flex布局

任何一个容器都可指定为flex布局

块级元素:

```
div{
    display:flex;
}
```

行内元素:

```
span{
    display:inline-flex;
}
```

注: 设为flex布局后，容器的子元素的`float`,`clear`,`vertical-align`属性将失效

基本概念

1. flex容器（flex container）

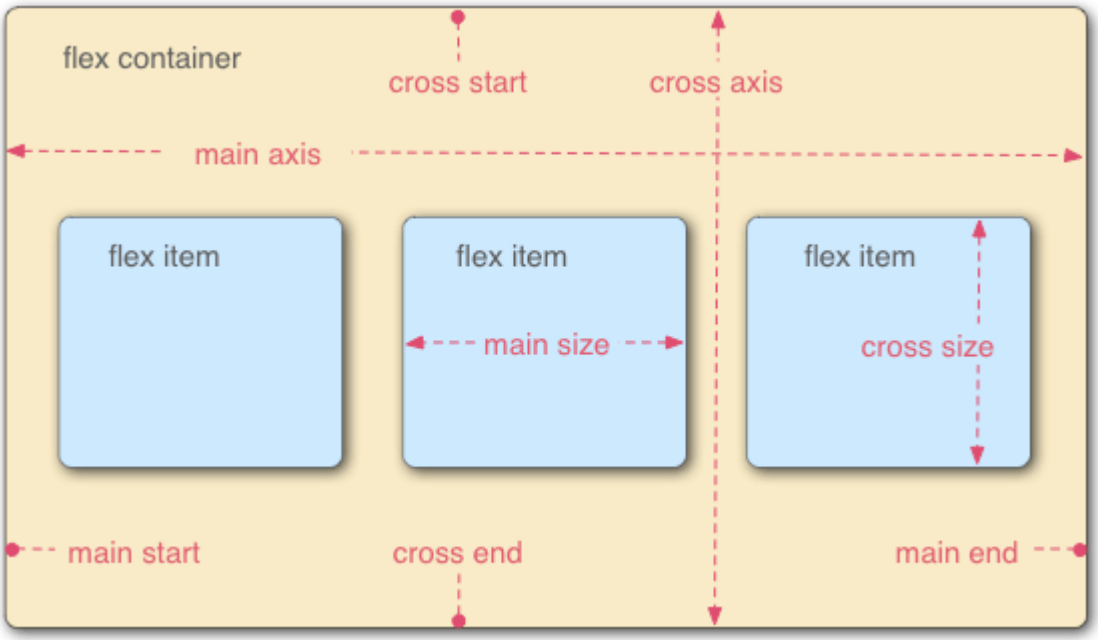
   采用flex布局的元素

2. flex项目（flex item）

   flex容器的所有子元素

3. 主轴（main axis）：默认为水平方向（flex item的排列方向）

4. 交叉轴（cross axis）：与主轴垂直



属性

容器属性

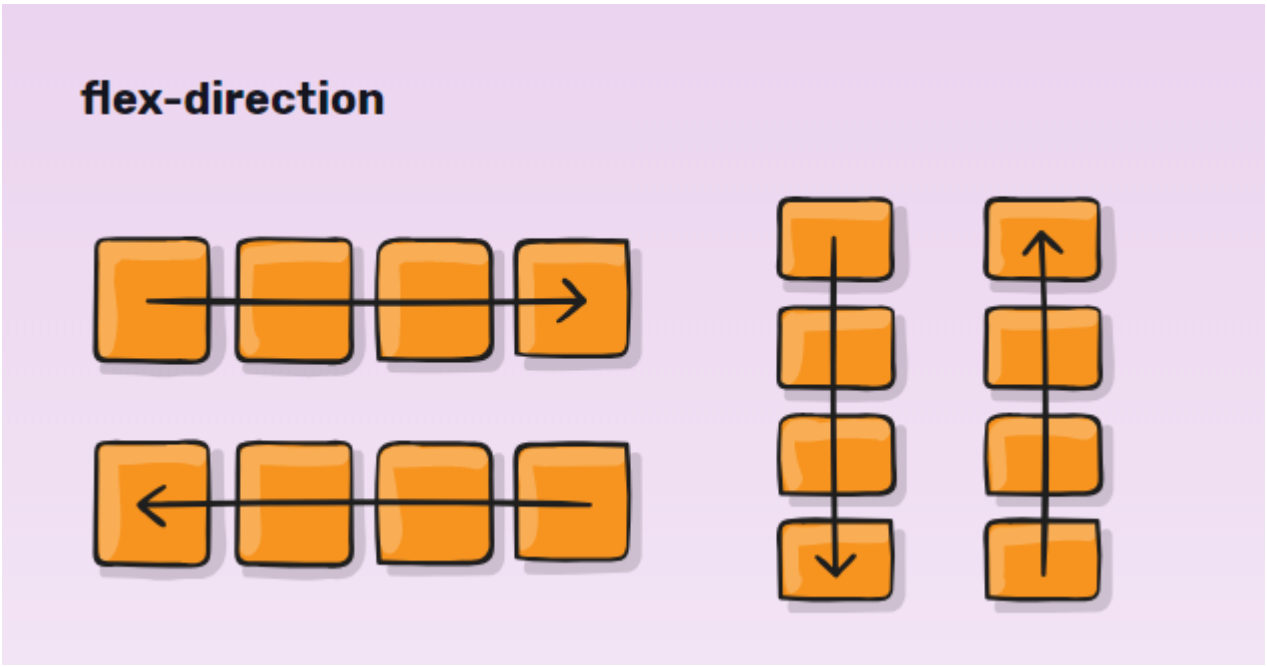
properties for flex container

> - flex-direction
> - flex-wrap
> - flex-flow
> - justify-content
> - align-items
> - align-content

1. flex-direction

设置主轴方向（即flex item排列方向)

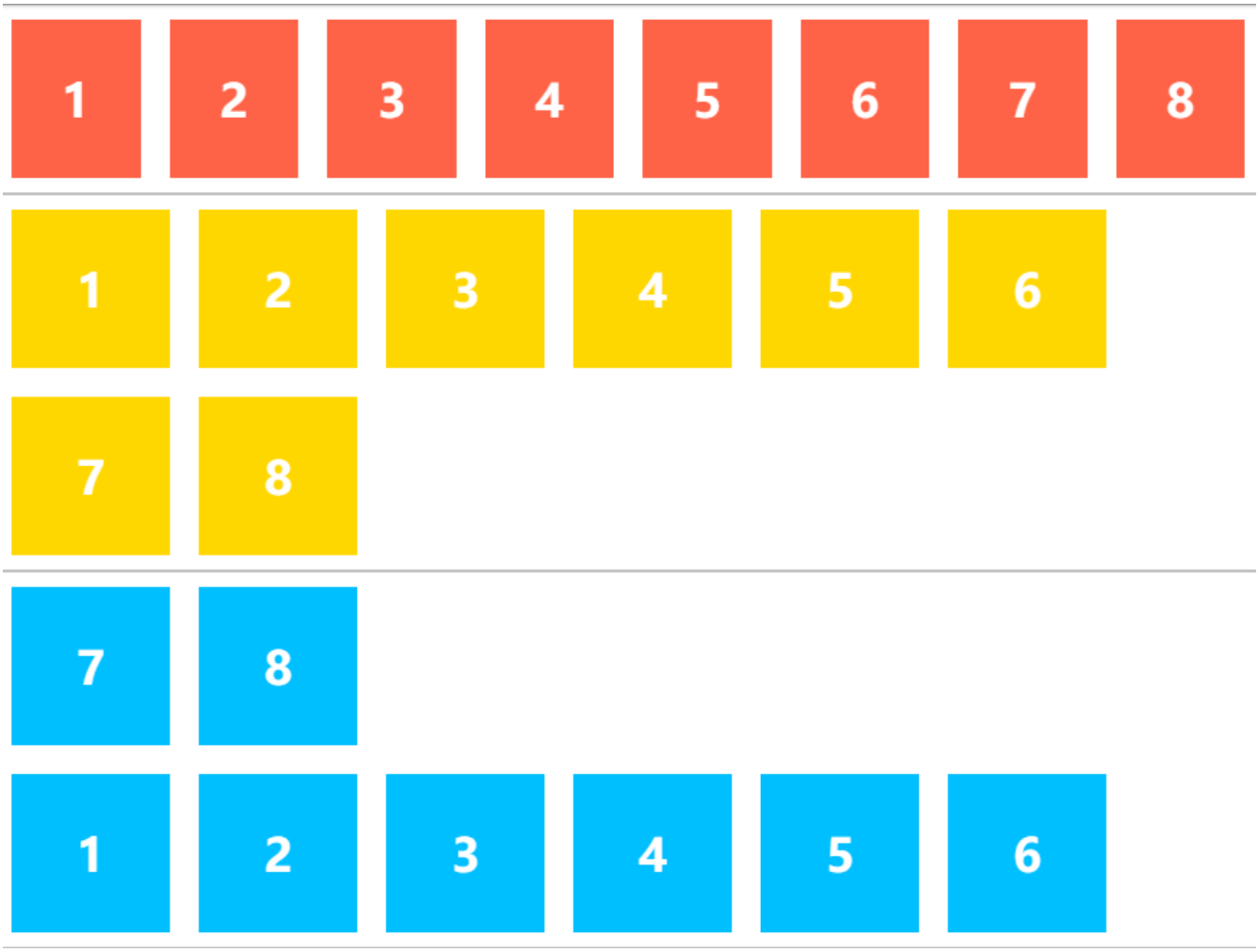
```
.flex_box{
    flex-direction:row | row-reverse | column | column-reverse;
    /*默认值为row,即水平从左往右排列*/
}
```



