

Flex Box 布局——动手实验

主讲人： 和凌志

Case1: 创建 flex 布局

flex-grow 用法

a flex container with some flex items



1 2 3

```
<style>
  .container {
    display: flex;
  }

  .red {
    background: red;
    flex-grow: 1;
  }

  .green {
    background: green;
  }

  .blue {
    background: blue;
  }

  .container > div {
    font-size: 1.5em;
    padding: .5em;
    color: white;
  }
</style>
```

```
<body>
  <div class="container">
    <div class="red">1</div>
    <div class="green">2</div>
    <div class="blue">3</div>
  </div>
</body>
```


flex-grow 也可简写为 flex

```
.container {  
  display: flex;  
}
```

We use **display: flex** to declare the outer element as a *flex container*.

That's all we need in order to start using flexbox.

all of that flex container's in-flow children automatically become *flex items* and are therefore laid out using the **flex layout model**.

```
.red {  
  background:  red;  
  flex-grow: 1;  
}
```

The CSS `flex-grow` property sets the flex grow factor to the provided number.

It specifies how much the flex item will grow relative to the rest of the flex items in the flex container when positive free space is distributed.

case1: 完整代码

```
<style>
  .container {
    display: flex;
  }

  .red {
    background: ■ red;
    flex: 1;
  }

  .green {
    background: ■ green;
  }

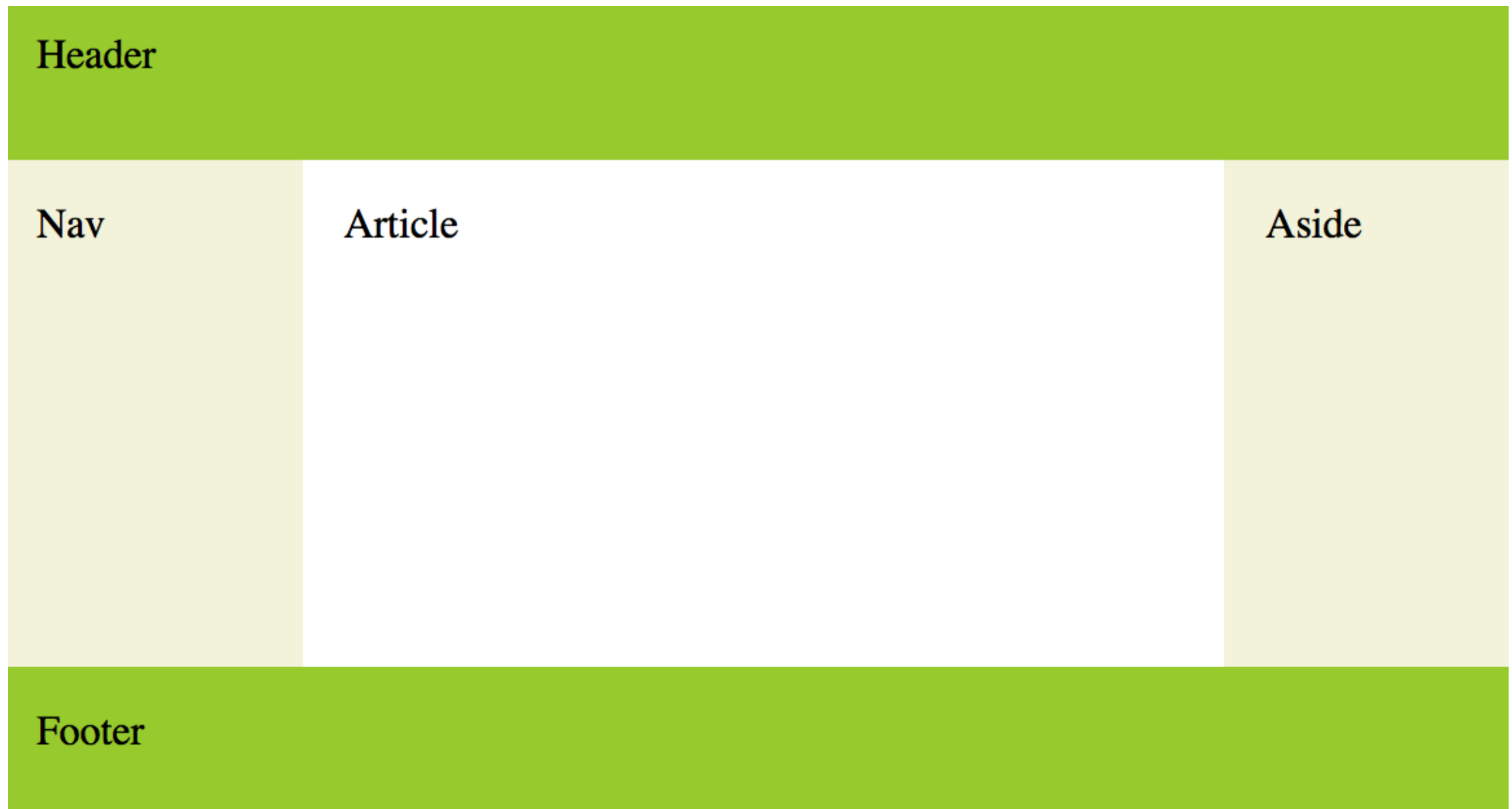
  .blue {
    background: ■ blue;
  }

  .container > div {
    font-size: 3em;
    padding: .5em;
    color: ■ white;
  }
</style>
```

```
<body>
  <div class="container">
    <div class="red">1</div>
    <div class="green">2</div>
    <div class="blue">3</div>
  </div>
</body>
```

