- **移动端基础**
  - **浏览器现状**
  - 手机屏幕现状
  - 移动端调试方法
- **视口**
  - meta视口标签
- **二倍图**
  - 物理像素&物理像素比
  - 多倍图
  - 背景缩放 background-size
  - 解决移动端图片模糊问题
  - 多倍图切图工具
- **移动端技术解决方案**
  - CSS初始化nomalize.css
  - CSS3盒子模型 box-sizing
  - 特殊样式
- **移动端开发的选择**
  - 单独制作移动端页面（主流）
  - 响应式页面兼容移动端（其次）
- **流式布局（百分比布局）**
- **flex布局**
  - 布局原理
  - 常见父项元素
    - flex-direction
    - justify-content
    - flex-wrap
    - align-items（单行）
    - align-content（多行）
    - flex-flow
  - 常见的子项元素
    - flex
    - align-self
    - order
- **居中对齐总结**
- **行内元素水平居中**
  - text-align: center;
  - fit-content
- **行内元素和行内块元素垂直居中**
- **块级元素水平居中**
- **块级元素水平垂直居中**
  - 定位方法
    - 方法一：必须知道宽高
    - 方法二：不必知道宽高
    - 方法三：margin配合定位
    - padding 方法
      - 方法一：给父元素加 padding
      - 方法二：给子元素加 padding
    - flex 布局方法
    - 伪元素方法

css属性书写顺序

1. 布局定位属性：display / position / float / clear / visibility / overflow
2. 自身属性：width / height / margin / padding / border / background
3. 文本属性：color / foont / text-decoration / text-align / vertical-align / white-space / break-word
4. 其他属性（CSS3）：content / cursor / border-radius / box-shadow / text-shadow /background:linear-gradient...

选择器

基础选择器

1. 标签选择器：`div {}`
2. 类选择器：`.header {}`
3. id选择器：`#box {}`
4. 通配符选择器：`* {}`

复合选择器

1. 后代选择器：`.header p {}`
2. 子代选择器：`.header > p {}`
3. 并集选择器：`.header, #box {}`
4. 伪类选择器：`a:link{未点击的链接} a:visited{访问过的} a:hover{鼠标经过} a:active{鼠标按下还未抬起} input:focus{获取到焦点的表单}`

伪类选择器具有使用顺序否为无效，link>visited>hover>active

font text background属性

属性	表示	注意点
font-size	字号	通常单位是px像素
font-family	字体	字体之间用逗号隔开，如果字体名字有空格需要用单引号
font-weight	字体粗细	加粗是700或blod，不加粗是normal或400，数字后面不加单位
font-style	字体样式	斜体是italic，不倾斜是normal
font	字体复合属性连写	属性连写有顺序style>weight>size/height>family。其中字号和字体必须出现，否则无效。
color	文本颜色	通常用十六进制，可以写简写，如#aa00ff可以写成#a0f
test-align	文本对齐	可以设定文字水平对齐方式
text-indent	文本缩进	通常用于段落缩进两个字的距离 2em
text-decoration	文本修饰	添加下滑线和取消下划线
line-height	行高	控制行与行之间的距离，其中又上边距，字体大小，下边距组成

属性	表示	注意点
background-color	背景颜色	预定义的颜色值/十六进制/RGB代码
background-image	背景图片	url(图片路径)
background-repeat	是否平铺	repeat/no-repeat/repeat-x/repeat-y
background-position	背景位置	length/position 分别是 x 和 y 坐标
background-attachment	背景附着	scroll（背景滚动）/fixed（背景固定）
背景简写	书写更简单	背景颜色 背景图片地址 背景平铺 背景滚动 背景位置;
背景色半透明	背景颜色透明	background: rgba(0,0,0,0.3); 后面必须是4个值

元素的显示模式

元素的显示模式就是*元素以生么方式进行显示*，比如`<div>`自己占一行，比如一行可以放多个`<span>`。

HTML元素一般分为**块元素**和**行内元素**。

块元素

常见的块元素有`<h1>~<h6>,<p>,<div>,<ul>,<li>`等，其中`<div>`是最典型的块元素。

特点

- 块元素独占一行。
- 高度、宽度、外边距以及内边距都可以控制。
- 宽度默认是容器（父级元素）的100%。
- 是一个容器及盒子，里面可以放行内或块级元素。

注意

- 文字类的元素内不能使用块元素。

行内元素

常见的行内元素有`<a>,<strong>,<b>,<em>,<i>,<del>,<s>,<ins>,<u>,<span>`等，其中`<span>`标签是最典型的行内元素。有的地方也将行内元素称为内联元素。

特点

- 相邻的行内元素在一行上，一行可以显示多个。
- 高、宽直接设置是无效的。
- 默认宽度就是他本身内容的宽度。
- 行内元素只能容纳文本或其他行内元素。

注意

- 链接里面不能放链接。
- 特殊情况链接`<a>`里面可以放块级元素，但是需要转换为块级模式。

行内块元素

在行内元素中有几个特殊的标签，`<img/>,<input/>,<td>`他们同时拥有块元素和行内元素的特点。

特点

- 相邻行内元素在一行上，但是他们之间会有空白缝隙。一行可以显示多个。
- 默认宽度就是它本省内容的宽度。
- 高度，行高，外边距以及内边距都可以控制。

CSS三大特性

层叠性

相同的选择器设置相同的样式，此时一个样式就会**覆盖**另一个样式。解决了样式冲突的问题。

层叠性原则

- 样式冲突，遵循的原则是**就近原则**，哪个样式离结构近，就执行那个样式
- 样式不冲突，不会层叠

继承性

CSS中子标签会继承父标签的某些样式，如文本颜色和字号。简单的理解就是：子承父业。

- 恰当的使用继承可以简化代码，降低CSS样式的复杂性
- 子元素可以继承父元素的样式（text-，font-，line-这些元素开头的可以继承，以及color属性）

优先级

当同一个元素指定多个选择器，就会有优先级产生。

- 选择器相同则执行层叠性
- 选择器不同，则根据**选择器权重**执行

选择器	**选择器权重**
继承 或者 *	0，0，0，0
元素选择器	0，0，0，1
类选择器，伪类选择器	0，0，1，0
ID选择器	0，1，0，0
行内样式 style=“”	1，0，0，0
!important 重要的	无穷大

注意

1. 继承到的父标签中的样式的权重，在子标签这里永远是0
2. a标签在浏览器中会被浏览器默认定制一个样式，所以如果a标签继承父类样式会被浏览器默认的样式覆盖掉。

复合选择器的权重叠加

当使用复合选择器时i，这个选择器的权重等于复合选择器中的几个选择器的权重的和。

- div ul li ——————————————> 0,0,0,3
- .nav ul li ——————————————->0,0,1,2
- a:hover ——————————————>0,0,1,1

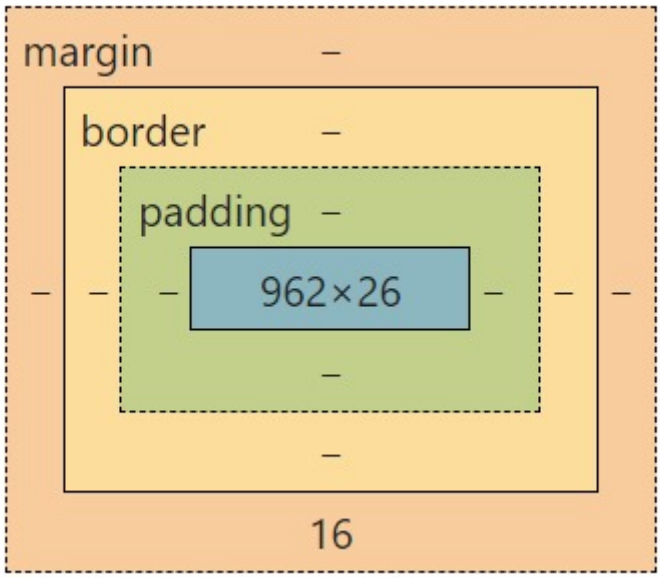
盒子模型

盒子模型组成

盒子模型就是吧HTML页面中的布局元素看作是一个矩形盒子，也就是一个承装内容的容器。

CSS盒子模型的本质就是一个盒子，封装周围的HTML元素，它包括：边距(border)、外边距(margin)、内边距(padding)和实际内容(content)。

- **Margin(外边距)** - 清除边框外的区域，外边距是透明的。
- **Border(边框)** - 围绕在内边距和内容外的边框。
- **Padding(内边距)** - 清除内容周围的区域，内边距是透明的。
- **Content(内容)** - 盒子的内容，显示文本和图像。



标准盒子模型和怪异盒子模型

在盒子模型中有两种分别是标准盒子模型和怪异盒子模型（IE盒子模型）他们盒子的范围有着不同的计算方式。

标准盒子模型：`box-sizing: content-box`

如果给定了width，这个width默认给的是内容大小。

width = content

盒子范围 = width + padding + border + margin。

怪异盒子模型：`box-sizing: border-box`

如果给定了width，这个width默认给的是content + padding + border的大小。

width = content + padding + border

盒子范围 = width + margin

边框（border）

border可以设置元素的边框。边框有三部分组成：边框宽度（粗细、边框样式、边框颜色。

属性	作用
border-width	定义边框粗细，单位是px
border-style	边框样式
border-color	边框颜色
border	复合写法，没有固定顺序
border-top	上边框复合写法
border-bottom	下边框复合写法
border-right	右边框复合写法
border-left	左边框复合写法
border-collapse	合并相邻的边框

border-style

- `dotted` - 定义点线边框
- `dashed` - 定义虚线边框
- `solid` - 定义实线边框
- `double` - 定义双边框
- `groove` - 定义 3D 坡口边框。效果取决于 border-color 值
- `ridge` - 定义 3D 脊线边框。效果取决于 border-color 值
- `inset` - 定义 3D inset 边框。效果取决于 border-color 值
- `outset` - 定义 3D outset 边框。效果取决于 border-color 值
- `none` - 定义无边框
- `hidden` - 定义隐藏边框

