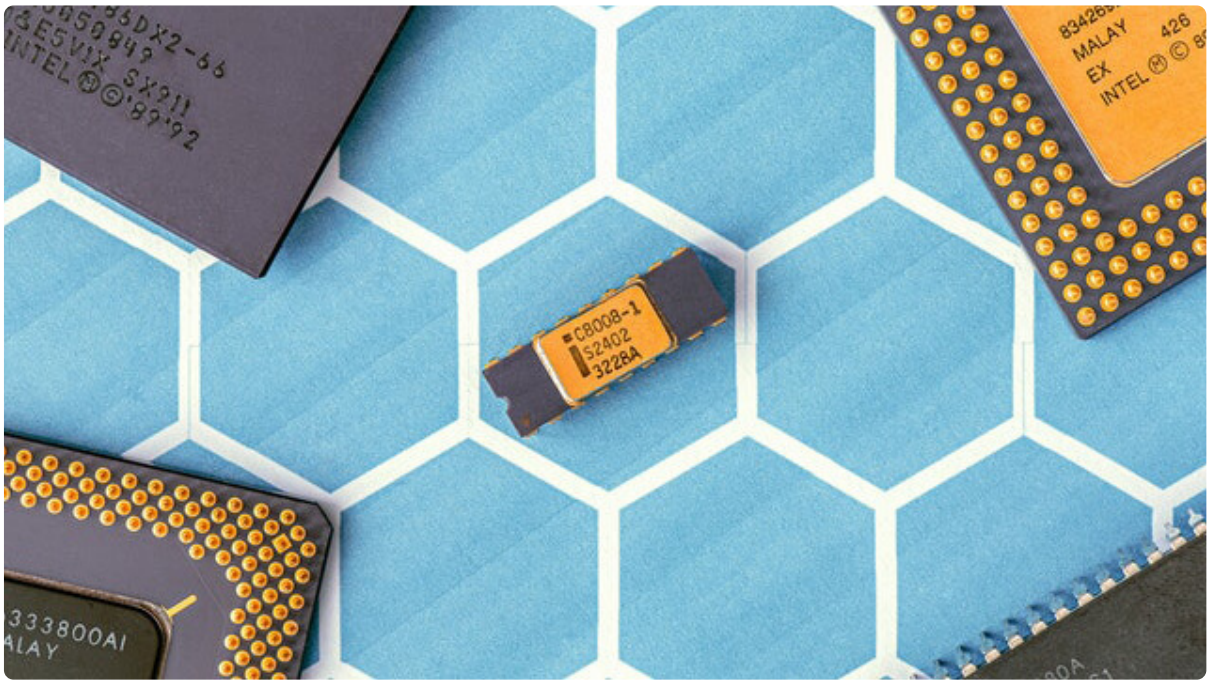
18 布局组件的开发

更新时间：2019-07-08 11:07:06



> 散步促进我的思想。我的身体必须不断运动，脑筋才会开动起来。
>
> —— 卢梭

上一节我们完成了页面的布局，安排好了标题栏、内容区、导航栏和页面蒙版的位置。这一节我们就要进一步开发布局样式，完成标题栏和导航栏的内容。

相关知识

我们在做布局组件开发的时候，将使用到弹性布局，所以我们这里提前把需要用到的弹性布局的知识介绍一下。弹性布局（Flex布局）是 CSS3 中提出的一种布局方式，和传统的布局方式不同的是，弹性布局不再指定一个元素具体的尺寸，而是描述这个元素该如何去填充空间。这样做会把布局变得简单，我们只需要提出布局的要求，剩下计算的部分就交给浏览器完成了。

Flex 布局中有两个比较重要的东西，一个是弹性容器，另一个是弹性盒子。弹性容器和我们之前用的盒子没什么区别，我们通过“display: flex;”来把一个盒子指定成弹性容器，这样对这个容器本身没什么影响，但是会影响到这个容器里面元素的排布。当容器被指定为弹性容器以后，它里面的盒子就变成了弹性盒子。弹性盒子会有如下特点：