2. flex-wrap

默认情况下，项目将试着排列在一行（main axis方向），该属性设置如何进行换行

```
.flex_box{
    flex-wrap:nowrap | wrap | wrap-reverse;
    /*nowrap(默认) : 即不换行，项目多时，将压缩每个项目尺寸，项目挤在同一行*/
    /*wrap : 从上到下换行*/
    /*wrap-reverse : 从下到上换行*/
}
```



3. flex-flow

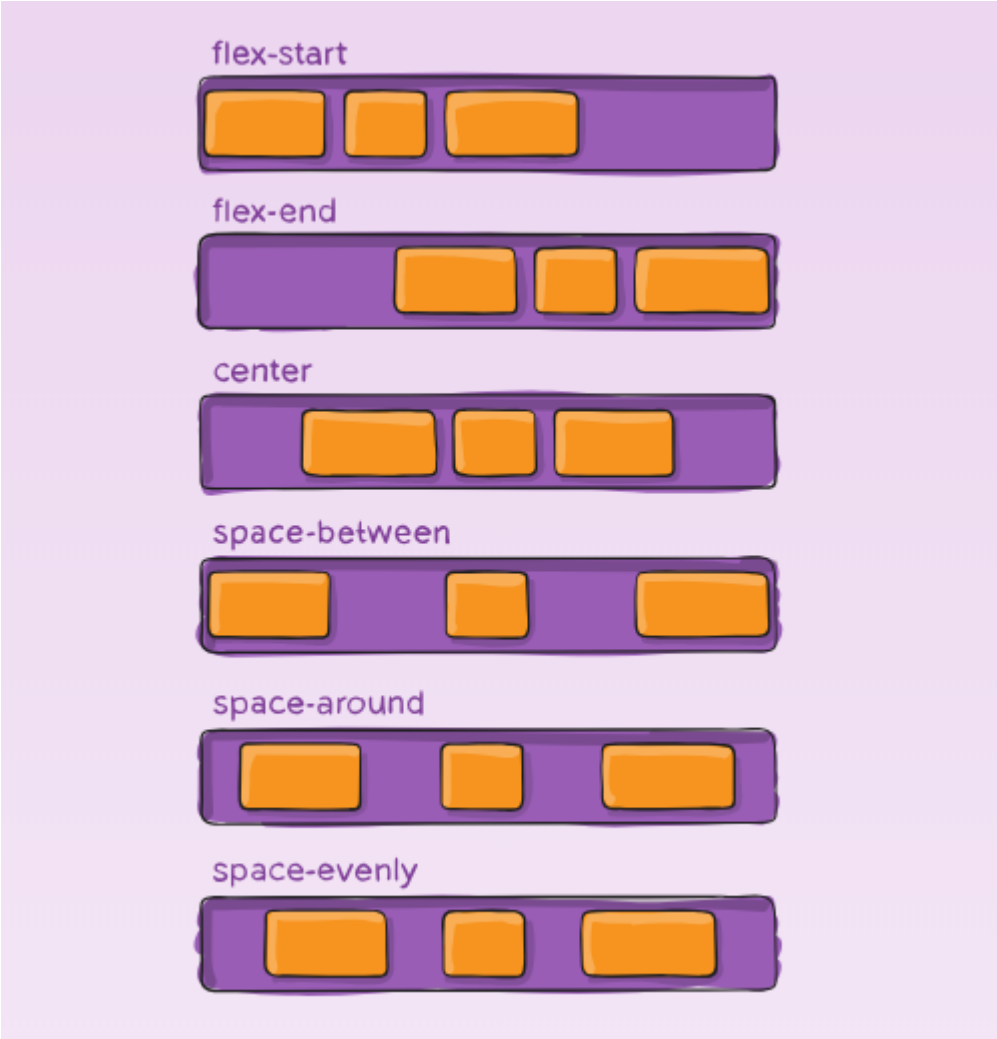
复合属性，是`flex-direction`和`flex-wrap`属性的简写形式

```
.flex_box{
    flex-flow:row wrap;
}
```

4. justify-content

定义项目在主轴上的对齐方式

```
.flex_box{
    justify-content:flex-start | flex-end | center | space-between | space-around |
space-evenly;
    /*flex-start(默认)*/
}
```