边框会影响盒子的实际大小

边框会额外增加盒子的实际大小。因此我们有两种解决方案：

1. 测量盒子的大小的时候，不测量边框。
2. 如果测量的时候包含了边框，则需要width/height减去边框宽度。

内边距（padding）

padding 用于设置内边距，即边框与内容之间的距离。

属性	作用
padding-left	左内边距
padding-right	右内边距
padding-top	上内边距
padding-bottom	下内边距
padding：5px	一个值，上下左右内边距都有5像素
padding：5px 10px	两个值，上下内边距是5像素，左右内边距是10像素
padding：5px 10px 20px	三个值，上内边距5像素，左右内边距10像素 下内边距20像素
padding：5px 10px 20px 30px	四个值，上5像素 右10像素 下20像素 左30像素 顺时针

内边距会影响盒子的实际大小

当我们给盒子指定 padding 值后，发生了两件事：

1. 内容和边框有了距离，添加了内边距。
2. padding 影响了盒子实际大小。

也就是说，如果盒子已经有了宽度和高度，此时再指定内边距，会撑大盒子。

解决方案

如果保证盒子跟效果图大小保持一致，则让 **width/height 减去多出来的内边距大小**即可

如果盒子本身没有指定width/height属性，则此时padding不会撑开盒子大小

外边距（margin）

margin 属性用于设置外边距，即控制盒子和盒子之间的距离。

属性	作用
margin-left	左外边距
margin-right	右外边距
margin-top	上外边距
margin-bottom	下外边距

maigin 的复合写法和 padding 完全一致。

块级盒子水平居中对齐

外边距可以让块级盒子水平居中，但是必须满足两个条件：

1. 盒子必须指定了宽度（width）。
2. 盒子左右的外边距都设置为 auto。

```
.header {width: 960px; margin: 0 auto;}
```

常见的写法，以下三种都可以

- margin-left: auto; margin-right: auto;
- margin: auto;
- margin: 0 auto;

注意：以上方法是让块级元素水平居中，行内元素或者行内块元素水平居中给其父元素添加 text-align:center 即可

嵌套块元素垂直外边距的坍塌

对于两个嵌套关系（父子关系）的块元素，父元素有上外边距同时子元素也有上外边距，此时父元素会塌陷较大的外边距值。

解决方案

1. 可以为父元素定义上边框。
2. 可以为父元素定义上内边距。
3. 可以为父元素添加 overflow: hidden。

还有其他方法，比如浮动、固定，绝对定位的盒子不会有塌陷问题，后期再总结。

清除内外边距

网页元素中有很多默认的内外边距，不同浏览器的默认也不一致。因此我们在布局前，首先要清楚网页元素的默认内边距。

```
* {
    padding: 0;
    margin: 0;
}
```

注意：行内元素为了照顾兼容性，尽量只设置左右内外边距，不要设置上下内外边距。但是转换为块级和行内块元素就可以了

圆角边框

CSS3中新增了圆角边框样式，这样我们的盒子就可以设置圆角。

border-radius 属性用于设置元素的外边框圆角。

radius 半径（园的半径）原理：园与边框的交集形成圆角效果。

```
border-radius: length;
```

- 参数值可以为**数值**或**百分比**的形式
- 如果是正方形，想要设置一个圆，把数值修改为高度或者宽度的一半即可，或者直接写为50%
- 如果是个 矩形，设置为高度的一般就可以做出两边半圆。
- 该属性是一个简写属性，可以跟四个值，分别代表左上角、右上角、右下角、左下角

盒子阴影

CSS3中新增了盒子阴影模型，我们可以使用 box-shadow 属性为盒子添加阴影。

语法：

```
box-shadow: h-shadow v-shadow blur spread color inset
```

值	描述
h-shadow	必须。水平阴影的位置。允许负值。
v-shadow	必须。垂直阴影的位置。允许负值。
blur	可选。模糊距离。
spread	可选。阴影的尺寸。
color	可选。阴影的颜色。请参阅CSS颜色值。
inset	可选。将外部阴影（outset）改为内部阴影。

注意：

1. 默认的是外阴影（outset），但是不可以写这个单词，否则导致阴影无效

2. 盒子阴影不占用空间，不会影响其他盒子排列

文字阴影

在CSS3中，可以使用text-shadow属性将阴影应用于文本。

语法：

```
text-shadow: h-shadow b-shadow blur color;
```

值	描述
h-shadow	必须。水平阴影的位置。允许负值。
v-shadow	必须。垂直阴影的位置。允许负值。
blur	可选。模糊距离。
color	可选。阴影的颜色。请参阅CSS颜色值。

传统网页布局的三种模式

网页布局的本质就是用CSS来摆放盒子。把盒子摆放到相应的位置。

CSS 提供了三种传统布局方式 （通俗说来就是盒子如何进行排序）

- 标准流（普通流/文档流）
- 浮动
- 定位

标准流

所谓的标准流：就是标签按照规定好默认方式排列。

1. 块级元素会独占一行，从上向下顺序排列。

- 常用元素：div、hr、p、h1~h6、ul、ol、dl、form、table

2. 行内元素会按照顺序，从左到右顺序排列，碰到父元素边缘则自动换行。

- 常用元素：span、a、i、em 等

以上都是标准流布局，我们前面学习的就是标准流，标准流是最基本的布局方式。

这三种布局方式都是用来摆放盒子的，盒子摆放到合适位置，布局自然就完成了。

注意：实际开发中，一个页面基本都包含了这三种布局方式（后面移动端学习新的布局方式）。

浮动

有很多的布局效果，标准流没有办法完成，此时就可以利用浮动完成布局。因为浮动可以改变元素标签默认的排列方式。

浮动最典型的应用：可以让多个块级元素一行内排列显示。

网页布局第一准则：多个块级元素纵向排列找标准流，多个块级元素横向排列找浮动

float 属性用于创建浮动框，将其移动到一边，知道左边缘或者右边缘触及包含块或另一个浮动框的边缘。

语法：

```
选择器 { float: 属性值;}
```

属性	描述
none	元素不浮动（默认值）
left	元素向左浮动
right	元素向右浮动

浮动特性

加了浮动之后的元素，会具有很多特性，需要我们装我的。

1. 浮动元素会脱离标准流（脱标）
    - 脱离标准流的控制（浮）移动到指定位置（动），（俗称脱标）
    - 浮动的盒子不再保留原先的位置，其它盒子会占用这个位置

    注意：浮动的元素是相互贴靠在一起的（不会有缝隙），如果父级宽度装不下这些浮动的盒子，多出的盒子会另起一行对齐

2. 浮动的元素会一行内显示并且元素顶部对齐。

3. 浮动的元素会具有行内块元素的特性。
    - 任何元素都可以浮动。不管原来是什么模式的元素，添加浮动之后具有行内块元素的相似的特性。
    - 如果块级盒子没有设置宽度，默认宽度和父级一样宽，但是添加浮动后，它的大小根据内容来决定。
    - 浮动的盒子中间是没有缝隙的，是紧挨着一起的。
    - 行内元素同理

浮动元素经常和标准流的父元素搭配使用

为了约束浮动元素的位置，我们的网页布局一般采用的策略是：

先用标准流的父元素排列上下位置，之后内部子元素采取浮动排列左右位置，复合网页布局第一准则。

浮布局注意点

1. 浮动和标准流的父盒子搭配

先用标准流的父元素排列上下位置，之后内部子元素采用浮动排列左右位置。

2. 一个元素浮动了，理论上其余的兄弟元素也要浮动。

一个盒子里面有多个子盒子，如果其中一个盒子浮动了，那么其他兄弟也应该浮动，以防止发生问题。

浮动的盒子只会影响浮动盒子后面的标准流，不会影响前面的标准流。

清除浮动

由于父级盒子很多情况下，不方便给高度，但是子盒子浮动又不占有位置，最后父级盒子高度为0时，就会影响下面的标准流盒子。

为什么需要清除浮动

1. 父级没有高度
2. 子盒子浮动了
3. 影响下面布局了，我们就应该清除浮动了

清除浮动的本质

- 清除浮动的本质是清除浮动元素造成的影响
- 如果父盒子本身有高度，则不需要清除浮动
- 清除浮动之后，父级就会根据浮动的子盒子自动检测高度。父级有了高度，就不会影响下面的标准流了。

语法：

```
选择器 {
    clear: 属性值;
}
```

属性值	描述
left	不允许左侧有浮动元素（清除左侧浮动的影响）
right	不允许右侧有浮动元素（清除右侧浮动的影响）
both	同时清除左右两侧浮动的影响（常用）