1. 弹性容器的默认宽度是 100%，和块级元素一样。
2. 横向布局的弹性盒子里的块级元素不再占用一整行，而是像 float 元素一样按一个方向排列。
3. 弹性盒子可以伸展，也可以压缩，尺寸可以随着容器大小而变化。
4. 多个弹性盒子的排布顺序、对齐方式等都是可以设置的。

弹性布局里，在这一节会用到以下样式属性：

1、**display: flex;**

这个属性是用来指定弹性容器的，给容器加上“display: flex”属性后，这个容器就变成弹性容器了。我们先来定义一个这样的弹性盒子：

```
<!-- HTML -->
<div class="flex-box">
    <span class="box box1">1</span>
    <span class="box box2">2</span>
    <span class="box box3">3</span>
</div>

/* CSS */
.flex-box{
    display: flex;
}
.flex-box > .box{
    width: 50px;
    height: 100px;
}
.flex-box > .box1{
    background: red;
}
.flex-box > .box2{
    background: green;
}
.flex-box > .box3{
    background: yellow;
}
```

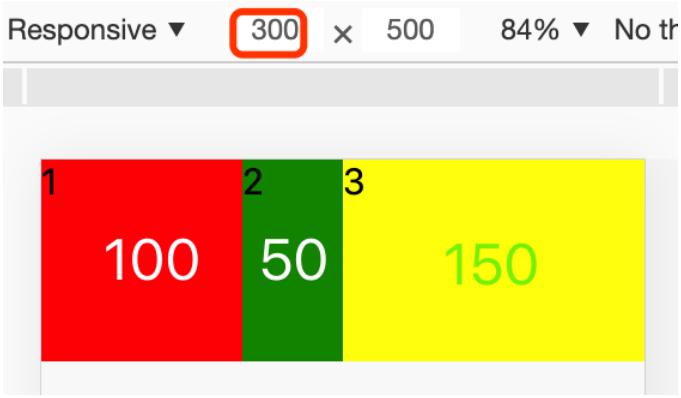
2、**flex-grow**

flex-grow 属性用来指定弹性盒子的拉伸方式。当一行内的弹性盒子不能充满整行时，就用这个属性来指定怎么去分配空闲的空间。flex-grow 的默认取值是 0，就是不去占用空闲区域。当 flex-grow 为非 0 数字的时候，这个弹性盒子就要进行拉伸了。具体拉伸多少，要看这一行里面有多少个需要拉伸的对象，这些需要拉伸的盒子按着 flex-grow 值的比例分配空闲区域的空间。以下面为例：

```
<!-- HTML -->
<div class="flex-box">
    <span class="box box1">1</span>
    <span class="box box2">2</span>
    <span class="box box3">3</span>
</div>

/* CSS */
.flex-box{
    display: flex;
}
.flex-box > .box{
    width: 50px;
    height: 100px;
}
.flex-box > .box1{
    flex-grow: 1;
    background: red;
}
.flex-box > .box2{
    background: green;
}
.flex-box > .box3{
    flex-grow: 2;
    background: yellow;
}
```

上面三个弹性盒子原始宽度都是 50px，整个屏幕是 300px宽，所以会有 150px 的空白区域。当我们给 box1 的“flex-grow”设为1；box2 没有“flex-grow”属性，默认为 0；box3的“flex-grow”设为 2。这样在分配空间的时候，box1 的“flex-grow”是 1，分得 1/3 的空白区域，也就是 50px；box2 不分配空闲空间；box3 的“flex-grow”是 2，分得 2/3 空白区域，也就是 100px。这样 box1 和 box3 就会拉伸，新的宽度是原宽度加上占用空白区域的宽度之和。box1 的宽度会变成 100px，box3 的宽度会变成 150px，结果如下：



