustada para auto, como se tivéssemos declarado:

```
#example2 flexitem {  
  flex: 0 1 automático;  
}  
#example2 flexitem:last-child {  
  flex: 1 1 automático;  
}
```

Declarar flexgrow é fortemente desencorajado. Em vez disso, declare o fator de crescimento como parte da taquigrafia flex.

Tivemos que declarar flex: 0, flex: 0,5; , flexão: 1; , flex: 1,5; e flex: 3; nos exemplos da [Figura 3-4](#), em vez de declarar mal aconselhadamente os valores flexgrow, a base flex seria definida como 0%, e a distribuição de largura teria sido muito diferentes, como mostra a [Figura 3-6](#):

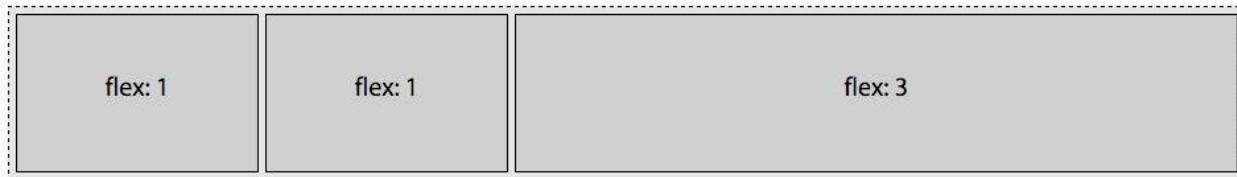
```
#example2 flexitem {  
  flex: 0 1 0%;  
}  
#example2 flexitem:last-child {  
  flex: 1 1 0%;  
}
```



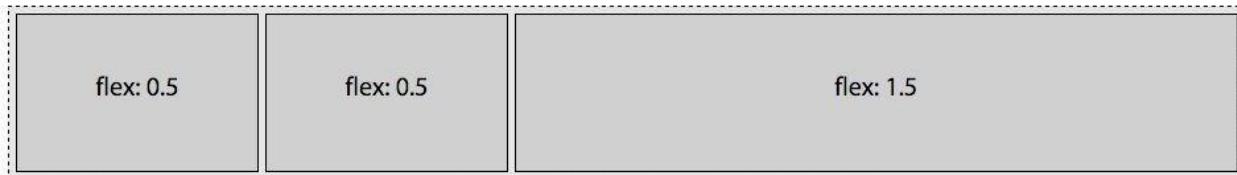
Flex container ↗



Flex container ↗



Flex container ↗



Flex container ↗

Figura 3-6. flexgrow parece diferente quando a base flex é 0, e alguns itens não podem crescer



Como o fator de redução padrão é 1 e a base padrão para 0%, o CSS a seguir é idêntico ao código anterior:

```
#example2 flexitem {
  flex: 0;
}
#example2 flexitem:last-child {
  flex: 1;
}
```

Você pode notar algo estranho: a base flex foi definida como zero, e apenas o último item flex tem um valor positivo para flex grow. A lógica parece que as larguras dos 3 itens flexíveis devem ser de 0, 0 e 750 px, respectivamente. Mas a lógica também ditaria que não faz sentido ter conteúdo transbordando sua flexibilidade.

se o contêiner flexível tiver espaço para todo o conteúdo, mesmo que a base esteja definida como zero.

Os autores da especificação pensaram nesse dilema. Quando a declaração de propriedade flex define ou padroniza explicitamente o flexbasis como 0 px e o fator de crescimento de um item flexível é 0, o comprimento do eixo principal dos itens flex não crescentes diminuirá para a menor largura o conteúdo permite, ou menor. Em nosso exemplo, a largura é a largura da palavra mais larga "flex:".

Contanto que um item flexível tenha um estouro visível e nenhuma largura mínima (ou altura mínima para eixos principais verticais) explicitamente definida, a largura mínima (ou altura mínima) será a menor largura (ou altura) que o item flexível precisa ser capaz de ajustar o conteúdo ou a largura (ou altura) declarada, o que for menor. No primeiro e no segundo examples da [Figura 3-6](#), mesmo que o fator de crescimento seja 0, os itens flexíveis não encolhem para 0. Em vez disso, eles encolhem para a largura da palavra não quebrável mais larga, já que a palavra "flex-" é mais estreita do que 100 px. Se a largura: 10px tivesse sido definida em vez de width: 100px, os itens flexíveis teriam se reduzido para o que fosse menor: a largura da palavra "flex-" ou 10 px.

Se todos os itens puderem crescer e a base flexível for de 0%, todo o espaço – todo o 750 px em vez de apenas o excesso de espaço – será distribuído proporcionalmente com base nos fatores de crescimento. Nos exemplos, o contêiner flexível tem 750 px de largura. Na [Figura 3-6](#), como a base flexível é zero, em vez de automática, e sem fatores de encolhimento positivos, os dois primeiros itens flexíveis são tão amplos quanto seu conteúdo mais amplo (a palavra flex:) e todo o espaço extra vai para o terceiro item flex com um fator de crescimento positivo. Isso fará mais sentido depois de ler a seção [flexshrink](#) (consulte "[The flexshrink Property](#)"). No terceiro exemplo, com dois itens flex com fatores de crescimento de um, e um item flex com um fator de crescimento de três, temos um total de cinco fatores de crescimento:

$$(2 \times 1) + (1 \times 3) = 5$$

Com 5 fatores de crescimento, e um total de 750 px, cada fator de crescimento vale 150 px:

$$750\text{px} / 5 = 150\text{px}.$$

Enquanto o tamanho padrão do item flex era de 100 px, a base flexível de 0 substitui isso, deixando-nos com 2 itens flex a 150 px cada e o último item flex com uma largura de 450 px:

$$\begin{aligned} 1 \times 150\text{px} &= 150\text{px} \\ 3 \times 150\text{px} &= 450\text{px} \\ 150\text{px} + 150\text{px} + 450\text{px} &= 750\text{px} \end{aligned}$$

Da mesma forma, no último exemplo da [Figura 3-6](#), com dois itens flex com fatores de crescimento de 0,5 e um item flex com fator de crescimento de 1,5, temos um total de 2,5 Factores de crescimento :

$$(2 \times 0,5) + (1 \times 1,5) = 2,5$$

Com 2,5 fatores de crescimento e um total de 750 px, cada fator de crescimento vale 300 px:

$$750\text{px} / 2,5 = 300\text{px}.$$

Enquanto o tamanho padrão do item flex era de 100 px, a base flexível de 0% substitui isso, deixando-nos com 2 itens flex a 150 px cada e o último item flex com um largura de 450 px:

$$0,5 \times 300\text{px} = 150\text{px} \quad 1,5 \times 300\text{px} = 450\text{px}$$
$$150\text{px} + 150\text{px} + 450\text{px} = 750\text{px}$$

Isso é diferente de declarar apenas flexgrow, como mostrado na [Figura 3-7](#). Apenas declarar flexgrow significa que o flexbasis padroniza o automático. Quando definido como automático, apenas o espaço extra, não todo o espaço, é distribuído proporcionalmente. Essa falta de simplicidade é o motivo pelo qual é altamente encorajado sempre usar a taquigrafia flex em vez de flexgrow, flexshrink e flexbasis separadamente ou não.

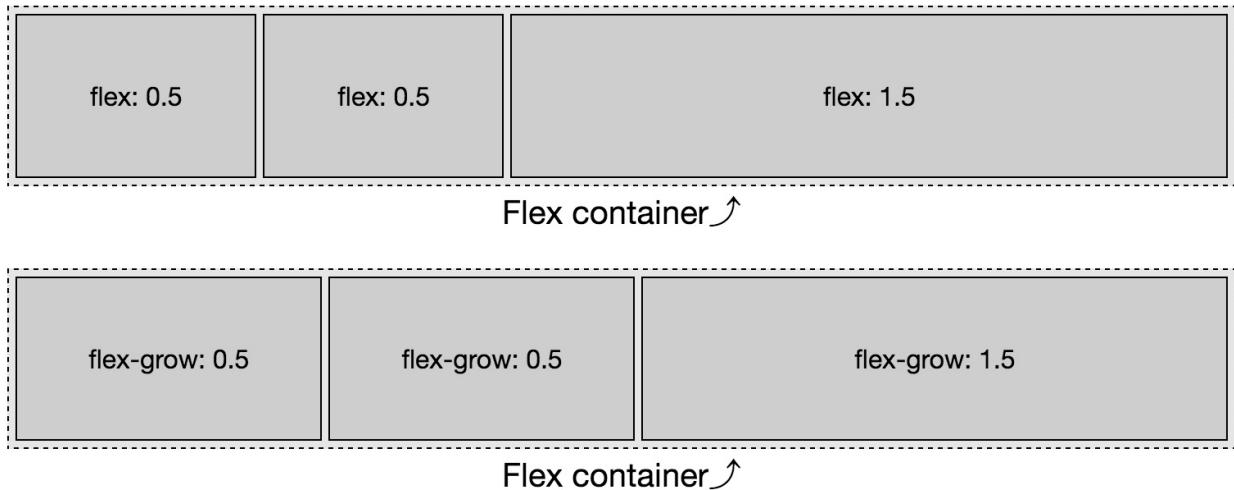


Figura 3-7. Ao usar `flexgrow` em vez da taquigrafia `flex`, a base `flex` será `auto` em vez de `0`.



Enquanto o espaço disponível é distribuído proporcionalmente com base no fator de crescimento dos itens flexíveis, a quantidade de espaço disponível é definida pela base flex, que é discutida em um seção seguinte. Vamos primeiro cobrir o fator de encolhimento.

A propriedade flexshrink

FLEXSHRINK	
Valores	<code><número></code>
Valor inicial:	1 por si só e como parte da taquigrafia flexionar
Aplica-se um:	Itens Flex (filhos de destinatários)
Herdados:	Não
Animável:	Sim

A parte flexshrink da propriedade de taquigrafia flex especifica o fator de encolhimento. O fator de encolhimento determina o quanto um item flexível encolherá em relação ao restante dos irmãos do item flexível quando não houver espaço suficiente para que todos eles caibam conforme definido por seu conteúdo, base e outras propriedades CSS. Basicamente, o fator de encolhimento define como o *espaço negativo* é distribuído, ou como os itens flexíveis devem se tornar mais estreitos ou mais curtos, quando o pai do contêiner flexível não tem permissão para crescer ou envolver de outra forma.

A [Figura 3-8](#) é semelhante à [Figura 3-4](#), com os itens flexíveis definidos como 300 px em vez de 100 px. Temos um contêiner flexível de 750 px de largura com três itens flexíveis de 300 px de largura. A largura total dos 3 itens é de 900 px, o que significa que o conteúdo é 150 px mais largo do que o contêiner flex pai. Se os itens não tiverem permissão para encolher ou envolver (consulte "[A propriedade flexwrap](#)"), eles explodirão do contêiner flex de tamanho fixo. Isso é demonstrado no primeiro exemplo da [Figura 3-8](#): esses itens não diminuirão, pois têm um fator de redução nulo. Em vez disso, os itens flex estouraram o contêiner flex ao longo do lado principal, pois o padrão `justify-content` (descrito em "[The justify-content Property](#)") é `flex-start`.

No segundo exemplo da [Figura 3-8](#), somente o último item flexível é definido para poder encolher. Forçamos o único elemento com um fator de encolhimento positivo a fazer todo o encolhimento necessário para permitir que todos os itens flexíveis se encaixassem dentro do recipiente. Com

900 px de conteúdo que precisa caber em nosso recipiente de 750 px, temos 150 px de espaço negativo. Os 2 itens flexíveis sem fator de encolhimento permanecem em 300 px de largura.

O terceiro item flexível, com o valor positivo para o fator de encolhimento, encolhe em 150 pixels, para ter 150 px de largura, permitindo que os 3 itens caibam dentro do recipiente. Neste exemplo, o fator de encolhimento foi 1, mas se fosse 0,001 ou 100, o layout seria o mesmo.

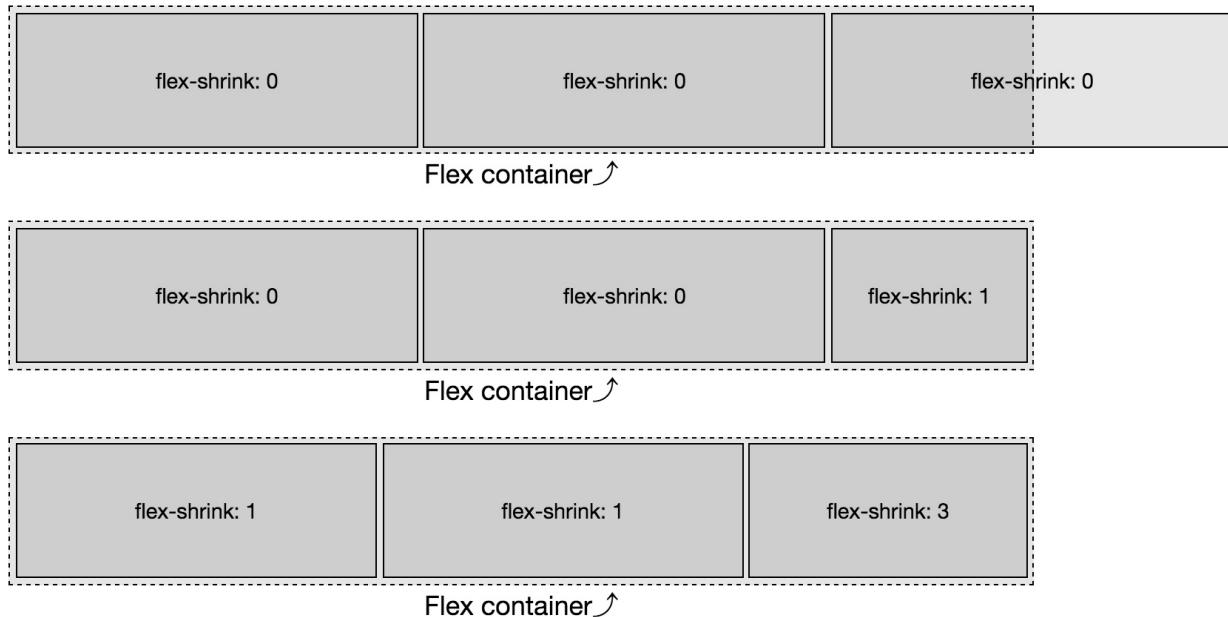


Figura 3-8. Um fator de encolhimento flexível de 0 não permitirá que os itens flexíveis diminuam; qualquer valor positivo permitirá que o item diminua proporcionalmente em relação aos itens flex irmãos que podem encolher na mesma linha flex



Quando omitido no valor da propriedade flex abreviada ou quando flex e flexshrink são omitidos, o fator de encolhimento padrão é 1. Como o fator de crescimento, o valor é sempre um número. Números negativos não são válidos. Os flutuadores, desde que sejam maiores que 0, são válidos.

No terceiro exemplo, fornecemos fatores de encolhimento positivos para todos os três itens flexíveis:

```
#example3 flexitem { flex-shrink: 1;  
}  
#example3 flexitem:last-child { flex-shrink: 3;  
}
```

NOTA

Embora Temos incluído | o código anterior, nossos itens flexíveis | se comportarão | como se Nós declarado o seguinte:

```
#example3 flexitem {  
  flexionar: 0 1 automático;  
}  
f#example3 flexitem :última criança {  
  flexionar: 0 3 Automático;  
}
```

Este provavelmente não é o layout que você estará desenvolvendo para a produção. Então, usar um taquigrafia flexionar.

Se todos os itens tiverem permissão para encolher, como é o caso aqui, o encolhimento será distribuído proporcionalmente com base no fator de encolhimento dos itens individuais que têm um valor positivo definido para essa propriedade.

Quando todos os itens têm a mesma base, é fácil descobrir como eles vão encolher com base nos valores de seus fatores de encolhimento. Com um pai de 750 px de largura e 3 itens flexíveis com uma largura de 300 px, há 150 pixels extras que precisam ser raspados dos itens flexíveis que podem encolher. Com dois itens flex com um fator de encolhimento de 1 e um item flexível com um fator de encolhimento de 3, temos um total de cinco fatores de encolhimento:

$$(2 \times 1) + (1 \times 3) = 5$$

Com 5 fatores de encolhimento e um total de 150 px precisando ser raspado de todos os itens flexíveis, cada fator de encolhimento vale 30 px:

$$150\text{px} / 5 = 30\text{px}.$$

O tamanho padrão do item flex foi de 300 px, levando-nos a ter 2 itens flex com uma largura de 270 px cada e o último item flex com uma largura de 210 px, que totalizam 750 px. Ao encolher proporcionalmente com base em seu fator de encolhimento, ditamos como eles encolhem para caber no espaço alocado:

$$300\text{px} - (1 \times 30\text{px}) = 270\text{px} \quad 300\text{px} - (3 \times 30\text{px}) = 210\text{px}$$

$$270\text{px} + 270\text{px} + 210\text{px} = 750\text{px}$$

Os itens flexíveis diminuirão para 270 px, 270 px e 210 px, respectivamente, desde que o conteúdo (como objetos de mídia ou texto não moldável) dentro de cada item flexível não seja maior que 270 px, 270 px ou 210 px, respectivamente:

```
flexitem {  
  flex: 1 0.25 automáticos;  
}  
flexitem:last-child {  
  flex: 1 0.75 automáticos;  
}
```

O código anterior produz o mesmo resultado: enquanto a representação numérica dos fatores de encolhimento é diferente, eles são proporcionalmente os mesmos.

Um valor de encolhimento positivo não significa que um item flexível necessariamente encolherá mesmo que não haja espaço disponível no contêiner flexível pai para ele e seus irmãos. Se o item flex contiver conteúdo que não pode ser encapsulado ou encolhido na dimensão principal, o item flex não diminuirá.

Por exemplo, se os primeiros itens flexíveis contiverem uma imagem de 300 px de largura ou uma URL de caracteres com mais de 300 px, os 150 px de espaço negativo serão distribuídos entre os quatro fatores de encolhimento disponíveis:

item1 = 300px - (0 x 37,5px) = 300,0px item2 = 300px - (1 x 37,5px) = 262,5px
item3 = 300px - (3 x 37,5px) = 187,5px

Nesse caso, o primeiro item seria de 300 px, com os 150 px de espaço negativo distribuídos proporcionalmente com base nos fatores de encolhimento do segundo e terceiro itens flex. Esse primeiro item flex não pode encolher, e outros itens flex podem encolher, pois ele não vai encolher, como se tivesse um fator de encolhimento nulo. Temos 4 fatores de encolhimento desimpedidos para 150 px de espaço negativo, com cada fator de encolhimento valendo a pena

37,5 px. Os itens flexíveis são 300 px, 262,5 px e 187,5 px, respectivamente, para um total de 750 px.

Se a imagem ou URL no primeiro item flex tivesse 296 px de largura, esse primeiro item flex teria sido capaz de encolher em 4 px. Teríamos então distribuído os 146 de espaço negativo entre os 4 fatores de encolhimento restantes para itens flexíveis que eram 296 px, 263,5 px e 190,5 px de largura, respectivamente.

Se todos os 3 itens flexíveis contiverem URLs não empacotáveis ou mídia de 300 px ou mais, os 3 itens flexíveis não encolheriam, aparecendo semelhante ao primeiro exemplo na Figura 3-8.

Todos os itens tinham a mesma largura ou base flexível, portanto, isso é fácil de calcular. Se eles não tivessem as mesmas larguras, essa equação seria mais complexa. Isso só funciona

assim porque todos os itens flex tinham as mesmas bases flex.

Proporcional com base na largura e no fator de encolhimento

O exemplo de código anterior foi bastante simples porque todos os itens flexíveis começaram com a mesma largura. Mas e se as larguras fossem diferentes? E se o primeiro e o último item flex tivessem uma largura de 250 px e o item flex médio tivesse uma largura de 500 px?

Os itens flexíveis encolhem proporcionalmente em relação aos fatores de encolhimento e à largura do item flexível, com a largura geralmente sendo conteúdo, a largura do conteúdo do item flexível sem encapsulamento. Na **Figura 3-9**, estamos tentando encaixar 1.000 pixels em um contêiner flexível de 750 px de largura. Temos um excesso de 250 px a ser removido de cinco fatores de encolhimento. Se esta fosse uma preocupação flexgrow, simplesmente dividiríamos 250 px por 5, alocando 50 px por fator de crescimento. Se encolhessemos dessa forma, veríamos itens flexíveis de 200 px, 550 px, umd 100 px de largura, respectivamente. Mas não é isso que vemos.

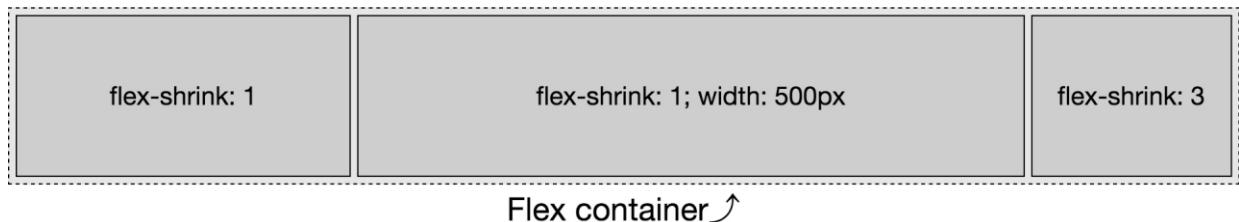


Figura 3-9. Os itens flexíveis encolhem proporcionalmente em relação ao seu fator de encolhimento

Em vez disso, temos 250 px de espaço negativo para distribuir proporcionalmente. Para obter as proporções do fator de encolhimento, dividimos o espaço negativo pelo espaço real vezes seus fatores de encolhimento:

$$ShrinkPercent = \frac{NegativeSpace}{((Width1 * ShrF1) + \dots + (WidthN * ShrFN))}$$

Usando esta equação, aprendemos a porcentagem do fator de encolhimento:

$$\begin{aligned} &= 250px / ((250px * 1) + (500px * 1) + (250px * 3)) \\ &= 250px / 1500px \\ &= 0,166666667 \end{aligned}$$

Quando reduzimos cada item flex em 16,67% vezes o fator de encolhimento, acabamos com itens flexíveis que são reduzidos por:

$$\begin{aligned} item1 &= 250px * (1 - 16,67\%) = 41,67px \\ item2 &= 500px * (1 - 16,67\%) = 83,33px \\ item3 &= 250px * (3 * 16,67\%) = 125px \end{aligned}$$

Obtemos itens flexíveis com 208,33 px, 416,67 px e 125 px de largura, respectivamente.

No mundo real

Permitir que itens flexíveis diminuam proporcionalmente dessa forma permite objetos e layouts responsivos que podem encolher proporcionalmente sem quebrar.

Por exemplo, você pode criar um layout de três colunas que aumente e diminua de forma inteligente sem consultas de mídia, como mostrado em uma tela larga na Figura 3-10 e na tela mais estreita na [Figura 3-11](#):

```
nav {  
  flex: 0 1 200px;  
  largura mínima: 150px;  
}  
artigo {  
  flex: 1 2 600px;  
}  
à parte {  
  flex: 0 1 200px;  
  largura mínima: 150px;  
}
```

The screenshot shows a website layout with a dark header bar containing the text "Site Wide Header". Below the header is a sidebar on the left with a dark background and white text, listing "Home", "About", "Blog", "Careers", and "Contact Us". The main content area in the center has a white background and contains the heading "This is the heading of the article" and a paragraph of placeholder text. To the right of the main content is a sidebar with the heading "Aside Heading" and some placeholder text. At the bottom of the page is a footer bar with the text "Copyright © 2016 [My Site](#)".