清除浮动的策略是：闭合浮动。

清除浮动的方法

1. 额外标签法（隔墙法）

额外标签法会在浮动元素的末尾添加一个空的标签。例如`<div style="clear:both"></div>`，或者其他标签如`</br>`等。

- 优点：通俗易懂，书写方便
- 缺点：添加许多无意义的标签，结构化较差

注意：新增的标签必须是块级元素

2. 父级添加 overflow

可以给父级添加 overflow 属性，将其属性设置为 hidden、auto 或 scroll 。

- 优点：代码简洁
- 缺点：无法显示溢出的部分

3. after 为元素法

:after 方式是额外标签法的升级版。也是给父元素添加。

```
.clearfix:after {
    content: "";
    display: block;
    herght: 0;
    clear: both;
    visibility: hidden;
}

.clearfix {
    /* ie6、ie7* 清除浮动*/
    *zoom: 1;
}
```

- 优点：没有增加标签，结构简单
- 缺点：照顾低版本浏览器

4. 双为元素清除浮动

```
.clearfix:before,
.clearfix:after {
    content: "";
    display: table;
}
```

```
.clearfix:after {
    clear: both;
}

.clearfix {
    /* 兼容低版本ie6、ie7* 清除浮动*/
    *zoom: 1;
}
```

定位

为什么需要定位

有些元素的位置比较刁钻，使用标准流和浮动很难实现。定位为我们解决了这个困难。

定位可以让某一元素自由的在一个盒子中移动位置或者固定屏幕中某个位置，并且压住其他盒子。

定位组成

定位：将盒子定在某一个位置，所以定位也是在摆放盒子，按照定位的方式移动盒子。

定位 = 定位模式 + 边偏移

定位模式用于指定一个元素在文档中的定位方式。边偏移则决定了该元素的最终位置。

边偏移

边偏移就是定位盒子移动到最终位置。有`top、bottom、left、right`四个属性。

边偏移属性	示例	描述
top	top: 80px	**顶端**偏移量，定位元素相对于父元素**上边线的距离**
bottom	bottom: 80px	**底部**偏移量，定位元素相对于父元素**下边线的距离**
left	left: 80px	**左侧**偏移量，定位元素相对于父元素**左边线的距离**
right	right: 80px	右侧偏移量，定位元素相对于父元素右边线的距离

注意：left和right同时存在时并不冲突，不会层叠。当两个属性同时存在时优先选择left。同理top和bottom优先选择top。

定位模式

定位模式决定元素的定位方式，它通过CSS的`position`属性来设置，一共有四个可选值：

值	语义
static	静态定位
relative	相对定位
absolute	绝对定位
fixed	固定定位

静态定位

静态定位是元素的默认定位方式，就是无定位的意思。

语法：

```
选择器 { position: static; }
```

- 静态定位按照标准流特性摆放位置，它没有边偏移
- 静态定位在布局中很少使用

相对定位（重要）

相对定位是元素在移动位置的时候，是**相对于它原来的位置来说**的（自恋型）

语法：

```
选择器 { position: relative; }
```

相对定位的特点：（务必记住）

1. 他是相对于自己原来的位置来移动的（**移动位置的时候参照点是自己原来的位置**）。
2. 原来在标准流的位置继续占有，后面的盒子仍然以标准流的方式对待它。（**不脱标，继续保留原来位置**）

因此，相对定位并没有脱标。它最典型的应用是给绝对定位当爹的。

绝对定位 absolute （重要）

绝对定位是元素在移动位置的时候，是相对于它祖先元素来说的（拼爹型）。

语法：

```
选择器 { position: absolute; }
```

绝对定位的特点：（务必记住）

1. 如果没有祖先元素或者祖先元素没有定位，则以浏览器为准定位（Document文档）。
2. 如果祖先元素有定位（相对、绝对、固定定位），则以最近一级的有定位的祖先元素为参考点移动位置。
3. **绝对定位不再占有原来的位置。（脱标）**

子绝父相

子绝父相非常重要，意思是：子级使用绝对定位的话，父级要用相对定位

1. 子级绝对定位，不会占有位置，可以放到父盒子里面的任何一个地方，不会影响其他盒子。
2. 父盒子需要加定位限制子盒子在父盒子内显示。
3. 父盒子布局时，需要占有位置，因此父亲只能是相对定位。

这就是子绝父相的由来，所以相对定位经常用来作为绝对定位的父级。

总结：因为父级需要占有位置，因此是相对定位，子盒子不需要占有位置，则是绝对定位。

绝对定位盒子居中

加了绝对定位的盒子不能通过 margin: 0 auto; 水平居中，但是可以通过下面的计算方法实现水平和垂直居中。

方法：

1. 先给盒子增加绝对定位
2. 让盒子的绝对定位走页面宽度的一半
3. 让盒子往左边走自己的一半

```
.box {
    position: absolute;
    left: 50%;
    margin-left: -100px
    width: 200px;
    height: 200px;
    background-color: pink;
}
```

固定定位

固定定位是元素固定于浏览器可视区的位置。

主要使用场景：**可以在浏览器页面滚动时元素的位置不会改变。**

语法：

```
选择器 {position: fixed; }
```

固定定位的特点：（务必记住）

1. 以浏览器的可视窗口为参照点移动元素。
    - 跟父元素没有任何关系
    - 不随滚动条的滚动而滚动
2. 固定定位不占有原来的位置
    - **固定定位也是脱标的，其实固定定位也可以看作是一种特殊的绝对定位。**

固定定位小技巧 贴近版心右侧

小算法：

```
1. 让固定定位的盒子 left：50%，走到浏览器可视区（也可以看作版心）的一半位置。
2. 让固定定位的盒子 margin-left: 版心宽度的一半巨鹿。多走 版心宽度的一般位置。
```

就可以让固定定位的盒子贴着版心对齐了。

粘性定位

粘性定位可以被认为是相对定位和固定定位的混合。Sticky粘性的：

语法：

```
选择器 {position: sticky; top: 10px; }
```

粘性定位的特点：

1. 以浏览器的可视窗口为参照点移动元素（固定定位特点）
2. 粘性定位占有原来的位置（相对定位特点）
3. 必须添加 top、left、bottom、right 其中一个才有效

不常用，兼容性差，ie不支持，常用的类似效用使用JS来完成

定位的叠放次序 z-index

在使用定位布局时，可能会出现盒子重叠的情况。此时可以使用 `z-index` 来控制盒子的前后顺序（z轴）

语法：

```
选择器 { z-index: 1; }
```

- 数值可以是正整数、负整数或0，默认是auto，数值越大，盒子越靠上。
- 如果属性相同，则按照书写顺序，后来者居上。
- 数字后面不能加单位。
- 只有定位的盒子才有z-index属性

定位的页数属性

绝对定位和固定定位也和浮动类似。

1. 行内元素添加绝对定位或者固定定位，可以直接设置高度和宽度。
2. 块级元素添加绝对或者固定定位，如果不给宽度或者高度，默认大小是内容大小。
3. 脱标的盒子不会发生外边距塌陷
    - 浮动元素、绝对定位（固定定位）元素都不会出发外边距合并的问题。

定位的拓展

绝对定位（固定定位）会完全压住盒子

浮动元素不同，浮动元素只会压住它下面标准流的盒子，但是不会压住下面标准流盒子里面的文字（图片）

但是绝对定位（固定定位）会压住下面标准流所有的内容。

浮动之所以不会压住文字，因为浮动产生的最终目的是为了做文字环绕效果的。文字会围绕浮动元素。

CSS高级使用技巧

精灵图

在网页中，往往会用到许多图像来修饰网页元素，将这些所需要的元素都放到一张图中，这就是精灵图。

为什么需要精灵图

如果网页中所使用的图修饰较多，那么就会导致网页频繁请求服务器，造成服务器压力过大，倒是页面的加载速度变慢。

为了有效降低服务器接收和发送请求的次数，提高页面加载速度，就出现了CSS的精灵图技术（也成为CSS Sprites、CSS雪碧）。

核心原理：将网页中的一些小背景图像整合到一张大图中，这样服务器只需要一次请求就可以了

精灵图的使用

1. 给盒子设定一个大小。
2. 通过 `background-position` 参数一定精灵图的位置，以便于让需要显示的内容在盒子中显示。

```
.box {
    width: 30px;
    height: 30px;
    background: url(../img/img.png);
    background-position: -155px -105px;
}
```

- 复合写法：`background: url(../img/img.png) -155px -105px;`
- 分开写法：通过 `background` 指定精灵图，通过 `background-position` 指定位置
- 指定位置也可以拆分成 `background-position-x` 和 `background-position-y`

字体图标

字体图标的产生

> 在网页布局中有许多的小图标，这些图标都是由字体图标来进行实现的。

在早期这些字体图标也使用精灵图进行实现，但是精灵图也有其自身缺点：

1. 图片文件还是比较大的。
2. 图片本身放大和缩小会失真。
3. 一旦图片制作完毕想要变更非常复杂。

此时，字体图标的技术就完美的解决了以上的问题

字体图标可以为前端工程师提供同一种方便高效的图标使用方式，展示的是图标，本质属于字体。

字体图标的优点

- 轻量级：一个图标字体要比一系列的图像要小 。一旦字体加载了，图标马上就会渲染出来，减少了服务器的请求。
- 灵活性：本质其实是文字，可以很随意的改变颜色、产生阴影、透明效果、旋转等。
- 兼容性：几乎所有浏览器都支持。