3、**flex-shrink**

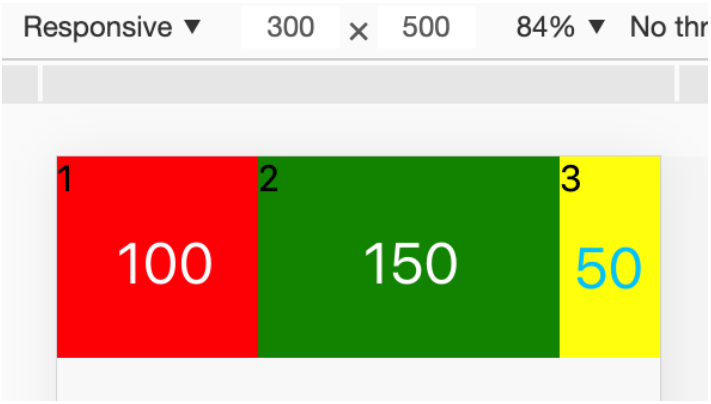
我们在使用弹性布局的时候，通常是使用它可以拉伸的特性，但是弹性盒子也可以是收缩的。flex-shrink 属性就是用来指定弹性盒子的收缩方式的，当所有盒子的宽度加起来超过了容器的总宽度，每个盒子就要缩减一定的尺寸来保证所有盒子都能在容器中。flex-shrink 的默认值是 1，也就是默认情况下所有盒子平分超出部分的尺寸。

```
/* CSS */
.flex-box{
    display: flex;
}
.flex-box > .box{
    width: 150px;
    height: 100px;
}
.flex-box > .box1{
    flex-shrink: 1;
    background: red;
}
.flex-box > .box2{
    flex-shrink: 0;
    background: green;
}
.flex-box > .box3{
    flex-shrink: 2;
    background: yellow;
}
```

我们把刚才的例子稍微修改一下，每个盒子设置为 150px宽，容器还是 300px，这样三个盒子宽度加起来会多出150px。这时候如果不给三个盒子指定收缩方式，那么多出来的 150px 就会由三个盒子平分，每个盒子收缩 50px，最终就是每个盒子都是 100px 宽。但如果我们给 box1 加上“flex-shrink: 1;”，给box2加上“flex-shrink: 0;”，给 box3 加上“flex-shrink: 2;”，这样压缩的情况就会不一样了。这时候“flex-shrink”不为0的元素就要通过收缩来消化掉多余的宽度，收缩的多少也是由“flex-shrink”值所占的比例来决定。

- box1 会消化超出部分的 1/3，也就是收缩 50px，宽度变为100px。
- box2 的“flex-shrink”值设置为 0，不参与压缩。
- box3 会消化超出部分的 2/3，也就是收缩 100px，宽度变为 50px。

最后的结果就是：



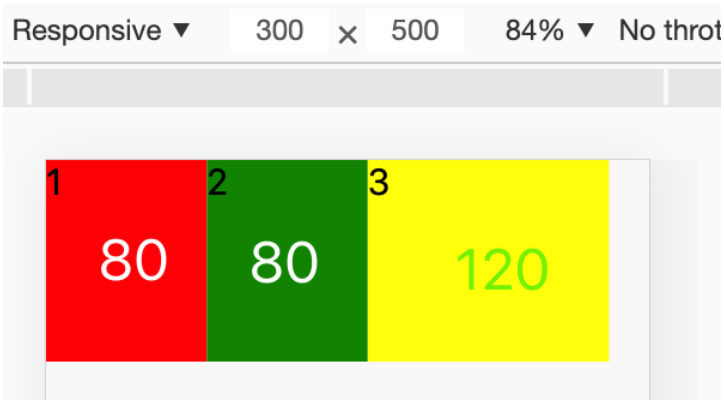
在使用 flex-shrink 的时候，有一点要注意，盒子被压缩的时候是有限度的，最小只能被压缩到里面内容的宽度。假如box3 里面有个宽 30px 的盒子，那么无论 box3 的“flex-shrink”有多大，他都将占用 30px 的宽度。如果这个弹性容器实在放不下这几个盒子，就会出现滚动条。

4、flex-basis

flex-basis 属性是用来指定弹性盒子基准宽度的。在弹性布局里，弹性盒子最终的宽度通常不是 width 属性指定的值，所以这里用 width 就显得不太准确。作为替代，弹性布局里给出了 flex-basis，它的作用和 width 属性很类似，但是把这个宽度叫做基准值，就显得严谨的多了。flex-basis 属性的优先级要高于 width，无论使用什么样的选择器，当两个属性同时作用在同一个弹性盒子上的时候，都是以 flex-basis 为准。

```
.flex-box{
    display: flex;
}
.flex-box > .box1{
    flex-basis: 80px;
    background: red;
}
.flex-box > .box2{
    flex-basis: 80px;
    background: green;
}
.flex-box > .box3{
    flex-basis: 120px;
    background: yellow;
}
.flex-box > .box{
    width: 50px;
    height: 100px;
}
```

