HTML5新特性

概述

HTML5 的新增特性主要是针对于以前的不足，增加了一些新的标签、新的表单和新的表单属性等。

这些新特性都有兼容性问题，基本是 **IE9+ 以上版本的浏览器**才支持，如果不考虑兼容性问题，可以大量使用这些新特性。

HTML5的发展

```
1-HTML  的历史：
    1993 年 HTML 首次以因特网草案的形式发布，然后经历了 2.0、3.2 和 4.0，直到 1999 年的 HTML 4.01 版本稳定下来。 后来逐渐被更加严格的 XHTML 取代。

2-XHTML 的兴衰史：
    自从 HTML 4.01 版本之后，掌握着 HTML 规范的万维网联网（W3C）组织没有再发布新的标准，而是围绕着 XHTML 1.0 以及之后的 XHTML 2.0 展开工作。XHTML 是基于 XML 、致力于实现更加严格并且统一的编码规范的 HTML 版本，解决之前的 HTML 4.01 版本时，由于编码不规范导致浏览器的各种古怪行为。对于开发者来说，XHTML 极大的好处，就是强迫开发者养成良好的编码习惯，放弃 HTML 的凌乱写法，最终降低了浏览器解析页面的难度，方便移植到更多平台。
    但是XHTML 带来的确实毁灭性的灾难，XHTML 2.0 规范了更严格的错误处理规则，强制要求浏览器拒绝无效的 XHTML 2 页面，强制 Web 开发者写出绝对正确规范的代码，同时不得向下兼容，这将意味着，之前已经写好的数亿的页面将无法兼容，并且Web开发者的难度又被加大，最终被抛弃了哦！

3-HTML 5 的到来：
    2008 年 W3C 发布了 HTML 5 的工作草案，2009 年停止了 XHTML 2 计划。随着时间推移，HTML 5 规范进一步解决了诸多非常实际的问题，各大浏览器厂商开始对旗下的产品进行升级，以便支持 HTML 5 。HTML 5 规范得到了持续的进步和完善，从而迅速融入到 Web 平台的实质性改进中。
    和 XHTML 2.0 不同，制定 HTML 5 规范的并不想挑出以往 HTML 的各种毛病为其改正，而是尽可能的补全 Web 开发者急需的各种功能。这些功能包括更强大的 CSS3、表单验证、音频视频 、本地存储、地理定位、绘画（Canvas）、Web 通信等等。
```

html5概念

狭义概念： 是html4的升级版,新增了标签，对标签进行了优化。

广义概念：HTML5代表浏览器端技术的一个发展阶段。在这个阶段，浏览器呈现技术得到了一个飞跃发展和广泛支持，它包括：HTML5，CSS3，Javascript，API在内的一套技术组合。