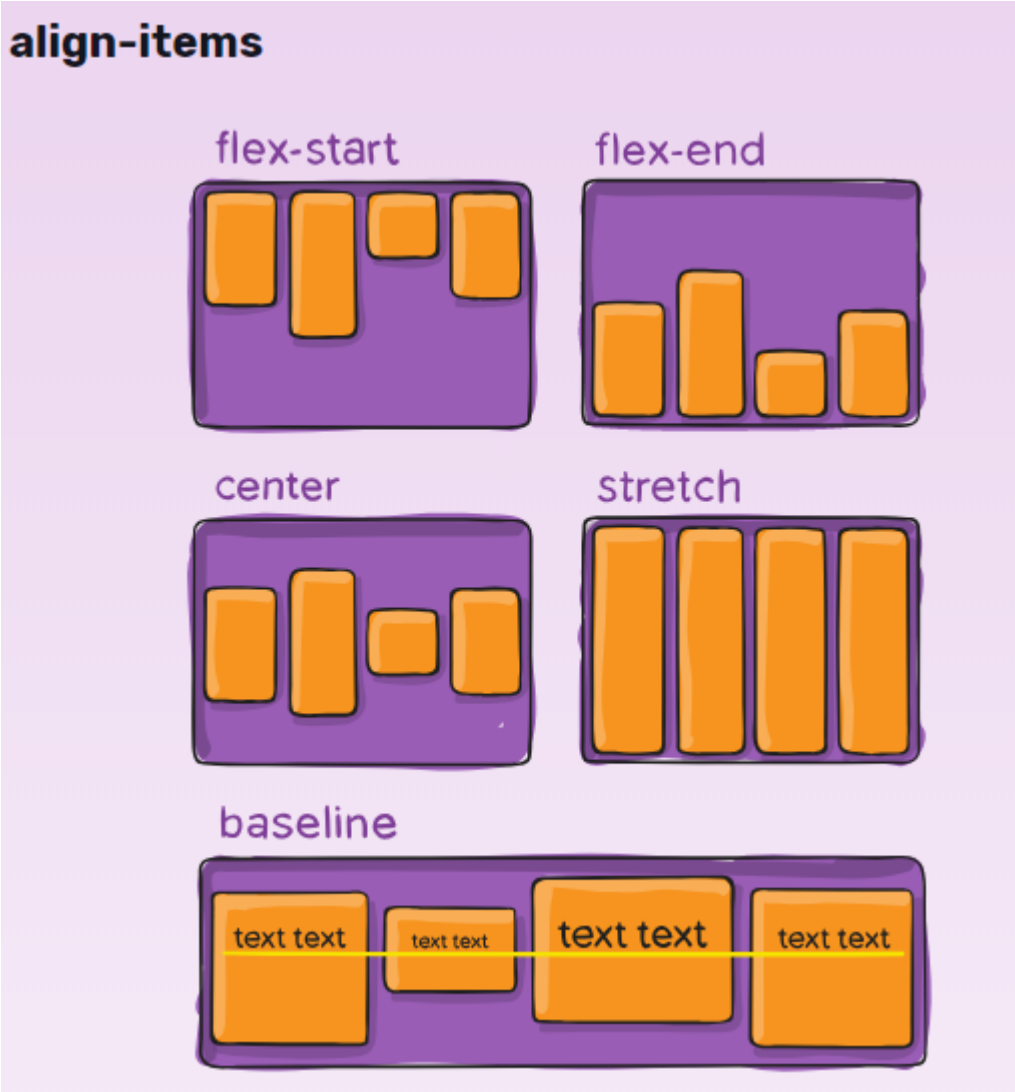


5. align-items

This defines the default behavior for how flex items are laid out along the cross axis **on the current line**. Think of it as the justify-content version for the cross-axis.

(注意区分该属性和 `align-content`)

```
.flex_box{
    align-items:flex-start | flex-end | center | stretch | baseline;
/*stretch(默认): 若flex item未设置高度或设为auto, flex item将占满整个flex container的高度*/
}
```



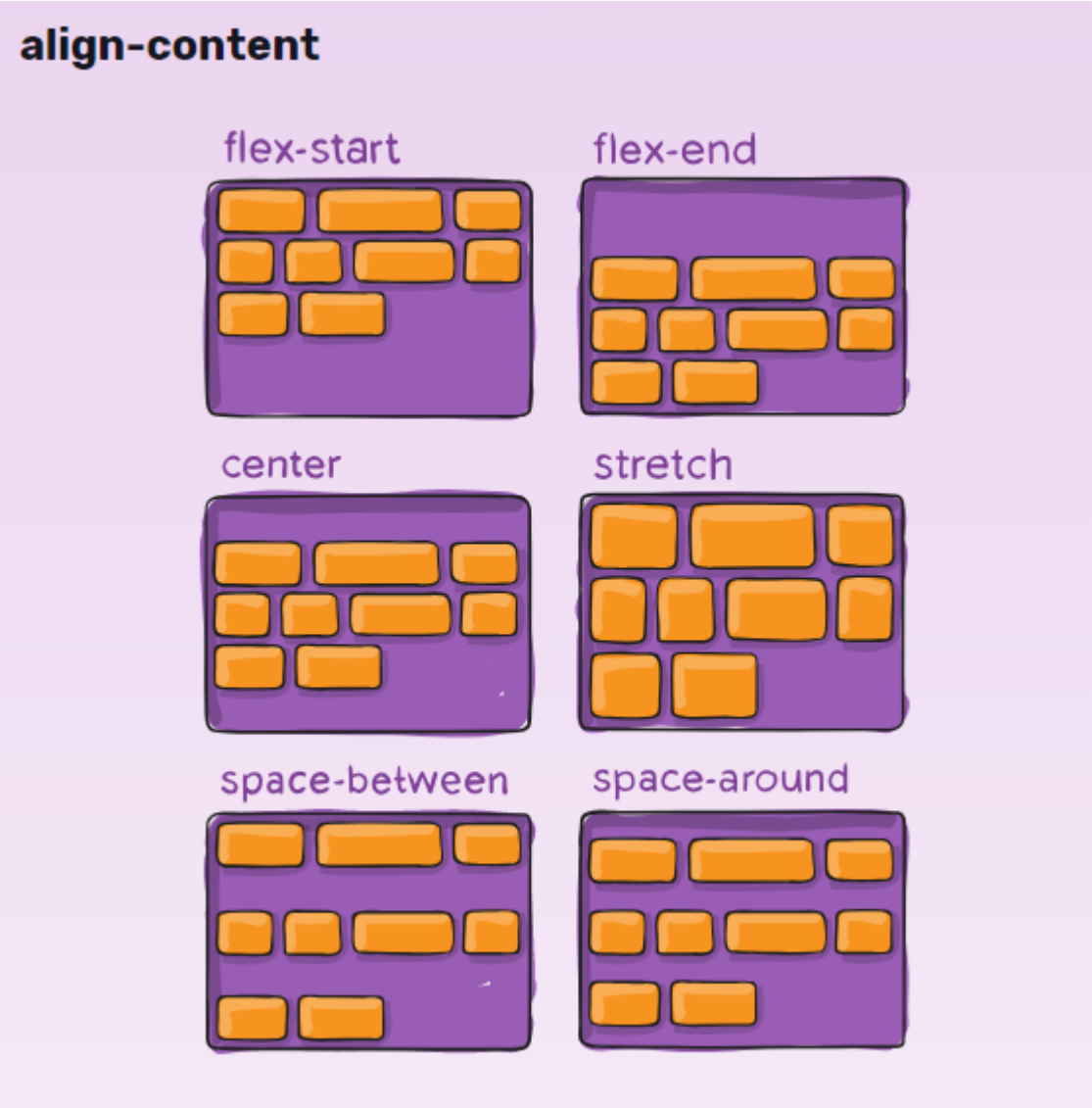
6. align-content

This aligns a flex container's lines within when there is extra space in the cross-axis, similar to how `justify-content` aligns individual items within the main-axis.

（多行在交叉轴方向上的排列方式）

Note: this property has **no effect** when there is **only one line** of flex items.

```
.flex_box{
    align-content:flex-start | flex-end | center | stretch | space-between | space-
around;
    /*stretch(默认)*/
}
```



项目属性

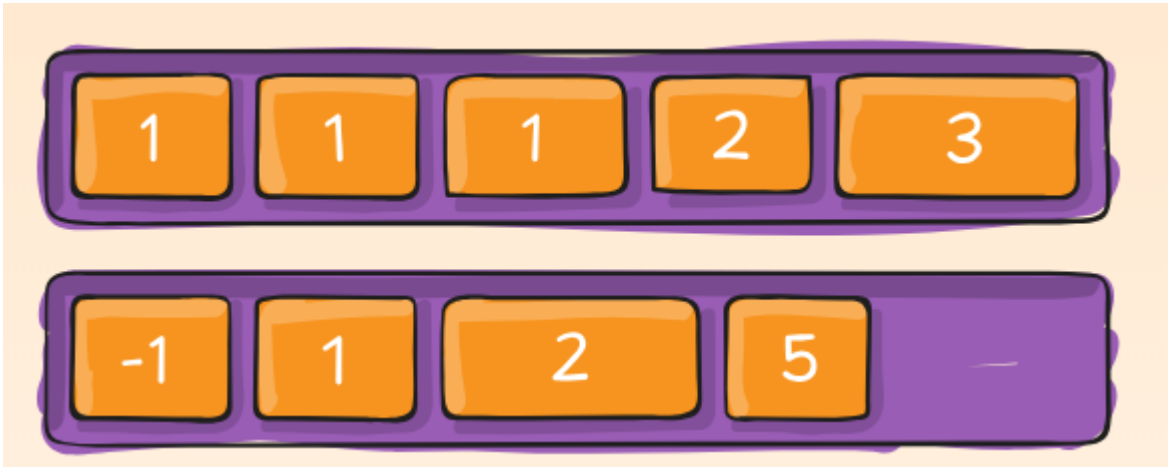
properties for flex items

> - order
> - flex-grow
> - flex-shrink
> - flex-basis
> - flex
> - align-self

1. order

By default, flex items are laid out in the source order. However, the `order` property controls the order in which they appear in the flex container.(定义flex item在主轴上的排列顺序，数值越小，排列越靠前)

```
.flex_item{
    order:3;
    /*default is 0*/
}
```