注意：字体图标并不能代替精灵技术，只是对工作中图标部分技术的提升和优化。

字体图标的使用

字体图标可以去相应的图标网站进行下载，这里推荐两个网站：

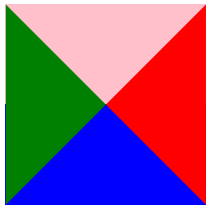
1. icomoon
2. iconfont https://www.iconfont.cn/

三角形的制作

三角形的制作使用 border 来进行制作。

当我们给一个盒子指定了大小为0，四个边框有大小且不同颜色时，这四个边框就会以三角的形式展现。

```
.box {
            width: 0;
            height: 0;
            border-top: 10px solid pink;
            border-right: 10px solid red;
            border-bottom: 10px solid blue;
            border-left: 10px solid green;
            margin: auto;
        }
```



这时我们要使用三角时，只需要将其他四个边框的颜色变为透明（transparent）即可。

根据此知识我们可以写出类似对话框效果。

```
<head>
    <style>
        .box {
            position: relative;
            width: 150px;
            height: 200px;
            background-color: antiquewhite;
            margin: 100px auto;
```

```
        }

        .box .angle {
            position: absolute;
            top: -20px;
            right: 10px;
            width: 0;
            height: 0;
            border-top: 10px solid transparent;
            border-right: 10px solid transparent;
            border-bottom: 10px solid antiquewhite;
            border-left: 10px solid transparent;
            margin: auto;
        }
    </style>
</head>

<body>
    <div class="box">
        <div class="box2"></div>
    </div>
</body>
```


鼠标样式 cursor

可以给元素添加 `cursor` 属性来控制经过其的鼠标样式。

```
{ cursor: pointer; }
```

属性	描述
default	小白 默认
pointer	小手
move	移动十字架
text	文本
not-allowed	禁止

```
<ul>
        <li style="cursor: default;">我是默认的小白鼠标样式</li>
        <li style="cursor: pointer;">我是小手鼠标样式</li>
        <li style="cursor: move;">我是鼠标移动鼠标样式</li>
        <li style="cursor: text;">我是鼠标文本样式</li>
        <li style="cursor: not-allowed;">我是鼠标禁止样式</li>
</ul>
```

- 我是默认的小白鼠标样式
- 我是小手鼠标样式
- 我是鼠标移动鼠标样式
- 我是鼠标文本样式
- 我是鼠标禁止样式

表单的轮廓线 outline

给表单添加 `outline: 0;` 或者 `outline: none;` 样式之后就可以去掉默认的蓝色边框。

```
input {
            outline: none;
        }
```

未去除轮廓

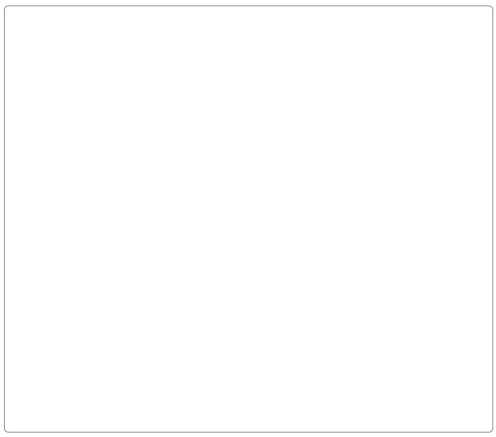
去除轮廓

文本域拖动缩放 resize

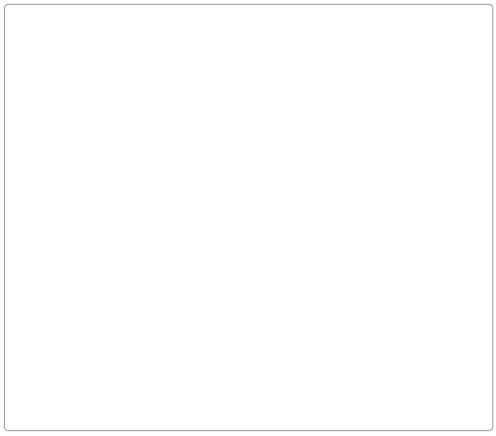
```
textarea {
            resize: none;
            outline: none;
        }
```

未去除

去除后



注意：文本域如果`<textarea></textarea>`不再同一行，那么显示出的文本框中会有自动的类似padding效果挤开里面的文字



图片文字居中对齐

vertical-align

CSS的`vertical-align`属性的使用场景：经常设置图片或者表单（行内块元素）和文字垂直对齐。

只针对行内元素或行内块元素起效

语法：

```
vertical-align: baseline | top | middle | bottom
```

值	描述
baseline	默认。元素放置在父元素的基线上。
top	把元素顶端与行中最高元素的顶端对齐
middle	把元素放置在父元素的中部
bottom	把元素的底端与行中最低元素的底端对齐

默认基线对齐

顶端对齐

底端

垂直对齐

图片底测空白缝隙

看上面第一个文本域与第二个文本域存在空白缝隙，这是因为行内块元素和文字的基线对齐

解决：

1. 给行内块元素加上 vertical-align即可
2. 将行内块（图片）转换为块级元素display: block;

溢出文字省略号显示

单行文本

我们文字的显示有时候会由于盒子不够宽而显示不开，出现换行的情况。

这时我们可以使用 `white-space: nowrap;` 来强制一行显示。

然后使用 `overflow: hidden;` 将溢出的部分隐藏起来。

最后使用 `text-overflow: ellipsis;` 对溢出部分的文字进行省略。

```
{
/* 1. 强制文本在一行内显示 */
/* 默认未normal自动换行 */
white-space: nowrap;
/* 2. 超出的部分隐藏 */
overflow: hidden;
/* 3. 文字用省略号替代超出的部分 */
text-overflow: ellipsis;
}
```

我们文字的显...

多行文本

多行文本溢出显示省略号，有非常大的兼容性问题，适合webKit浏览器或移动端（移动端大多是webKit内核）

```
{
    overflow: hidden;
    text-overflow: ellipsis;
    /* 弹性伸缩盒子模型显示 */
    display: -webkit-box;
    /* 限制一个块蒜素显示的文本的行数 */
    -webkit-line-clamp: 2;
    /* 设置检索伸缩盒子对象的子元素排列方式 */
    -webkit-box-orient: vertical;
}
```

我们文字的显示
有时候会由于...

这种方法只会省略指定的行，所以指定行下面的照常显示，因此需要配合盒子高度来进行设置。

margin 负值技巧

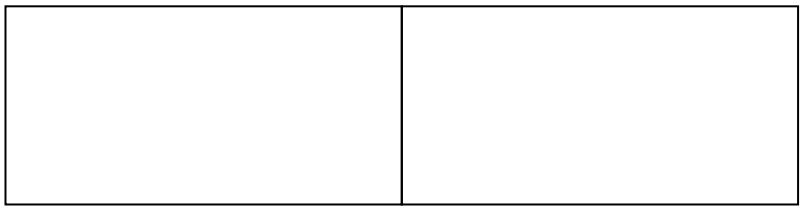
当我们使用浮动展示商品的时候，如果商品需要加边框，那么前面一个盒子的右边框和后面一个盒子的左边框就重合了，宽度为2倍，这时就需要给一个负值margin，来让一个盒子的边框压住一个盒子的边框，这样视觉效果就是一个没有重合的边框了。

```
<style>
        * {
            padding: 0;
            margin: 0;
        }
        li {
            list-style: none;
        }
        .box {
            margin: auto;
```

```
            width: 1000px;
            height: 100px;
            background-color: aquamarine;
        }
        .box ul li {
            float: left;
            width: 199px;
            height: 100px;
            background-color: beige;
            border: 1px solid #000;
            margin-left: -1px;
        }
    </style>
<body>
    <div class="box">
        <ul>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
        </ul>
    </div>
</body>
```

解读：这时候有些人就会有问题，为什么给`li`一个`margin-left: -1px`之后盒子没有整体向左移动溢出父盒子。

因为给了浮动，`li`首先会向左对齐，对齐后才会向左移动-1px。



边框选中高亮

由于我们的这个盒子是左右边框相互覆盖，来解决重合的问题。

所以当我们要边框高亮时，就会出现有一边被覆盖无法高亮的情况。

这时我们给高亮的盒子增加一个相对定位，并根据境况给一个z-index值便可以使得选中的盒子覆盖在最上面。

注意：相对定位显示在浮动之上

文字围绕浮动元素

当我们给图片添加浮动时，这时候文字就会围绕在浮动边上，并不会被浮动压住。

因为浮动的出现的初衷就是为了做文字环绕的。

行内块元素的妙用

行内元素和行内块元素可以通过给父类添加`text-align`来实现水平居中对齐。

而且行内块元素何以设置宽度和高度。

三角的巧妙运用

三角的形状可以通过控制不同`border`边框的大小来进行调整。

三角可以用`i`标签制作，专门用来做三角。

CSS初始化

不同浏览器对有些标签的默认值是不同的，为了消除不同浏览器对HTML文本呈现的差异，照顾浏览器兼容，我们需要对CSS进行初始化。

这里我们以京东代码为例

```
/* 把所有标签的内外边距清零 */
* {
    margin: 0;
    padding: 0
}

/* em 和 i 斜体文字不倾斜 */
em,
i {
    font-style: normal
}

/* 去掉li的小圆点 */
li {
    list-style: none
}

img {
    /* 照顾低版本浏览器，如果图片外面包含了链接会有边框的问题 */
    border: 0;
    /* 取消图片有空白缝隙的问题 */
    vertical-align: middle
}
/* 鼠标经过button时变成小手 */
button {
    cursor: pointer
}
/* 改变a颜色并取消下划线 */
a {
    color: #666;
    text-decoration: none
}

/* 京东特色，鼠标经过a变红色 */
a:hover {
    color: #c81623
}

button,
input {
```

```
    font-family: Microsoft YaHei, Heiti SC, tahoma, arial, Hiragino Sans GB,
"\5B8B\4F53", sans-serif
}

body {
    /* css3抗锯齿性，让文字更加清晰 */
    -webkit-font-smoothing: antialiased;
    background-color: #fff;
    font: 12px/1.5 Microsoft YaHei, Heiti SC, tahoma, arial, Hiragino Sans GB,
"\5B8B\4F53", sans-serif;
    color: #666
}

/* 专门定义了隐藏样式，方便调用 */
.hide,
.none {
    display: none
}

/* 清除浮动，利用伪元素清除 */
.clearfix:after {
    visibility: hidden;
    clear: both;
    display: block;
    content: ".";
    height: 0
}

.clearfix {
    *zoom: 1
}
```