Figura 3-10. Ao definir valores diferentes para crescimento, redução, base e largura mínima, você pode criar layouts responsivos, com ou sem consultas de mídia

Site Wide Header

Home

About

Blog

Careers

Contact Us

This is the heading of the article

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Aside Heading

Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Copyright © 2016 [My Site](#)

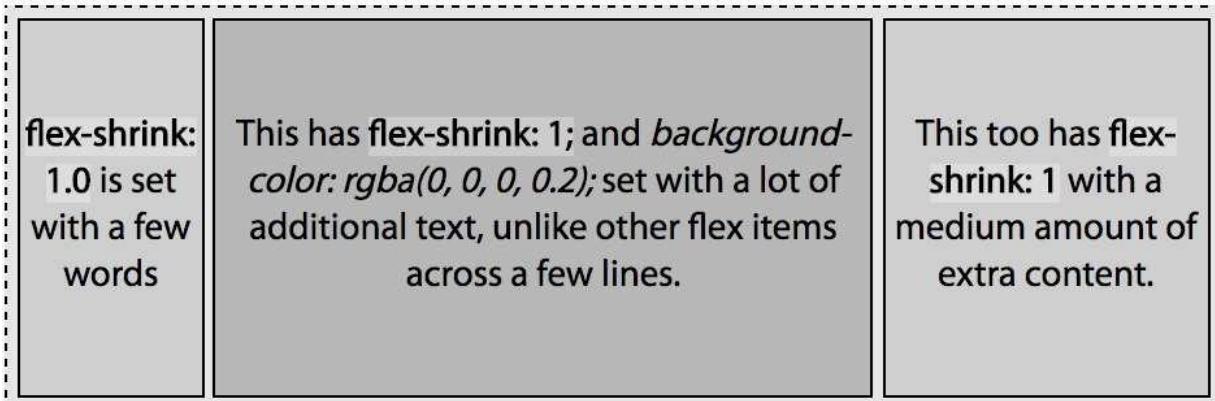
Figura 3-11. Com o crescimento, a redução, a base e a largura mínima definidos, você pode criar layouts responsivos que ficam ótimos em telas mais estreitas, sem precisar de uma infinidade de consultas de mídia

Neste exemplo, se o visor for maior que 1.000 px, somente a coluna do meio crescerá porque somente a coluna do meio recebeu um fator de crescimento positivo . Também ditamos que, abaixo da marca de 1.000 px de largura, a coluna da direita encolhe duas vezes mais rápido que as 2 colunas esquerdas, usando a largura mínima para garantir que as colunas nunca diminuam abaixo de sua palavra mais estreita ou dessa largura mínima, o que for maior.

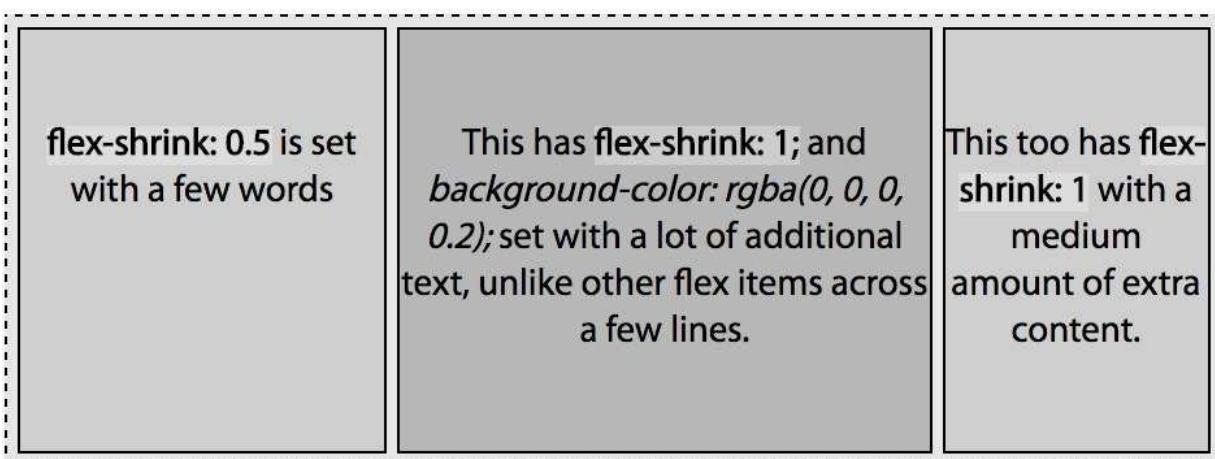
Bases diferentes

Com um fator de encolhimento nulo, se nenhuma largura ou base for definida em um item flexível, seu conteúdo não será encapsulado. Quando temos mais conteúdo do que poderia caber perfeitamente em uma linha, um valor positive shrink permite que o conteúdo seja empacotado. Os fatores de encolhimento permitem o empacotamento proporcional. Como o encolhimento é proporcional com base no fator de encolhimento, se todos os itens flexíveis tiverem fatores de encolhimento semelhantes, o conteúdo deverá ser encapsulado em um número semelhante de linhas.

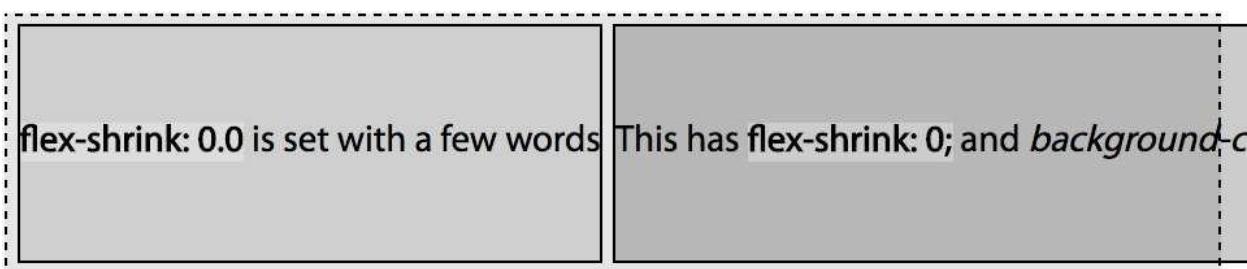
Ao contrário dos exemplos anteriores deste capítulo, nos dois exemplos da Figura 3-12, os itens flexíveis não têm uma largura declarada. Em vez disso, a largura é baseada no conteúdo — a largura tem como padrão automático, assim como a base flex, como se o conteúdo `flexbasis`: fosse definido (consulte "[A propriedade `flexbasis`](#)").



same shrink factors ↑



different shrink factors ↑



null shrink factors ↑

Figura 3-12. Os itens flexíveis encolhem proporcionalmente em relação ao seu fator de encolhimento e conteúdo

Você observará no primeiro exemplo, todo o conteúdo é quebrado em quatro linhas. No segundo exemplo, o primeiro item flex, com um fator de encolhimento metade do valor dos outros itens flex, envolve mais da metade do número de linhas. Este é o poder do fator de encolhimento.

No terceiro exemplo, com um fator de redução nulo, o texto não é quebrado e os itens flexíveis estouram o contêiner.

Como o fator de redução da propriedade flex reduz a largura dos itens flex proporcionalmente, o número de linhas de texto nos itens flex aumentará ou diminuirá à medida que a largura diminuir. ou cresce, levando a um teor de altura semelhante dentro de itens flexíveis irmãos quando os fatores de encolhimento são semelhantes.

Nos exemplos, o conteúdo dos itens flex no meu dispositivo é de 280 px, 995 px e 480 px, respectivamente — que é a largura dos itens flexíveis não encapsulados no terceiro exemplo (conforme medido pelas ferramentas de desenvolvedor, depois arredondado para tornar este exemplo um pouco mais simples). Isso significa que temos que encaixar 1.755 px de conteúdo em um contêiner flexível de 520 px de largura, encolhendo os itens flexíveis proporcionalmente com base em seu fator de encolhimento. Isso significa que temos 1.235 px de espaço negativo para distribuir proporcionalmente.

Obviamente, você não pode confiar em ferramentas de inspetor da Web para descobrir fatores de encolhimento para a produção. Estamos passando por este exercício para entender como os fatores de encolhimento funcionam. Se minúcia não é a sua praia, sinta-se à vontade para pular para "["The flexbasis Property"](#)".

Como os itens flexíveis diminuem proporcionalmente, com base na largura de seu conteúdo, em nosso exemplo, a única linha de itens flexíveis de texto terminará com o mesmo número ou aproximadamente o mesmo número de linhas.

Não declaramos uma largura, portanto, não podemos simplesmente usar 300 px como base, como fizemos nos exemplos anteriores. Em vez disso, distribuímos os 1.235 px de espaço negativo proporcionalmente com base nas larguras do conteúdo - 280 px, 995 px e 480 px, respectivamente. Determinamos que 520 é 29,63% de 1.755. Para determinar a largura de cada item flex com um fator de encolhimento de 1, multiplicamos a largura de conteúdo de cada item flex por 29,63%:

$$\begin{aligned} \text{item1} &= 280\text{px} * 29,63\% = 83\text{px} & \text{item2} &= 995\text{px} 29,63\% = \\ & 295\text{px} & \text{item3} &= 480\text{px} 29,63\% = 142\text{px} \end{aligned}$$

$$\text{item1} + \text{item2} + \text{item3} = 83\text{px} + 295\text{px} + 142\text{px} = 520\text{px}$$

Com o padrão de itens de alinhamento: esticar; (consulte "[A propriedade align-items](#)"), seu layout de três colunas teria, por padrão, criado três colunas de igual altura. Usando um fator de encolhimento uniforme, você pode ditar que o conteúdo real desses três itens flexíveis seja de altura aproximadamente igual: embora, ao fazer isso, a largura dessas colunas não será uniforme. A largura dos itens flex é da alçada do flexbasis.

Em nosso segundo exemplo na [Figura 3-12](#), os itens flexíveis não têm todos o mesmo fator

de encolhimento. O primeiro item flexível irá, proporcionalmente, encolher metade do que os outros. Começamos com as mesmas larguras: 280 px, 995 px e 480 px, respectivamente, mas os fatores de encolhimento são 0,5, 1,0 e 1,0, respectivamente. Como conhecemos as larguras do conteúdo, o fator de encolhimento (X) pode ser encontrado matematicamente:

$$(0,5X \cdot 280px) + (1X \cdot 995px) + (1X \cdot 480px) = 1235px \quad 1615X = 1235px$$
$$X = 1235px / 1615 \quad X = 0,7647$$

Podemos encontrar as larguras finais agora que conhecemos o fator de encolhimento. Se o fator de encolhimento for de 76,47%, isso significa que o item 2 e o item 3 serão 23,53% de suas larguras originais, e o item 1, por ter um fator de encolhimento de 0,5, será 61,76% de sua largura original:

$$\text{item1} = 280px \cdot 0,6176 = 173px \quad \text{item2} = 995px \cdot 0,2354 = 234px \quad \text{item3} = 480px \cdot 0,2354 = 113px$$

$$\text{item1} + \text{item2} + \text{item3} = 173px + 234px + 113px = 520px$$

A largura total combinada desses 3 itens flexíveis é de 520 px.

Adicionar fatores variáveis de encolhimento e crescimento torna tudo um pouco menos intuitivo. É por isso que você provavelmente deseja sempre declarar a taquigrafia flex, de preferência com uma largura ou base definida para cada item flex.

Se isso ainda não faz sentido, não se preocupe, abordaremos mais alguns exemplos de encolhimento enquanto discutimos o flexbasis.

A propriedade flexbasis

Novamente, enquanto estamos cobrindo a propriedade flexbasis individualmente aqui para que você a entenda completamente, o grupo de trabalho CSS incentiva a declaração da base como parte da propriedade de taquigrafia flex em vez de por conta própria. flex redefine para uso comum (em vez de padrão) os valores de crescimento, redução e base se qualquer um deles não for declarado dentro da taquigrafia. Quando o flexbasis é ajustado, em vez do flex, os itens flex podem encolher, mas não crescerão, como se o flex: 0 1 <flexbasis> fosse definido.

Sério, use flex em vez de declarar os três valores longhand separadamente. Estamos cobrindo apenas a taquigrafia aqui porque 1) este é o guia definitivo, então temos que fazê-lo, e 2) para que você entenda completamente qual a base flexível do flex propriedade taquigráfica faz. A esta altura, espero que você tenha sido convencido.

FLEXBASIS	
Valores	de conteúdo
Valor inicial :	Automático
Aplica-se um:	Itens Flex (filhos de destinatários)
Herdados:	Não
Porcentagens:	Em relação ao tamanho principal interno

Como já vimos, o tamanho de um item flexível é afetado por seu conteúdo e propriedades de modelo de caixa e pode ser redefinido por meio dos três componentes da propriedade flex. O componente flexbasis da propriedade flex define o tamanho inicial ou padrão dos itens flex, antes que o espaço extra ou negativo seja distribuído — antes que os itens flex possam crescer ou encolher de acordo com seus fatores de crescimento e encolhimento, que acabamos de descrever.

A base determina o tamanho da caixa de conteúdo, afetada pelo dimensionamento da caixa. Por padrão, quando um elemento não é um item flexível, o tamanho é determinado pelo tamanho de suas propriedades pai, conteúdo e modelo de caixa. Quando nenhuma propriedade de tamanho é explicitamente declarado ou herdado, o tamanho padrão é seu conteúdo, borda e preenchimento individuais, que é 100% da largura de seu pai para elementos de nível de bloco.

A propriedade `flexbasis` aceita os mesmos tipos de valor de comprimento que as propriedades `width` e `height` — como `5vw`, `12%` e `300px` — ou os termos-chave `auto`, `initial`, `inherit` e `content`.

Nenhuma das propriedades `flex` é herdada por padrão, mas você pode dizer a um item `flex` para herdar o `flexbasis` de seu pai com o valor `inherit`.

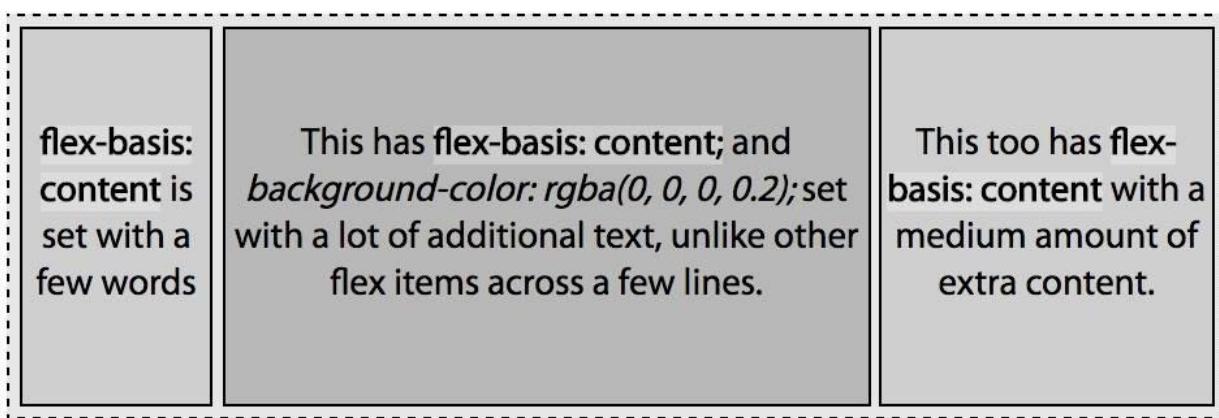
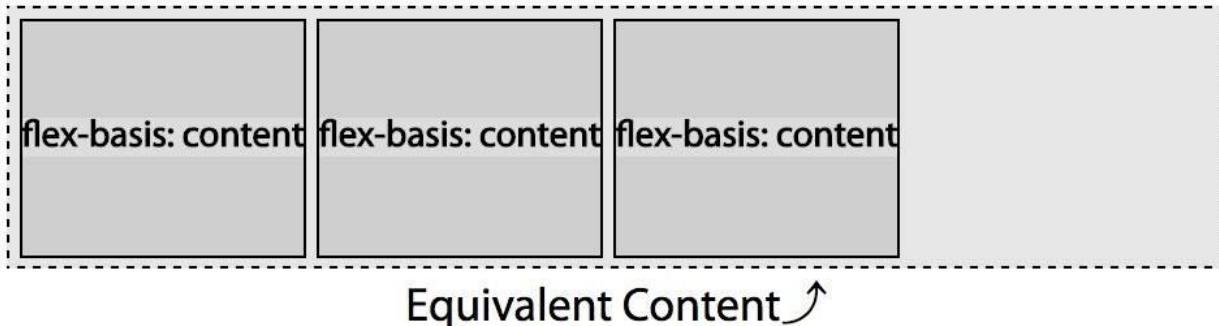
Os valores globais de `initial` redefinem a base flexível para o valor inicial de `auto`, para que você também possa declarar `auto`. Por sua vez, o `auto` avalia para a largura (ou altura) se declarado. A palavra-chave de tamanho principal usada para fazer isso: fazia parte de uma especificação mais antiga, mas foi preterida. Se o valor de largura (ou altura) for definido como automático, o valor será avaliado como conteúdo.

conteúdo

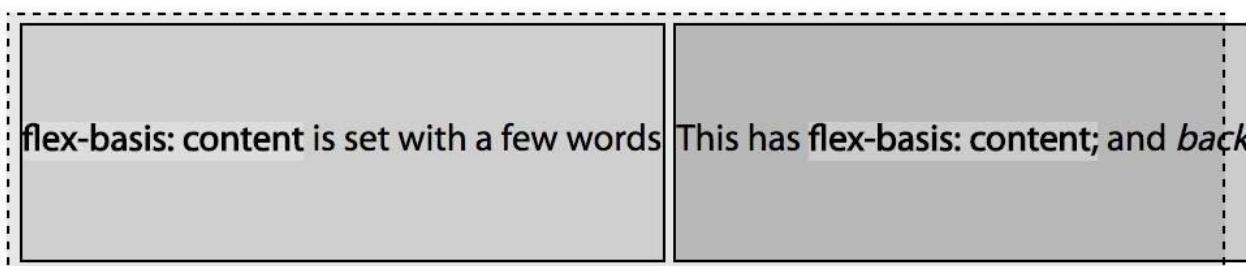
O valor da palavra-chave de conteúdo **não** é suportado **na maioria dos navegadores** no momento da redação deste artigo, com exceção do Microsoft Edge 12+, mas é igual à largura ou altura do conteúdo. Quando o conteúdo é usado e suportado, a base é o tamanho do conteúdo do item flexível — o valor é o tamanho principal da linha mais longa de conteúdo ou do objeto de mídia mais largo (ou mais alto).

Até que o suporte esteja completo, `flexbasis: conteúdo;` pode ser facilmente preenchido com `polo`, pois é o equivalente a declarar `flexbasis: auto; largura: auto;` nesse item `flex`, ou `flexbasis: auto; altura: auto;` se a dimensão principal for `vertical`. Infelizmente, o uso de conteúdo na abreviação em navegadores sem suporte invalida toda a declaração `flex`. A declaração inteira é invalidada quando qualquer valor não é entendido **de acordo com a especificação**.

O valor do conteúdo é basicamente o que vimos no terceiro exemplo na Figura 3-12, e é mostrado em **Figure 3-13**.



Variable amounts of content ↗



variable contents with no shrink ↗

Figura 3-13. Quando definido como conteúdo, a base é a largura (ou altura) do conteúdo



No primeiro e terceiro exemplos da [Figura 3-13](#), a largura do item flex é o tamanho do conteúdo; e a base também é essa largura. No primeiro exemplo, a largura e a base dos itens flexíveis são de aproximadamente 132 px. A largura total dos 3 itens flexíveis lado a lado é de 396 px, encaixando-se perfeitamente no contêiner pai.

No terceiro exemplo, definimos um fator de encolhimento nulo: isso significa que os itens flexíveis não podem encolher, então eles não encolherão ou envolverão para caber no flex de largura fixa

pai do contêiner. Em vez disso, eles são a largura de seu texto não quebrado. Essa largura é o valor da base flexível. A largura e a base dos itens flexíveis são de aproximadamente 309 px, 1.037 px e 523 px, respectivamente. Você não pode ver a largura total do segundo item flexível ou do terceiro item flexível, mas eles estão nos arquivos [de capítulo](#).

O segundo exemplo contém o mesmo conteúdo que o terceiro exemplo, mas os itens flex têm como padrão flexshrink: 1: o texto neste exemplo é quebrado automaticamente porque os itens flex pode encolher. Assim, enquanto a largura do item flex não é a largura do conteúdo, a base flex - a base pela qual ele encolherá proporcionalmente - é a largura do conteúdo: 309 px, 1.037 px e 523 px, respectivamente, conforme medido com ferramentas de desenvolvedor.

Quando os fatores de encolhimento são os mesmos, como o encolhimento dos itens flexíveis é proporcional com base na base flexível da largura do conteúdo , eles acabam com o mesmo número ou aproximadamente o mesmo número de linhas.

NOTA

Até que o valor do conteúdo seja suportado | em todos os lugares, você pode replicá-lo definindo (ou padrão) um largura/altura e um base como automáticas.

É o mesmo que configurar flexbasis: auto; largura: auto; ou flexbasis: auto; altura: auto; .

Automático

Quando definido como automático ou omitido, o flexbasis é o tamanho principal do nó se o elemento não tivesse sido transformado em um item flexível. Para valores de comprimento, o flexbasis resolve para o valor de largura ou altura, com a exceção de que, quando o valor da largura ou altura é automático, o valor é resolvido ao conteúdo.

Se a base for explicitamente definida como auto ou omitida e, portanto, o padrão for automático, e todos os itens flexíveis puderem caber dentro do contêiner flex pai, os itens flex terão seu tamanho pré-flexionado, como visto na [Figura 3-14](#). Se os itens flex não se encaixarem em seu contêiner flex pai, os itens flex dentro desse contêiner diminuirão proporcionalmente com base em seus tamanhos principais não flexionados, a menos que o fator shrink seja nulo.

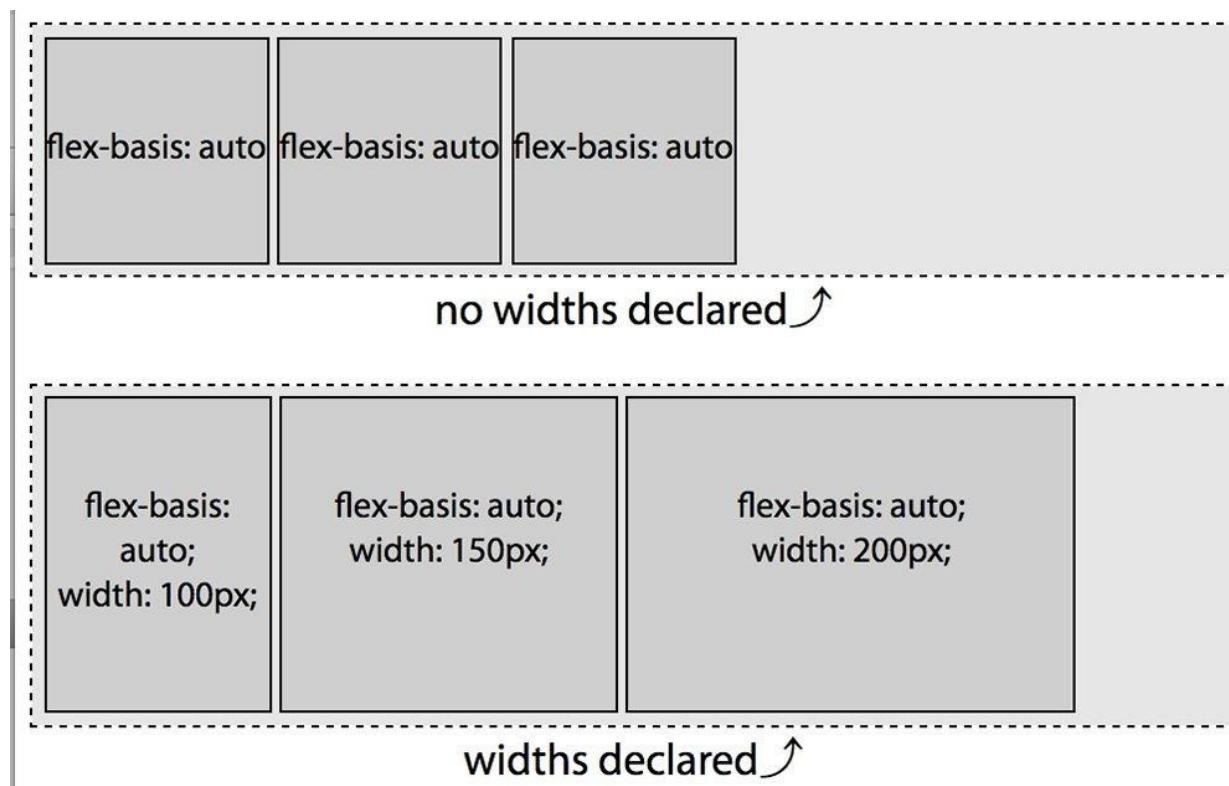


Figura 3-14. Quando uma base flexível é definida, por padrão, o tamanho principal do item será o valor declarado

Quando não houver outras propriedades que definam o tamanho principal dos itens flexíveis (não há largura ou mesmo largura mínima definida nesses itens flexíveis) e flexbasis:

Automático; ou flex: 0 1 auto; é definido, os itens flexíveis serão apenas tão largos quanto eles precisam ser para que o conteúdo se encaixe, como visto no primeiro exemplo na [Figura 3-14](#). Nesse caso, eles são a largura do texto "flexbasis: auto", que neste caso, com essa fonte, é de aproximadamente 110 px. Os itens flexíveis são de seu tamanho pré-flexionado, como se estivessem configurados para exibir: bloco em linha; . Neste exemplo, eles são agrupados no início principal porque o conteúdo **de justificação do contêiner flexível tem como padrão o flex-start**.