2. flex-grow

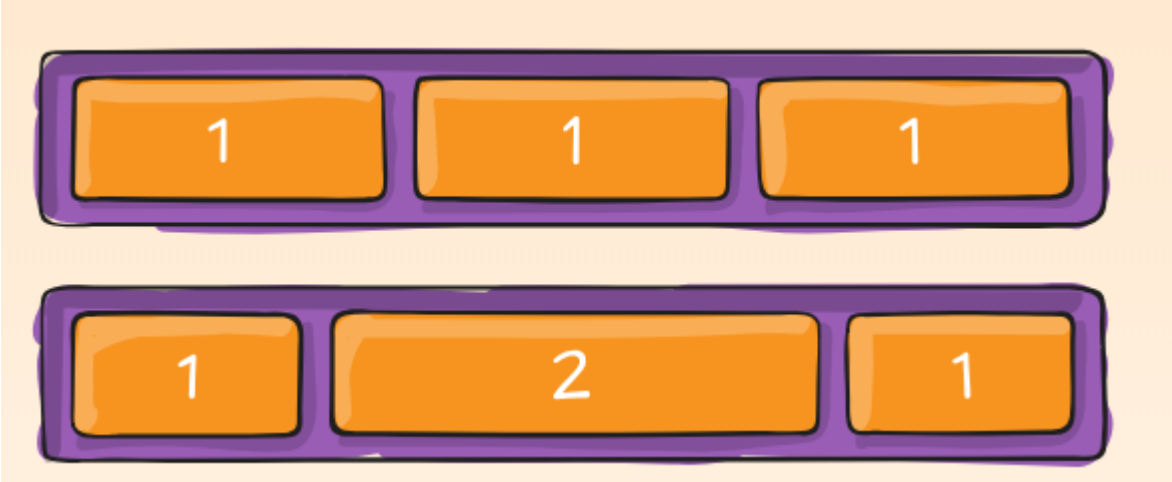
This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up.

If all items have `flex-grow` set to 1, the remaining space in the container will be di

stributed equally to all children. If one of the children has a value of 2, the remaining space would take up twice as much space as the others (or it will try to, at least).

定义flex item的放大比例，默认为0，即，若存在剩余空间，也不进行放大，关于flex布局的空间与剩余空间划分，见文末参考文献

```
.flex_item{
    flex-grow:1;
    /*default:0*/
}
```



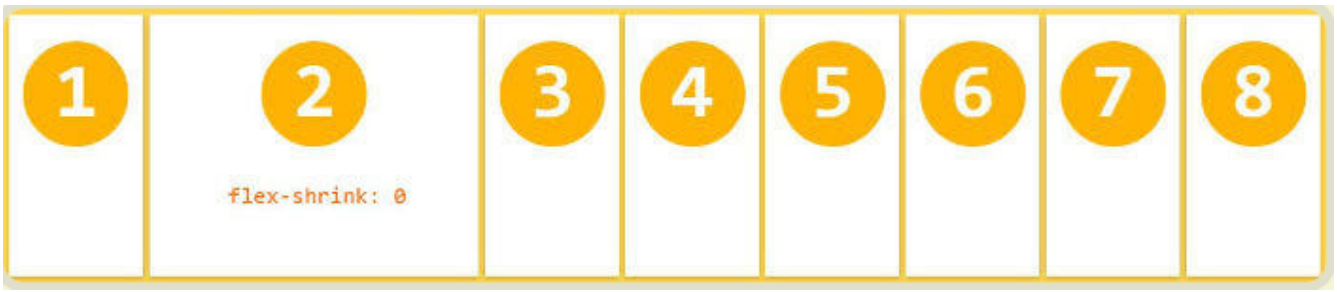
3. flex-shrink

This defines the ability for a flex item to shrink if necessary.

note:Negative numbers are invalid

定义flex item的缩小比例，默认为1，即，如果空间不足，flex item将进行缩小，负值无效

```
.flex_item{
    flex-shrink:0;
    /*default is 1*/
}
```



4. flex-basis

This defines the default size of an element before the remaining space is distributed. It can be a length (e.g. 20%, 5rem, etc.) or a keyword. The `auto` keyword means "look at my width or height property" (which was temporarily done by the `main-size` keyword until deprecated). The `content` keyword means "size it based on the item's content" - this keyword isn't well supported yet, so it's hard to test and harder to know what its brethren `max-content`, `min-content`, and `fit-content` do.

If set to `0`, the extra space around content isn't factored in. If set to `auto`, the extra space is distributed based on its `flex-grow` value.

可以理解为计算剩余空间之前，为flex item预留的空间，若设为`auto`，则根据flex item的内容来决定预留空间。

可将该属性近似理解为`width`，即为元素设置初始宽度

```
.flex_item{
    flex-basis:<length> | auto;
    /*default auto*/
}
```

5. flex

`flex-grow`,`flex-shrink`,`flex-basis`的简写（复合属性）

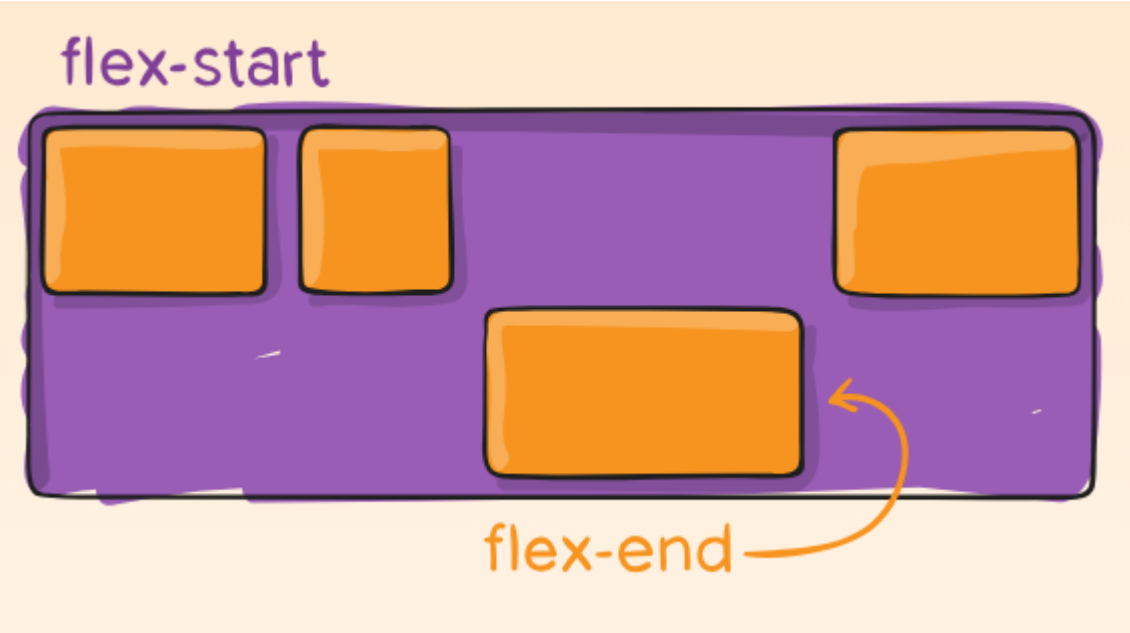
```
.flex_item{
    flex:0 1 auto;
    /*default*/
}
```

6. align-self

This allows the default alignment (or the one specified by `align-items`) to be overridden for individual flex items.