还是刚才的例子，为了体现两个属性优先级的关系，我们把“.flex-box > .box”选择器放在了最后来提高它的优先级。这个例子运行的结果如下：



从图中可以看出，三个盒子的宽度都是以 flex-basis 指定的值为准，而 width 属性没有生效。

5、flex

下来介绍flex属性，这个属性的格式如下：

```
flex: <flex-grow> <flex-shrink> <flex-basis>
```

flex 属性是对前面讲的 flex-grow、flex-shrink、flex-basis 三个属性的缩写。如果把三个属性都按顺序指定出来是很容易使用的，但通常我们会进一步缩写，把三个属性值合成一个来使用：

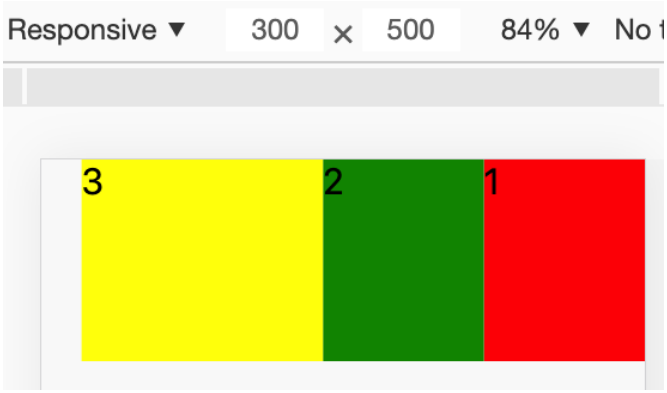
- **默认情况**，当不指定flex值时，默认的就是“flex: 0 1 auto;”，和分开写的三个属性默认情况一样。
- **“flex: none;”**，flex的值为none时，表示的意思是“flex: 0 0 auto;”，就是这个盒子既不伸展也不收缩。
- **“flex: auto;”**，flex值为auto时，表示的意思是“flex: 1 1 auto;”，就是这个盒子是既能伸展也能收缩。
- **“flex: 1;”**，flex值为1（也可以是其他非0数值）时，表示的意思是“flex: 1 1 0;”，这表示容器内的盒子会平分空间。

6、flex-direction

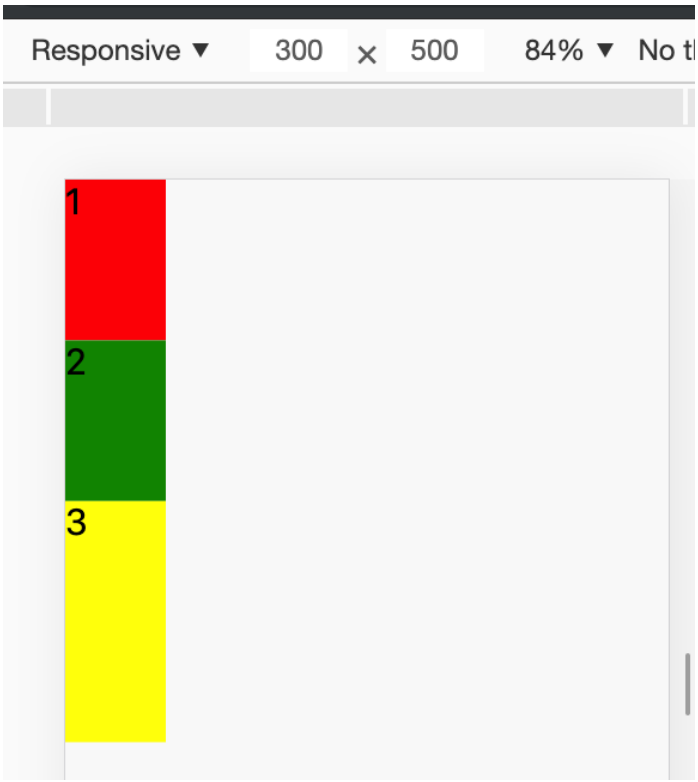
最后要说的是 flex-direction，我们前面讲的例子都是按着默认的从左到右一行排列。但是弹性布局中是支持弹性盒子排布方向的。这个属性是作用在弹性容器上的，取值可以是“row”、“row-reverse”、“column”和“column-reverse”。默认的取值就是 row，也就是我们之前用的由左到右排列的方式，我们把上面例子中容器的排布方式修改一下。