Case2: use flexbox to create a three column layout

use flexbox to create a three column layout



创建 flex 容器

In the example, we make the `#main` element the flex container, and leave the `header` and `footer as block elements`. In other words, only the middle bit is flexbox.

```
.main {  
  display: flex;  
  height: calc(100vh - 40vh);  
}
```

We simply use **display: flex** to make it a flex container.

```
height: calc(100vh - 40vh);
```

- 高度占比通过计算设置： 60%
- `calc` 是一个函数， 减号两边留有空格

set the main area to 100 percent of the viewport's height, *minus* the height of the header and footer (20vh each).

This will ensure that the layout takes up the full height of the screen, even when there's not much content.

This means that the footer will never rise up and leave blank space underneath if the content doesn't take up the full screen height.

- 侧边栏各占 20%
- 剩余空间全部留给内容区

```
.nav {  
  width: 20%;  
  background: blue;  
}  
.article {  
  flex: 1;  
}  
.aside {  
  width: 20%;  
  background: blue;  
}
```

知识点延伸：

1. 侧边栏和内容区，完全通过百分比设置是否可行？
2. 通过设置 flex 占比是否可行？

设置 flex 比例，如下：

```
.nav {  
  flex: 1;  
  background:blue;  
}  
.article {  
  flex: 8;  
}  
.aside {  
  flex: 1;  
  background:blue;  
}
```

case2 : 完整代码

```
<style>
.main {
display:flex;
height: calc(100vh - 40vh);
}

.nav {
width: 20%;
background:blue;
}
.article {
flex: 1;
}
.aside {
width: 20%;
background:blue;
}
.header, .footer {
background: yellowgreen;
height: 20vh;
}
</style>
```

```
<div class="header">Header</div>

<div class="main">
<div class="nav" > Nav </div>
<div class ="article" >Article</div>
<div class = "aside" > Aside </div>
</div>
<div class="footer">Footer</div>
```