Unicode编码字体：

把中文字体名称用相应的Unicode编码来代替，这样就可以有效的避免浏览器解析CSS代码时候出现乱码的问题。

比如：

黑体 \9ED1\4F53

宋体 \5B8B\4F53

微软雅黑 \5FAE\8F6F\96C5\9ED1

CSS3新特性

CSS3 新增选择器

属性选择器

属性选择器可以根据元素特定的属性来选择元素。这样就可以不借助于类或者id选择器。

1. 利用属性选择器可以不用借助类和id选择器。

```
input[value] {

}
```

以上选择器只选择具有 value 属性的 input。

1. 属性选择器还可以选择 属性=值 的某些元素

```
input[type=text] {

}
```

1. 属性选择器可以选择属性值开头的某些元素

```
div[class^=icon] {

}

<div class="icon1">小图标1</div>
<div class="icon2">小图标2</div>
<div class="icon3">小图标3</div>
<div class="icon4">小图标4</div>
```

1. 属性选择器可以选择属性值结尾的某些元素

```
div[class$=data]{

}

<div class="icon1-data">小图标1</div>
<div class="icon2-data">小图标2</div>
<div class="icon3-ico">小图标3</div>
```

1. 属性选择器可以选择属性值存在某些值的元素

```
div[class*=con]{

}

<div class="icon1-data">小图标1</div>
<div class="icon2-data">小图标2</div>
<div class="icon3-ico">小图标3</div>
```

注意：类选择器，伪类选择器，属性选择器的权重都是10

结构伪类选择器

结构伪类选择器主要根据文档结构来选择元素，常用于根据父级选择里面的子元素。

选择符	简介
E:first-child	匹配父元素中的第一个子元素E
E:last-child	匹配父元素中的最后一个子元素E
E:nth-child(n)	匹配父元素中的第 n 个子元素E
E:first-of-type	指定类型 E 的第一个
E:last-of-type	指定类型 E 的最后一个
E:nth-of-type(n)	指定类型 E 的第 n 个

nth-child（n）

选择某个父元素中的一个或者多个特定的子元素。

- **其中 n 可以是数字，关键字和公式**
- 如果n是数字，那么就是选择第n个子元素，里面的数字从1开始
- n 可以是关键字：even偶数，odd奇数
- n 可以是公式：常见的公式如下（如果n是公式，则从0开始计算，但是第0个元素或者超出元素的个数会被忽略）

公式	取值
2n	偶数
2n+1	奇数
5n	5，10，15...
n+5	从第五个开始（包含第五个）到最后
-n+5	前5个（包含第五个）...

nth-child 和 nth-of-type 的区别

E:nth-child 给所有的孩子标号

E:nth-of-type 仅仅给要选择的孩子E标号

详情看如下

```
<style>
    /* 由于 nth-child(n) 给所有孩子都标号，所以这里的第一个孩子选择的是p 光头强，但是光头强不是指定的div标签，所以不做更改 */
    section div:nth-child(1) {
    color: pink;
    }
    /* 由于 nth-of-type(n) 仅仅给指定孩子标号，所以这里只给div标号，选取第1个，就是div中标号的第一个，因此将熊大选出 */
    section div:nth-of-type(1) {
        color: pink;
    }
</style>
```

```
<body>
    <section>
    <p>光头强</p>
      <div>熊大</div>
      <div>熊二</div>
    </section>
</body>
```

伪元素选择器（重点）

为元素选择器可以利用CSS创建新标签元素，而不需要HTML标签，从而简化HTML结构。

选择符	简介
::before	在元素内部的前面插入内容
::after	在元素内部的后面插入内容

注意：

- before 和 after 创建一个元素，但是属于行内元素
- 新创建的这个元素在文档树中是找不到的，所以我们称为伪元素
- 语法：element::before{}
- before 和 after 必须有 content 属性
- before 在父元素内容的前面创建元素，after 在父元素内容的后面插入元素
- 伪元素选择器和标签选择器一样，权重为1

伪元素清除浮动

```
.clearfix::after {
            content: "";
            display: block;
            height: 0;
            clear: both;
            visibility: hidden;
        }
.clearfix::before,
.clearfix::after {
  content:"";
    /* 转换为块级元素并且在一行内显示 */
    display:table;
}

.clearfix::after {
    clear:both;
}
```

CSS3 盒子模型

CSS3中可以通过`box-sizing`来指定盒子模型，有2个值：可以指定为 content-box、border-box，这样一来我们计算盒子大小的方法也发生了改变。

可以分为两种情况：

1. box-sizing: content-box; 盒子大小为 width + padding + border （以前默认的）
2. box-sizing: border-box; 盒子大小为 width

如果盒子模型设置为第二种，那么 padding 和 border 就不会撑大盒子了 （前提是 padding 和 border 不会超过 width 宽度

CSS3 图片模糊处理 filter

filter 属性将模糊或颜色偏移等图形效果应用于元素。

```
filter: 函数（）; 例如：filter:blur(5px); blur模糊处理数值越大越模糊
```

CSS3 计算盒子宽度 calc 函数

可以使用 calc 函数在声明属性值时执行一些计算。

```
width: calc(100%-80px);
```

括号里可以使用 + - * / 来计算

CSS3 过渡

过渡（transition）是 CSS3 中具有颠覆性的特征之一， 我们可以在不使用 Flash 动画或者 JavaScript 的情况下，当元素从一种样式变换为另一种样式时为元素添加效果。

过渡动画：是一个状态 渐可以渐的过度到另一个状态

可以让我们页面更加好看，更动感十足，虽然低版本浏览器不支持（ie9以下版本）但是不会影响页面布局。

我们现在经常和 :hover 一起搭配使用

```
transition: 要过渡的属性 花费时间 运动曲线 何时开始:
```

1. 属性：想要变化的 css 属性，宽度高度 背景颜色 内外边距都可以。如果想要所有的属性都变化过度，写一个 all 就可以。
2. 花费时间：单位时 秒（必须写单位）比如 0.5s
3. 运动曲线：默认是 ease（可以省略）
4. 何时开始：单位时 秒（必须写单位）可以设置延迟触发时间 默认是 0s（可以省略）

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>过渡</title>
    <style>
        .box {
            width: 100px;
            height: 100px;
            margin: auto;
            background-color: pink;
```

```
            /* transition: width 1s ease-in-out 1s, height 1s ease-in-out 1s; */
            transition: all 0.5s;
        }

        .box:hover {
            width: 200px;
            height: 200px;
        }
    </style>
</head>
<body>
    <div class="box">

    </div>
</body>
</html>
```

CSS3 2D转换

转换（transform）是CSS3中具有颠覆性的特征之一，可以实现元素的唯一、旋转、缩放等效果。

转换可以简单理解为变形。

- 移动：translate
- 旋转：rotate
- 缩放：scale

移动translate

2D移动是2D转换里面的一种功能，可以改变元素在页面中的位置，类似**定位**。

语法：

```
transform: translate(x, y);
transform: translateX(n);
transform: translateY(n);
```

demo：

```
<style>
    div {
        width: 200px;
        height: 200px;
        background-color: pink;
        /* x就是x轴上移动位置，y就是y轴上移动位置 */
        /* transform: translate(200px, 100px); */
    }
    div:first-child {
        transform: translate(30px, 30px);
    }
    div:last-child {
        background-color: purple;
    }
</style>
```

```
<body>
    <div></div>
    <div></div>
</body>
```

重点：

- 定义2D转换中的移动，沿着X和Y轴移动元素。
- translate最大的优点是不会影响其它元素的位置。
- translate中的百分比单位是相对于自身元素的`translate: (50%, 50%)`。
- 对行内标签没有效果。

实例：水平垂直居中

在没有transform的时候我们要实现一个盒子在父盒子中水平垂直居中都需要我们用到计算。比如：先用定位将盒子上和坐各移动50%这时盒子并不在最中央，我们还需要让其向上移动自己高度的一半，向左走自己宽度的一半才可实现。

由于transform中的百分比是相对于自身的，所以我们可以使用**定位+transform**来实现免计算水平垂直居中。

```
<style>
    .box {
        position: relative;
        width: 600px;
        height: 800px;
        background-color: pink;
    }