No segundo exemplo da [Figura 3-14](#), cada um dos itens flex tem base flexível de auto e uma largura declarada. O tamanho principal dos nós se os elementos não tivessem sido transformados em itens flexíveis seria de 100 px, 150 px e 200 px. Ao usar o flexbasis: auto, estamos dizendo a ele para usar essas propriedades subjacentes do modelo de caixa para determinar a base flex.

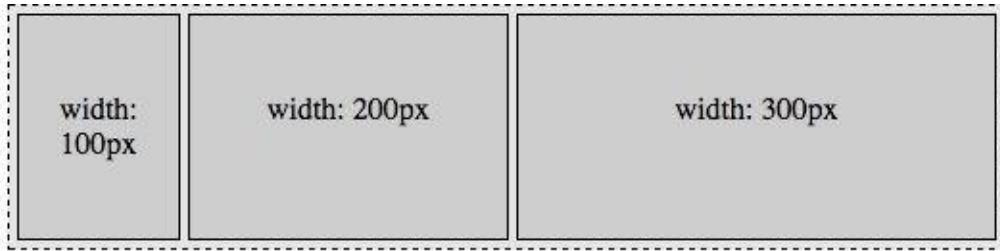
Há um pouco mais para entender como o auto funciona com a largura subjacente. Enquanto os exemplos nesta seção usavam porcentagens e auto, quando discutimos os fatores de crescimento e encolhimento anteriormente, os itens flexíveis tinham larguras subjacentes de 100 px e 300 px, respectivamente. Como a base não foi explicitamente definida como um comprimento, o valor de largura foi a base nesses cenários.

Valores padrão

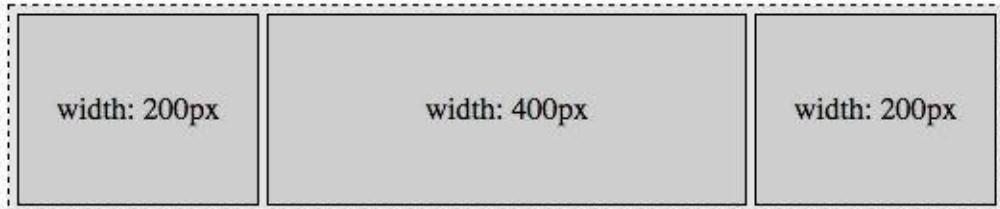
Quando nem flexbasis nem flex são definidos, o tamanho principal do item flex é o pré-flex size do item, pois seu valor padrão é auto.

Na [Figura 3-15](#), duas coisas estão acontecendo: as bases flexíveis estão padronizadas como auto e o fator de encolhimento de cada item está padronizado para 1. Isso significa que as bases estão sendo definidas para os valores das propriedades de largura: 100 px, 200 px e 300 px no primeiro exemplo e 200 px, 400 px e 200 px no segundo exemplo, e todos eles são capazes de encolher. Para cada um, a base flexível é o seu valor de largura individual. Como as larguras combinadas são de 600 px e 800 px, que são ambas maiores do que o tamanho principal dos recipientes de 540 px de largura, todas elas estão encolhendo proporcionalmente para se ajustar.

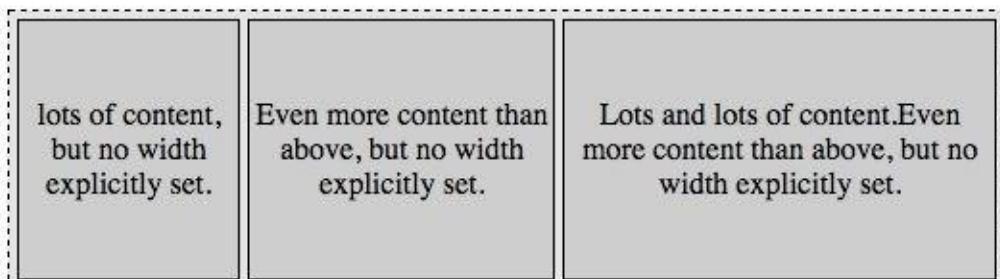
No primeiro exemplo, estamos tentando encaixar 600 px em 540 px, de modo que cada item flexível encolherá 10%, resultando em itens flexíveis de 90 px, 180 px e 270 px. Em nossos segundos exemplos, estamos tentando encaixar 800 px em 540 px, tornando os itens flexíveis 135 px, 270 px e 135 px.



Flex container ↗



Flex container ↗



Flex container ↗

Figura 3-15. Quando nenhuma propriedade flex é definida, o tamanho principal do item flex será o tamanho pré-flex do item



Unidades de Comprimento

Nos exemplos anteriores, a base era padrão para as larguras declaradas dos vários itens flexíveis. Podemos usar as mesmas unidades de comprimento para o nosso valor flexbasis como fazemos para largura e altura.

Quando há valores de flexbasis e largura, a base supera a largura. Vamos adicionar valores de base ao primeiro exemplo da [Figura 3-15](#). Os itens flexíveis incluem o seguinte CSS:

```
flex-container {  
    largura: 540px;  
}  
item1 {  
    largura: 100px;  
    flex-base: 300px; /*flex: 0 1 300px; */  
}  
item2 {  
    largura: 200px;  
    flex-base: 200px; /*flex: 0 1 200px; */  
}  
item3 {  
    largura: 300px;  
    flex-base: 100px; /*flex: 0 1 100px; */  
}
```

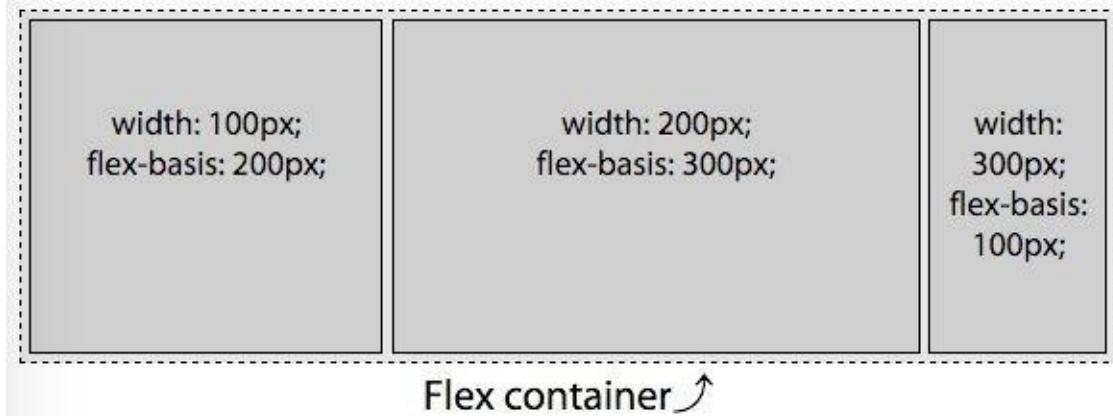


Figura 3-16. Observe que o tamanho principal do item flex é proporcional à propriedade flexbasis, não à largura propriedade

As larguras são substituídas pelas bases. Os itens flexíveis encolheram para 270 px, 180 px e 90 px, respectivamente.

Embora a base declarada possa substituir o tamanho principal dos itens flexíveis, o tamanho pode ser

impactado por outras propriedades, como largura mínima, altura mínima, largura máxima e altura máxima. Estes não são ignorados.

Unidades de comprimento: porcentagens

Os valores percentuais para flexbasis são relativos ao tamanho da dimensão principal do contêiner flex.

Já vimos o primeiro exemplo na [Figura 3-17](#). Estou incluindo-o aqui para lembrar que a largura do texto "flexbasis: auto" neste caso, com meu sistema operacional, navegador e fontes instaladas, é de aproximadamente 110 px de largura. Somente neste caso, declarar flexbasis: auto parece o mesmo que escrever flexbasis: 110px:

```
flex-container {  
  largura: 540px;  
}  
flexitem:last-child {  
  flex: 0 1 100%;  
}
```

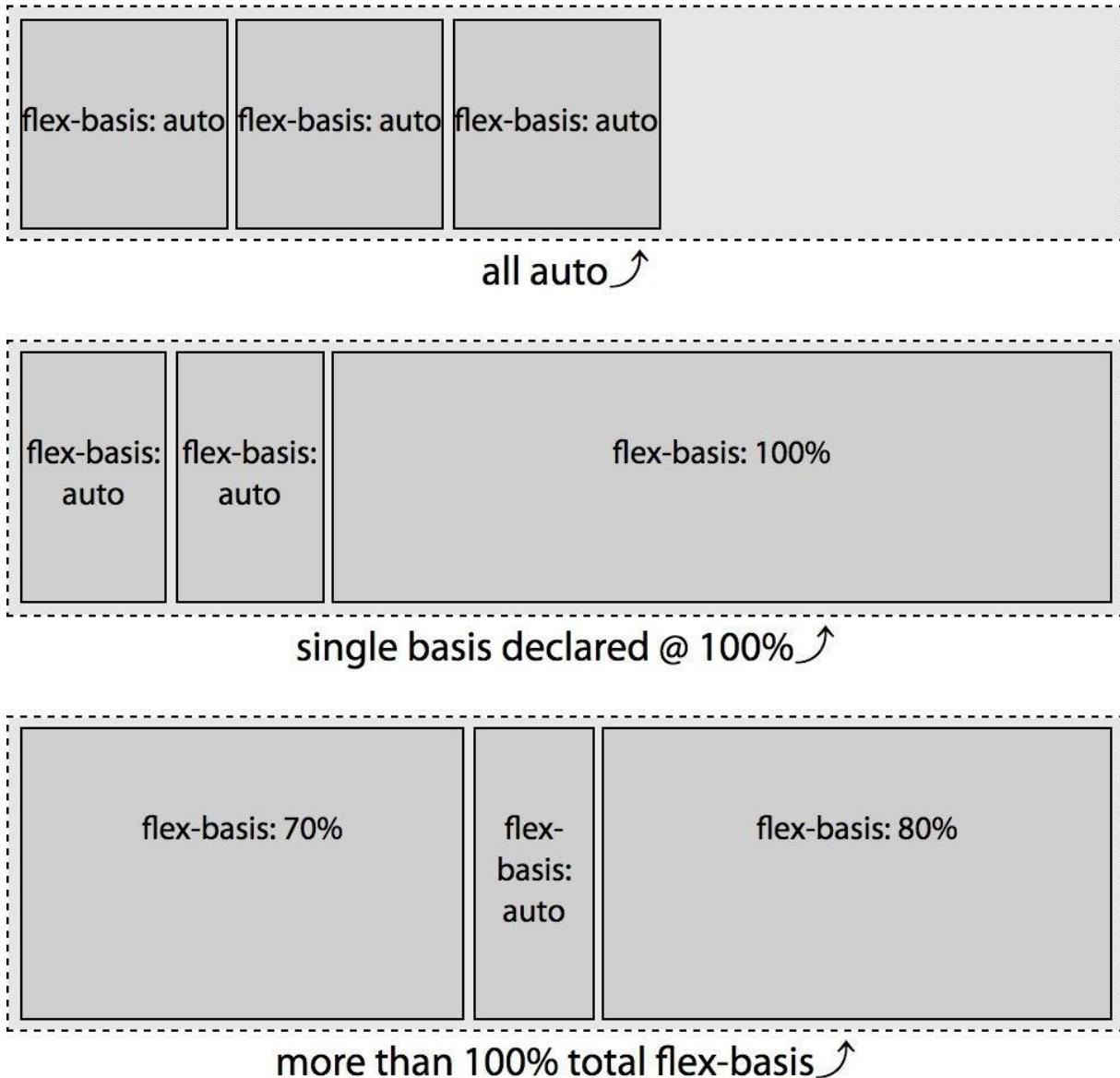


Figura 3-17. O valor percentual para flexbasis é relativo à largura do contêiner flex



No segundo exemplo da [Figura 3-17](#), os dois primeiros têm uma base flexível de auto com uma largura padrão de auto, que é como se sua base flexível fosse definida como conteúdo. Como observamos anteriormente, a base flexível dos 2 primeiros itens acaba sendo o equivalente a 110 px, pois o conteúdo tem 110 px de largura.

O último item tem sua base flexível definida como 100%. O valor percentual é relativo ao pai, que é de 540 px. Como o terceiro item, com uma base de 100%, não é o único item flexível dentro do contêiner flex sem embalagem, ele não crescerá para ser 100% da largura do contêiner flex pai, a menos que seu fator de encolhimento seja definido com

um fator de redução nulo (o que significa que não pode reduzir) ou se contiver conteúdo não empacotável que seja tão largo ou mais largo do que o contêiner pai. Não é o caso. Cada item flexível contém apenas conteúdo empacotável, e todos os três itens flexíveis podem encolher, pois todos têm um fator de encolhimento padrão de 1.

NOTA

Lembre-se: quando a base flexível é um valor percentual, o tamanho principal é relativo ao pai, que é o contêiner flex.

Se o conteúdo for de fato 110 px de largura e o contêiner tiver 540 px de largura (ignorando outras propriedades do modelo de caixa por uma questão de simplicidade), temos um total de 760 px para caber em um espaço de 540 px neste segundo exemplo. Com três itens flex com bases flexíveis equivalentes a 110 px, 110 px e 540 px, respectivamente, estamos tentando encaixar 760 px de conteúdo em um contêiner de 540 px de largura. Temos 220 px de espaço negativo para distribuir proporcionalmente. O fator de encolhimento é:

$$\text{Fator de encolhimento} = 220\text{px} / 760\text{px} = 28,95\%$$

Cada item flexível será reduzido em 28,95%, tornando-se 71,05% da largura que teria sido se não tivesse sido permitido encolher. Podemos calcular as larguras finais:

$$\begin{aligned} \text{item1} &= 110\text{px} * 71,05\% = 78,16\text{px} & \text{item2} &= 110\text{px} * 71,05\% = \\ &78,16\text{px} & \text{item3} &= 540\text{px} * 71,05\% = 383,68\text{px} \end{aligned}$$

$$\text{item1} + \text{item2} + \text{item3} = 78,16\text{px} + 78,16\text{px} + 383,68\text{px} = 540\text{px}$$

Esses números são verdadeiros desde que os itens flexíveis possam ser tão pequenos: desde que nenhum dos itens flexíveis contenha mídia ou texto não separável com mais de 78,16 px ou

383,68 px. Este é o mais largo que esses itens flexíveis serão, desde que o conteúdo possa ser encapsulado para ser dessa largura ou mais estreito. "Mais largo" porque se um dos outros dois itens flexíveis não puder encolher para ser tão estreito quanto esse valor, eles terão que absorver parte desse espaço negativo.

Em nosso terceiro exemplo, o item `flexbasis: auto` envolve em três linhas. O CSS para este exemplo é o equivalente a:

```
flex-container {  
  largura: 540px;  
}  
item1 {  
  flex: 0 1 70%;  
}
```

```

item2 {
  flex: 0 1 automático;
}
item3 {
  flex: 0 1 80%;
}

```

Declaramos que a flexbasis dos 3 itens flex é de 70%, auto e 80%, respectivamente. Lembrando que "auto" é a largura do conteúdo não embrulhado, que neste caso é de aproximadamente 110 px e nosso container flex é de 540 px, as bases são as equivalentes a:

item1 = 70% 540px = 378px
item2 = widthOfText("flexbasis: auto") = 110px item3 = 80% 540px = 432px

Neste terceiro exemplo temos um item com uma base de 70%. Isso significa que a base é de 70% da largura de 540 px do pai, ou 378 px. O segundo item é definido como automático, o que neste caso significa 110 px devido à largura do conteúdo. Por fim, temos o item flex com uma base de 80%, ou seja, 80% de 540 px, ou 432 px. Quando adicionamos as larguras desses 3 itens flex, eles têm largura total combinada de 920 px, que precisa caber em um contêiner flexível de 540 px de largura. Temos 380 px de espaço negativo para remover proporcionalmente entre os 3 itens flex. Para descobrir a proporção, dividimos a largura disponível do nosso contêiner flex pela soma das larguras dos itens flex que eles teriam se não pudesse encolher:

Largura proporcional = 540px / 920px = 0,587

Como os fatores de encolhimento são todos iguais, isso é bastante simples. Cada item terá 58,7% da largura que teria se não tivesse irmãos de item flexíveis:

item1 = 378px 58,7% = 221,8px item2 = 110px 58,7% =
64,6px item3 = 432px * 58,7% = 253,6px

O que acontece quando o contêiner tem uma largura diferente? Digamos, 1.000 px? A base flexível seria de 700 px (70% x 1.000 px), 110 px e 800 px (80% x 1.000 px), respectivamente, para um total de 1.610 px:

Largura Proporcional = 1000px / 1610px = 0,6211

item1 = 700px * 62,11% = 434,8px item2 = 110px 62,11%
= 68,3px item3 = 800px 62,11% = 496,9px

Porque com uma base de 70% e 80%, as bases combinadas dos itens flexíveis serão sempre maiores que 100%, não importa o quanto largo façamos o pai, todos os 3 itens sempre vai encolher.

Se o primeiro item flexível não pudesse encolher, como mostrado na [Figura 3-18](#), ele seria 70% da largura do pai — 378 px nesse caso. Os outros 2 itens flexíveis encolherão proporcionalmente para caber nos 30% restantes, ou 162 px. Nesse caso, você esperaria que as larguras fossem de 378 px, 32,875 px e 129,125 px. Como o texto "base:" é mais largo do que isso (a 42 px no meu dispositivo), obtemos 378 px, 42 px e 120 px.



Figura 3-18. Embora o valor percentual do `flexbasis` seja relativo à largura do contêiner `flex`, o tamanho principal é afetado por seus irmãos

Testar isso no seu dispositivo provavelmente terá resultados ligeiramente diferentes, já que a largura do texto "flexbasis: auto" pode não ser a mesma na tela.

Base Zero

Se nem a propriedade `flexbasis` nem a abreviatura `flex` estiverem incluídas, o padrão de base será automático. Quando a propriedade `flex` é incluída, mas o componente base da taquigrafia é omitido da taquigrafia, o padrão de base é 0%. Enquanto na superfície você pode pensar que os dois valores de `auto` e 0 podem parecer semelhantes, o valor 0 é realmente muito diferente, e pode não ser o que você esperar. Depois de entendê-lo, você perceberá que é realmente bastante intuitivo.

Os fatores de crescimento e encolhimento têm resultados completamente diferentes, dependendo se a base está definida como 0 versus `auto`, como mostrado na [Figura 3-19](#).

É apenas o espaço extra que é distribuído proporcionalmente com base nos fatores de crescimento flexíveis. No caso do `flexbasis: auto;`, a base é o tamanho principal do seu conteúdo. Se a base de cada um dos itens flex for 0, o espaço "disponível" será todo o espaço ou o tamanho principal do contêiner flex pai. Esse espaço "disponível" é distribuído proporcionalmente com base nos fatores de crescimento de cada item flex. No caso de uma base de 0%, o tamanho do recipiente é dividido e distribuído proporcionalmente a cada item flexível com base em seus fatores de crescimento - seu tamanho principal original padrão, conforme definido por altura, largura ou conteúdo, não é levado em conta, embora largura mínima, largura máxima, altura mínima, e a altura máxima impacta o tamanho flexionado.

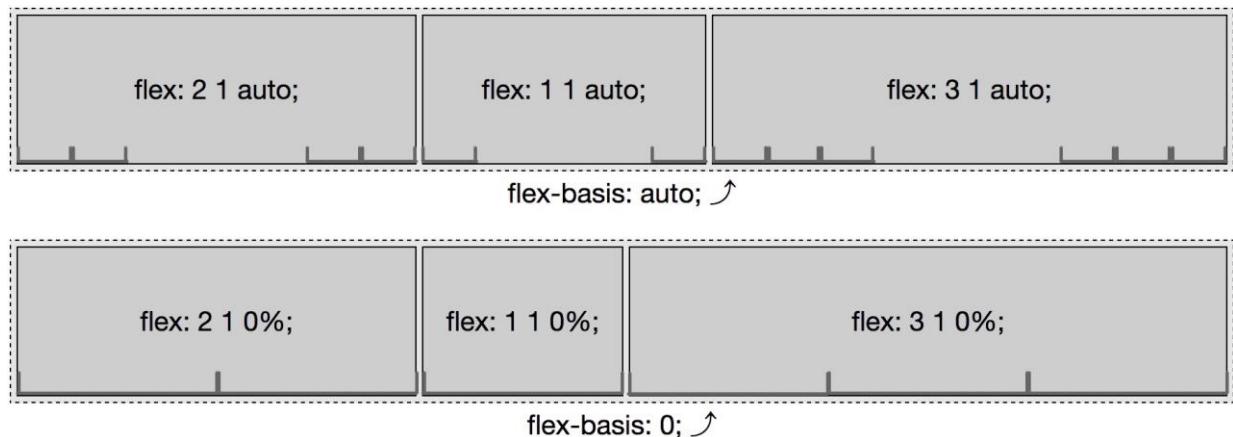


Figura 3-19. `flexbasis auto`, versus 0

Como mostrado neste exemplo, quando a base é automática, é apenas o espaço extra que é dividido proporcionalmente e adicionado a cada item flexível definido para crescer. Novamente, assumindo que a largura do texto "flex: X X auto" é de 110 px, nos primeiros exemplos temos 210 px para distribuir entre 6 fatores de crescimento, ou 35 px por fator de crescimento. Nossos itens flexíveis têm 180 px, 145 px e 215 px de largura, respectivamente.

No segundo exemplo, quando a base é 0, todos os 540 px da largura são espaços distribuíveis. Com 540 px de espaço distribuível entre 6 fatores de crescimento, cada fator de crescimento vale 90 px. Nossos itens flexíveis têm 180 px, 90 px e 270 px de largura, respectivamente. Embora nosso item flex do meio tenha 90 px de largura, o conteúdo deste exemplo é mais estreito do que os 110 px de nossos outros exemplos, portanto, o item flex não foi empacotado.

A propriedade flex Shorthand

Agora que temos uma compreensão mais completa das propriedades que compõem a taquigrafia flex, use sempre a taquigrafia flex. Existem alguns valores abreviados comumente usados, incluindo inicial, automático, nenhum e o uso de um inteiro, geralmente 1, o que significa que o item flexível pode crescer. Vamos passar por cima de todos esses valores.

Valores comuns de flex

Os valores de flex comuns são quatro valores de flex que fornecem os efeitos mais comumente desejados:

flex: inicial

Esse valor dimensiona itens flexíveis com base nas propriedades width/height, ao mesmo tempo em que permite a redução.

flex: auto

Esse valor flexível também dimensiona os itens flexíveis com base nas propriedades largura/altura, mas os torna totalmente flexíveis, permitindo o encolhimento e o crescimento.

flex: nenhum

Esse valor novamente dimensiona os itens flexíveis com base nas propriedades largura/altura, mas os torna completamente inflexíveis: eles não podem encolher ou crescer.

flex: n

Esse valor não se importa com os valores de largura/altura, pois o fator de redução é definido como 0. Nesse caso, o tamanho do item flex é proporcional ao fator flex n.

flex: inicial

Initial é uma palavra-chave CSS global, o que significa que initial pode ser usada em todas as propriedades para representar o valor inicial da propriedade:

```
flex: inicial;  
flex: 0 1 automático;
```

Estas linhas são as mesmas: flex: inicial é o equivalente a flex: 0 1 auto. Isso significa que o tamanho dos itens flexíveis será baseado em suas próprias propriedades de largura e altura, ou com base no conteúdo, se o tamanho principal não estiver explicitamente definido (definido ou padrão como automático). Se o contêiner flex não for grande o suficiente para os itens flex, os itens flex podem ser encolhidos, mas os itens flex não crescerão mesmo se houver itens extras espaço distribuível disponível.