case3: Nested (嵌套) Flex Containers with Flexbox

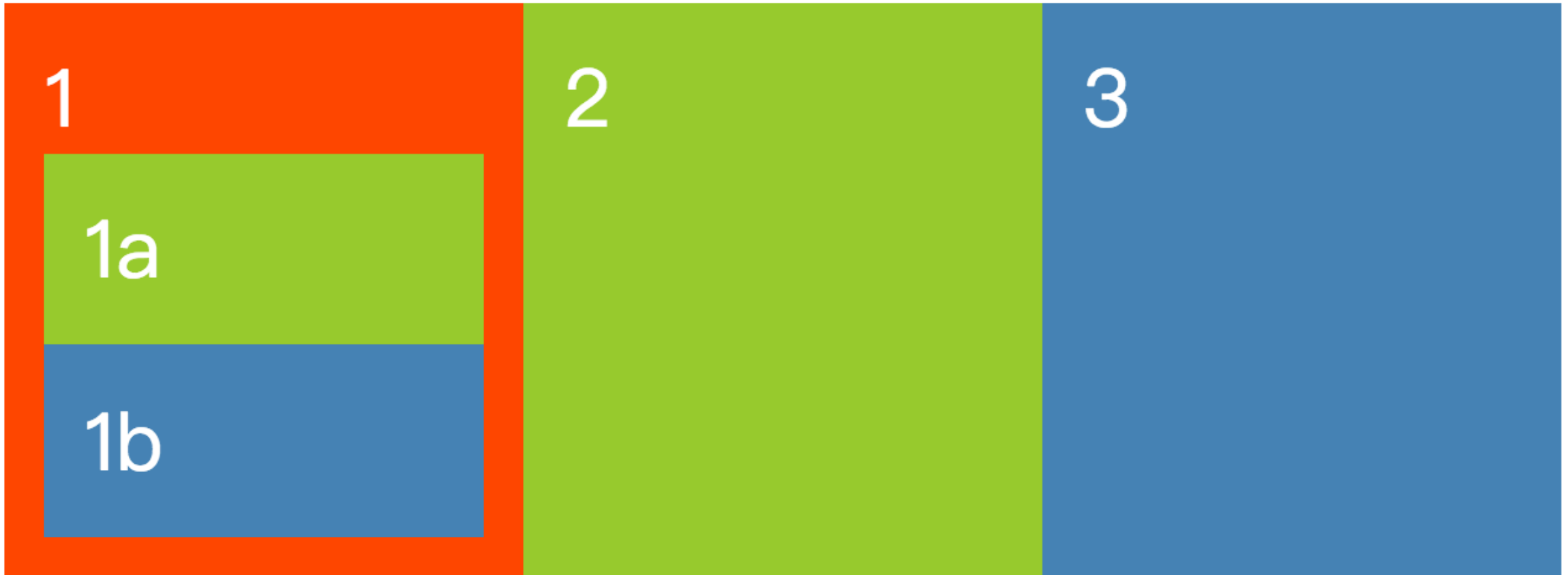
Nested Flex Containers with Flexbox

Flexbox is inherently **a one dimensional layout model**.

Flex items within a flex container can be laid out either horizontally or vertically, but not both.

If you want to lay out items in both dimensions, you'll need to **nest a flex container inside another one**.

Nested Flex Containers with Flexbox



Nested Flex Containers with Flexbox

Flexbox is inherently **a one dimensional layout model**.

Flex items within a flex container can be laid out either horizontally or vertically, but not both.

If you want to lay out items in both dimensions, you'll need to **nest a flex container inside another one**.


实现思路：

In this example we apply `display: flex` to both the outer container *and* to the red flex item.

But with the red flex item, we use `flex-direction: column`, which causes its flex items to stack up vertically on top of each other.

The default value for `flex-direction` is `row`, so that's why we didn't bother using this property for the outer container — the flex items are laid out in a row by default.

flex-direction: column

```
.container {  
  display: flex;  
}  
  
.red {  
  background:  orangered;  
  display: flex;  
  flex-direction: column;  
}
```

case3: 完整代码

```
<style>
  .body {
    box-sizing: border-box;
  }
  .container {
    display: flex;
  }

  .red {
    background: orangered;
    display: flex;
    flex-direction: column;
  }

  .green {
    background: yellowgreen;
  }
  .blue {
    background: steelblue;
  }
  .container div {
    font-size: 1.5em;
    padding: 1em;
    color: white;
    flex: 1;
  }
</style>
```

```
<div class="container">
  <div class="red">1
    <div class="green">1a</div>
    <div class="blue">1b</div>
  </div>
  <div class="green">2</div>
  <div class="blue">3</div>
</div>
```

case 4: Two Dimensional Website Layouts

Two dimensions

use flexbox across both dimensions — the row *and* the column.

use **one flex container for the vertical layout**, and another for the **horizontal layout**.

Flex Container

(flex-direction: column)



`<body>`

header

#main

nav

article

aside

Flex
Container

(flex-direction: row)



footer

实现思路：

```
body {  
  display: flex;  
  min-height: 100vh;  
  flex-direction: column;  
  margin: 0;  
}
```

The **display: flex** makes it a flex container and the **flex-direction: column** makes its flex items display in a column.