可以为单个flex item覆盖其父元素的`align-items`属性

```
.flex_item{
    align-self:flex-start | flex-end | center | stretch | baseline;
/*stretch(默认)：若flex item未设置高度或设为auto，flex item将占满整个flex container的高度*/
}
```



参考文献： 深入理解flex-grow,flex-shrink,flex-basis

Examples

1. 用flex实现顶部导航栏

(实现了响应式布局)

效果图:

(浏览器宽度较大时)

(浏览器宽度适中时)

(浏览器宽度较小时)

代码:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title></title>
```

```
    <link rel="stylesheet" type="text/css" href="test.css">
    </style>
</head>
<body>
    <ul class="nav">
        <li><a href="#">Home</a></li>
        <li><a href="#">Video</a></li>
        <li><a href="#">Article</a></li>
        <li><a href="#">Game</a></li>
        <li><a href="#">MsgBoard</a></li>
    </ul>
</body>
</html>
```

```
@navBgColor:deepskyblue;

*{
    margin:0px;
    padding:0px;
}
.nav{
    list-style: none;
    display: flex;
    flex-flow: row wrap;
    justify-content: flex-end;
    background:@navBgColor;
}

.nav a{
    display: block;
    text-decoration: none;
    color:white;
    padding:1em;
    &:hover{
        background:darken(@navBgColor, 2%);
    }
}

@media all and (max-width: 800px){
    .nav{
        justify-content: space-around;
    }
}

@media all and (max-width: 500px){
    .nav{
        flex-direction: column;
        & a{
            text-align: center;
            padding:10px;
            border-top: 1px solid rgba(255,255,255,.3);
            border-bottom:1px solid rgba(0,0,0,.1);
        }
```

```
        & li:last-of-type a{
            border-bottom:none;
        }
    }
}
```

2. 用flex实现骰子 

```
<div id="root">
        <div class="pad" id="d1">
            <div class="dot"></div>
        </div>
        <div class="pad" id="d2">
            <div class="dot"></div>
            <div class="dot"></div>
        </div>
        <div class="pad" id="d3">
            <div class="dot"></div>
            <div class="dot"></div>
            <div class="dot"></div>
        </div>
        <div class="pad" id="d4">
            <div class="row">
                <div class="dot"></div>
                <div class="dot"></div>
            </div>
            <div class="row">
                <div class="dot"></div>
                <div class="dot"></div>
            </div>
        </div>
        <div class="pad" id="d5">
            <div class="row">
                <div class="dot"></div>
                <div class="dot"></div>
            </div>
            <div class="row">
                <div class="dot"></div>
            </div>
            <div class="row">
                <div class="dot"></div>
                <div class="dot"></div>
            </div>
        </div>
        <div class="pad" id="d6">
            <div class="row">
                <div class="dot"></div>
                <div class="dot"></div>
            </div>
```

```
            <div class="row">
                <div class="dot"></div>
                <div class="dot"></div>
            </div>
            <div class="row">
                <div class="dot"></div>
                <div class="dot"></div>
            </div>
        </div>
    </div>
```

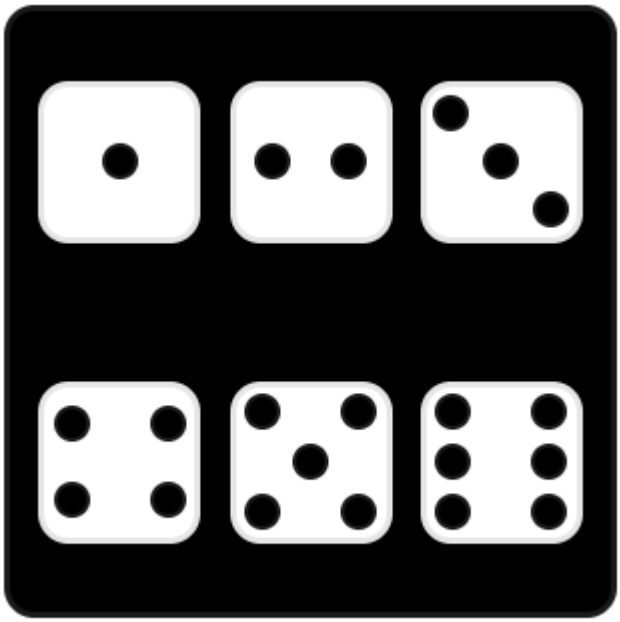
```
/*LESS文件*/
@myMargin:3px;
#root{
    display: flex;
    width:200px;
    height:200px;
    margin:auto;
    background:black;
    border:2px solid lighten(black,10%);
    border-radius:10px;
    flex-flow: row wrap;
    justify-content: space-evenly;
    align-items:center;
}
.pad{
    display:flex;
    width:50px;
    height:50px;
    background:white;
    border:2px solid darken(white,10%);
    border-radius:10px;
    flex-flow: row wrap;
    justify-content: space-around;
    align-items:center;
}
.dot{
    width:10px;
    height:10px;
    background: black;
    border:1px solid lighten(black,10%);
    border-radius:50%;
}
#d3{
    & .dot:first-of-type{
        margin-top:@myMargin;
        align-self:flex-start;
    }
    & .dot:last-of-type{
        margin-bottom:@myMargin;
        align-self:flex-end;
```

```
    }
}
#d4{
    & .row{
        display:flex;
        flex-flow: row wrap;
        flex-basis: 100%;
        justify-content: space-between;
        & .dot:first-of-type{
            margin-left:@myMargin;
        }
        & .dot:last-of-type{
            margin-right: @myMargin;
        }
    }
}

#d5{
    & .row{
        display:flex;
        flex-flow: row wrap;
        flex-basis: 100%;
        &:not(:nth-of-type(2)){
            justify-content: space-between;
            & .dot:first-of-type{
                margin-left:@myMargin;
            }
            & .dot:last-of-type{
                margin-right: @myMargin;
            }
        }
        &:nth-of-type(2){
            justify-content: center;
        }
    }
}

#d6{
    & .row{
        display:flex;
        flex-flow: row wrap;
        flex-basis: 100%;
        justify-content: space-between;
        & .dot:first-of-type{
            margin-left:@myMargin;
        }
        & .dot:last-of-type{
            margin-right: @myMargin;
        }
    }
}
```

效果图



以上Flex布局的内容参考至：CSS-Trick，以及 阮一峰的教程

grid布局初探

Definition:

CSS Grid Layout is the most powerful layout system available in CSS. It is a 2-dimensional system, meaning it can handle both columns and rows, unlike flexbox which is largely a 1-dimensional system. You work with Grid Layout by applying CSS rules both to a parent element (which becomes the Grid Container) and to that element's children (which become Grid Items).

3分钟入门grid布局

指定一个元素作为grid容器：

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

将该容器划分成什么样的网格：

```
.container{
    grid-template-columns:100px 100px 100px;
    grid-template-rows:100px 100px 100px;
}
```

放入9个items，效果图如下

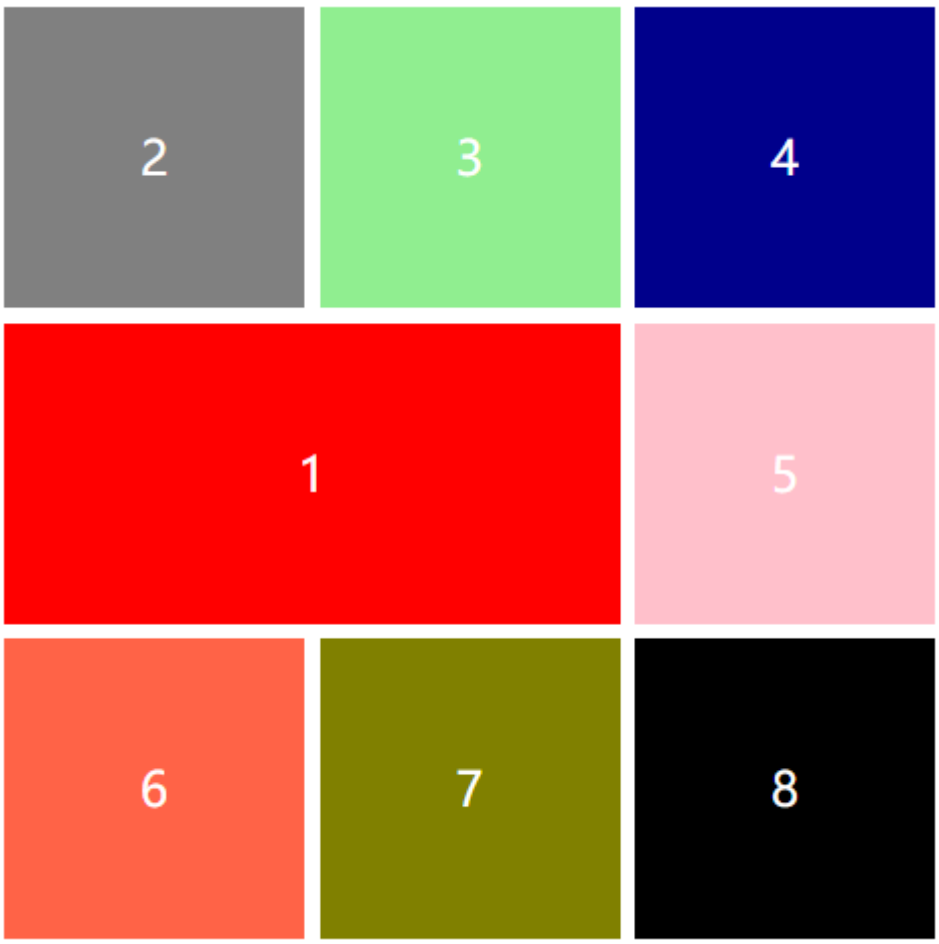
可以对每个item指定其行列的开始和结束位置