把“flex-direction”修改成“ row-reverse”时，就是从右至左排列，运行结果如下：

```
.flex-box{
    display: flex;
    flex-direction: row-reverse;
}
```



当把“flex-direction”修改成 “colume” 时，就是从上到下排列，运行结果如下：



这里要注意下，当我们把排列方式修改以后，每个盒子的宽度又变回 50px了，而每个盒子的高度不一样了。这时因为在竖向排列的时候，flex-basis 的值就会代表高度了，所以会有这种变化。

关于 Flex 的知识就先介绍这么多，后面的开发过程中还会遇到其他的用法，我们后面再讲。

标题栏的开发

介绍完基础知识，我们就该进入开发了。经过刚才对弹性布局的介绍，后面的开发就显得简单多了。

首先我们要开发标题栏部分，标题栏的格式很固定，我们在微信里找两个标题栏来看下：

这两张图片分别是微信里聊天页面和支付页面的标题栏，标题栏就分为左中右三部分。我们要做的标题栏能满足如下要求即可：

1. 左侧的按钮宽度固定，不使用弹性盒子，文本垂直居中。
2. 标题部分宽度可以自适应。
3. 标题文字水平竖直居中，且文本超长后可以自动截断。
4. 右侧按钮宽度固定，不使用弹性盒子，文本垂直居中。
5. 容器有背景色，能遮盖住底层内容区。

我们先新建一个文件 /demo/layout-component.html，把上一节建好的 /demo/layout.html 内容复制过去，然后去掉蒙版层的 HTML 代码，结果如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0,user-scalable=no">
    <link rel="stylesheet" href="../src/tuitui-ui.css">
    <title>页面布局组件</title>
  </head>
  <body>
    <div class="tt-header">
      标题栏
    </div>
    <div class="tt-navbar">
      导航栏
    </div>
    <div class="tt-content">
      内容区
    </div>
  </body>
</html>
```

下来我们完成 tt-header 里的 HTML 部分，直接在 /demo/layout-component.html 的 tt-header 部分中修改：