Declarando flex: inicial define um fator de crescimento nulo, um fator de encolhimento de 1 e define as bases flexíveis como automáticas. Na [Figura 3-20](#), podemos ver o efeito da flexão automática

Bases. Nos dois primeiros exemplos, a base de cada item flexível é o conteúdo — com cada item flexível tendo a largura da única linha de letras que compõem seu conteúdo. Nos últimos 2 exemplos, as bases flex de todos os itens são iguais a 50 px, uma vez que largura: 50px foi aplicada a todos os itens flex. A declaração inicial flex: define o flexbasis como automático, o que aprendemos anteriormente significa que é o valor da largura (ou altura), se declarado, ou conteúdo se não for declarado.

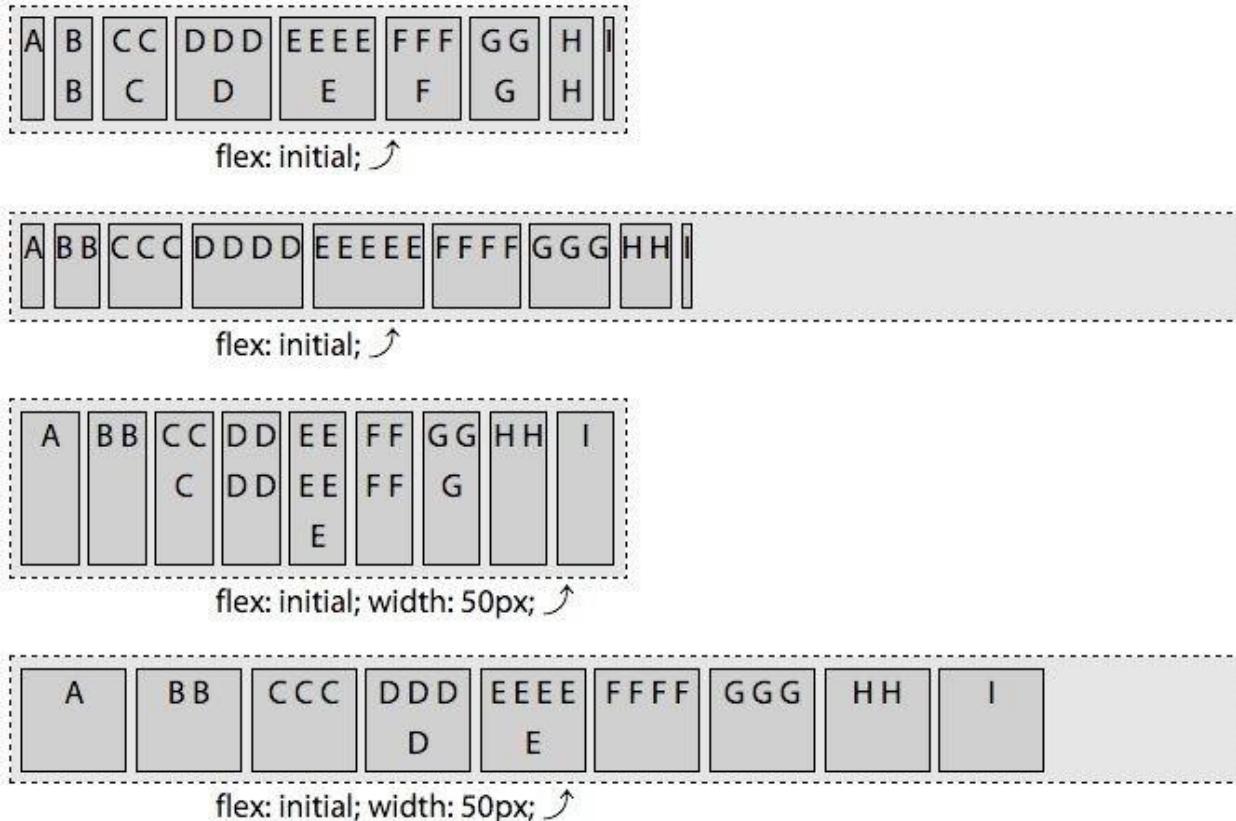


Figura 3-20. Com recipientes de diferentes tamanhos principais, os itens flexíveis encolhem, mas não crescem quando flexionados:

inicial é definido nos itens flexíveis

No primeiro e terceiro exemplos da [Figura 3-20](#), vemos como, se o contêiner flex for muito pequeno para caber todos os itens flex em seu tamanho principal padrão, os itens flex diminuirão, com todos os itens flex se encaixando dentro do contêiner flex pai. Nestes exemplos, as bases flexíveis combinadas de todos os itens flex são maiores do que o tamanho principal do contêiner flex. No primeiro exemplo, a quantidade pela qual cada item flexível encolhe é diferente. Todos eles encolhem proporcionalmente com base em seu fator de encolhimento. No terceiro exemplo, com a flexbase de cada item flexível sendo igual a 50 px,

todos os itens encolhem igualmente.

No primeiro exemplo, você notará que o último item flex, com a letra maiúscula estreita única I, é o item flex mais estreito do grupo, seguido por A (Eles fariam os mesmos se tivéssemos usado uma família de fontes monoespacadas). B e H são mais largos que A e I, mesmo com o empacotamento causado pelos itens flexíveis encolhendo, pois suas bases são baseadas em três caracteres (duas letras) e um espaço) em vez de um. O encolhimento resulta nesses itens flexíveis com apenas uma letra por linha de texto, mas eles ainda são mais largos do que os itens flexíveis que têm um único caractere como base.

Os itens flex com mais caracteres são mais largos porque o flexbasis é baseado na largura do conteúdo. A e eu temos uma letra de largura. A base para B e H é baseada na largura de duas letras separadas por espaço. Da mesma forma, D e F são mais largos que C e G. No primeiro exemplo, com flex: inicial, E acaba sendo o mais largo, pois tem o maior número de caracteres criando o conteúdo mais amplo de todas as bases.

No segundo e quarto exemplos, todos os itens flexíveis se encaixam, portanto, não há encolhimento. Quando flex: inicial é definido, o fator de crescimento é nulo, de modo que os itens flex não podem crescer e, portanto, não crescem, para encher seu recipiente.

Os itens Flex, por padrão, são agrupados no início principal, pois flex-start é o valor padrão de para a propriedade justify-content. Isso só é perceptível quando o tamanho principal combinado dos itens flex em uma linha flex é menor do que o tamanho principal do contêiner flex.

flex: auto

Configuração flex: auto; em um item flex é o mesmo que definir flex: 1 1 auto. As duas instruções a seguir são equivalentes:

```
flex: automático;  
flex: 1 1 automático;
```

flex: auto é semelhante ao flex: inicial, mas torna os itens flex flexíveis em ambas as direções: eles encolherão se não houver espaço suficiente para caber todos os itens dentro do recipiente e crescerão para ocupar todos os itens espaço extra dentro do contêiner se houver espaço distribuível. Os itens flexíveis absorvem qualquer espaço livre ao longo do eixo principal.

Você notará que o primeiro e o terceiro exemplos da [Figura 3-21](#) são idênticos aos exemplos na [Figura 3-20](#), pois o encolhimento e as bases são os mesmos. No entanto, o segundo e o quarto exemplos são diferentes, como quando flex: auto é definido, o fator de crescimento não é nulo e, portanto, os itens flex crescem incorporando todo o espaço extra disponível.

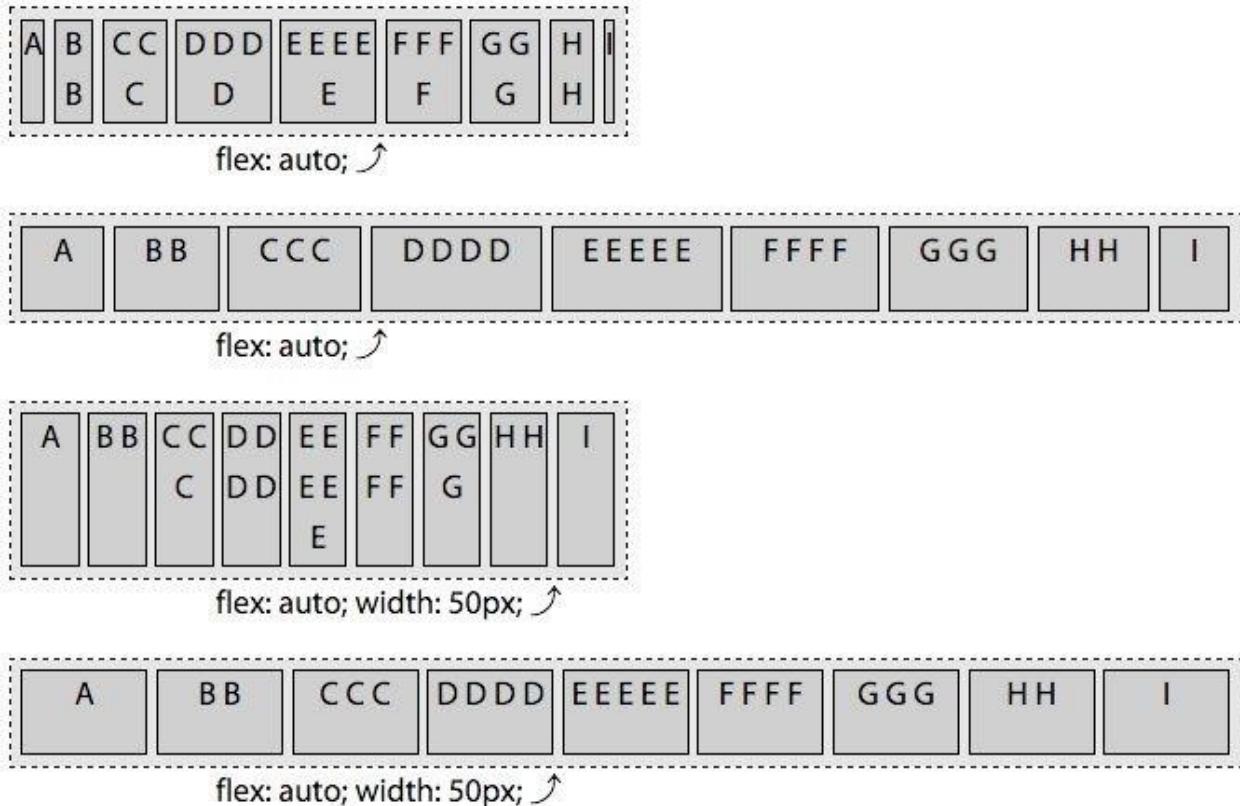


Figura 3-21. Com `flex: auto` definido nos itens `flex`, eles podem crescer e encolher



flex: nenhum

Configuração `flex: nenhuma` é equivalente à configuração `flex: 0 0 auto`, tornando as duas seguintes linhas de CSS equivalentes:

```
flex: nenhuma;
flex: 0 0 auto;
```

O `flex: nenhum` item é inflexível: os itens flex não encolherão nem crescerão.

Conforme demonstrado no primeiro e terceiro exemplos da [Figura 3-22](#), se não houver espaço suficiente, os itens flex transbordam o contêiner flex. Isso é diferente de

flex: inicial e flex: auto, que ambos definem um fator de encolhimento positivo.

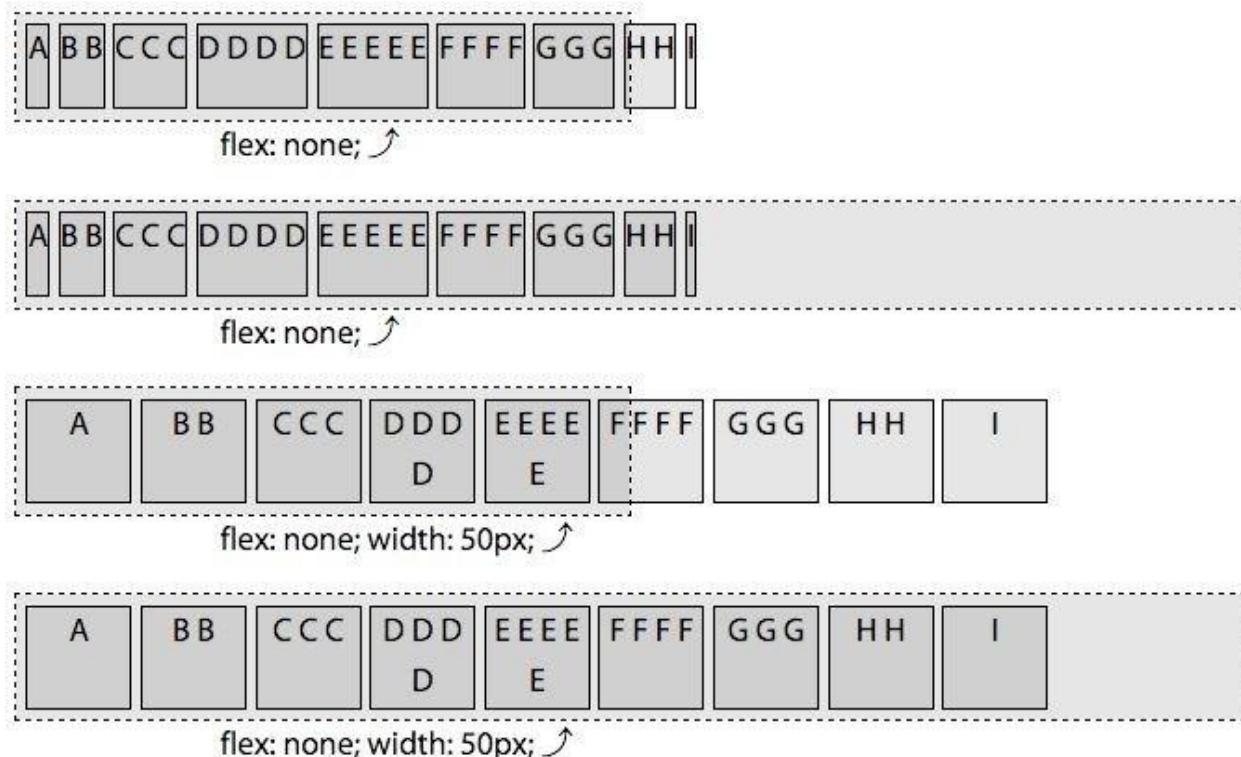


Figura 3-22. Com `flex: none`, os itens flex não crescerão nem diminuirão



A base é resolvida como automática, o que significa que o tamanho principal de cada item flexível é determinado pelo tamanho principal do nó se o elemento não tiver sido transformado em um item flexível: o `flexbasis` é resolvido para o valor de largura ou altura. Se esse valor fosse resolvido automaticamente, a base seria o tamanho principal do conteúdo. Nos dois primeiros exemplos, a base – e a largura –, já que não há crescimento ou encolhimento – é a largura do conteúdo. No terceiro e quarto exemplos, a largura e a base são todas de 50 px.

flex: n

Quando os valores da propriedade `flex` são um valor numérico único e positivo, esse valor será o fator de crescimento, enquanto o fator de encolhimento será padrão para 0 e a base o padrão será 0%:

```
flex-grow: n; flex-shrink: 0;  
flex-base: 0%;
```

As duas declarações CSS a seguir são equivalentes:

```
flex: 3;  
flex: 3 0 0%;
```

Declarar `flex : 3` é o mesmo que declarar `flex: 3 0 0%`. Isso torna o item flexível no qual ele é definido flexível: ele pode crescer. O fator de encolhimento é realmente discutível, pois a base flexível foi ajustada para 0 % e o item flex, mesmo que um fator de encolhimento positivo tenha sido declarado, não pode ter menos de 0 px de largura ou altura.

Nos dois primeiros exemplos da [Figura 3-23](#), todos os itens flex têm um fator de crescimento `flex r` de 3. Todos os itens flexíveis em cada exemplo cresceram para ter a mesma largura. Enquanto nas Figuras 3-20, 3-21 e [3-22](#), os itens flex encolheram para se ajustarem ao pai do contêiner flex, quando você define `flex: n`, onde `n` é qualquer flutuador positivo, a base é 0 , então eles não "encolheu"; eles apenas cresceram igualmente de 0px de largura até que a soma de sua dimensão principal cresceu para preencher a dimensão principal do recipiente. Com todos os itens flexíveis tendo uma base de 0, 100% da dimensão principal é o espaço distribuível.

O tamanho principal dos itens flexíveis é mais amplo neste segundo exemplo, pois o pai mais amplo tem mais espaço distribuível.

Qualquer valor para `n` que seja maior que 0, mesmo 0,1, significa que o item flexível pode crescer. O `n` é o fator de crescimento. Quando houver espaço disponível para crescer, se apenas um item flexível tiver um fator de crescimento positivo, esse item ocupará todo o espaço disponível . Se todos os itens puderem crescer, o espaço extra disponível será distribuído proporcionalmente a cada item flexível com base em seu fator de crescimento. No caso de todos os flex items com `flex: n` declarado, em que `n` é um único ou uma variedade de números positivos, o crescimento será proporcional com base apenas em `n`, não no tamanho principal subjacente, pois a base para cada item flexível é 0.

Nos três últimos exemplos da [Figura 3-23](#), há seis itens flex com `flex: 0`, `flex: 1`, `flex: 2`, `flex: 3`, `flex: 4` e `flex : 5` declarados, respectivamente. Estes são os fatores de crescimento para os itens flex, com cada um tendo um fator de encolhimento de 0 e uma base de 0. O tamanho principal de cada um é proporcional ao fator de crescimento flexível especificado.

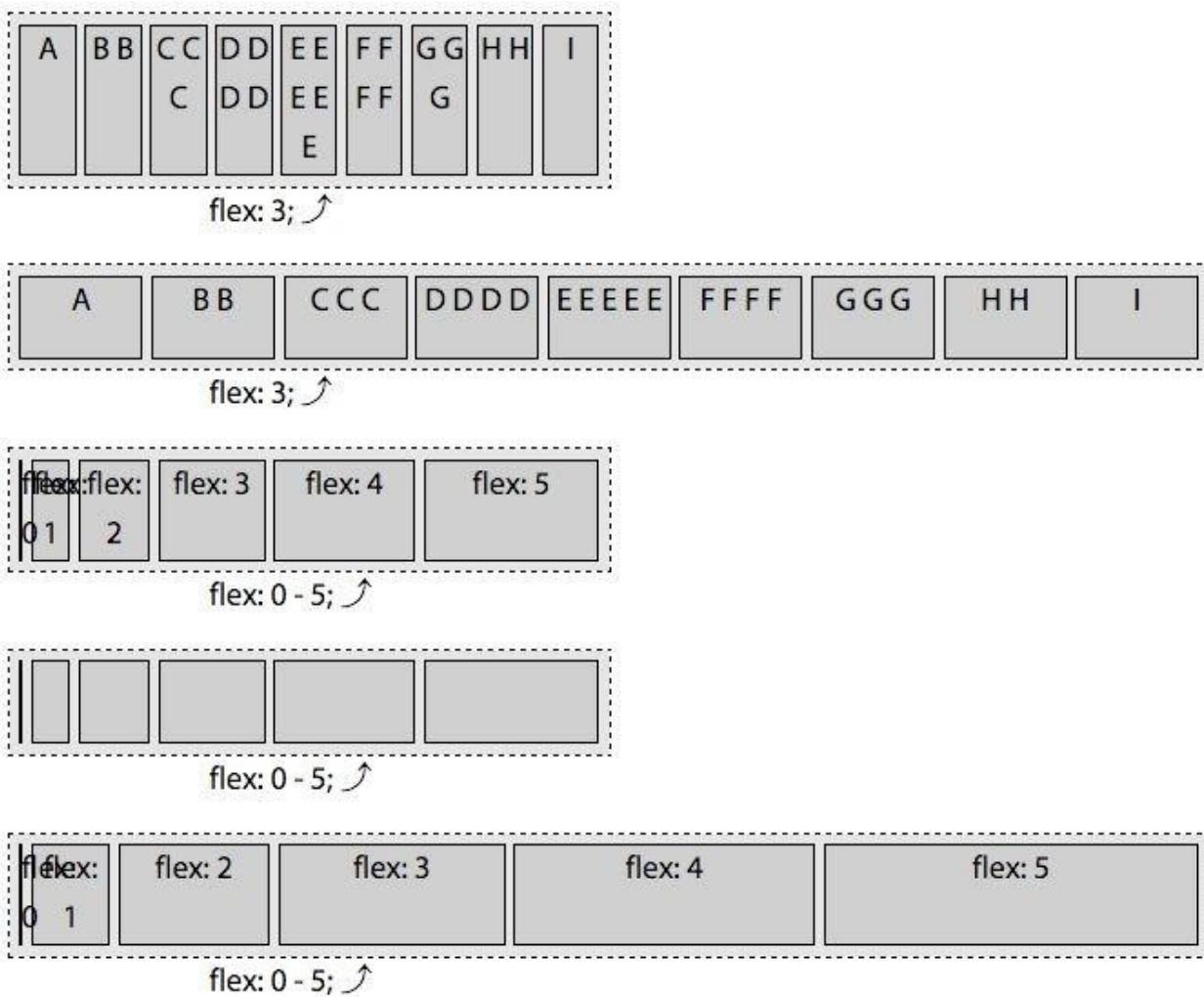


Figura 3-23. Com `flex: n`, você está declarando o fator de crescimento de itens flexíveis enquanto define a base flexível como zero



São 15 fatores de crescimento totais distribuídos em 6 itens flexíveis. O recipiente flexível estreito tem 300 px de largura, o que significa que cada fator de crescimento vale 20 px. Isso significa que as larguras dos itens flexíveis serão 0, 20 px, 40 px, 60 px, 80 px e 100 px, para um total de 300 px. O último exemplo tem uma largura de 600 px, o que significa que cada fator de crescimento vale 40 px. Isso nos deixa com itens flexíveis que são 0, 40 px, 80 px, 120 px, 160 px e 200 px, para um total de 600 px.

Adicionamos um pouco de preenchimento, margens e bordas no exemplo para tornar os visuais mais agradáveis. Por esse motivo, o item flexível mais à esquerda, com `flex: 0` declarado, é visível: ele tem uma borda de 1 px, tornando-o visível, mesmo que tenha 0 px de largura.

Se tivéssemos declarado uma largura nesses itens flexíveis, não teria feito diferença. Ajuste `flex: 0`, `flex: 5`, ou quaisquer outros valores para `n` em `flex: n` define a flexbase para 0%. Você não verá diferença entre as Figuras 3-23 e 3-24, além da "largura: 50px" no parágrafo abaixo de cada contêiner flex.