```
.item1{
    background:red;
    grid-column-start: 1;
    grid-column-end:3;
    /*可以合写成如下形式*/
    /*
    grid-column:1 / 3;
    */
    grid-row-start: 2;
    grid-row-end:3;
    /*
    grid-row:2 / 3;
    */

    /*
```

```
    注意：在less下，这样合写，会导致错误，因为sublime的less编译器会将1 / 3进行计算，最终得到的css文件里，
    结果是 grid-column:0.3333;    可以对网格线进行命名，使用名字来进行合写，less则不会报错.
        */
    }
```

效果图如下：



可以在对container声明时，对网格线进行命名，如：

```
.container {
  display: grid;
  color: black;
  height: 100%;
  grid-template-columns:[a] 100px [b] 100px [c] 100px [d];
  grid-template-rows: [i] 100px [ii] 100px [iii] 100px [iv]  ;
  grid-gap: 5px;
}
```

然后对item进行定位时，通过网格线的名字，来指定该item的column和row的始末位置

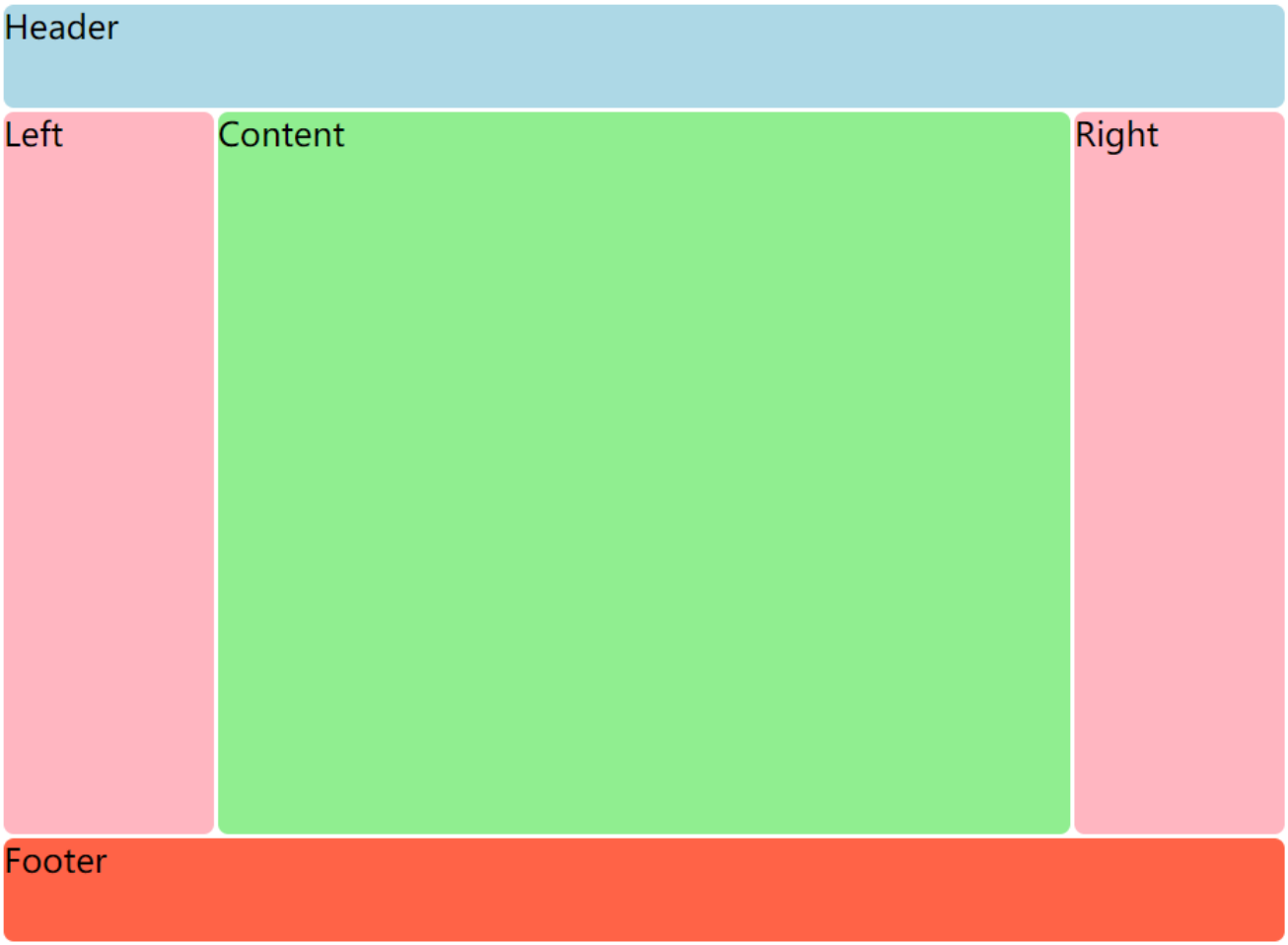
用grid实现一些经典布局

1. 圣杯布局

```
<div class="container">
        <div class="header">Header</div>
        <div class="left">Left</div>
        <div class="right">Right</div>
        <div class="content">Content</div>
        <div class="footer">Footer</div>
    </div>
```