```
<div class="tt-header">
    <div class="left"><i class="fa fa-chevron-left"></i> 返回</div>
    <div class="title">我是标题</div>
    <div class="right"><i class="fa fa-ellipsis-h"></i></div>
</div>
```

要完成之前提出的需求，我们要做的是：

1. 将 tt-header 改成弹性容器，并添加背景色。
2. 使 .left、.title 和 .right 的元素在自己的容器里都水平竖直居中。
3. 对 .title 部分做文本自动截断处理
4. 对 .left 和 .right 部分指定宽度，且不能伸缩。

完成上面四个要求，前面提到的需求就可以实现了。这里面水平竖直居中可以在容器上设置，通过继承三个子元素就都可以使用了。最终修改的 /src/header.css 的代码如下：

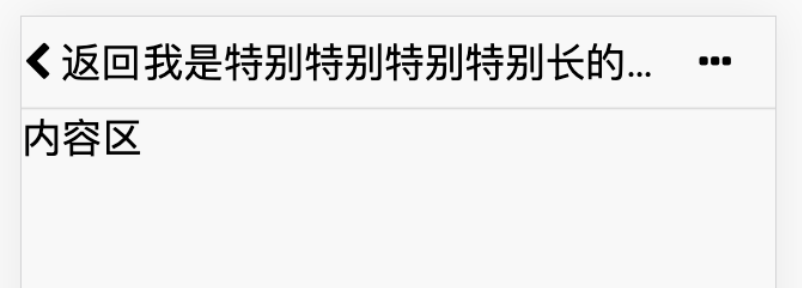
```
/* 头部导航条 */
.tt-header{
+   display: flex;
    position: fixed;
    box-sizing: border-box;
    width: 100%;
    max-width: 640px;
    height: 2.3rem;
+   line-height: 2.3rem;
+   text-align: center;
    top: 0;
    z-index: 200;
    border-bottom: 1px solid #ddd;
+   background: #f8f8f8;
}
/* 左侧功能区 */
.tt-header > .left{
    flex-basis: 3rem;
    text-align: center;
    flex-shrink: 0;
}
/* 中间标题部分 */
.tt-header > .title{
    flex-grow: 1;
    overflow: hidden;
    white-space: nowrap;
    text-overflow: ellipsis;
}
/* 右侧功能区 */
.tt-header > .right{
    flex-basis: 3rem;
    flex-shrink: 0;
}
```

这里面关于 Flex 的知识不再重复讲了，但有两个其他的点要注意：

1. “line-height: 2.3rem;” 配合上之前的“height: 2.3rem;”就可以使单行文本水平居中，通过在父元素上设置，三个子元素都可以继承的到。
2. .tt-header 使用的怪异盒模型，但在做垂直居中的时候height 和 line-height 使用的是同样的竖直，把 border-bottom 的宽度忽略也没问题。
3. “overflow: hidden; white-space: nowrap; text-overflow: ellipsis;”这三条语句组合使用，就可以对单行超长文本做截断，并且在截断的地方自动补充省略号。

这段代码运行的结果如下：

当标题文本超长时，就会自动截断：



这样标题栏的样式就完成了。

导航栏的开发

导航栏用于提供网站导航，也比较简单。我们可以看下微信的导航栏：

导航栏里就是几个导航区域平分导航栏，每个导航区域里包括一个图标和一个导航名称。在做这个导航栏的时候，满足以下条件就可以了：

1. 各个导航区域平分导航栏。
2. 各导航区域内的文字在区域内水平居中。
3. 导航栏有背景色，可以遮盖住内容区的内容。
4. 给导航项目设置选中状态的样式。

我们先完成 /demo/layout-component.html 的 tt-navbar 部分的HTML:

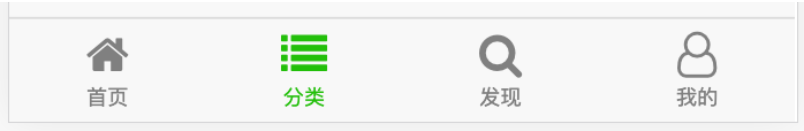
```
<div class="tt-navbar">
    <a class="navbar-item">
        <i class="fa fa-home icon"></i>
        <span class="name">首页</span>
    </a>
    <a class="navbar-item active">
        <i class="fa fa-list icon"></i>
        <span class="name">分类</span>
    </a>
    <a class="navbar-item">
        <i class="fa fa-search icon"></i>
        <span class="name">发现</span>
    </a>
    <a class="navbar-item">
        <i class="fa fa-user-o icon"></i>
        <span class="name">我的</span>
    </a>
</div>
```

导航栏里的项目一般都是链接，所以我们用a标签来当做导航项目的容器。里面我们分为“首页”、“分类”、“发现”和“我的”四个项目，把“分类”这一栏用 active 类设置成了选中项。

参考刚才开发标题栏时候的做法，这个导航栏也可以用弹性布局来实现。可以按着下面的代码来实现：