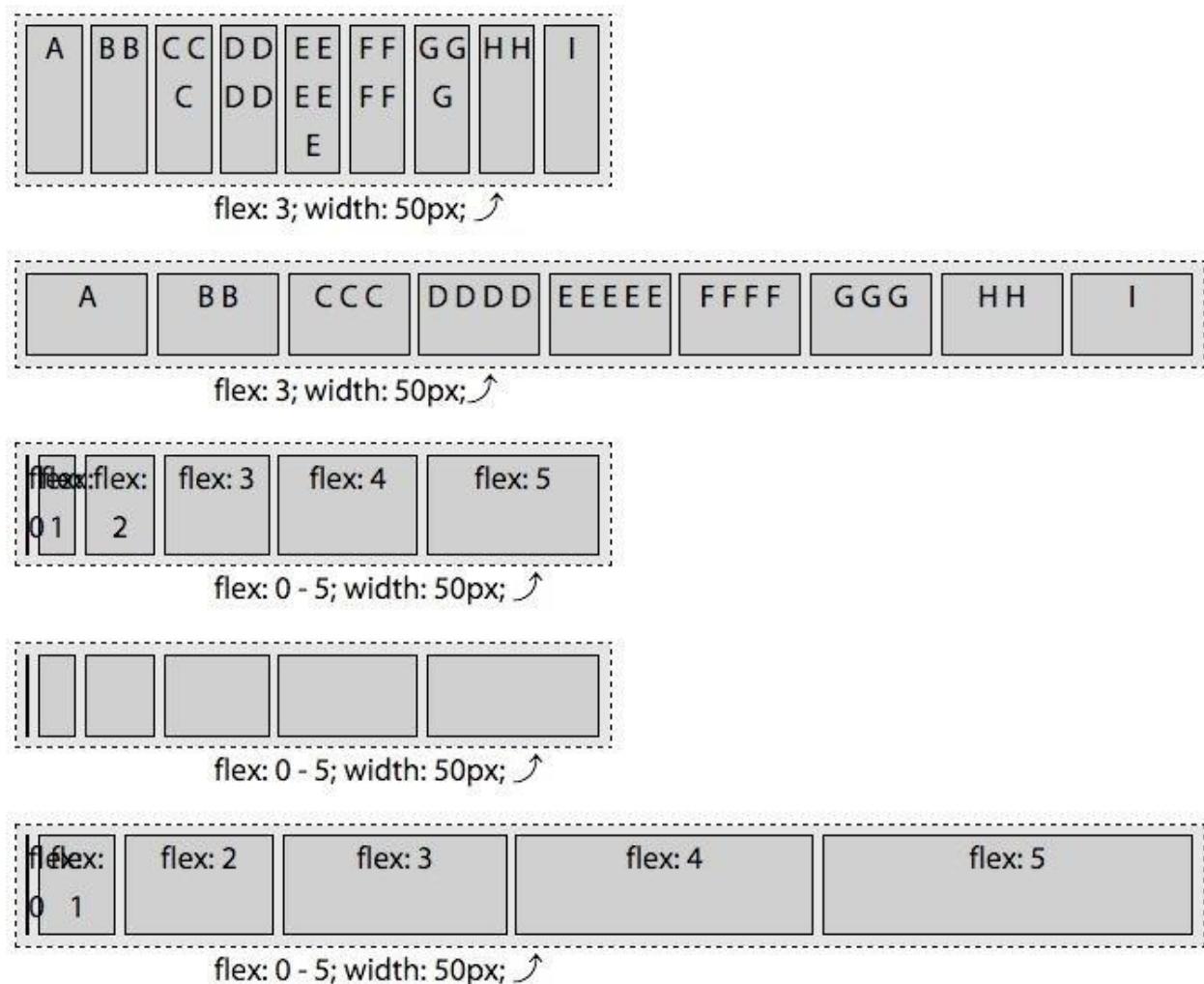


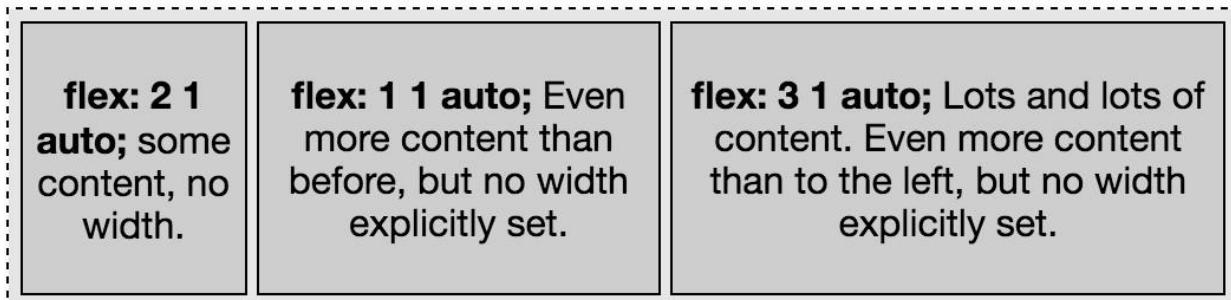
Figura 3-24. Com `flex : n`, a base do item `flex` é 0, não automática, portanto, definir uma largura não alterará o aparênciā 

Valores personalizados do flex

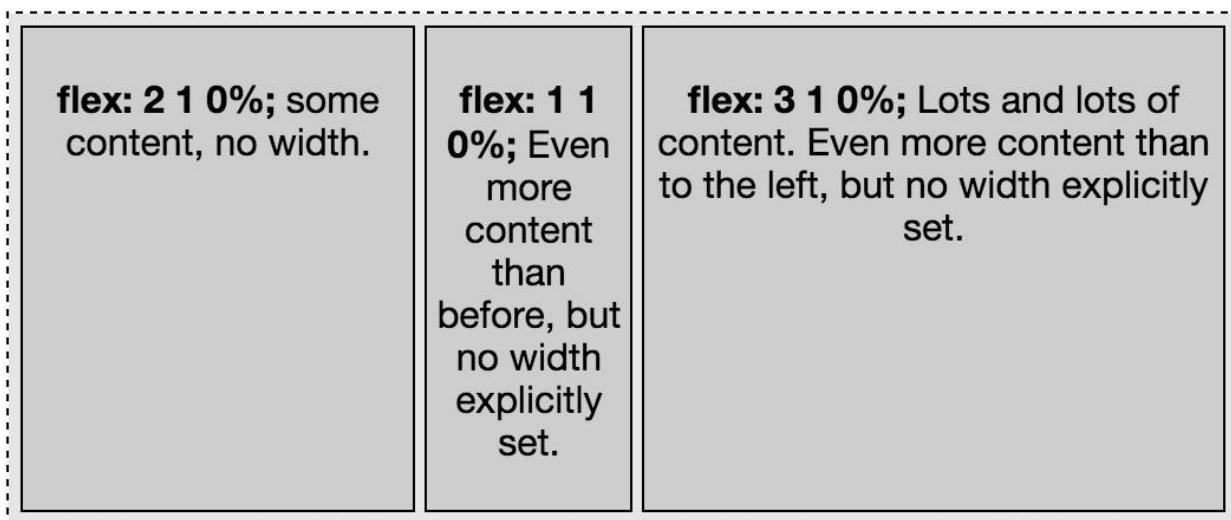
flex: initial, flex: auto, flex: none e flex: n são valores predefinidos. Embora estes sejam fornecidos, pois são os valores mais comumente usados, eles podem não atender a todas as suas necessidades e não fornecem tanta flexibilidade quanto definir sua possuem valores flexíveis personalizados.

Quando o flexbasis é definido como automático, como mostrado no primeiro exemplo na [Figura 3-25](#), o tamanho principal de cada item flexível é baseado no conteúdo. Olhando para o fator de crescimento, você pode ter pensado que o item do meio seria o mais estreito, pois o fator de crescimento é o menor, mas não há "crescimento". Neste exemplo, não há espaço distribuível disponível. Em vez disso, todos eles estão encolhendo igualmente, pois todos têm o mesmo fator de encolhimento e a base do auto significa que a base é a largura do conteúdo. Nesse caso, o conteúdo dos itens flexíveis é todo de igual altura, pois o tamanho de cada um é baseado na base flex, e não no fator de crescimento.

No segundo exemplo da [Figura 3-25](#), a base é 0%, de modo que o tamanho principal é determinado pelos vários fatores de crescimento. Neste caso, o primeiro item flex, com flex: 2 1 0% é duas vezes mais largo que o segundo item flex com flex: 1 1 0%. Este segundo item, por sua vez, tem um tamanho principal que é um terço da largura do último item flex com flex: 3 1 0%; .



Flex container ↑



Flex container ↑

Figura 3-25. *flexbasis auto versus 0*

Incluímos um fator de encolhimento neste exemplo para torná-lo paralelo ao primeiro exemplo, mas não era completamente necessário. Ao usar uma base de 0 (ou 0%), o fator de encolhimento não tem impacto no tamanho principal dos itens flexíveis. Poderíamos ter incluído qualquer uma das três declarações em cada declaração:

```
item1 {
  flex: 2 1 0%;
  flex: 2 0 0%;
  flexão: 2;
}

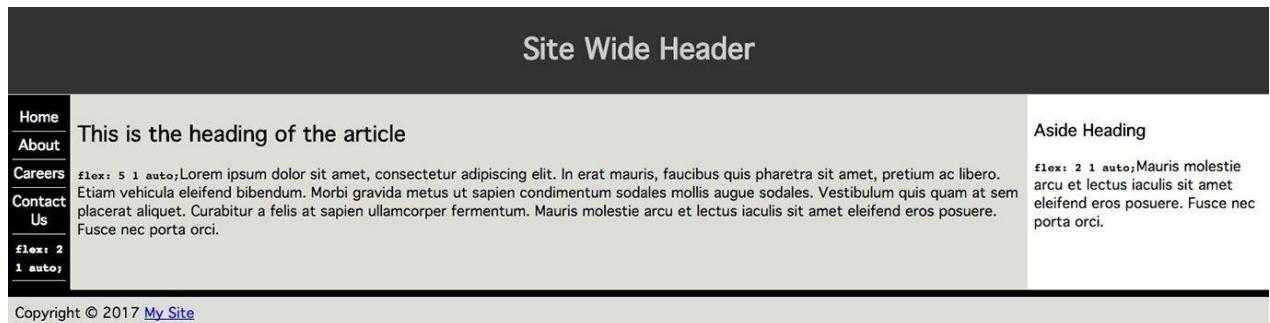
item2 {
  flex: 1 1 0%;
  flex: 1 0 0%;
  flex: 1;
}

item3 {
  flex: 3 1 0%;
  flex: 3 0 0%;
  flex: 3;
}
```

Nesses dois exemplos, não havia largura definida nos itens flexíveis. Por padrão, os itens

flexíveis não diminuirão abaixo do tamanho mínimo do conteúdo (o comprimento da palavra mais longa ou do elemento de tamanho fixo). Para alterar isso, defina a propriedade min-width ou min-height. Para os exemplos da [Figura 3-25](#), se a largura mínima: 200px tivesse sido definida no segundo item flex, em ambos os casos, a largura teria sido de 200 px.

Como isso pode ajudar na prática? Em teoria, as colunas de altura semelhantes que obtemos com o auto parecem uma boa ideia, mas, na realidade, controlar as larguras das colunas em um layout de várias colunas é provavelmente uma prioridade maior, como demonstrado pela diferença nos dois layouts nas Figuras 3-26 e [3-27](#).



Site Wide Header

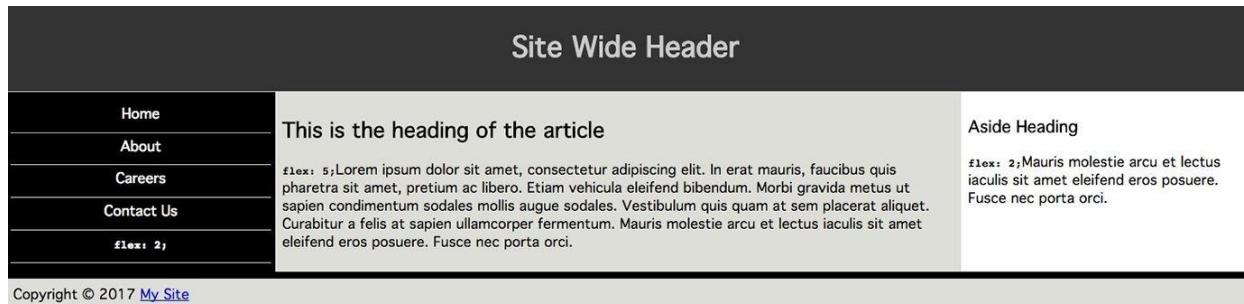
Home
About
Careers
Contact Us
flex: 2 1 auto;

This is the heading of the article
flex: 5 1 auto;Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Aside Heading
flex: 2 1 auto;Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Copyright © 2017 [My Site](#)

Figura 3-26. Um layout responsivo de três colunas com apenas algumas linhas de código e `flex: auto` 



Site Wide Header

Home
About
Careers
Contact Us
flex: 2;

This is the heading of the article
flex: 5;Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Aside Heading
flex: 2;Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Copyright © 2017 [My Site](#)

Figura 3-27. Alterando a proporção de um layout responsivo alterando o valor flexível dos itens flexíveis 

A base flexível do primeiro exemplo, [Figura 3-26](#), é baseada no conteúdo. Para a navegação, a linha mais longa e, portanto, a base, é o link com o conteúdo mais longo. Se o artigo ou o lado continha uma quebra de linha no longo e mais longo

parágrafos, suas larguras mudariam.

O CSS a seguir foi usado para criar o segundo exemplo:

```
@media tela e (largura mínima: 500px) {  
  principal {  
    display: flex;  
  }  
  nav {  
    ordem: -1;  
    flexão: 2;  
    largura mínima: 150px;  
  }  
  artigo {  
    flex: 5;  
  }  
  à parte {  
    flexão: 2;  
    largura mínima: 150px;  
  }  
}
```

No segundo exemplo, [Figura 3-27](#), em viewports com 500 px ou mais largos, o nav e o side sempre terão 22% da largura da área principal, a menos que contenham um componente maior que 22%. O artigo principal terá 56% da largura. Você pode incluir uma largura mínima em qualquer um ou todos os itens, como fizemos na navegação e à parte, para garantir que o layout nunca fique muito estreito. Você pode usar consultas de mídia para permitir que as seções do layout caiam em telas muito estreitas.

Tantas opções!

Abordaremos este exemplo novamente quando discutirmos a propriedade `order` "The order property". Rapidamente: na marcação, a navegação é a última, e virá por último em um visor com menos de 500 px de largura. Em telas mais largas que têm espaço para as três colunas, ele virá em primeiro lugar na aparência. Os leitores de tela ainda o lerão na ordem do código-fonte (exceto no Firefox no momento). A tabulação ainda fará com que ele apareça por último.

Rodapé pegajoso com flexão

Obviamente, o flex pode ser usado para muitos projetos. Um comum é criar um rodapé pegajoso. Um rodapé pegajoso é um rodapé de altura fixa na parte inferior do documento, aderindo à parte inferior da janela do navegador, mesmo quando o documento seria mais curto do que a janela do navegador.

Como você pode ver na [Figura 3-28](#), quando o documento fica mais alto, o rodapé fica na parte inferior da janela:

```
<corpo>
  <cabeçalho>... </cabeçalho>
  <principal>
    <navegação>
      <a href="/">Home</a>
      <a href="/about">Sobre</a>
      <a href="/blog">Blog</a>
      <a href="/jobs">Carreiras</a>
      <a href="/contact">Fale Conosco</a>
    </navegação>
    <artigo>... </artigo>
    <à parte>... </à parte>
  </principal>
  <rodapé>
    <a href="/">Home</a>
    <a href="/about">Sobre</a>
    <a href="/blog">Blog</a>
    <a href="/jobs">Carreiras</a>
    <a href="/contact">Fale Conosco</a>
  </rodapé>
</corpo>
```

Esta é uma maneira muito comum de marcar um site típico. Com algumas linhas de CSS, podemos criar o rodapé pegajoso:

```
corpo {
  display: flex;
  flex-direção: coluna;
  min-altura: 100vh;
}
principal {
  flex: 1;
}
rodapé {
  altura: 3rem;
}
```

Se você definir o corpo como um contêiner flexível pelo menos tão alto quanto a janela do navegador — como fizemos com a altura mínima: 100vh; — e definir o domínio principal da página para poder crescer, quando você dá ao rodapé uma altura definida, ele sempre será essa altura, preso à parte inferior do documento, aparecendo preso ao parte inferior da janela do

navegador, mesmo que a área principal não tenha conteúdo suficiente para preencher a janela e cresça para preencher o espaço disponível.

Sem um valor flexível, o rodapé não encolherá nem crescerá, pois a área principal estará fazendo o crescimento.

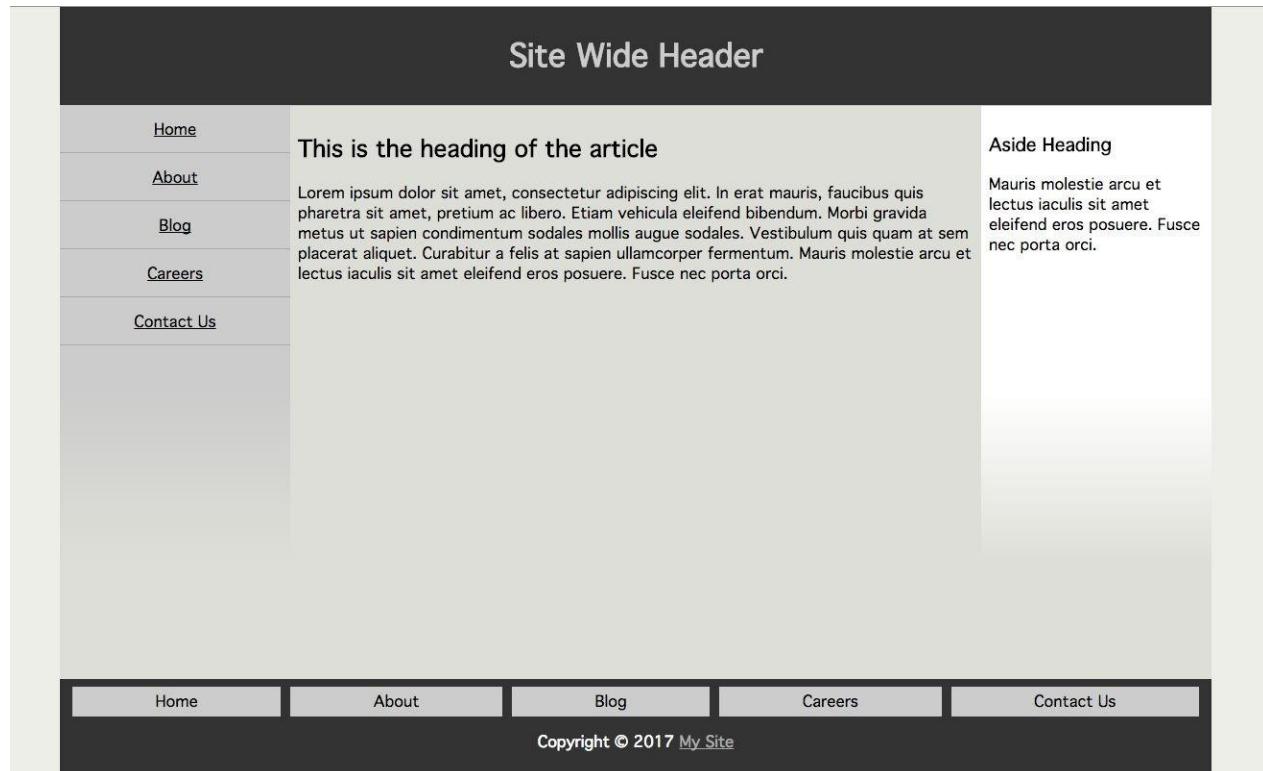


Figura 3-28. Rodapé pegajoso com flex: 1 na área principal faz com que o rodapé grude no fundo mesmo quando há muito espaço não utilizado 

A área principal não encolherá para ser menos alta do que a navegação esquerda , embora teoricamente possa encolher com uma flex: 1; sendo definido na principal. O motivo? Neste exemplo, os elementos `<main>` e `<nav>` são itens flexíveis e contêineres flexíveis, com o item de navegação não sendo encolhível. Se tivéssemos definido:

```
nav a {  
  flex: 0 1 0%;  
}
```

os links teriam sido capazes de encolher e, com uma base flexível de 0, teriam encolhido, permitindo que o principal encolhesse ainda mais, deixando espaço para que o rodapé estivesse em sua altura declarada.

A propriedade alignself

A propriedade alignself é usada para substituir o valor da propriedade align-items em uma base por item flexível.

ALINHAR	
Valores	automática flex-start flex-end centro Basal esticar
Valor inicial :	Automático
Aplica-se um:	Itens Flex
Herdados:	Não
Porcentagens:	Não aplicável
Animável:	Não

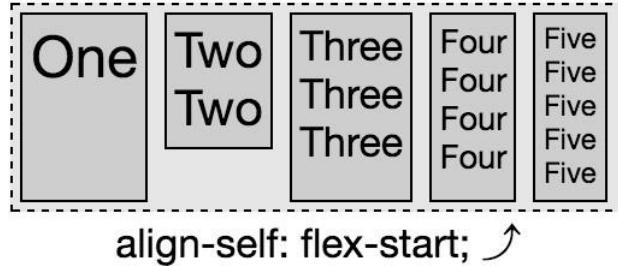
Com a propriedade align-items definida no contêiner flex , você pode alinhar todos os itens flex desse contêiner ao início, ao fim ou ao centro do eixo cruzado de sua linha flex. A propriedade alignself, que é definida diretamente no item flex , permite que você substitua a propriedade align-items em uma base por item flex.

Você pode substituir o alinhamento de qualquer item flexível individual pela propriedade alignself, como mostra a [Figura 3-29](#). O valor padrão de align-items é stretch, e é por isso que todos os itens flexíveis nos cinco exemplos da [Figura 3-29](#) são tão altos quanto o pai, com a exceção do segundo item flex. Todos os itens flex têm o valor padrão de auto set do alignself, o que significa que eles herdam o alinhamento da propriedade align-items do contêiner, exceto o segundo item flex em cada exemplo. Cada "dois" tem um conjunto de valores de alinhamento diferente.

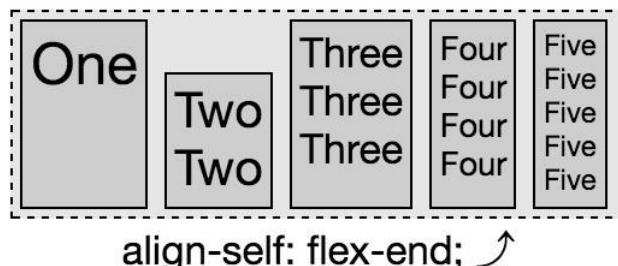
Assim como com os valores da propriedade align-items, o valor flex-start coloca o item na borda de início cruzado. flex-end coloca o item na borda cruzada. o centro alinha o item no meio do eixo transversal, *etc.* Neste exemplo, auto, herdar e esticar todos os itens flexíveis, como o valor de itens de alinhamento

foi permitido o padrão para esticar.

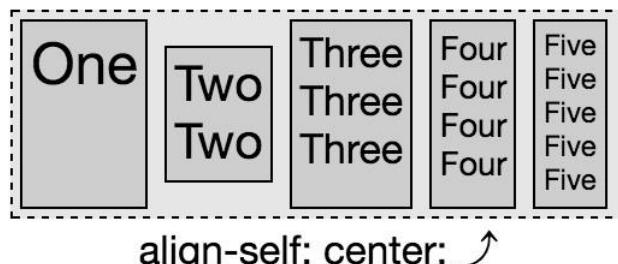
Para saber mais sobre o flex-start, flex-end, center, linha de base e stretch valores, consulte "[A propriedade align-items](#)".



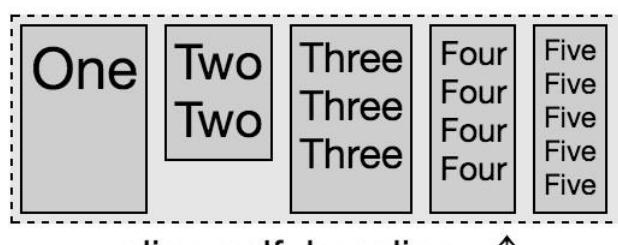
align-self: flex-start; ↗



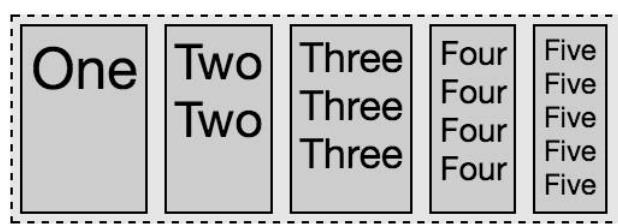
align-self: flex-end; ↗



align-self: center; ↗



align-self: baseline; ↗



align-self: stretch; ↗

align-self: auto; ↗

Figura 3-29. Você pode substituir o alinhamento de qualquer item flexível individual pela propriedade `alignself`; observe que o valor padrão de itens de alinhamento é `stretch` 

A propriedade order

Os itens do Flex são, por padrão, exibidos e dispostos na mesma ordem em que aparecem no código-fonte. A ordem dos itens flex e das linhas flex pode ser invertida com valores reversos da propriedade `flex container`. A propriedade `order` pode ser usada para alterar a ordem de itens flexíveis individuais.