```
.container{
    display: grid;
    grid-gap: 2px;
    grid-template-columns: repeat(12,1fr);
    /* 将列划分成12份，每份占1/12 */
    grid-template-rows: 50px 350px 50px;
    grid-template-areas:
    "h h h h h h h h h h h h"
    "l l c c c c c c c c r r"
    "f f f f f f f f f f f f";
}
.container div{
    border-radius:5px;
}
.header{
    grid-area: h;
    background:lightblue;
}
.left{
    grid-area: l;
    background: lightpink;
}
.right{
    grid-area: r;
    background: lightpink;
}
.content{
    grid-area: c;
    background: lightgreen;
}
.footer{
    background: tomato;
    grid-area: f;
}
```

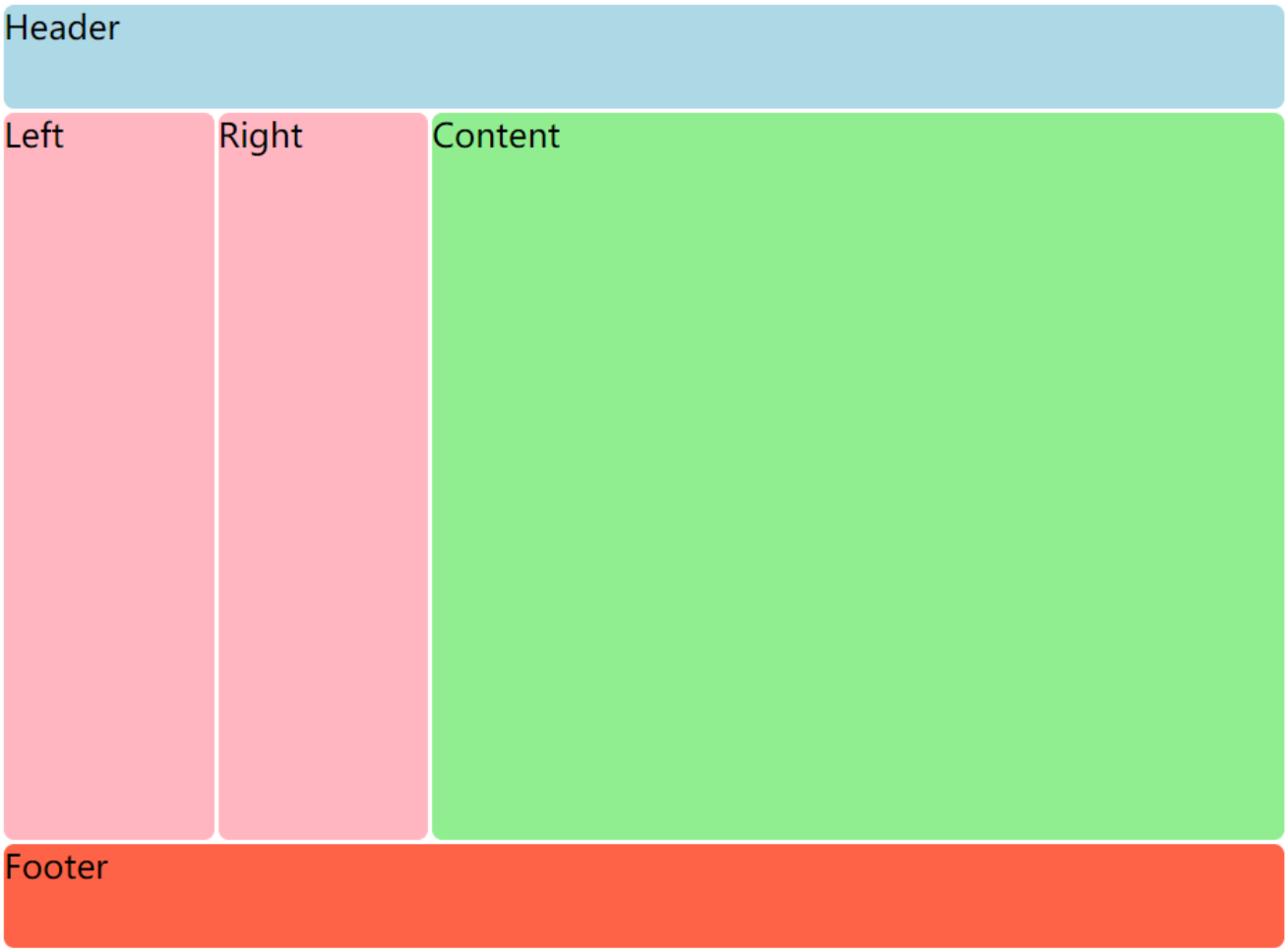
效果图:



尝试修改布局，如把left和right都移动到左边，那么只需要修改`grid-template-area`属性即可

```
.container{
    display: grid;
    grid-gap: 2px;
    grid-template-columns: repeat(12,1fr);
    grid-template-rows: 50px 350px 50px;
    grid-template-areas:
    "h h h h h h h h h h h h"
    "l l r r c c c c c c c c"
    "f f f f f f f f f f f f";
    /*将r r 拿到左边来就行了*/
}
```

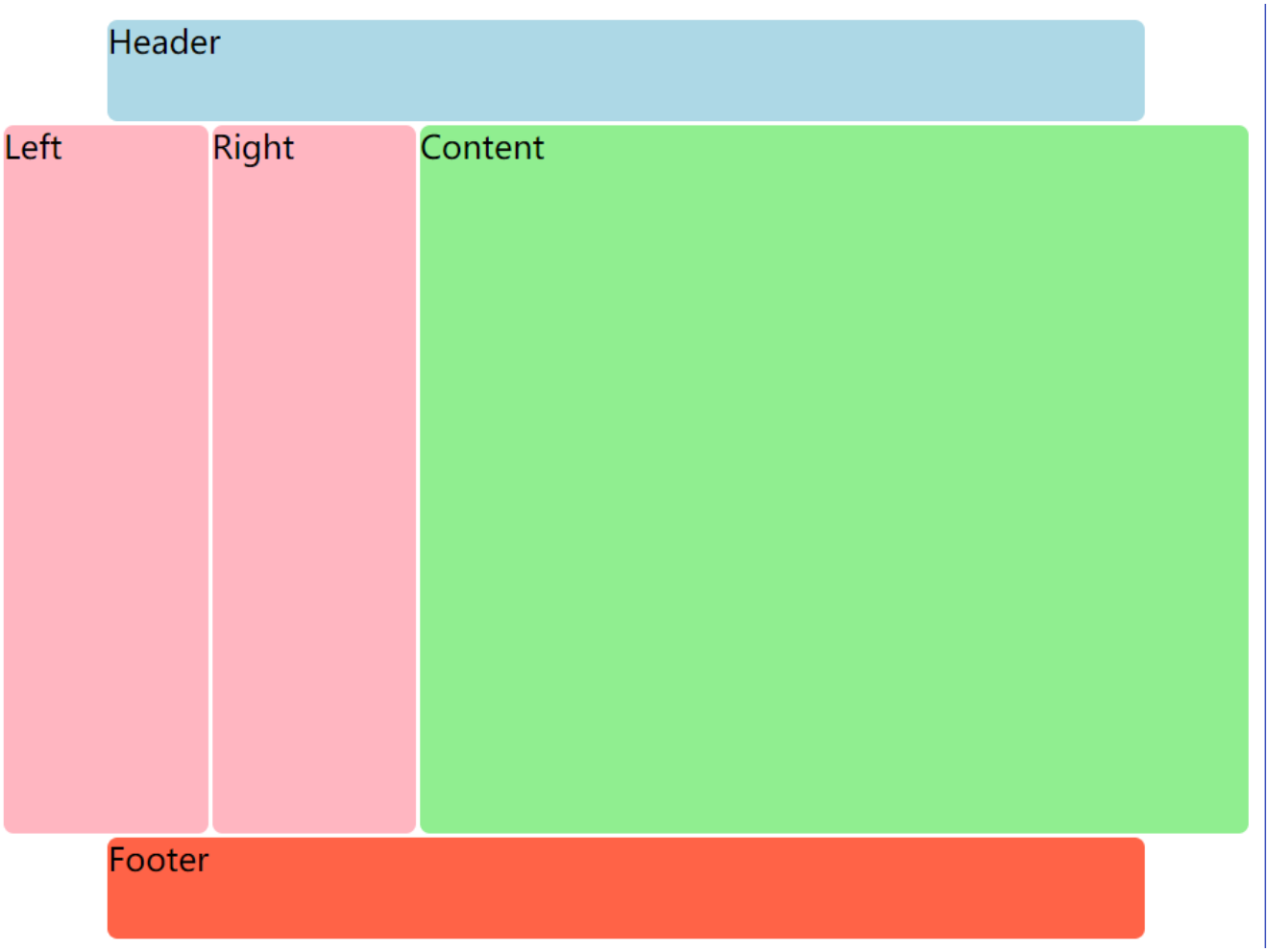
效果图如下：



也可以利用`.`来创建空白的网格单元格