    .box>div {
        position: absolute;
        top: 50%;
        left: 50%;
        width: 200px;
        height: 200px;
        background-color: purple;
        transform: translate(-50%, -50%);

    }
</style>
<body>
    <div class="box">
        <div></div>
    </div>
</body>
```

旋转rotate

2D旋转指的是让元素在2维平面内顺势站或者逆时针旋转。

语法：

```
transform: rotate(度数);
```

demo：

```
<style>
    div {
        width: 100px;
        height: 100px;
        background-color: pink;
        transform: rotate(15deg);
    }
</style>
<body>
    <div></div>
</body>
```

重点：

- rotate里面跟度数，单位是deg比如rotate（45deg）。
- 角度为正时，顺时针；负时，逆时针。
- 默认旋转的中心点是元素的中心点。

2D转换中心点transform-origin

我们可以设置元素转换的中心点。

```
transform-origin: x y;
```

重点：

- 注意后面的参数x和y用空格隔开。
- x y默认转换中心是元素的中心点（50% 50%）。
- 还可以给x y设置**像素**或者**方位名词**（top、bottom、left、right、center）。

案例：三角

我们可以利让正方形旋转45度，然后只显示其两个border，省去了导入字体图标的麻烦。

```
<style>
    div {
        position: relative;
        width: 300px;
        height: 35px;
        margin: 100px auto;
        border: 1px solid #000;
    }
    div::after {
        content: "";
        position: absolute;
        width: 7px;
        height: 7px;
        top: 14px;
        right: 15px;
        border-top: 2px solid #000;
        border-right: 2px solid #000;
        transform: rotate(45deg);
    }
</style>
<body>
```

```
    <div></div>
</body>
```

案例：旋转遮罩显示

```
<style>
    div {
        width: 200px;
        height: 200px;
        border: 1px solid pink;
        margin: 100px auto;
        overflow: hidden;
    }

    div::before {
        content: "brokyz";
        display: block;
        width: 100%;
        height: 100%;
        background-color: hotpink;
        transform: rotate(180deg);
        transform-origin: left bottom;
        transition: all 0.3s;
    }

    div:hover::before {
        transform: rotate(0deg);
    }
</style>
<body>
    <div></div>
</body>
```

缩放scale

缩放，顾名思义，可以放大和缩小。只要给元素加上了这个属性就能控制它放大还是缩小。

语法：

```
transform: scale(x, y);
```

注意：

- 注意其中的x和y用逗号分隔
- `transform: scale(1, 1)` 宽高都放大1倍，等同于没有放大。
- `transform: scale(2, 3)` 宽放大2倍，高放大3倍。
- `transform: scale(2)` 宽高都放大2倍。
- `transform: scale(0.5)` 缩小。78z
- scale缩放的优势：可以设置中心点缩放，而不影响其它盒子的位置。

demo：

```
<style>
    div {
```

```
        width: 200px;
        height: 200px;
        margin: 300px auto;
        background-color: pink;
        transform-origin: left bottom;
        transition: all 0.3s;
    }

    div:hover {
        /* transform: scale(2, 3); */
        transform: scale(2);
    }
</style>
<body>
    <div></div>
</body>
```

2D转换综合写法

我们可以将2D转换的变换进行综合。

```
transform: translate() rotate() scale ...;
```

注意：

- 变换的书写顺序会影响转换的效果（先旋转会改变原来的坐标轴方向）。
- 当我们同时有位移和其它属性的时候，记得要**把位移放在最前面。**

CSS3动画

动画（animation）是CSS3中最具有颠覆性的特征之一，可以通过设置多个节点来精确控制一个或一组动画，常用来实现复杂的动画效果。

制作动画分为两步：

1. 先定义动画。
2. 再使用（调用）动画。

定义动画keyframes

demo：

```
/* 方法1 */
@keyframes 动画名称 {
    0% {
        width: 100px;
    }
    100% {
        width:200px;
    }
}

/* 方法2 */
@keyframes 动画名称 {
```

```
    from {
        width: 100px;
    }
    to {
        width:200px;
    }
}

/* 规定不同阶段  */
@keyframes 动画名称 {
    0% {
        width: 100px;
    }
    50% {
        width: 200px;
    }
    100% {
        width:100px;
    }
}
```

动画序列

- 0%是动画的**开始**，100%是动画的**完成**。这样的规则就是动画序列。
- 再**@keyframes**中规定某项CSS样式，就能创建当前样式逐渐改为新样式的动画效果。
- 动画是使元素从一种样式逐渐变化为另一种样式的效果。您可以改变任意多的样式任意多的**次数**。
- 请用百分比来规定变化发生的时间，或用关键词**from**和**to**，等同于**0%*和*100%**。

调用动画

当有元素想要使用动画时，我们就给那个元素的CSS中加上已经定义好的动画名`animation-name`和完成动画所需要的时间`animation-duration`就可以调用我们所定义的动画了。

demo：

```
div {
                width: 200px;
                height: 200px;
                background-color: pink;
                animation-name: move;
                animation-duration: 3s;
            }
```

实例：环绕

```
<style>
    /* 1. 定义动画 */
    @keyframes move {
        /* 开始状态 */
        0% {
            transform: translate(0, 0);
        }
        25% {
            transform: translate(1000px, 0);
```

```
        }
        50% {
            transform: translate(1000px, 500px);
        }
        75% {
            transform: translate(0, 500px);
        }
        /* 结束状态 */
        100% {
            transform: translate(0, 0);
        }
    }
    /* 2. 调用动画 */
    div {
        width: 200px;
        height: 200px;
        background-color: pink;
        animation-name: move;
        animation-duration: 3s;
    }
</style>
<body>
    <div></div>
</body>
```

动画常用属性

属性	描述
@keyframes	定义动画
animation	所有动画属性的简写属性，除了animation-play-state属性。
animation-name	规定@keyframes动画的名称。（必须的）
animation-duration	规定动画完成一个周期所花费的秒或毫秒，默认是0。（必须的）
animation-timing-function	规定动画的速度曲线，默认是ease。
animation-delay	规定动画何时开始，默认是0。
animation-iteration-count	规定动画被播放的次数，默认是1，还有infinite。
animation-direction	规定动画是否在下一周期逆向播放，默认是normal，alternate逆播放
animation-play-state	规定动画是否在运行或暂停。默认是running，还有pause。
animation-fill-mode	规定动画结束后保持状态，保持forwards回到起始backwards。

- 伴随opicity 使用来调整元素透明度。

动画属性简写

```
animation: 动画名称 持续时间 运动曲线 何时开始 播放次数 是否反方向 动画起始或动画结束的状态
```

demo：

```
animation: myfirst 5s linear 2s infinite alternate;
```

注意：

- 前面两个属性name duration一定要写。
- 其余属性保持默认的可以省略不写。
- animation-play-state需要单独写，不能写在属性简写里面。

案例：小圆点扩散动画

```
<style>
    .box {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
    }

    .dotted {
        margin: auto;
        width: 8px;
        height: 8px;
        background-color: #09f;
        border-radius: 50%;
    }

    .box div[class^="pulse"] {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
        width: 8px;
        height: 8px;
        box-shadow: 0 0 20px #009dfd;
        border-radius: 50%;
        animation: pulse 2.4s linear infinite;
    }
    .box div.pulse2 {
        animation-delay: 0.8s;
    }
    .box div.pulse3 {
        animation-delay: 1.6s;
    }
    @keyframes pulse {
        0% {
        }
        70% {
```

```
            width: 40px;
            height: 40px;
            opacity: 1;
        }
        100% {
            width: 70px;
            height: 70px;
            opacity: 0;
        }
    }
</style>
<body>
    <div class="box">
        <div class="dotted"></div>
        <div class="pulse1"></div>
        <div class="pulse2"></div>
        <div class="pulse3"></div>
    </div>
</body>
```

- 里面小圆点的扩散不使用scale进行扩散的原因是，scale会将阴影部分也放大，实际效果不好

速度曲线细节

animation-timing-function：规定动画的速度曲线，默认是ease。

值	描述
linear	动画从头到尾的速度是相同的。匀速。
ease	默认。动画以低速开始，然后加快，在结束前变慢。
ease-in	动画以低速开始。
ease-out	动画以低速结束。
ease-in-out	动画以低速开始和结束。
steps()	指定了时间函数中的间隔数量（步长）。

案例：打字机效果

```
<style>
    div {
        width: fit-content;
        height: 30px;
        font-size: 20px;
        animation: type 2s steps(8) infinite;
        background-color: antiquewhite;
        overflow: hidden;
    }
    @keyframes type {
        0% {
```

```
            width: 0;
        }
        100% {
            width: 160px;
        }
    }
</style>
<body>
    <div>这是打字机效果</div>
</body>
```

CSS3 3D转换

我们的生活中的环境是3D的，照片就是3D物体在2D平面呈现的例子。

特点：

- 近大远小。
- 物体后面遮挡不可见。

三维坐标系

三维坐标系其实就是指立体空间，立体空间是由3个轴共同组成的。

- x轴：水平向右
- y轴：垂直向下
- z轴：垂直屏幕向外

3D 移动 translate3d

3D 移动在 2D 基础上多加了一个可移动的方向，就是 z 轴方向。

语法：

- `transform: translateX(100px)`：仅仅在 X 轴移动
- `transform: translateY(100px)`：仅仅在 Y 轴移动
- `transform: translateZ(100px)`：仅仅在 Z 轴移动
- `transform: translate3d(x, y, z)`：其中 x、y、z 分别指要移动的方向的距离

透视 perspective

在 2D 平面产生近大远小视觉立体，但是只是二维效果的。

- 如果想要在网页生成 3D 效果需要透视（可以理解为3D物体投影在2D平面内）。
- 模拟人类的视觉位置，可以认为安排一只眼睛去看。
- 透视我们也称为视距：视距就是人的眼睛到屏幕的距离。
- 距离视觉点越近的在电脑平面成像越大，越远成像越小。
- 透视的单位是像素。