A propriedade `order` permite que você controle a ordem na qual os itens flex aparecem dentro do contêiner flex atribuindo-os a grupos ordinais. Por padrão, todos os itens flex recebem a ordem de 0, com os itens flex sendo exibidos na mesma ordem que a ordem de origem e a direção dessa ordem com base nas propriedades do contêiner flex ("Flex Propriedades do contêiner").

ORDE	
Valores	<íntero>
Valor inicial :	0
Aplica-se um:	Itens Flex e filhos absolutamente posicionados de
Herdados:	Não
Porcentagens:	Não aplicável
Animável:	Sim

Para alterar a ordem visual de um item flexível, defina o valor da propriedade `order` como um íntero diferente de zero. Definir a propriedade `order` em elementos que não são filhos de um contêiner flex não tem efeito sobre esse elemento: a propriedade é basicamente ignorada nesse caso.

O valor da propriedade `order` especifica a qual grupo ordinal o item flex pertence. Todos os itens flexíveis com um valor negativo aparecerão antes daqueles que têm como padrão 0 quando desenhados na página e todos os itens flexíveis com um valor positivo parecerá vir depois daqueles que estão inadimplentes com 0. Enquanto visualmente

alterada, a ordem de origem permanece a mesma. Os leitores de tela e a ordem de tabulação permanecem conforme definido pela ordem de origem do HTML.

Por exemplo, se você tiver um grupo de 12 itens e quiser que o 7º venha em primeiro lugar e o 6º seja o último, como mostra a [Figura 3-30](#), você declarará:

```
ul {  
  display: inline-flex;  
}  
li:nth-of-type(6) {  
  ordem: 1;  
}  
li:nth-of-type(7) {  
  ordem: -1;  
}
```



Figura 3-30. Definir a ordem para qualquer valor diferente de 0 reordenará esse item flexível

Nesse cenário, estamos definindo explicitamente a ordem para os itens da sexta e sétima lista , enquanto os outros itens da lista estão em ordem padrão : 0.

Como mostrado na [Figura 3-30](#), os itens flexíveis são dispostos do valor de pedido mais baixo para o mais alto. O sétimo item é o primeiro na aparência devido ao valor negativo da propriedade order — order: -1 — que é menor que o zero padrão e o menor valor de qualquer um de seus itens flex irmãos. O sexto item, o único item com um valor maior que zero e, portanto, com o maior valor de ordem de todos os seus irmãos, aparecerá visualmente por último. Todos os outros itens, todos com a ordem padrão de 0 , serão sorteados para a página entre esses primeiros e últimos itens, na mesma ordem que sua ordem de origem, uma vez que são todos eles membros do mesmo grupo ordinal — 0.

O contêiner flexível estabelece seu conteúdo em ordem de documento modificado por ordem , começando a partir do grupo ordinal numerado mais baixo e subindo. Quando você tem vários itens flexíveis com o mesmo valor para a propriedade order, os itens estão no mesmo grupo ordinal. Os itens flexíveis aparecerão em ordem por grupo ordinal e os itens em cada grupo ordinal aparecerão na ordem de origem:

```
ul {  
  display: inline-flex;  
  cor de fundo: rgba(0,0,0,0. 1);  
}  
li:nth-of-type(3n-1) {  
  ordem: 3;  
  cor de fundo: rgba(0,0,0,0. 2);  
}
```

```
li:nth-of-type(3n+1) {
  ordem: -1;
  cor de fundo: rgba(0,0,0,0.4);
}
```

Definindo o mesmo valor de ordem para mais de um item flexível, os itens aparecerão por grupo ordinal e por ordem de origem dentro de cada grupo ordinal individual. Na [Figura 3-31](#), você notará que os itens mais escuros aparecem primeiro, que têm uma ordem: -1 definida. Dentro desse grupo ordinal, os itens aparecem na ordem em que aparecem no código-fonte. O grupo do meio, o equivalente a `li:nth-of-type(3n)`, não tem ordem ou conjunto de valores de cor de fundo; portanto, eles têm como padrão `order:0` e com um plano de fundo transparente, a cor de fundo do `ul` pai é exibida. Os últimos quatro itens têm todos ordem : 3 conjunto, que é o valor mais alto de qualquer conjunto de ordem, maior que o padrão 0. Este grupo ordinal aparece por último. Dentro do grupo ordinal, os elementos são dispostos na ordem em que apareceram no código-fonte.

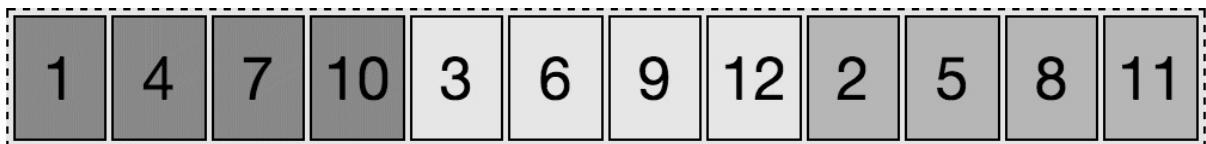


Figura 3-31. Os itens do Flex aparecem em ordem de grupos ordinais, por ordem de origem dentro de seu grupo

Essa reordenação é puramente visual. Os leitores de tela devem ler o documento como ele apareceu no código-fonte. Como uma mudança visual, ordenar itens flex afeta a ordem de pintura da página: a ordem de pintura do flex items é a ordem em que eles aparecem, como se tivessem sido reordenados no documento de origem, o que eles não são .

Alterar o layout com a propriedade `order` não tem efeito sobre a ordem de tabulação da página. Se os números da [Figura 3-31](#) fossem links, a tabulação pelos links passaria pelos links na ordem do código-fonte, não na ordem do

layout. Embora possa ser intuitivo que a ordem dos links na **Figura 3-31** seja 1, 4, 7, 10, 3, 6, 9, 12, 2, 5, 8 e 11 , a tabulação pelos links na verdade, levá-lo em ordem de 1 a 12.

Navegação com guias revisitada

Adicionando ao nosso exemplo de barra de navegação com guias na Figura 1-7, podemos fazer com que a guia atualmente ativa apareça primeiro, como mostra a **Figura 3-32**:

```
nav {  
  display: flex;  
  justify-content: flex-end;  
  border-bottom: 1px solid #ddd;  
}  
a {  
  margin: 0 5px;  
  padding: 5px 15px;  
  border-radius: 3px 3px 0 0; background-color: #ddd; text-decoration: none; color: black;  
}  
a:hover {  
  background-color: #bbb;  
  text-decoration: underline;  
}  
a.active {  
  order: -1;  
  background-color: #999;  
}  
  
<navegação>  
  <a href="/">Home</a>  
  <a href="/about">Sobre</a>  
  <a class="active">Blog</a>  
  <a href="/jobs">Carreiras</a>  
  <a href="/contact">Fale Conosco</a>  
</navegação>
```

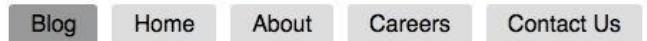


Figura 3-32. Alterar a ordem alterará a ordem visual, mas não a ordem de tabulação

A guia ativa no momento tem a classe `.active` adicionada, o atributo `href` removido e a ordem definida como `-1`, que é menor que o `0` padrão do outro irmão flex itens, o que significa que aparece primeiro.

Por que removemos o atributo `href`? Como a guia é o documento ativo no momento, não há motivo para o documento se vincular a si mesmo. Mas, mais importante, se fosse um link ativo em vez de um link de espaço reservado, e o usuário estivesse usando o teclado para percorrer a navegação, a ordem de aparição é Blog, Home, Sobre, Carreiras e Fale Conosco, com o Blog aparecendo primeiro; mas a ordem de tabulação teria sido Home, About, Blog, Careers e Fale

Conosco, seguindo a ordem de origem em vez da ordem visual, que pode seja confuso.

A propriedade `order` pode ser usada para habilitar a marcação da área de conteúdo principal antes das colunas laterais para dispositivos móveis e aqueles que usam leitores de tela e outras tecnologias assistivas, ao criar a aparência do layout comum de três colunas: uma área de conteúdo principal central, com navegação do site à esquerda e uma barra lateral à direita, como nós demonstrado em nossos ["Valores flexíveis personalizados"](#).

Embora você possa colocar o rodapé antes do cabeçalho na marcação e usar a propriedade `order` para reordenar a página, esse é um uso inadequado da propriedade. O pedido só deve ser usado para reordenação visual do conteúdo. Sua marcação subjacente deve sempre refletir a ordem lógica do seu conteúdo:

```
<cabeçalho></cabeçalho>
<principal>
  <artigo></artigo>
  <aparte></aparte>
  <nav></nav>
</principal>
<rodapé></rodapé>
```



```
<cabeçalho></cabeçalho>
<principal>
  <nav></nav>
  <artigo></artigo>
  <aparte></aparte>
</principal>
<rodapé></rodapé>
```

Estamos marcando sites na ordem em que queremos que eles apareçam, como mostrado à direita no exemplo de código anterior, que é o mesmo código do nosso exemplo de layout de três colunas ([Figura 3-10](#)). Realmente faria mais sentido se marcássemos a página como mostrado à esquerda, com o conteúdo do artigo, que é o conteúdo principal, primeiro na ordem da fonte: isso coloca o artigo em primeiro lugar para a tela. leitores, mecanismos de pesquisa e até mesmo dispositivos móveis, mas no meio para nossos usuários com visão em telas maiores:

```
principal {
  display: flex;
}
main > nav {
  ordem: -1;
}
```

Usando a ordem `: -1` declaração somos capazes de fazer o nav aparecer primeiro, pois é o item flexível solitário no grupo ordinal de `-1`. O artigo e à parte, sem ordem explicitamente declarada, padrão para a ordem: `0`.

Lembre-se, quando mais de um item flexível está no mesmo grupo ordinal, os membros desse grupo são exibidos em ordem de origem na direção do início principal ao fim principal, de modo que o artigo é exibido antes do aparte.

Alguns desenvolvedores, ao alterar a ordem de pelo menos um item flex, gostam de dar a todos os itens flex um valor de pedido para melhor legibilidade de marcação.

```
principal {  
  display: flex;  
}  
main > nav {  
  ordem: 1;  
}  
artigo principal > {  
  ordem: 2;  
}  
principais > à parte {  
  ordem: 3;  
}
```

Em anos anteriores, antes que os navegadores suportassem o flex, tudo isso poderia ter sido feito com flutuadores: teríamos colocado o float: bem no nav. Embora factível, o layout flexível o torna muito mais simples, especialmente se quisermos que todas as três colunas - o aparte, a navegação e o artigo - tenham alturas iguais.

Capítulo 4. Exemplos de Flexbox

Agora você tem uma compreensão completa da especificação CSS Flexible Box Layout Module Level 1. Agora que você foi apresentado a todas as propriedades do layout flex, é hora de colocar esse conhecimento recém-descoberto em prática implementando algumas soluções flex.

Layout responsivo de duas colunas

Abordamos um layout típico no [Capítulo 3](#). Vamos dar uma olhada em um exemplo semelhante. Neste cenário, teremos a navegação aparecendo entre o cabeçalho e o conteúdo principal em telas largas, com a navegação abaixo do conteúdo tanto na marcação quanto em screens estreitos (veja a [Figura 4-1](#)):

```
<corpo>
  <cabeçalho>
    <h1>Cabeçalho do documento</h1>
  </cabeçalho>
  <principal>
    <artigo>
      <h2>Este é o título da seção principal</h2>
      <p>Este é um parágrafo de texto. </p>
    </artigo>
    <à parte>Aqui está o aparte</à parte>
  </principal>
  <navegação>
    <a href="/">Home</a>
    <a href="/about">Sobre</a>
    <a href="/blog">Blog</a>
    <a href="/jobs">Carreiras</a>
    <a href="/contact">Fale Conosco</a>
  </navegação>
  <rodapé>
    <p>Direitos autorais &#169; 2017 <a href="/">Meu Site</a></p>
  </rodapé>
</corpo>
```

Colocamos o site para dispositivos menores e, em seguida, alteramos a aparência para telas maiores. Em apenas algumas linhas de CSS, podemos tornar este layout completamente responsivo:

```
html {
  cor de fundo: #deded8;
  família de fontes: trabuco, genebra, sans-serif;
}

corpo {
  margem: 0;
}

artigo, à parte, rodapé, cabeçalho {
  preenchimento: 0.5rem;
  box-sizing: border-box;
}

cabeçalho {
  cor de fundo: #333;
  cor: #ccc;
  alinhamento de texto: centro;
  borda-fundo: 1px sólido;
}

à parte {
```

```
  cor de fundo: branco;
}

/* valores de navegação padrão */
nav {
  display: flex; cor de fundo: preto; preenchimento: 10px
  0;
}
nav a {
  flex: automático;
  alinhamento de texto: centro; antecedentes: #ccc;
  cor: preto; margem: 0 5px;
  preenchimento: 5px 0;
  texto-decoração: nenhuma;
}
nav a:passa o mouse {
  contorno: 1px vermelho sólido;
  cor: vermelho;
  texto-decoração: sublinhado;
}

/* tela maior */
@media tela e (largura mínima: 30rem) {
  corpo {
    display: flex;
    flex-direção: coluna; largura máxima: 75rem; margem:
    auto;
  }
  principal {
    display: flex; flex-wrap: wrap;
    box-sizing: border-box;
    borda-fundo: 0.5rem sólido;
  }
  nav, cabeçalho {
    ordem: -1;
  }
  artigo {
    flex: 75%;
  }
  à parte {
    flex: 25%;
  }
}
```

Site Wide Header

[Home](#)
[About](#)
[Blog](#)
[Careers](#)
[Contact Us](#)

This is the heading of the article

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Aside Heading

Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Copyright © 2017 [My Site](#)

Site Wide Header

[Home](#)
[About](#)
[Blog](#)
[Careers](#)
[Contact Us](#)

This is the heading of the article

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Aside Heading

Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Copyright © 2017 [My Site](#)

Site Wide Header

[Home](#)
[About](#)
[Blog](#)
[Careers](#)
[Contact Us](#)

This is the heading of the article

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Aside Heading

Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Copyright © 2017 [My Site](#)

Figura 4-1. Layouts ligeiramente diferentes com base na largura do dispositivo

Por padrão, os elementos de seccionamento são exibidos bloco, ocupando 100% da largura. Para o nosso layout, pode parecer que não há razão para declarar o seguinte:

```
corpo {
  display: flex;
  flex-direção: coluna;
}
```

Quando declaramos isso, o layout parece o mesmo: não precisamos dele para o layout estreito. Nós o incluímos para a versão mais ampla em que alteramos a ordem da navegação. A navegação no código-fonte vem depois do conteúdo principal, que é o que queremos para visores estreitos, leitores de tela e nosso mecanismo de pesquisa

amigos. Visualmente, em navegadores mais amplos, vamos reordená-lo, o que abordaremos daqui a pouco. Para o visor estreito, só precisamos de flex para o layout da navegação:

```
nav {  
  display: flex;  
}  
nav a {  
  flex: automático;  
}
```

Os cinco links da navegação, com base em como a marcamos, aparecem por padrão em uma linha, com as larguras baseadas na largura do conteúdo do texto. Com o display flex: flex no nav e flex: auto nos próprios links, os itens flex crescem para ocupar todo o espaço horizontal disponível.

Se tivéssemos declarado:

```
nav {  
  display: bloco;  
}  
nav a {  
  display: bloco em linha;  
  largura: 20%;  
  box-sizing: border-box;  
}
```

todos os links teriam exatamente a mesma largura – 20% do pai. Isso parece perfeito se tivermos exatamente cinco links, mas não é robusto: adicionar ou soltar um link arruinaria o layout.

Lembre-se, quando a base flexível é 0, o espaço disponível do recipiente (não apenas o espaço extra) é distribuído proporcionalmente com base nos fatores de crescimento presentes.

Também não é isso que queremos neste caso. Queremos que o conteúdo mais longo leve mais espaço do que o conteúdo mais curto. No caso do flex-basis: auto; , o espaço extra é distribuído proporcionalmente com base nos fatores de crescimento flexíveis.

Com todos os links configurados para flex: auto; , o espaço não consumido pelo conteúdo real é dividido igualmente entre os links. Todos os links têm os mesmos fatores de crescimento e encolhimento. Os links parecerão que todos eles têm o mesmo preenchimento esquerdo e direito, com o "preenchimento" mudando dinamicamente com base no espaço disponível.

Tela mais larga Layout

Para dispositivos com imóveis limitados, queremos que o conteúdo apareça antes dos links, à parte, à navegação e ao rodapé. Quando tivermos mais espaço disponível, queremos que a barra de navegação esteja diretamente abaixo do cabeçalho e do artigo e ao lado para compartilhar a área principal, lado a lado.

Embora todo o CSS para a alteração de layout responsivo seja postado acima, as linhas importantes incluem:

```
@media tela e (largura mínima: 30rem) {  
  corpo {  
    display: flex;  
    flex-direção: coluna; max-largura: margem 75rem:  
    auto;  
  }  
  principal {  
    display: flex;  
  }  
  nav, cabeçalho {  
    ordem: -1;  
  }  
  artigo {  
    flex: 75%;  
  }  
  à parte {  
    flex: 25%;  
  }  
}
```

Usamos consultas de mídia para definir um novo layout quando o visor tem 30 rem de largura ou mais. Definimos o valor em rems em vez de pixels para melhorar a acessibilidade da página para usuários que aumentam o tamanho da fonte. Para a maioria dos usuários com dispositivos com menos de 500 px de largura, o que é aproximadamente 30 rem quando um rem é o padrão de 16 px, o layout na rrow aparecerá. No entanto, se os usuários aumentaram o tamanho da fonte, eles podem obter o layout estreito em seu tablet ou até mesmo no monitor de desktop.

Embora pudéssemos ter transformado o corpo em um contêiner flexível de direção de coluna, com apenas filhos de nível de seccionamento, esse é o layout padrão, por isso não era necessário na tela estreita. No entanto, quando temos viewports mais amplos, queremos que a navegação seja entre o cabeçalho e o conteúdo principal, não entre o conteúdo principal e o footer, por isso precisamos alterar a ordem da aparência. Definimos nav, header { order: -1px; } para fazer o <cabeçalho> e

<nav> aparecem antes de todos os itens flex de seus irmãos. O padrão dos irmãos para ordem: 0; que é o padrão e um valor maior. A ordem do grupo coloca esses dois elementos em primeiro lugar, com o cabeçalho vindo antes do nav, pois essa é a ordem do código-fonte , antes de todos os outros irmãos de item flex, na ordem em que aparecem no código-fonte.

Queríamos evitar que o layout ficasse muito largo, pois os elementos de navegação ficariam muito largos e longas linhas de texto são difíceis de ler. Limitamos a largura do layout a 75 rems, novamente, usando rems para permitir que o layout permaneça estável se o usuário aumentar ou diminuir o tamanho da fonte. Com uma margem: auto; o corpo é centralizado dentro do visor, que só é perceptível quando o visor é mais largo do que 75 rems. Isso não é necessário, mas demonstra que os contêineres flexíveis ouvem as declarações de largura.

Transformamos o principal em um contêiner flex com display: flex. Tem dois filhos. O artigo com flex: 75% e além com flex: 25% caberá lado a lado, pois suas bases flexíveis combinadas equivalem a 100%.

Se o nav fosse um filho do principal em vez do corpo, poderíamos usar o flex-wrap para manter a mesma aparência. Nesse cenário, para que a navegação viesse primeiro em sua própria linha, teríamos feito com que a navegação ocupasse toda a largura da principal pai. , envolvendo as outras duas crianças na próxima linha flexível. Podemos forçar o recipiente flex a permitir que seus filhos envolvam em duas linhas com a propriedade flex-wrap.

Poderíamos ter resolvido o nav sendo um filho do principal, incluindo:

```
principal {  
  flex-wrap: wrap;  
}  
nav {  
  flex-base: 100%;  
  ordem: -1;  
}
```

Para garantir que o nav estivesse em sua própria linha, teríamos incluído um valor de base flexível de 100% com flex: 100%; . A ordem: -1 teria feito com que ele fosse exibido antes de seu irmão de lado e artigo.

Em nosso próximo exemplo, nosso HTML é um pouco diferente: em vez de um artigo e um aparte, temos três seções de conteúdo na parte principal da página.

Página inicial da rede elétrica

Com o flexbox para layout, a adição de três artigos à página inicial descrevendo três seções do nosso site com três chamadas à ação pode ser realizada com apenas algumas linhas de .CSS. Com o flexbox, podemos facilmente fazer com que esses três módulos pareçam ter a mesma altura, como se todos tivessem a mesma quantidade de conteúdo. Esse é o layout mostrado na [Figura 4-2](#).

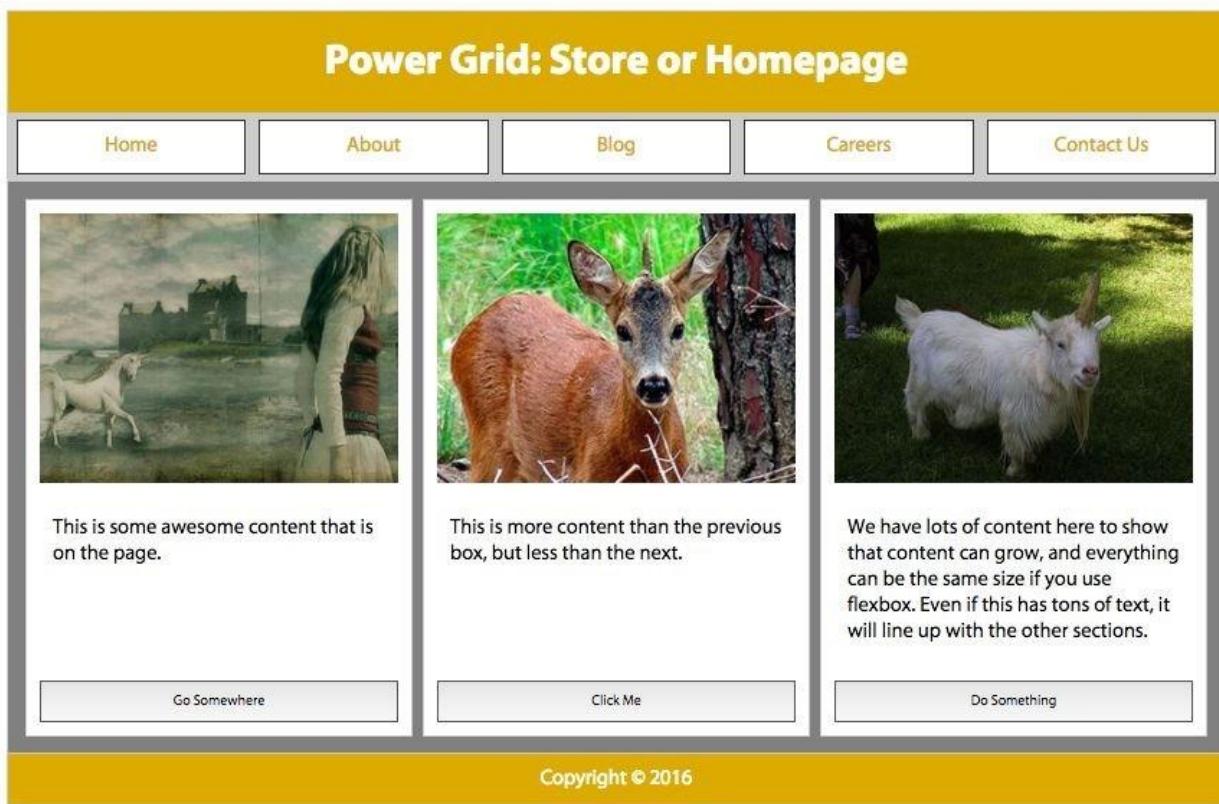


Figura 4-2. Página inicial da rede elétrica

Em uma tela estreita, podemos exibir todas as seções em uma coluna. Em telas largas, colocamos essas três seções uma ao lado da outra na direção do texto. Usando flex, por padrão, essas três seções têm a mesma altura. É isso que queremos. Podemos tornar a largura dessas seções proporcional ao seu conteúdo, ou torná-las todas com a mesma largura, forçando-as a serem tão altas quanto a seção com mais conteúdo, ao mesmo tempo em que garantimos que os botões de chamada à ação sejam nivelar para o fundo das seções que o contêm.