```
.container{
    grid-template-areas:
    ". h h h h h h h h h ."
    "l l r r c c c c c c c"
    ". f f f f f f f f f .";
}
```

效果图如下：

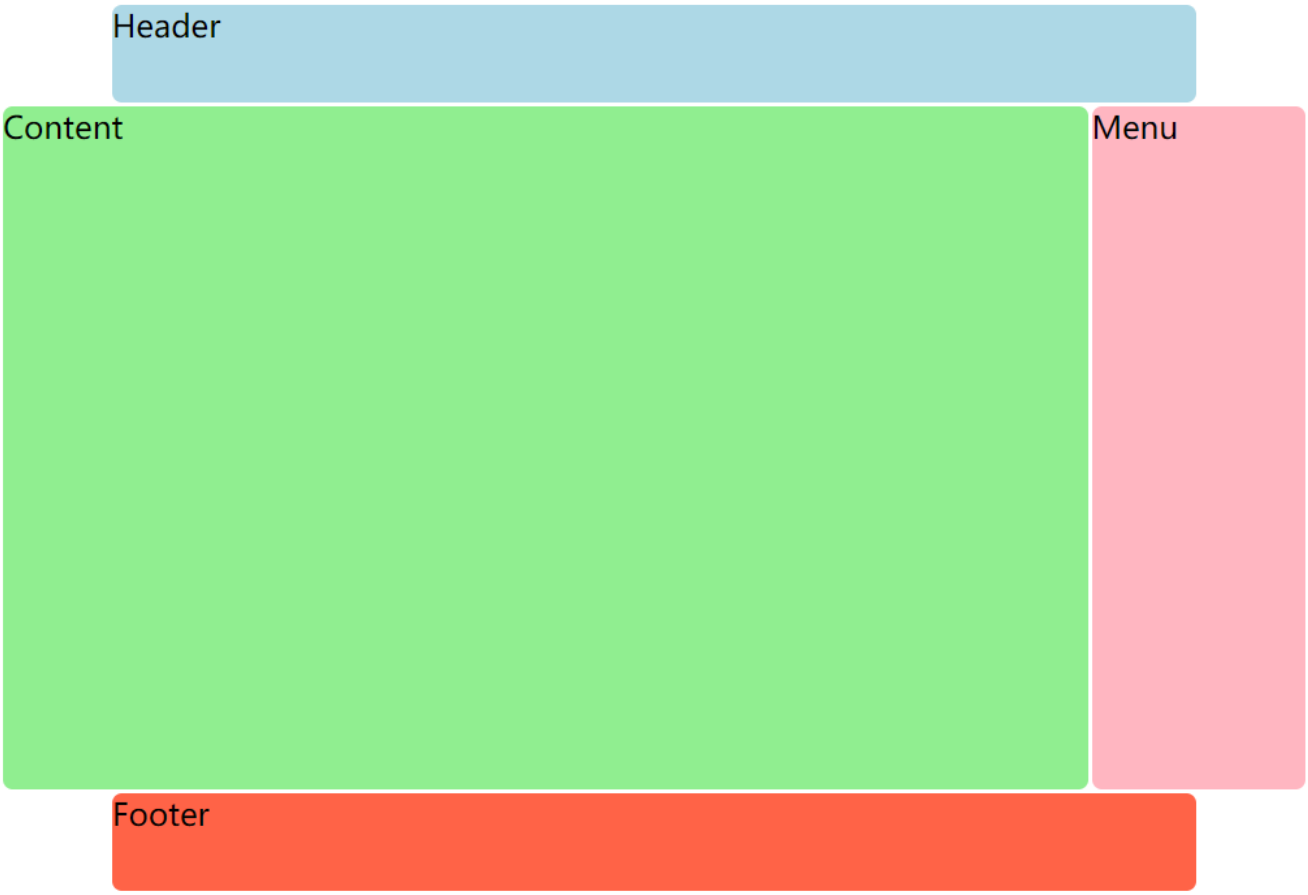


2. Grid与响应式布局

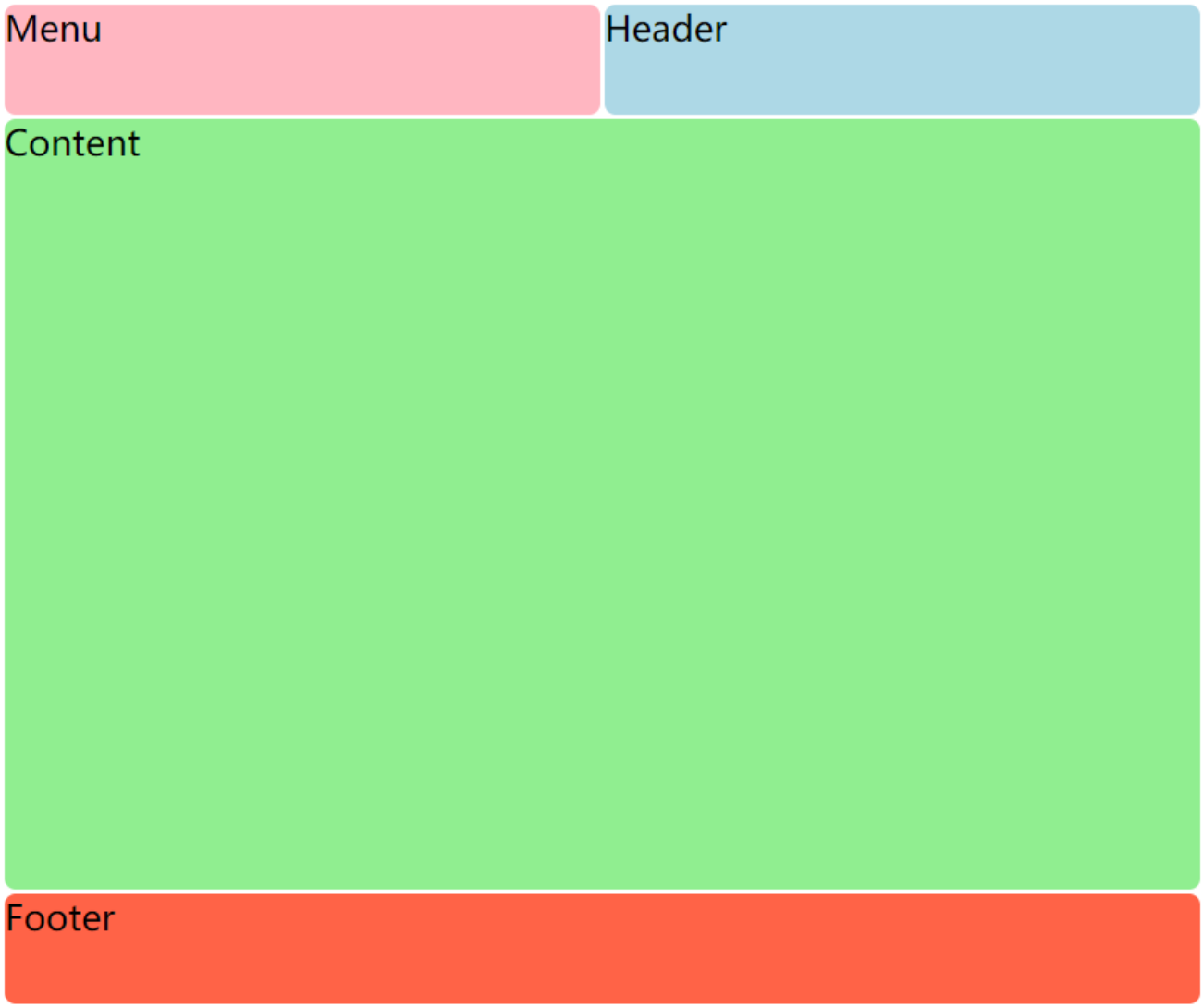
```
.container{
    display:grid;
    grid-gap:5px;
    grid-template-columns:repeat(12,1fr);
    grid-template-rows:50px 350px 50px;
    grid-template-areas:
        ". h h h h h h h h h h ."
        "c c c c c c c c c c m m"
        ". f f f f f f f f f f .";
}
@media screen and (max-width:640px){
    .container{
        grid-template-areas:
            "m m m m m m h h h h h h"
            "c c c c c c c c c c c c"
            "f f f f f f f f f f f f";
    }
}
```

效果图如下：

可显示区域宽度大于640px时：



可显示区域宽度小于等于640px时：



所有的更改都是通过纯CSS完成的，无需修改HTML，而且grid里的item排在什么位置（哪个网格），我们完全可以在CSS里随意调整（通过`grid-template-areas`），HTML文档中各个item的源顺序无关紧要。通过**媒体查询**和**Grid布局**，使得*结构*和*样式*分离，非常牛逼。

Grid布局的以上内容参考至：这里

关于Grid更多更完整的信息

见 Grid布局完全指南，CSS-Trick的Grid完全指南(英文版)