Another Flex Container

```
#main {  
  display: flex;  
  flex: 1;  
}
```

Adding **display: flex** makes it a flex container just like the other one. Now its in-flow children become flex items. The **flex: 1** ensures that it grows so that it occupies the maximum amount of space available.

We could've used **flex-direction: row** to explicitly specify its direction, but **row** is the default value anyway.

Flex row Layout

```
#main {  
  display: flex;  
  flex: 1;  
}  
#main > article {  
  flex: 1;  
}  
#main > nav,  
#main > aside {  
  width: 20%;  
  background: beige;  
}
```

case4: 完整代码

```
<style>
.container {
  display: flex;
  min-height: 100vh;
  flex-direction: column;
}
#main {
  display: flex;
  flex: 1;
}
#main > article {
  flex: 1;
}
#main > nav,
#main > aside {
  width: 20%;
  background: beige;
}
header, footer {
  background: yellowgreen;
  height: 20vh;
}
</style>
```

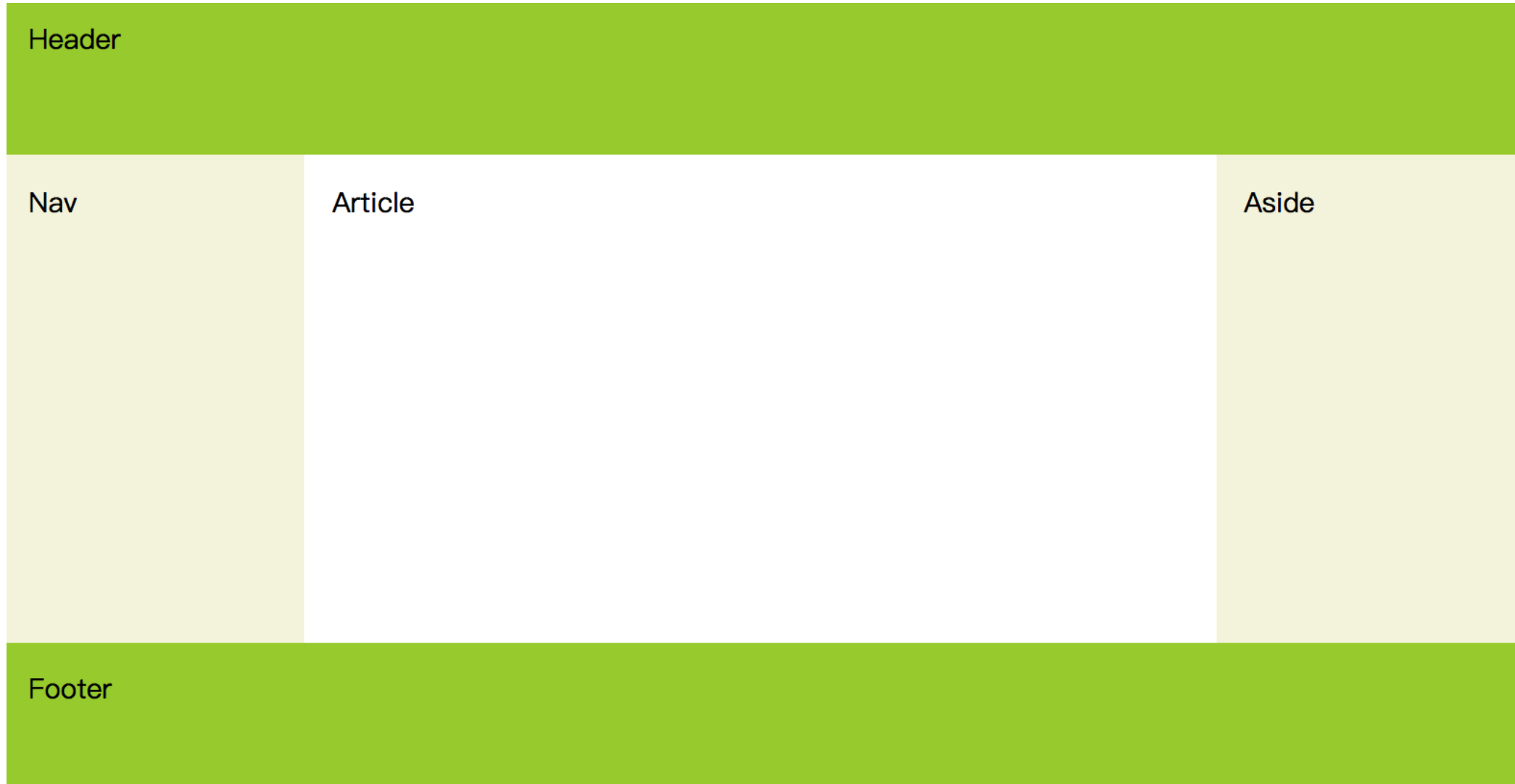
```
<header>Header</header>
  <div id="main">
    <nav>Nav</nav>
    <article>Article</article>
    <aside>Aside</aside>
  </div>
<footer>Footer</footer>
```