```
perspective: 400px;
```

透视是需要写在观察元素的父盒子里面的（注意只能写在父元素中）

- 视距：人眼到电脑屏幕的距离。
- z轴：物体距离屏幕的距离，z轴越大我们看到的物体就越大。

3D旋转roate3d

3D旋转指可以让元素在三维平面内沿着x轴，y轴，z轴或者自定义轴进行旋转。

语法：

- `transform: rotateX(45deg)`：沿着x轴方向旋转45度。
- `transform: rotateY(45deg)`：沿着y轴方向旋转45度。
- `transform: rotateZ(45deg)`：沿着z轴方向旋转45度。
- `transform: rotate3d(x, y, z, 45deg)`：沿着自定义轴旋转自定义度。

左手法则

左手大拇指指向轴的正方向，四指的指向就是对应轴旋转的正方向。

3D呈现 transform-style

- 控制子元素是否开启三维立体环境。
- `transform-style: flat`子元素不开启3d立体空间。默认。
- `transform-style: preserve-3d`子元素开启3d立体空间。
- 代码写给父级，但是影响子盒子。
- 这个属性非常重要。

浏览器私有属性

浏览器私有前缀是为了兼容老版本的写法，比较新版本的浏览器无需添加。

私有前缀

- `-moz-`：代表Firefox浏览器私有属性。
- `-ms-`：代表ie浏览器私有属性。
- `-webkit-`：代表safari、chrome私有属性。
- `-o-`：代表Opera私有属性。

提倡写法

```
-moz-border-radius: 10px;
-webkit-border-radius: 10px;
-o-border-radius: 10px;
border-radius: 10px;
```

移动端基础

浏览器现状

国内的UC、QQ和百度等手机浏览器都是根据Webkit修改的内核，国内尚无自主研发的内核。

移动端常见浏览器

UC浏览器、QQ浏览器、欧鹏浏览器、百度手机浏览器、360安全浏览器、谷歌浏览器、搜狗手机浏览器、猎豹浏览器等等。

注意

兼容移动端主流浏览器，只需要处理Webkit内核即可。

手机屏幕现状

- 移动端设备屏幕尺寸非常多，碎片化严重。
- Android设备有多种分辨率：480x800,480x854,540x960,720x1280,1080x1920等，还有传说中的2K,4k屏
- 近年来iPhone的碎片化也加剧了，其设备的主要分辨率有：640x960,640x1136,750x1334,1242x2208等。
- 作为开发者无需关注这些分辨率，因为我们常用的尺寸单位是px。

注意：作为前端开发，不建议大家纠结dp、dpi、pt、ppi等单位。

移动端调试方法

- Chrome DevTools 的模拟手机调试
- 搭建本地web服务器，手机和服务器在一个局域网内，通过手机访问服务器
- 使用外网服务器，直接IP或域名访问

视口

视口（viewpot）就是浏览器显示页面内容的区域。视口可以分为布局视口、视觉视口和理想视口。

布局视口

- 一般移动端的浏览器默认设置了一个布局视口，用于解决早期的PC端页面在手机上显示的问题。
- IOS，Android基本都将这个视口分辨率设置为980px，所以PC上的网页大多数都能在手机上呈现，只不过元素看上去很小，一般默认可以通过手动缩放网页。

视觉视口

- 字面意思，它是用户正看到的网站的区域。注意：是网站的区域。
- 我们可以通过缩放取操作视觉视口，但不会影响布局视口，布局视口仍保持原来的宽度。

理想视口

- 为了使网站在移动端有最理想的浏览器和阅读宽度而设定。
- 理想视口，对于设备来讲，是最理想的尺寸。
- 需要手动添加meta视口标签通知浏览器操作。
- meta视口标签的主要目的：布局视口的宽度应该与理想视口宽度一致，简单理解就是设备有多宽，我们布局的视口就有多宽。

meta视口标签

给`meta`标签指定`name="viewport"`来规定其为视口标签

```
<meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimim-scale=1.0">
```

属性	解释说明
width	宽度设置的是viewport宽度，可以设置device-width特殊值
initial-scale	初始缩放比，大于0的数字
maximum-scale	最大缩放比，大于0的数字
minimum-scale	最小缩放比，大于0的数字
user-scalable	用户是否可以缩放，yes或no（1或0）

标准的viewport设置

- 视口宽度和设备保持一致
- 视口默认缩放比1.0
- 不允许用户缩放
- 最大允许缩放比1.0
- 最小允许缩放比1.0

二倍图

物理像素&物理像素比

- 物理像素点指的是屏幕显示的最小颗粒，是物理真是存在的。这时厂商在出厂时就设置好的，比如苹果6\7\8是750*1334，也就是我们所说的分辨率。
- 我们开发时候的1px不一定等于一个物理像素的。
- PC端页面，1个px等于一个物理像素的，但是移动端就不尽相同。
- 一个px的能显示的物理像素点的个数，称为物理像素比或屏幕像素比。
- PC端和早期的手机屏幕/普通手机屏幕：1CSS像素 = 1物理像素。
- Retina（视网膜屏幕）是一种显示技术，可以把更多的物理像素点压缩至一块屏幕里，从而达到更高的分辨率，并提高屏幕显示的细腻程度。

多倍图

- 对于一张50px*50px的图片，在手机Retina屏中打开，按照刚才的物理像素比会放大倍数，这样会造成图片模糊。
- 在标准的viewporti设置中，使用倍图来提高图片质量，解决在高清设备中的模糊问题。
- 通常使用二倍图，因为iPhone6/7/8的影响，但是现在还存在3倍图4倍图的情况，这个看实际开发公司需求。
- 背景图片注意缩放问题。

背景缩放 background-size

background-size 为我们规定了背景图像的尺寸。

```
background-size: 50px 50px;
background-size: 50px;
background-size: cover;
background-size: contain;
```

- `background-size: 50px 60px;`：指定了背景图片**宽度**和**高度。**
- `background-size: 50px;`：指定了背景图片**宽度，高度随指定的宽度等比例拉伸。**

- `background-size: cover;`：背景图片完全覆盖盒子。（部分图片可能显示不全）
- `background-size: contain;`：背景图片高度和宽度等比例拉伸，直到宽或高其中一种拉满盒子为止。

小技巧

由于使用`background-size: cover;`时部分图片边缘可能由于需要覆盖整个盒子而超出屏幕，为了优化显示效果可以搭配`background-position: center center;`来使用。

解决移动端图片模糊问题

在移动端由于屏幕的Retina技术会存在一个物理像素比的问题，一般来说物理像素比普遍为2。也就是说我们在开发时写的50px在物理像素比为2的手机上将会展现出100px。

如果我们书写的元素时纯色那么不会有任何显示问题，但是如果我们使用图片时，由于我们图片本身像素为50px*50px，在手机上被以100px*100px*100px显示出来后，将不可避免会产生模糊的问题。

为了解决这个问题，我们可以在准备图片时，就准备一个100px*100px的二倍图，然后我们在写的时候将其缩放为50px*50px，这样即使被移动端以100px*100px显示，也不会出现模糊的问题。

demo：

```
div {
    width: 50px;
    height: 50px;
    background: url(100px*100px.png) no-repeat;
    background-size: 50px 50px;
    border: 1px solid red;
}
```

多倍图切图工具

- Cutterman(已收费)：https://www.cutterman.cn/zh
- NB Cutter：https://pslkzs.com/nbCutter/index.php

移动端技术解决方案

移动端浏览器基本以webkit内核为主，因此我们就考虑webkit兼容性问题。

我们可以放心使用H5标签和CSS3样式。

同时对于浏器的私有前缀我们只需要考虑添加webkit即可。

CSS初始化nomalize.css

移动端CSS初始化推荐使用normalize.css

- Normalize.css:保护了有价值的默认值
- Normalize.css:修复了浏览器的bug
- Normalize..css:是模块化的
- Normalize.css:拥有详细的文档

官网地址：http://necolas.github.io/normalize.css/

CSS3盒子模型 box-sizing

- 传统模式宽度计算：盒子的宽度=CSS中设置的width+border+padding
- CSS3盒子模型：盒子的宽度=CSS中设置的宽度width里面包含了border和padding

也就是说我们CSS3盒子模型中，padding和border不会撑大盒子了。

```
/* CSS3盒子模型 */
box-sizing: border-box;
/* 传统盒子模型 */
box-sizing: content-box;
```

传统or CSS3盒子模型?

- 移动端可以全部CSS3盒子模型
- PC端如果完全需要兼容，我们就用传统模式，如果不考虑兼容性，我们就选择CSS3盒子模型

特殊样式