```
/* 底部导航栏 */
.tt-navbar{
+   display: flex;
    position: fixed;
    box-sizing: border-box;
    bottom: 0;
    width: 100%;
    max-width: 640px;
    height: 2.5rem;
    border-top: 1px solid #ddd;
    z-index: 200;
+   background: #f8f8f8;
+   text-align: center;
}
/* 导航项目 */
.tt-navbar > .navbar-item{
    flex: 1;
    color: #808080;
}
/* 被选中的样式 */
.tt-navbar > .navbar-item.active{
    color: #09BB07;
}
/* 导航图标 */
.tt-navbar > .navbar-item > .icon{
    padding: .3rem 0 .1rem;
    font-size: 1.1rem;
}
/* 导航名称 */
.tt-navbar > .navbar-item > .name{
    display: block;
    font-size: .5rem;
}
```

这段代码中，首先给容器tt-navbar设置成了弹性容器，并修改了对齐方式、背景色；然后在.navbar-item上设置了“flex: 1;”和字体颜色，使得各个导航区域平分导航条（这里没有在.navbar-item父元素上设置字体颜色，是为了可以覆盖链接自带的样式）；再然后通过并集选择器.navbar-item.active给选中的导航项加上了绿色的字体颜色，最后通过.icon和.name两个选择器调整了图标和字体的样式。这样整个导航条就实现出来了，最终效果如下：

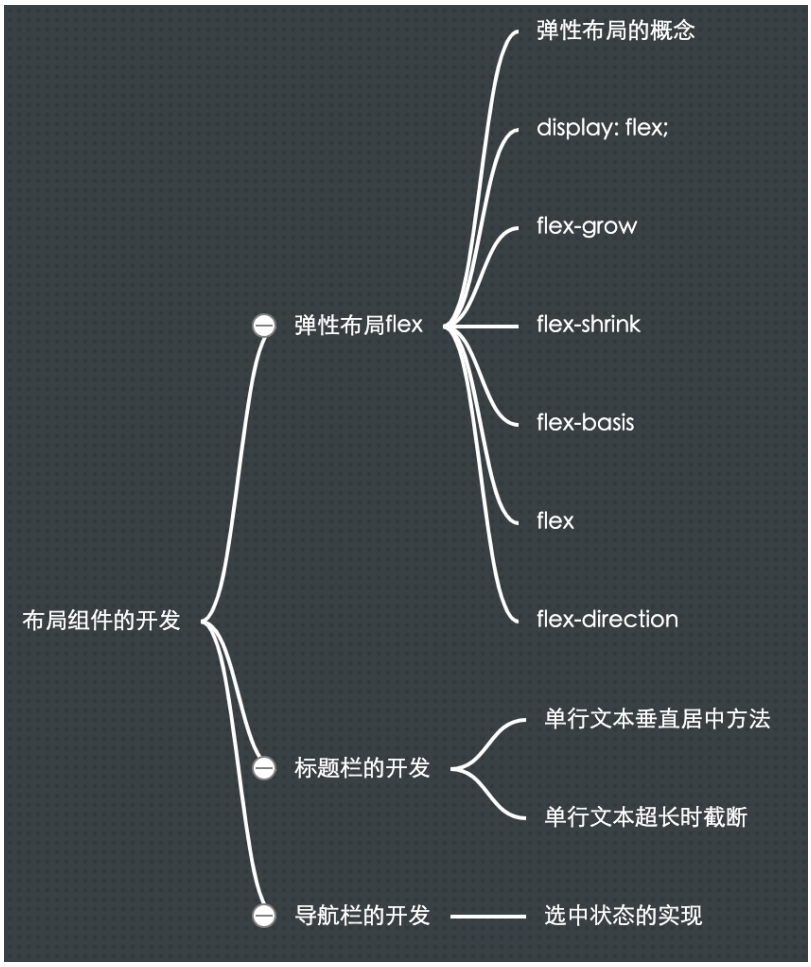


总结

写着写着，这一节的内容又不少，最主要的是介绍了 Flex 布局的基础用法，此外还开发了标题栏和导航栏两个布局组件。在后面的章节还会用到很多次的 Flex，也会有更复杂的使用方法，后面遇到了我们再谈。

在使用Flex布局的时候，很多同学可能会有疑问，我们所有使用的样式都是 CSS3 的样式，而没有加 -webkit- 或者 -moz- 这些前缀。这是因为在移动端浏览器基本上都是使用的 webkit 内核，已经能对 CSS3 有很好的兼容性了，另外在最后集成代码的时候还会使用工具对明显会不兼容的属性自动加上兼容的写法。这样我们在前期开发的时候可以先不写兼容，等集成好以后再在各种设备上测试，发现有不兼容的现象再做调整。同学们如果想查某个属性的兼容性，可以使用这个网站：https://caniuse.com/。

这一节的内容结构如下：



这一节的内容就到这里，同学们可以访问【页面布局组件在线预览】来查看这一节的演示效果，下一节我们开始开发表单组件。

17 页面布局的设计与开发
19 表单元素的开发