case5 : Create a Responsive Layout with Flexbox

Mobile First Layout

combine **flexbox** with **media queries** to create a layout that responds to different sized screens.

Layout 1 for wide screen



Layout 2 for medium or narrow screens

Header

Article

Nav

Aside

Footer

实现思路： change display flex to block

```
@media screen and (max-width: 575px) {  
  #main {  
    display: block;  
  }  
}
```

We did this on the **#main** element so that its children are no longer flex items.

This results in them stacking up on top of each other in source order.

Case4: 完整代码

```
<style>
* {
  box-sizing: border-box;
}
body {
  display: flex;
  min-height: 100vh;
  flex-direction: column;
  margin: 0;
}
#main {
  display: flex;
  flex: 1;
}
#main > article {
  flex: 1;
}
#main > nav,
#main > aside {
  flex: 0 0 20vw;
  background: #beige;
}

header, footer {
  background: #yellowgreen;
  height: 20vh;
}
header, footer, article, nav, aside {
  padding: 1em;
}
@media screen and (max-width: 575px) {
  #main {
    flex-direction: column;
  }
}
</style>
```

```
<header>Header</header>
<div id="main">
  <nav>Nav</nav>
  <article>Article</article>
  <aside>Aside</aside>
</div>
<footer>Footer</footer>
```

注意“临界值”的设定！

```
@media screen and (max-width: 900px) {  
  #main {  
    flex-direction: column;  
  }  
}
```

case6: Mobile First Layout

Mobile First Layout (从小屏着手，再考虑大屏)

Mobile first is a term for layouts that are created primarily for mobile devices, but include a media query that changes the layout on larger devices.

That's the opposite of what we did above, where **our default layout was for large devices**, and we added the media query for smaller devices.

```
@media screen and (min-width: 576px) {  
  #main {  
    flex-direction: row;  
  }  
  #main > nav,  
  #main > aside {  
    flex: 0 0 20vw;  
  }  
}
```

Note that the media query uses [min-width](#) this time, to match all devices of that width and wider. （大屏情况）

On the above, we used [max-width](#) to match only devices that were that width or narrower. （小屏情况）

注意临界值的设定

```
@media screen and (min-width: 900px) {  
  #main {  
    flex-direction: row;  
  }  
  
  #main>nav,  
  #main>aside {  
    flex: 0 0 20vw;  
  }  
}
```


#main 的 flex-direction: column

```
#main {  
  display: flex;  
  flex: 1;  
  flex-direction: column;  
}
```

Case6: 完整代码 (media query 变化)

```
<style>
* {
  box-sizing: border-box;
}
body {
  display: flex;
  min-height: 100vh;
  flex-direction: column;
  margin: 0;
}

#main {
  display: flex;
  flex: 1;
  flex-direction: column;
}

#main > article {
  flex: 1;
}
#main > nav,
#main > aside {
  background: #beige;
}

header, footer {
  background: #yellowgreen;
  height: 20vh;
}
header, footer, article, nav, aside {
  padding: 1em;
}

@media screen and (min-width: 576px) {
  #main {
    flex-direction: row;
  }
  #main > nav,
  #main > aside {
    flex: 0 0 20vw;
  }
}
</style>
```

```
<header>Header</header>
<div id="main">
  <nav>Nav</nav>
  <article>Article</article>
  <aside>Aside</aside>
</div>
<footer>Footer</footer>
```

小结:

1. Flex box layout 的container
2. row 与 Column 的应用
3. Media query 的应用
4. Mobile first 模式

感谢聆听