```
/* CSS3盒子模型 */
box-sizing:border-box;
-webkit-box-sizing:border-box;
/* 点击高亮我们需要清除清除;
设置为transparent完成透明 */
-webkit-tap-highlight-color:transparent;
/* 在移动端浏览器默认的外观在iOS上加上这个属性才能给按钮和输入框自定义样式 */
-webkit-appearance:none;
/* 禅用长按页面时的弹出菜单 */
img,a {-webkit-touch-callout:none;
```

移动端开发的选择

单独制作移动端页面（主流）

- 流式布局（百分比布局）：京东
- flex弹性布局（强烈推荐）：携程网
- less+rem+媒体查询布局：苏宁
- 混合布局

响应式页面兼容移动端（其次）

- 媒体查询
- bootstrap

流式布局（百分比布局）

- 流式布局，就是百分比布局，也称为非固定像素布局。
- 通过盒子的宽度设置成百分比来根据屏幕的宽度来进行伸缩，不受固定像素的限制，内容向两侧填充。
- 流式布局方式是移动web开发者比较常见的布局方式。
- max-width：不允许超过最大值。
- min-width：不允许小于最大值。

```
max-width: 1000px;
min-width: 400px
```

flex布局

布局原理

flex是flexible Box的缩写，意为**弹性布局**，用来为盒状模型提供最大的灵活性，**任何一个容器**都可以指定为flex布局。

- 当我们为父盒子设为flex布局以后，子元素的float、.clear和vertical-align属性将失效。
- 伸缩布局=弹性布局=伸缩盒布局=弹性盒布局=flex布局

采用Flex布局的元素，称为Flex容器(flex container),简称"容器"。它的所有子元素自动成为容器成员，称为Flex项目(flex item),简称"项目"。

- 体验中div就是flex父容器。
- 体验中span就是子容器flex项目。
- 子容器可以横向排列也可以纵向排列。

总结flex布局原理：

- 就是通过给父盒子添加flex属性，来控制子盒子的位置和排列方式。

常见父项元素

以下6个属性是给父亲设置的：

- flex-direction：设置主轴的方向。
- justify-content：设置主轴上的子元素排列方式。
- flex-wrap：设置子元素是否换行。
- align-content：设置侧轴上的子元素的排列方式（多行）。
- align-tems：设置侧轴上的子元素排列方式（单行）。
- flex-flow：复合属性，相当于同时设置了flex-direction和flex-wrap。

flex-direction

`flex-direction`属性可以用来**设置主轴和其方向**，剩下的另一个轴就是侧轴。我们的子元素是跟着主轴来进行排列的。

属性值	说明
row	主轴为x轴，方向从左到右。**此项为默认。**
row-reverse	主轴为x轴，方向从右到左。
column	主轴为y轴，方向从上到下。
column-reverse	主轴为y轴，方向从下到上。

justify-content

`justify-content`属性定义了项目在**主轴上的对齐**方式。

属性值	说明
flex-start	从头部开始，如果x轴是主轴，则从左向右。**默认值。**
flex-end	从尾部开始排列。
center	在主轴居中对齐。（如果x轴是主轴则居中对齐）
space-around	平分剩余空间。
space-between	先两边贴边，再平分剩余空间（重要）。

注意：使用之前一定要确定好主轴是哪个。

flex-wrap

`flex-wrap`属性定义项目是否换行，flex布局中默认是不换行的，项目都会排列在同一行（又称轴线）上。

如果为默认不换行，当子元素总宽度超出时，会自动调整子元素的宽度，依旧不进行换行。

属性值	说明
nowrap	默认值。不换行。
wrap	换行。

align-items（单行）

该属性控制子元素在**侧轴**上的排列方式。仅仅在子元素为**单行**时有效。

属性值	说明
flex-start	从上到下
flex-end	从下到上
center	挤在一起居中（垂直居中）
stretch	拉伸（默认值）

align-content（多行）

设置子元素在**侧轴**上的排列方式并且只能用于子项出现**多行**的情况下有效，在单行情况下没有效果。

属性值	说明
flex-start	在侧轴的头部开始排列。默认值。
flex-end	在侧轴的尾部开始排列
center	在侧轴的中间显示
space-around	子项在侧轴平分剩余空间
space-between	子项在侧轴先分布在两头，在平分剩余空间
stretch	设置子项元素高度平分父元素高度。

flex-flow

`flex-flow`属性时flex-direction和flex-wrap属性的复合属性。

```
flex-flow: row wrap;
```

常见的子项元素

flex

`flex`属性定义子项目**分配剩余空间**，用`flex`来表示占多少**份数。**

默认属性值为0

```
.item {
    flex: 1;
}
```

align-self

`align-self`属性允许单个项目有与其他项目不一样的对齐方式，可覆盖`align-items`属性。属性值与`align-items`相同。

默认值为auto，表示继承父元素的`align-items`属性，如果没有父元素，则等同于stretch。

order

`order`属性定义项目的排列顺序。**数值越小，排列越靠前**，默认值为0。

注意：和z-index不同。

居中对齐总结

行内元素水平居中

text-align: center;

```
<style>
    .parent {
        background-color: aliceblue;
        text-align: center;
    }
    .child {
        background-color: orange;
    }
</style>
<body>
    <div class="parent">
        <span class="child">content</span>
    </div>
</body>
```

content

fit-content

```
<style>
    .parent {
        background-color: aliceblue;
        width: fit-content;
        margin: auto;
    }
    .child {
        background-color: orange;
    }
</style>
<body>
    <div class="parent">
        <span class="child">content</span>
    </div>
</body>
```

content

行内元素和行内块元素垂直居中

```
<style>
    .parent {
        height: 200px;
        line-height: 200px;
        background-color: aliceblue;
    }
    .child {
        background-color: orange;
    }
```

```
</style>
<body>
    <div class="parent">
        <span class="child">content</span>
    </div>
</body>
```

content

块级元素水平居中

```
<style>
    .parent {
        height: 200px;
        width: 200px;
        background-color: aliceblue;
    }
    .child {
        height: 100px;
        width: 200px;

        background-color: orange;
    }
</style>
<body>
    <div class="parent">
        <div class="child">content</div>
    </div>
</body>
```

content

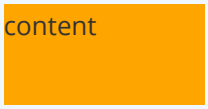
块级元素水平垂直居中

定位方法

方法一：必须知道宽高

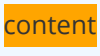
```
<style>
    .parent {
        position: relative;
        height: 100px;
        background-color: aliceblue;
    }
    .child {
        width: 100px;
        height: 50px;
```

```
        position: absolute;
        background-color: orange;
        left: 50%;
        top: 50%;
        margin-top: -25px;
        margin-left: -50px;
    }
</style>
<body>
    <div class="parent">
        <div class="child">content</div>
    </div>
</body>
```



方法二：不必知道宽高

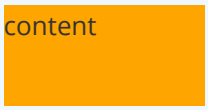
```
<style>
    .parent {
        position: relative;
        height: 100px;
        background-color: aliceblue;
    }
    .child {
        position: absolute;
        background-color: orange;
        left: 50%;
        top: 50%;
        transform: translate(-50%, -50%);
    }
</style>
<body>
    <div class="parent">
        <div class="child">content</div>
    </div>
</body>
```



方法三：margin配合定位

当我们给子元素绝对定位后，当我们给各边界距离都为0后，子元素就会填充父元素的所有可用空间，这样一来，在水平和垂直方向上就有了可以分配的空间，这时设置`margin: auto;`就可以自动居中。

```
<style>
    .parent {
        position: relative;
        height: 100px;
        background-color: aliceblue;
    }
    .child {
        position: absolute;
        background-color: orange;
        left: 0;
        top: 0;
        right: 0;
        bottom: 0;
        margin: auto;
    }
</style>
<body>
    <div class="parent">
        <div class="child">content</div>
    </div>
</body>
```



padding 方法

方法一：给父元素加 padding

```
<style>
    .parent {
        background-color: aliceblue;
        padding: 20px;
    }
    .child {
        height: 100px;
        background-color: orange;
    }
</style>
<body>
    <div class="parent">
        <div class="child">content</div>
    </div>
</body>
```

方法二：给子元素加 padding

```
<style>
    .parent {
        background-color: aliceblue;
        width: 200px;
        height: 100px;
    }
    .child {
        width: 100px;
        height: 50px;
        background-color: orange;
        padding: 25px 50px;
        background-clip: content-box;
    }
</style>
<body>
    <div class="parent">
        <div class="child">content</div>
    </div>
</body>
```

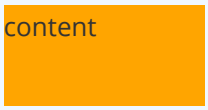
flex 布局方法

`justify-content: center;` 控制水平居中

`align-items: center;` 控制垂直居中

```
<style>
    .parent {
        height: 200px;
        background-color: aliceblue;
        display: flex;
        align-items: center;
        justify-content: center;
    }
    .child {
        height: 50px;
        width: 100px;
        background-color: orange;
    }
</style>
<body>
    <div class="parent">
        <div class="child">content</div>
```

```
    </div>
</body>
```



伪元素方法

```
<style>
    .parent {
        height: 100px;
        text-align: center;
        background-color: aliceblue;
    }
    .child {
        height: 50px;
        width: 100px;
        display: inline-block;
        background-color: orange;
        vertical-align: center;
    }
    .parent::before {
        content: "";
        height: 100px;
        display: inline-block;
        vertical-align: middle;
        background-color: yellow;
    }
</style>
<body>
    <div class="parent">
        <div class="child">content</div>
    </div>
</body>
```

为了方便观察，这里效果中给模拟的伪元素增加了5px的宽度，实际不用给伪元素设置宽度。