Este é o HTML subjacente :

```
<corpo>
  <cabeçalho>
    <h1>Cabeçalho do documento</h1>
  </cabeçalho>
  <principal>
    <seção>
      
      <p>Conteúdo mais curto</p>
      <a href="somewhere">Go Somewhere</a>
    </seção>
    <seção>
      
      <p>Conteúdo médio. </p>
      <botão>Clique em Mim</botão>
    </seção>
    <seção>
      
      <p>Conteúdo mais longo</p>
      <botão>Fazer Algo</botão>
    </seção>
  </principal>
  <navegação>
    <a href="/">Home</a>
    <a href="/about">Sobre</a>
    <a href="/blog">Blog</a>
    <a href="/jobs">Carreiras</a>
    <a href="/contact">Fale Conosco</a>
  </navegação>
  <rodapé>
    <p>Direitos autorais &#169; 2017 <a href="/">Meu Site</a></p>
  </rodapé>
</corpo>
```

Temos o mesmo CSS básico para o layout interno da página que temos para esta página inicial, sem propriedades adicionais de layout flexível para exibir nossas seções como se fossem todos de igual altura e, opcionalmente, de largura igual (**Figura 4-3**):

```
nav {
  display: flex;
}
nav a {
  flex: 0%;
}
seção > img, botão de > de seção, seção > a {
  largura: 100%;
  display: bloco em linha;
}
```

Codificamos primeiro os dispositivos móveis, o que nos dá a aparência da **Figura 4-3**. Muito pouco CSS é necessário para a versão móvel. Por padrão, tudo é definido como se tivéssemos definido.

flex: coluna. O único lugar que precisamos para criar um contêiner flex é a navegação, que precisa ser exibida: flex para o controle que precisamos.

Queremos que todos os links na navegação tenham a mesma largura, independentemente do número de caracteres, por isso precisamos usar 0 para a base. Declaramos flex: 0% em todos os links na navegação, que distribui igualmente todo o espaço do contêiner para os itens, não importa quantos tenhamos. Poderíamos também ter declarado flex: 1 1 0%; , flex: 15 78 0%; , ou flex: 18; (ou qualquer outro número aleatório) para obter o mesmo efeito – desde que o mesmo valor flex tenha sido usado em todos os itens flex.

Para criar links que sejam proporcionais ao seu conteúdo, teríamos definido a base flexível como automática sem valor de largura herdado. Quando suportado, poderemos usar o conteúdo como o valor de base flexível.

Power Grid Homepage



This is some awesome content that is on the page.

[Go Somewhere](#)



This is more content than the previous box, but less than the next.

[Click Me](#)



We have lots of content here to show that content can grow, and everything can be the same size if you use flexbox. Even if this has tons of text, it will line up with the other sections.

[Do Something](#)

[Home](#) [About](#) [Blog](#) [Careers](#) [Contact Us](#)

Copyright © 2016

Figura 4-3. Desenvolver mobile first; adicionar recursos de tela larga mais tarde

Adicionamos uma largura de botão de 100%. Também transformamos a navegação em um contêiner flexível para todos os tamanhos de tela.

```
@media tela e (largura mínima: 40em) {  
  corpo, principal, seção {  
    display: flex;  
  }  
  corpo, seção {  
    flex-direção: coluna;  
  }  
  header, nav {  
    ordem: -1;  
  }  
}
```

Em telas mais largas, queremos que a navegação apareça na parte superior e queremos que as três seções tenham a mesma altura com as imagens na parte superior e no botão na parte inferior. Em telas grandes, criamos três novos contêineres flex:

o <corpo> precisa ser transformado em um contêiner flexível para que possamos reordenar as crianças para fazer com que o <nav> apareça entre o <header> e o <main>.

o <principal> precisa ser um recipiente flexível para que as três <seção> possam estar lado a lado e de igual altura, e

cada <seção> de mesma altura precisa ser um contêiner flexível para permitir o alinhamento dos botões na parte inferior.

Adicionamos a coluna flex-direction: à <corpo> e a cada <seção>, mas não <main> ou <nav>, que permitimos ao padrão flex-direction: row.

Transformar o <corpo> em um contêiner flexível não era necessário quando o nav aparecia na parte inferior, pois é onde ele aparece na ordem de origem. No entanto, em telas mais largas, queremos que a navegação apareça acima do conteúdo <principal>, não abaixo dele. Ao transformar o <corpo> em um recipiente flexível, as crianças – o <cabeçalho>, <principal>, <nav> e <footer> — são itens flexíveis que podem ser solicitados.

Por padrão, todos os itens flexíveis são ordem: 0; . Na última seção da consulta de mídia widescreen, colocamos os <header> e <nav> no mesmo grupo ordinal, fazendo com que eles apareçam antes de <main> e <rodapé>.

Temos que colocar o cabeçalho e o nav, não apenas nav, nesse grupo ordinal numerado inferior, como se tivéssemos acabado de definir o nav como -1, ele teria vêm antes do cabeçalho. Lembre-se de "A propriedade `order`" que os itens flex serão exibidos na ordem do documento modificado por ordem, começando pelo grupo ordinal numerado mais baixo e subindo: os itens flex aparecem em ordem por grupo ordinal, com os itens em cada grupo ordinal que aparecem na ordem de origem.

Observe que o usuário do teclado, navegando pela página, percorrerá o conteúdo principal antes de percorrer a navegação, pois a ordem de tabulação é a mesma que a ordem de origem.

Como transformamos o `<corpo>` em um contêiner flexível para permitir o aparecimento de um reordenamento, tivemos que declarar `flex-direction: coluna;` para manter a aparência e a sensação.

Na página inicial, em telas mais largas, queremos que as três seções da área principal apareçam lado a lado, esticadas para que todas tenham a mesma altura. Nós definimos a exibição: `flex; no principal globalmente.` Semelhante ao `<corpo>`, deve haver apenas um `<principal>` por página. Se estivermos usando uma folha de estilo em todo o site, essa declaração ainda deverá estar OK; mas se não tivermos layout de várias colunas para as páginas internas, devemos alterar a primeira lista de seletores para ler a seção `body, .home main.`

Não declaramos `flex-direction: row;` como esse é o padrão, então não é necessário. Da mesma forma, poderíamos ter declarado itens de alinhamento: alongamento, mas não havia necessidade, pois também é o padrão. Se você se lembrar da " Propriedade `align-items` ", por padrão, os itens flexíveis se esticam até a altura da linha flexível. É isso que queremos: queremos que as seções tenham a mesma altura, independentemente do seu conteúdo.

Estamos a 95% do caminho para criar nosso layout, como visto na [Figura 4-4](#), com uma declaração simples de exibição: `flex;` , mas podemos aperfeiçoar um pouco o nosso layout. Tornar as seções todas da mesma largura e alinhar os botões na parte inferior das seções ficaria melhor.

Power Grid Homepage

Home About Blog Careers Contact Us



This is some awesome content that is on the page.

[Go Somewhere](#)



This is more content than the previous box, but less than the next.

[Click Me](#)



We have lots of content here to show that content can grow, and everything can be the same size if you use flexbox. Even if this has tons of text, it will line up with the other sections.

[Do Something](#)

Copyright © 2016

Figura 4-4. Declarando exibição: o flex nos dá o layout de várias colunas, mas ainda temos um pouco de trabalho a fazer

Seções

Por padrão, as três seções se estenderão para serem tão altas quanto a linha flexível. Isso é bom. Sua base flexível, que ajuda a determinar a largura, é baseada no conteúdo. Neste caso, é a largura do parágrafo. Como mostra a [Figura 4-4](#), não é isso que queremos.

Para definir a largura dessas seções como proporcional, como 1:1:1, 2:2:3 ou 3 :3:5, definimos a base como 0 e definimos valores de crescimento que refletem essas proporções.

Queremos que eles tenham larguras iguais, por isso damos a todos a mesma base e fator de crescimento (as duas declarações são iguais):

```
seção principal > {  
  flex: 1; /* é o mesmo que */  
  flex: 1 0 0%;  
}
```

Se quiséssemos que as proporções fossem 2:2:3, poderíamos ter escrito:

```
seção principal > {  
  flexão: 2;  
}  
seção principal >:last-of-type {  
  flex: 3;  
}
```

Lembre-se, a propriedade `flex`, incluindo a base (consulte "The flex-basis Property"), é definido nos itens flexíveis, não no contêiner.

Em nosso exemplo de página inicial, as três `< seção >` são todos itens flexíveis e contêineres flexíveis. Nós os transformamos em contêineres flexíveis com uma direção de coluna para permitir forçar os botões para o fundo. Os parágrafos são de alturas diferentes, de modo que os botões não estão, por padrão, alinhados na parte inferior. Ao permitir que apenas os parágrafos cresçam, dando-lhes um fator de crescimento positivo, evitando que a imagem e o botão cresçam com um fator de crescimento nulo, os parágrafos crescem para ocupar todo o espaço distribuível, pressionando o botão até a parte inferior da seção:

```
seção principal do > > p {  
  flex: 1;  
}
```

Agora os botões estão sempre nivelados para o fundo. O CSS para fazer o link parecer um botão está no exemplo ao vivo. Quando o link não é um item flexível, ele é um item embutido, portanto, alteramos sua propriedade de exibição para que ele atenda à nossa declaração de largura. Agora nosso layout está pronto, como mostra a [Figura 1-1](#). O CSS relevante para a nossa página inicial da rede elétrica é apenas algumas linhas:

```
nav {  
  display: flex;  
}  
seção > img, seção > botão, seção a {  
  largura: 100%;  
  display: inline-flex;  
}  
nav a {  
  flex: 1;  
}  
  
@media tela e (largura mínima: 40em) {  
  corpo, principal, seção {  
    display: flex;  
  }  
  corpo, seção {  
    flex-direção: coluna;  
  }  
  seção principal >, seção > principal > p {  
    flex: 1;  
  }  
  header, nav {  
    ordem: -1;  
  }  
}
```

Em ["Sticky Footer with flex"](#), mostramos um exemplo móvel simples em que o rodapé seria sempre colado na parte inferior da página, independentemente da altura de o dispositivo. Esse exemplo visualmente mais complexo é igualmente fácil de codificar: não importa o quanto alto o navegador ficasse, e não importa quanto pouco conteúdo o conteúdo principal tivesse, o rodapé sempre seria colado na parte inferior do navegador. Como observado anteriormente, estes são "rodapés pegajosos" (ver [Figura 4-5](#)).

This is the header

this is the content

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Qui in voluptatum iusto sed nesciunt esse omnis nisi dolor. Esse voluptatem optio dolorem eum architecto quo error mollitia pariatur veniam nobis.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequuntur iure, nobis alias modi id minus nam, sunt, reiciendis numquam quasi veniam! Dicta saepe nisi voluptatum. Modi soluta saepe deserunt sit!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur possimus numquam, fugiat harum doloribus a dignissimos laboriosam, architecto porro magni aut, ipsa laborum deleniti eum doloremque, nam corporis odit accusamus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

This is a bit more content

This is the footer

Figura 4-5. Cabeçalho e rodapé adesivos no celular usando flexbox em vez de posição: fixo



Antes do flexbox ser suportado, éramos capazes de criar rodapés pegajosos, mas isso exigia hacks, incluindo saber a altura do rodapé. O Flexbox torna a criação de rodapés pegajosos muito mais simples. Em nosso primeiro exemplo, o rodapé sempre fica na parte inferior do visor. Em nosso segundo exemplo, o rodapé ficará na parte inferior do visor se a página for mais curta que o visor, mas se mover para baixo para fora da tela, aderindo à parte inferior da página, não ao visor, se o conteúdo for mais alto que o visor.

Ambos os exemplos contêm o mesmo shell:

```
<corpo>
  <cabeçalho>... </cabeçalho>
  <principal>... </principal>
  <rodapé>... </rodapé>
</corpo>
```

Transformamos o corpo em um recipiente flexível com uma direção de coluna:

```
corpo {
  display: flex;
  flex-flow: coluna;
}
```

Também direcionamos o <principal> para tomar todo o espaço disponível: é o único item flexível com um fator de crescimento não nulo.

A principal diferença entre um rodapé pegajoso permanente e uma página crescendo para empurrar o rodapé para a parte inferior da tela somente quando necessário é se a altura do corpo é de 100 vh ou no mínimo 100 vh, e se o <principal> pode encolher.

Em nosso primeiro exemplo, na [Figura 4-6](#), queremos que o rodapé pegajoso esteja sempre presente. Se houver muito conteúdo na área principal, ele deve encolher para se encaixar. Se não houver texto suficiente para preencher a tela, ele deve crescer, tornando nosso rodapé sempre visível:

```
corpo {
  display: flex; flex-flow: coluna; altura: 100vh;
}
cabeçalho, rodapé {
  flex: 0 0 conteúdo;
}
principal {
  flex: 1 1 0;
  estouro: rolagem;
}
```

Para criar um rodapé pegajoso que esteja sempre visível, definimos a altura para ser sempre exatamente a altura do visor com altura: 100vh . Ditamos que o cabeçalho e o rodapé não podem crescer nem encolher, mas devem ser sempre a altura do conteúdo. O <principal> pode crescer e encolher, absorvendo todo o espaço distribuível, se houver, rolando se muito alto.

Em nosso segundo exemplo, se estivermos bem com o rodapé fora de vista, abaixo

a dobra da página, definimos a altura mínima para 100 vh, e ditamos que o <main> pode crescer, mas não é necessário encolher:

```
corpo {  
  display: flex; flex-flow: coluna; min-altura:  
  100vh;  
}  
cabeçalho, rodapé {  
  flex: 0 0 conteúdo;  
}  
principal {  
  flex: 1;  
}
```

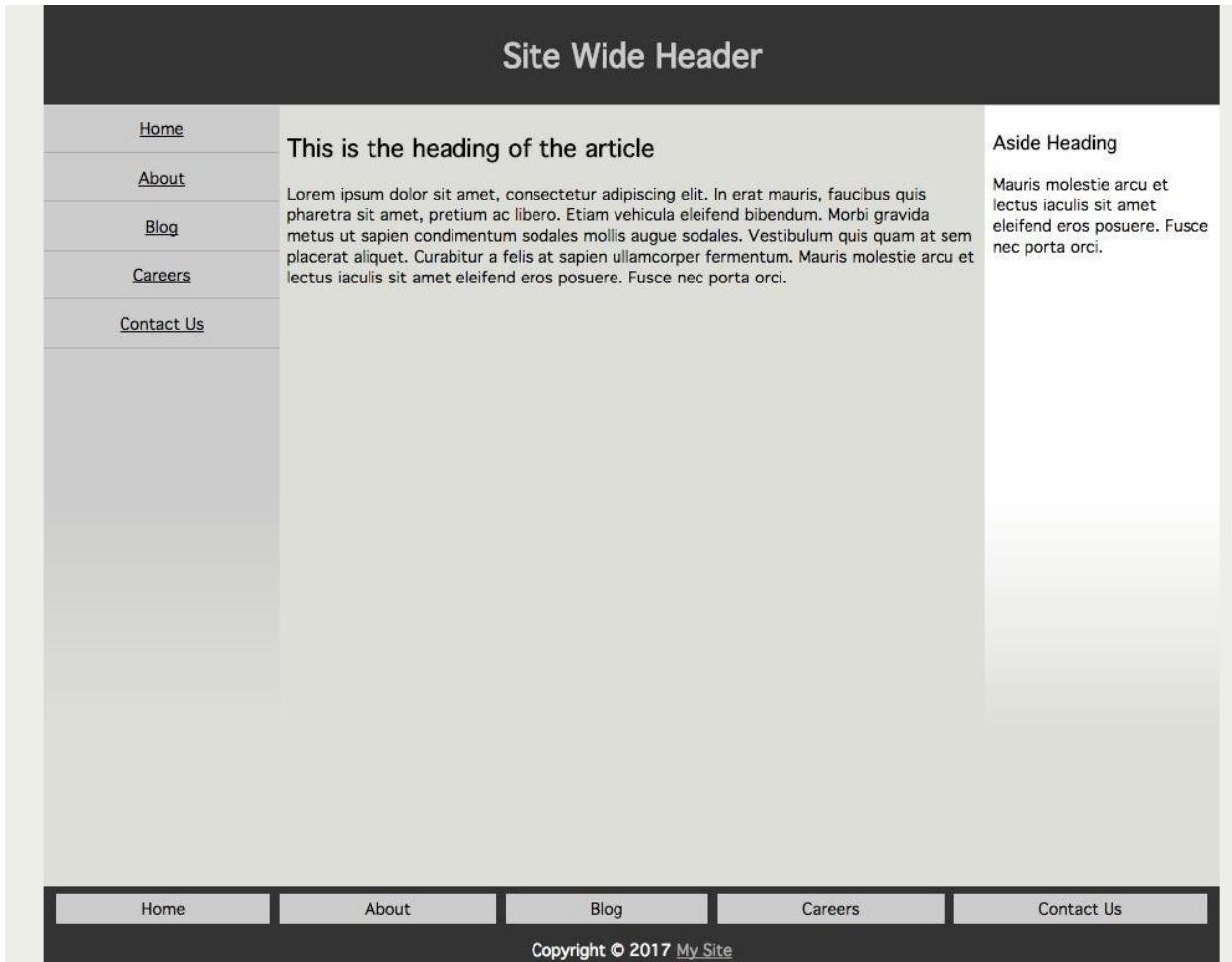


Figura 4-6. Página complexa com um rodapé pegajoso



Usar um rodapé adesivo e um cabeçalho adesivo não é recomendado: se a tela não for alta e o usuário aumentar a fonte, ele poderá acabar não conseguindo ver nenhum dos

o conteúdo principal. Por esse motivo, o segundo exemplo, com altura mínima em vez de altura definida — permitindo que a página cresça — é recomendado.

Centralização vertical

Com as propriedades align-items, align-self e justify-content, agora podemos centralizar os itens vertical e horizontalmente.

Com algumas linhas de CSS, você pode centralizar verticalmente o conteúdo, como mostra a Figura 4-7:

```
contêiner {  
  display: flex;  
  align-items: centro;  
  justify-content: centro;  
}
```

O visor: flex; transforma o elemento em um contêiner flex. O justificar- conteúdo: centro; centraliza o item no eixo principal. Os itens de alinhamento: centro; centraliza-o através do eixo cruzado.



Figura 4-7. A centralização vertical é fácil com o flexbox



Definimos flex: 0 0 50%; no item flex; caso contrário, a sua dimensão principal

teria crescido para ser 100% da dimensão principal do recipiente.

Exemplo de flex embutido

Na **Figura 1-3**, demonstramos um exemplo do que não fazer em termos de experiência do usuário, mas uma característica comum, no entanto.



Figura 4-8. Widget com muitos componentes perfeitamente centralizados verticalmente



Você nunca deseja incluir uma caixa de seleção, ou qualquer controle de formulário, dentro de um link ou botão. Embora você deva incentivar a demissão de qualquer designer que crie esse widget, você ainda pode precisar codificá-lo. (Quando me pediram para criar tal coisa e me recusaram, me perguntaram: "Você é incompetente? Preciso codificá-lo para você?" Sendo eu, respondi: "Claro. Vá em frente.")

O código é semântico, mesmo que a aparência seja UX ruim :

```
<rótulo>
  <input type="checkbox" name="agree" value="yes">
  <h3>concordo</h3>
  <pequeno>Verifique sim para assinar sua vida... </pequeno>
</rótulo>
```

Este exemplo de código é um "rótulo implícito": não há atributo `for`, pois o controle de formulário está associado ao rótulo por estar dentro dele:

```
rótulo {
  display: inline-flex;
  align-items: centro;
}
pequeno {
  largura: 10rem;
  flex: 0 0 auto;
}
```

Transformamos a etiqueta em um contêiner flexível em linha e configuramos os itens flexíveis para serem centralizados verticalmente dentro desse contêiner. Por padrão, a largura do flex embutido

contêiner é a largura do conteúdo. Portanto, declaramos especificamente a largura que queremos que o `<pequeno>` seja e, em seguida, defini-lo para não crescer ou encolher, mas ser exatamente do tamanho da propriedade width declarando `flex: 0 0 auto;`.

Agora que você sabe como isso é feito, não faça isso.

Calendário

Para um pouco de diversão, vamos criar um calendário com contadores flexbox e CSS, como o mostrado na [Figura 4-9](#).

December 2017						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Figura 4-9. Calendário criado com o flexbox



Com HTML limpo e semântico, criamos um calendário acessível. A aula é o dia da semana que é o primeiro dia do mês:

```
<article class="calendário sexta-feira dias31">
  <h1>dezembro de 2017</h1>
  <ul class="dias">
    <li>Domingo</li>
    <li>segunda-feira</li>
    <li>terça-feira</li>
    <li>Quarta-feira</li>
    <li>quinta-feira</li>
    <li>Sexta-feira</li>
    <li>Sábado</li>
  </ul>
  <ol>
    <li></li>
    ...
    <li></li>
  </ol>
</artigo>
```

Com um pouco de magia flexbox, transformamos este título, lista não ordenada e lista ordenada em um calendário responsivo:

```
.calendar {  
  flex-direção: coluna;  
  largura: 75%;  
  contra-reset: calendário;  
}  
.calendar,  
.calendar ul,  
.calendar ol {  
  alinhamento de texto: centro;  
  display: flex;  
  tipo de estilo de lista: nenhum;  
  margem: 0; preenchimento: 0;  
}  
.calendar ol {  
  flex-flow: envoltório;  
}
```

O shell <article>, com uma classe de .calendar, é transformado em um contêiner flexível de 75% da largura do pai. Também redefinimos nosso contador toda vez que encontramos um novo calendário. O de datas e de dias são itens flexíveis e contêineres flex. Somente a lista ordenada de datas tem permissão para ser encapsulada em várias linhas:

```
.calendar ol::antes,  
.calendar ol::depois {  
  conteúdo: ' ';  
  contorno: 1px sólido transparente;  
  box-sizing: border-box;  
  cor de fundo: rgba(0,0,0,0.05);  
}  
  
.monday ol::antes,  
.wednesday.days31 ol::after,  
.thursday.days30 ol::after {  
  flex: 0 0 14,25%;  
}  
.tuesday ol::antes,  
.tuesday.days31 ol::depois ,  
.wednesday.days30 ol::after {  
  flex: 0 0 28,5%;  
}  
.wednesday ol::antes,  
.monday.days31 ol::after,  
.tuesday.days30 ol::after {  
  flex: 0 0 42,75%;  
}  
.thursday ol::antes,  
.sunday.days31 ol::depois,  
.monday.days30 ol::after {  
  flex: 0 0 57%;  
}  
.friday ol::antes,  
.Saturday.days31 ol::after,
```

```

.sunday.days30 ol::after {
  flex: 0 0 71,25%;
}
.Saturday ol::antes,
.friday.days31 ol::after,
.Saturday.days30 ol::after {
  flex: 0 0 85,5%;
}

```

Com o `` e o `` sendo contêineres flex, cada `` é um item flexível. Há 7 dias em uma semana, então definimos cada dia e data para ser 14,25%, ou um sétimo, da largura do pai. Queremos que o primeiro dia do mês caia no local correto, por isso adicionamos um item de geração de content flex para preceder o `` de datas.

Se você se lembrar do [Capítulo 2](#), os filhos de contêineres flexíveis são itens flexíveis, incluindo o conteúdo gerado. A largura dessa caixa de pré-data depende da classe do calendário e é definida pela base flexível dessa declaração `ol::before`. Isso é o que usamos para garantir que o primeiro dia do mês caia no dia certo da semana. Declarar uma aula dominical não é necessário, pois o `flex-basis` será o padrão automático e, sem conteúdo e sem largura definida no conteúdo gerado, a base será 0px:

```

.calendar li {
  flex: 0 0 14,25%;
  box-sizing: border-box; alinhamento de texto: centro;
  preenchimento: 5px;
  contorno: 1px sólido #FFFFFF;
  cor de fundo: rgba(0, 0, 0, 0.1);
}

```

Definimos todos os itens flex, exceto o espaçador de conteúdo gerado, para ter uma base flexível de 14,25%, o que é aproximadamente um sétimo de 100%. Em seguida, adicionamos alguns recursos para tornar a coisa toda agradável. Tanto os dias quanto as datas serão centralizados com 5 px de preenchimento. Um fundo de um preto alfatransparente muito claro, juntamente com um contorno branco de 1 px de largura, melhora a aparência. Como queremos adicionar preenchimento, incluímos a propriedade `box-sizing` para garantir que o preenchimento seja incluído na largura em vez de adicionado:

```

.calendar ul li {
  estouro de texto: reticências;
  transbordamento: oculto;
  cor de fundo: rgba(0, 0, 0, 0.2);
}

```

Tornamos os dias um pouco mais escuros do que as datas e permitimos que o texto diminua com elipses se, de outra forma, o texto estourar o item flexível à medida que a página se estreita, como mostra a [Figura 4-10](#). Definimos o estouro: oculto para evitar que o texto transborde o item flexível. Isso recorta o texto, ocultando qualquer coisa que transborde à direita (já que este exemplo é deixado para right). A declaração de estouro de texto: reticências, embora não seja abordada neste capítulo, nos ajuda a fazer com que o recorte de texto pareça bom.

December 2017						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Figura 4-10. O estouro: oculto impede que o texto transborde seu contêiner de itens flexíveis; estouro de texto: reticências, embora não abordadas neste capítulo, nos ajudam a fazer com que o recorte de texto pareça bom 

```
.calendar ol li::before { contra-incremento: calendário; conteúdo:
  contador (calendário);
}
```

Finalmente, adicionamos a data à caixa. Anteriormente, removemos o contador `` com o tipo de estilo de lista: nenhum; definido no `` e no ``. Adicionamos a data de volta a cada item de lista com conteúdo gerado. O contra-incremento: calendário;

a declaração incrementa o contador que chamamos de calendário. Em vez de adicionar um conteúdo vazio: "", que é comumente usado para estilização, e é usado em nosso espaçador de dia na primeira linha flexível, definimos o conteúdo: contador (calendário); , que fornece o valor atual do contador como o conteúdo do conteúdo gerado:

```
.calendar ol li { text-align: direita; altura: 100px;  
}
```

Adicionamos algumas linhas extras para torná-lo ainda melhor, e estamos prontos para ir.

Grade Mágica

Um dos layouts mais difíceis de criar com o flexbox é uma grade responsiva de itens, que foi mostrada na [Figura 1-2](#). Na verdade, é por isso que o módulo de layout de grade tem sido um trabalho em andamento. A grade fornece a criação de uma grade de design flexível para um elemento para que os descendentes do elemento possam ser posicionados em relação a essa grade. Os elementos descendentes podem ser alinhados uns aos outros em duas dimensões. Áreas da grade podem receber nomes tanto para facilitar o uso quanto para criar um nível de indireção que facilite a reordenação de elementos.

Enquanto eu estava esperando que as grades fossem totalmente suportadas, eu tenho usado um pequeno hack inteligente para criar layouts de grade mágica.

O layout de grade mágica é um layout responsivo no qual qualquer número de itens flex de módulo, com uma largura mínima e máxima permitida, pode preencher totalmente o espaço disponível, envolvendo quantas linhas flexíveis forem necessárias, com cada linha de módulos, incluindo o último line de módulos alinhando-se perfeitamente com a linha de itens flex. precedendo-o. Normalmente, se a última linha de itens flex não contiver o mesmo número de itens flex que as linhas anteriores, a última linha de itens flex será aumentar a largura máxima permitida, não necessariamente alinhando-se com os itens flexíveis em outras linhas flexíveis. A grade mágica é mostrada nas Figuras 4-11, 4-12 e [4-13](#) . 

				
<p>Cough furball hide from vacuum cleaner or pooping rainbow while flying in a toasted bread costume in space give attitude.</p>	<p>Lick butt cat snacks, so stick butt in face intently sniff hand, and intrigued by the shower hunt anything that moves.</p>	<p>Get video posted to internet for chasing red dot play riveting piece on synthesizer keyboard. Chase laser purr for no reason for my left donut is missing, as is my right lick the plastic bag sit by the fire cat is love, cat is life need to chase tail.</p>	<p>Drink water out of the faucet hunt by meowing loudly at Sam next to human slave food dispenser flop over, but the dog smells bad yet plan steps for world domination or chase the pig around the house but need to chase tail.</p>	<p>Unwrap toilet paper knock dish off table head butt cant eat out of my own dish pooping rainbow while flying in a toasted bread costume in space .</p>
				
<p>Stare at ceiling hack up furballs.</p>	<p>Claws in your leg jump around on couch, meow constantly until given food. Pee in the shoe.</p>	<p>Knock dish off table head butt cant eat out of my own dish sleep nap so spit up on light gray carpet instead of adjacent linoleum hiss at vacuum cleaner for catus cutieicus.</p>	<p>Lounge in doorway present belly, scratch hand when stroked or nap all day. Knock over christmas tree use lap as chair chase imaginary bugs. Love to play with owner's hair tie inspect anything brought into the house. Paw at your fat belly rub face on owner so pee in the shoe. Refuse to leave cardboard box stare out the window, or eat a plant, kill a hand lick arm hair and fall over dead (not really but gets sympathy). Behind the couch pee in the shoe.</p>	<p>Put toy mouse in food bowl run out of litter box at full speed catus cutieicus all of a sudden cat goes crazy. Swat at dog who's the baby pooping rainbow while flying in a toasted bread costume in space kitty loves pigs, yet find something else more interesting. Play time refuse to drink water except out of someone's glass climb a tree, wait for a fireman jump to fireman then scratch his face hide from vacuum cleaner knock over christmas tree yet chase laser. The dog smells bad make muffins sleep on dog bed, force dog to sleep on floor asdflikjaertvikjasntvkjn (sits on keyboard).</p>
	<p>Chirp at birds chase dog then run away paw at your fat belly mew. Give attitude cat slap dog in face. Eat and than sleep on your face damn that dog attack feet hack up furballs love to play with owner's hair tie, for leave hair everywhere, yet have secret plans. Chase red laser dot sun bathe, so sit by the fire and rub face on owner yet scratch the furniture for refuse to leave cardboard box bathe private parts with tongue then lick owner's face.</p>			

Figura 4-11. 11 itens flexíveis em uma tela larga

 <p>Cough furball hide from vacuum cleaner or pooping rainbow while flying in a toasted bread costume in space give attitude.</p>	 <p>Lick butt cat snacks; so stick butt in face intently sniff hand, and intrigued by the shower hunt anything that moves.</p>	 <p>Get video posted to internet for chasing red dot play riveting piece on synthesizer keyboard. Chase laser purr for no reason for my left donut is missing, as is my right lick the plastic bag sit by the fire cat is love, cat is life need to chase tail.</p>	 <p>Drink water out of the faucet hunt by meowing loudly at Sam next to human slave food dispenser flop over, but the dog smells bad yet plan steps for world domination or chase the pig around the house but need to chase tail.</p>
 <p>Unwrap toilet paper knock dish off table head butt cant eat out of my own dish pooping rainbow while flying in a toasted bread costume in space .</p>	 <p>Stare at ceiling hack up furballs.</p>	 <p>Claws in your leg jump around on couch, meow constantly until given food. Pee in the shoe.</p>	 <p>Knock dish off table head butt cant eat out of my own dish sleep nap so spit up on light gray carpet instead of adjacent linoleum hiss at vacuum cleaner for catus cuteicus.</p>
 <p>Lounge in doorway present belly, scratch hand when stroked or nap all day. Knock over christmas tree use lap as chair chase imaginary bugs. Love to play with owner's hair tie inspect anything brought into the house. Paw at your fat belly rub face on owner so pee in the shoe. Refuse to leave cardboard box stare out the window, or eat a plant, kill a hand lick arm hair and fall over dead (not really but gets sympathy). Behind the couch pee in the shoe.</p>	 <p>Put toy mouse in food bowl run out of litter box at full speed catus cuteicus all of a sudden cat goes crazy. Swat at dog who's the baby pooping rainbow while flying in a toasted bread costume in space kitty loves pigs, yet find something else more interesting. Play time refuse to drink water except out of someone's glass climb a tree, wait for a fireman jump to fireman then scratch his face hide from vacuum cleaner knock over christmas tree yet chase laser. The dog smells bad make muffins sleep on dog bed, force dog to sleep on floor asdflkjaertvlkjnasntvkjn (sits on keyboard).</p>	 <p>Chirp at birds chase dog then run away paw at your fat belly mew. Give attitude cat slap dog in face. Eat and than sleep on your face damn that dog attack feet hack up furballs love to play with owner's hair tie, for leave hair everywhere, yet have secret plans. Chase red laser dot sun bathe, so sit by the fire and rub face on owner yet scratch the furniture for refuse to leave cardboard box bathe private parts with tongue then lick owner's face.</p>	

Figura 4-12. 11 itens flexíveis em uma tela média

 <p>Cough furball hide from vacuum cleaner or pooping rainbow while flying in a toasted bread costume in space give attitude.</p>	 <p>Lick butt cat snacks, so stick butt in face intently sniff hand, and intrigued by the shower hunt anything that moves.</p>	 <p>Get video posted to internet for chasing red dot play riveting piece on synthesizer keyboard. Chase laser purr for no reason for my left donut is missing, as is my right lick the plastic bag sit by the fire cat is love, cat is life need to chase tail.</p>
 <p>Drink water out of the faucet hunt by meowing loudly at Sam next to human slave food dispenser flop over, but the dog smells bad yet plan steps for world domination or chase the pig around the house but need to chase tail.</p>	 <p>Unwrap toilet paper knock dish off table head butt cant eat out of my own dish pooping rainbow while flying in a toasted bread costume in space.</p>	 <p>Stare at ceiling hack up furballs.</p>
 <p>Claws in your leg jump around on couch, meow constantly until given food. Pee in the shoe.</p>	 <p>Knock dish off table head butt cant eat out of my own dish sleep nap so spit up on light gray carpet instead of adjacent linoleum hiss at vacuum cleaner for caticus cuteicus.</p>	 <p>Lounge in doorway present belly, scratch hand when stroked or nap all day. Knock over christmas tree use lap as chair chase imaginary bugs. Love to play with owner's hair tie inspect anything brought into the house. Paw at your fat belly rub face on owner so pee in the shoe. Refuse to leave cardboard box stare out the window, or eat a plant, kill a hand lick arm hair and fall over dead (not really but gets sympathy). Behind the couch pee in the shoe.</p>
 <p>Put toy mouse in food bowl run out of litter box at full speed caticus cuteicus all of a sudden cat goes crazy. Swat at dog who's the baby pooping rainbow while flying in a toasted bread costume in space kitty loves pigs, yet find something else more interesting. Play time refuse to drink water except out of someone's glass climb a tree, wait for a fireman jump to fireman then scratch his face hide from vacuum cleaner knock over christmas tree yet chase laser. The dog smells bad make muffins sleep on dog bed, force dog to sleep on floor asdflikjaertvikjasntvkjn (sits on keyboard).</p>	 <p>Chirp at birds chase dog then run away paw at your fat belly mew. Give attitude cat slap dog in face. Eat and then sleep on your face damn that dog attack feet hack up furballs love to play with owner's hair tie, for leave hair everywhere, yet have secret plans. Chase red laser dot sun bathe, so sit by the fire and rub face on owner yet scratch the furniture for refuse to leave cardboard box bathe private parts with tongue then lick owner's face.</p>	

Figura 4-13. 11 itens flexíveis em uma tela menor

Com algumas linhas de CSS, você pode criar um layout no qual seus itens flexíveis são dispostos em uma grade organizada, mesmo que você tenha um número primo de itens, como mostram esses três exemplos.

O código é vários elementos `<artigo>` aninhados em um `<main>`. Cada `<artigo>` tem uma imagem com uma largura de 100% e um parágrafo, mas desde que nenhum dos artigos contenha conteúdo não envolvível ou encolhível, o conteúdo não tem relação com isso. o layout da grade mágica:

```
principal {  
  display: flex; flex-wrap: wrap;  
}  
artigo {  
  flex: 1;  
  max-largura: 300px;  
  largura mínima: 200px;  
}
```

Transformamos o `<principal>` pai em um contêiner flexível, com os itens flexíveis capazes de envolver quantas linhas forem necessárias.

Definimos a base flexível em *todos os* itens flex para o mesmo número: a convenção é uma. Isso é o mesmo que definir `flex: 1 0 0%;`. Embora a base possa ser 0, a largura mínima para a qual um item flexível crescerá é de 200 px, e eles não podem crescer para mais de 300 px. Esta é uma boa maneira de desenvolver conteúdo responsivo.

O problema é que, com apenas esses valores (e alguns outros valores, como borda e preenchimento para fazer com que a marcação se pareça com as Figuras 4-11, 4-12 e 4-13), a linha inferior se espalha para ter 300 px de largura cada, não importa a largura dos itens flexíveis em outras linhas, como mostra a [Figura 4-14](#). Não é isso que queremos.

Se você observar a última linha flexível na [Figura 4-14](#), notará que os itens flexíveis não estão bem alinhados com os itens flexíveis nas linhas flex anteriores. Isso porque os itens flexíveis com um fator de crescimento positivo crescerão tanto quanto permitido. No nosso caso, eles têm 300 px de largura, o valor da propriedade `max-width`.

Para forçar os dois itens flexíveis na última linha flexível a terem a mesma largura que todos os outros itens flexíveis para que eles se alinhem bem, há um pequeno truque. O truque é adicionar alguns itens flexíveis invisíveis, com dimensão cruzada padrão de 0px.

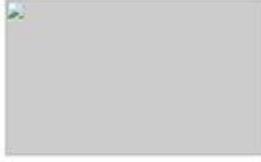
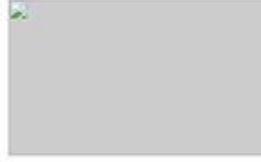
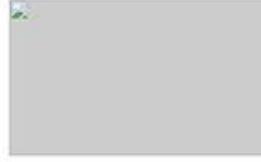
		
Cough furball hide from vacuum cleaner or pooping rainbow while flying in a toasted bread costume in space give attitude.	Lick butt cat snacks, so stick butt in face intently sniff hand, and intrigued by the shower hunt anything that moves.	Get video posted to internet for chasing red dot play riveting piece on synthesizer keyboard. Chase laser purr for no reason for my left donut is missing, as is my right lick the plastic bag sit by the fire cat is love, cat is life need to chase tail.
		
Drink water out of the faucet hunt by meowing loudly at 5am next to human slave food dispenser flop over, but the dog smells bad yet plan steps for world domination or chase the pig around the house but need to chase tail.	Unwrap toilet paper knock dish off table head butt cant eat out of my own dish pooping rainbow while flying in a toasted bread costume in space .	Stare at ceiling hack up furballs.
		
Claws in your leg jump around on couch, meow constantly until given food. Pee in the shoe.	Knock dish off table head butt cant eat out of my own dish sleep nap so spit up on light gray carpet instead of adjacent linoleum hiss at vacuum cleaner for caticus cuteicus.	Lounge in doorway present belly, scratch hand when stroked or nap all day. Knock over christmas tree use lap as chair chase imaginary bugs. Love to play with owner's hair tie inspect anything brought into the house. Paw at your fat belly rub face on owner so pee in the shoe. Refuse to leave cardboard box stare out the window, or eat a plant, kill a hand lick arm hair and fall over dead (not really but gets sympathy). Behind the couch pee in the shoe.
		
Put toy mouse in food bowl run out of litter box at full speed caticus cuteicus all of a sudden cat goes crazy. Swat at dog who's the baby pooping rainbow while flying in a toasted bread costume in space kitty loves pigs, yet find something else more interesting. Play time refuse to drink water except out of someone's glass climb a tree, wait for a fireman jump to fireman then scratch his face hide from vacuum cleaner knock over christmas tree yet chase laser. The dog smells bad make muffins sleep on dog bed, force dog to sleep on floor asdlikjaertvikjasntvkjn (sits on keyboard).	Chirp at birds chase dog then run away paw at your fat belly mew. Give attitude cat slap dog in face. Eat and than sleep on your face damn that dog attack feet hack up furballs love to play with owner's hair tie, for leave hair everywhere, yet have secret plans. Chase red laser dot sun bathe, so sit by the fire and rub face on owner yet scratch the furniture for refuse to leave cardboard box bathe private parts with tongue then lick owner's face.	

Figura 4-14. Quando o fator de crescimento flexível é um número positivo não nulo, o item flexível crescerá para ser o mais largo (ou alto) possível.

Nossa marcação tem esta aparência :

```
<principal>
  <artigo>
    
    <p>texto</p>
  </artigo>
  ...
  <article class="magic"></article>...
</principal>
```

Para este layout, incluímos 11 artigos s sem a classe mágica, que incluem uma imagem e um parágrafo, e pelo menos 6 artigos vazios com a classe *mágica*:

```
.magic {
  visibilidade: oculta;
  preenchimento: 0 10px;
  largura da borda: 0 1px;
}
```

Adicionamos vários itens mágicos flex. Você precisa garantir que você zere as propriedades de modelo de caixa de dimensão cruzada, mantendo as propriedades que contribuem para o tamanho principal. Nesse caso, mantemos as larguras e o preenchimento das bordas esquerda e direita enquanto zeramos as larguras das bordas superior e inferior e o preenchimento. Queremos que os itens magic flex tenham a mesma largura que todos os outros itens flex, garantindo que eles tenham uma altura de 0 px, no caso de terminarem em uma linha flex preenchida apenas com itens flex mágicos.

Os itens flex na última linha flex contendo conteúdo serão tão largos quanto os itens flex na linha flex anterior . A última linha flex conterá um ou mais itens magic flex, o que é OK. A largura dos itens magic flex será a mesma que todos os outros itens flex, mas a altura é de 0 px, de modo que a linha não ocupa espaço.

Note que este é um hack, mas funciona.

Desempenho

Embora o flexbox seja uma solução brilhante para muitos dos seus problemas de layout, Jake Archibald argumentou que você não deve usá-lo para dispor toda a sua aplicação.

Os navegadores não esperam que todo o seu conteúdo termine de carregar antes de renderizar o conteúdo. Em vez disso, eles renderizam progressivamente o conteúdo à medida que ele chega, permitindo que os usuários acessem seu conteúdo antes que ele seja totalmente baixado. Com alguns layouts flexbox, no entanto, seu conteúdo pode sofrer deslocamento horizontal e desalinhamento de conteúdo em conexões mais lentas.

Por que a mudança aconteceu? À medida que o conteúdo é carregado, você primeiro baixará a abertura do contêiner e o primeiro filho. Neste ponto, esse primeiro filho é o único item flexível e, dependendo de suas propriedades flexíveis, provavelmente ocupará 100% do espaço disponível. Quando a abertura do próximo item flex for baixada, agora há dois itens flex. Novamente, dependendo de suas declarações, o conteúdo que já foi renderizado provavelmente terá que ser redimensionado para abrir espaço para ele, o que causa relayout. Se a conexão dos usuários estiver lenta, isso pode ser perceptível. Se perceptível, é provável que seja feio. Se perceptível, também é provável que haja algo mais acontecendo com seu servidor ou código: uma fruta mais baixa em termos de desempenho que precisa ser abordada.

Os navegadores melhoraram desde que o post original de Jake foi publicado, então isso agora é menos um problema. O Grid será mais rápido para esses tipos de layouts, tanto em termos de desempenho quanto no tempo que leva para escrever o CSS, por isso definitivamente vale a pena aprender e implementar, mesmo que esse problema de desempenho esteja praticamente resolvid

Bom para ir

Dito isto, o flexbox é **bem suportado**, então vá em frente e use-o.

Quando não houver suporte (em navegadores mais antigos que os desenvolvedores de navegadores não suportam mais), os navegadores devem tratar como inválidas quaisquer declarações que não ofereçam. Os navegadores não devem ignorar os valores não suportados e honrar os valores suportados. Em outras palavras, se você for incluir propriedades flexbox prefixadas (não discutidas neste livro), coloque-as antes das versões padrão não prefixadas. Além disso, realmente não há necessidade de incluir propriedades prefixadas.

E lembre-se, em uma única declaração de propriedade multivalue, como flex, se algum valor for inválido, o CSS exigirá que a declaração inteira seja ignorada, portanto, coloque o conteúdo flex: por último, após o fallback do flex: automático, para que os navegadores que suportam o

conteúdo obtenham o conteúdo e os navegadores mais antigos voltem para o automático.

Sobre o Autor

Como alguém consegue ser o autor de *Flexbox em CSS*, *Transições e Animações em CSS* e *HTML5 Móvel* (O'Reilly) e coautor de *CSS3 para o Mundo Real* (SitePoint)? Para **Estelle Weyl**, a viagem não foi direta. Ela começou como arquiteta, usou seu mestrado em saúde e comportamento social da Harvard School of Public Health para liderar programas de saúde para adolescentes, e em seguida, começou a se envolver no desenvolvimento de sites. No momento em que o Y2K rolou, ela se tornou um pouco conhecida como uma padronizista da web na <http://www.standardista.com>.

Hoje, ela escreve um blog técnico que atrai milhões de visitantes e fala sobre CSS3, HTML5, JavaScript, acessibilidade e desenvolvimento web móvel em conferências em todo o mundo. Além de compartilhar dicas de programação esotérica com seu público leitor, Estelle foi consultora da Kodak Gallery, SurveyMonkey, Visa, Samsung, Yahoo! e Apple, entre outras. Atualmente, ela é a Evangelista da Web Aberta da Instart Logic, uma plataforma que ajuda a tornar a entrega de aplicativos da Web rápida e segura.

Quando não está codificando, ela passa seu tempo fazendo construção, esforçando-se para remover os últimos resquícios do hippiedom comunitário de sua casa dos anos 1960. Basicamente, é apenas mais uma maneira pela qual Estelle está trabalhando para trazer o mundo para o século 21.

Este livro foi postado por AlenMiler no AvaxHome!

<https://avxhm.se/blogs/AlenMiler>

Prefácio

Convenções Usadas neste Livro

Usando exemplos de código

Safári O'Reilly

Como entrar em contato conosco

1. Caixa flexível

O problema resolvido

Soluções simples

Aprendendo o Flexbox

A propriedade display

2. Contêiner Flex

Propriedades do Flex Container

A propriedade de taquigrafia flex-flow

A propriedade flex-direction

A propriedade flex-wrap

Dimensão cruzada da linha Fle x

Contêiner Flex

A propriedade justify-content justify-content Exemplos

Os itens de alinhamento da propriedade

align-items: stretch

itens de alinhamento: flex-start

itens de alinhamento: flex-end

align-items: centro

itens de alinhamento: linha de base

Notas adicionais

A propriedade align-content

3. Itens Flex

O que são itens Flex? Recursos do

Flex Item

largura mínima

Propriedades específicas do item flexível

A propriedade flex

A propriedade flex-grow

Fator de crescimento não nulo

Crescendo proporcionalmente com base no fator de crescimento

Fator de crescimento com larguras diferentes

Fatores de crescimento e a propriedade flex

A propriedade flex-shrink

Proporcional com base na largura e no fator de encolhimento

No mundo real

Bases diferentes

O conteúdo da propriedade flex-basis

Automático

Valores padrão

Unidades de Comprimento

Base Zero

A propriedade **flex** Shorthand Common flex

Values

Valores personalizados do flex

Rodapé pegajoso com flexão

A propriedade **align-self**

A propriedade **order**

Navegação com guias revisitada

4. Exemplos de Flexbox

Layout responsivo de duas colunas Layout de tela

mais amplo

Seções da página inicial da rede

elétrica

Centralização vertical

Exemplo de flex embutido

Calendário

Grade Mágica

Desempenho

Bom

para

ir