语义化标签

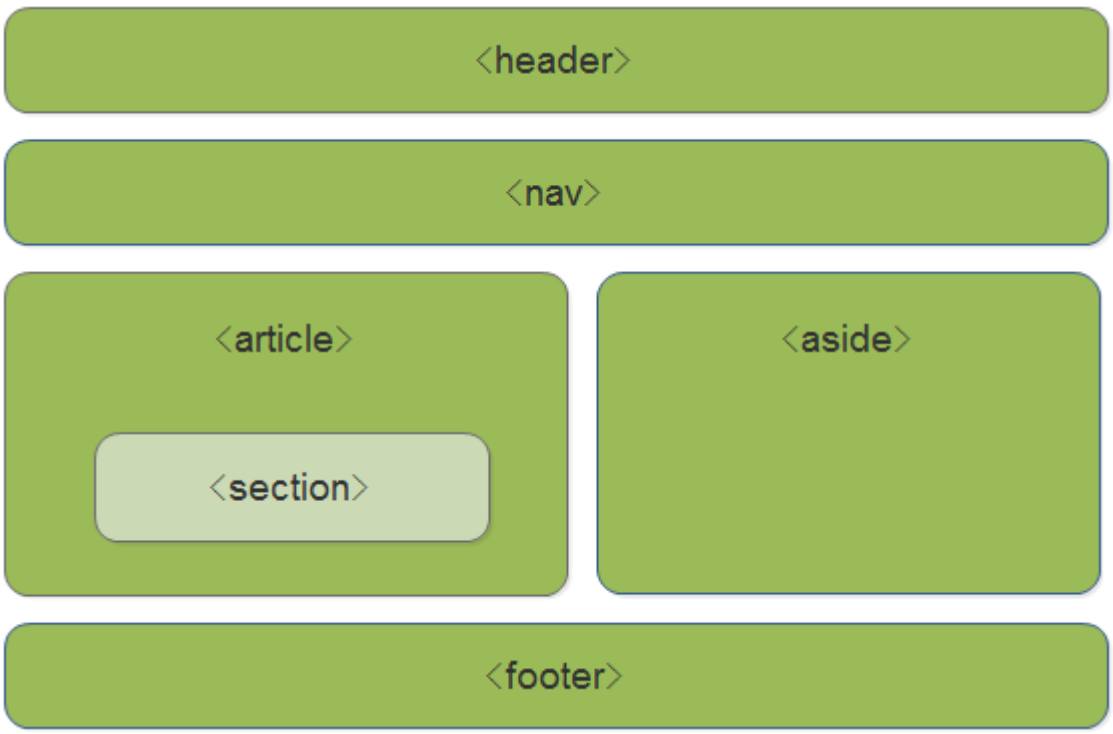
以前布局，我们基本用 div 来做。div 对于搜索引擎来说，是没有语义的

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

<div class="nav"> </div>
<div class="content"> </div>
<div class="footer"> </div>
```

发展到了HTML5后，新增了一些语义化标签，这样的话更加有利于浏览器的搜索引擎搜索，也方便了网站的seo（Search Engine Optimization，搜索引擎优化），下面就是新增的一些语义化标签

- `<header>` 头部标签
- `<nav>` 导航标签
- `<article>` 内容标签
- `<section>` 定义文档某个区域
- `<aside>` 侧边栏标签
- `<footer>` 尾部标签



> HTML5新增语义化标签

```
<style>
        header,
        nav {
            width: 800px;
            height: 100px;
            background-color: pink;
            border-radius: 15px;
            margin: 15px auto;
        }
        section {
            width: 800px;
            height: 200px;
            background-color: skyblue;
            margin: 15px auto;
```

```
        }
        article {
            width: 350px;
            height: 200px;
            background-color: red;
            float: left;
        }
        aside {
            width: 350px;
            height: 200px;
            background-color: green;
            float: right;
        }
        footer {
            width: 800px;
            height: 100px;
            background-color: purple;
            margin: 15px auto;
        }
    </style>
</head>
<body>
    <header>头部标签</header>
    <nav>导航栏标签</nav>
    <section>
        <article>
            文章标签
        </article>
        <aside>
            侧边栏标签
        </aside>
    </section>
    <footer>
        尾部标签
    </footer>
</body>
```

多媒体标签

多媒体标签分为 音频 **audio** 和视频 **video** 两个标签 使用它们，我们可以很方便的在页面中嵌入音频和视频，而不再去使用落后的flash和其他浏览器插件了。

因为多媒体标签的 属性、方法、事件比较多，因此我们需要什么功能的时候，就需要去查找相关的文档进行学习使用。

视频标签- video

基本使用

当前 元素支持三种视频格式： 尽量使用 **mp4格式**

使用语法：

```
<video src="media/mi.mp4"></video>
```

兼容写法

由于各个浏览器的支持情况不同，所以我们会有一种兼容性的写法，这种写法了解一下即可

```
  <style>
    video {
        width: 300px;
    }
  </style>
<!-- <video src="media/video.mp4" controls="controls"></video> -->
    <!-- 谷歌浏览器把自动播放功能给禁用了 有解决方案： 给视频添加静音属性 -->
    <video loop="loop" poster="media/01.jpg" controls autoplay muted="muted">
        <source src="media/video.mp4" type="video/mp4" />
        <source src="media/video.ogg" type="video/ogg" />
        您的浏览器太low了，不支持播放此视频
    </video>
```

上面这种写法，浏览器会匹配video标签中的source，如果支持就播放，如果不支持往下匹配，直到没有匹配的格式，就提示文本

video 常用属性

属性	值	描述
autoplay	autoplay	视频就绪自动播放（谷歌浏览器需要添加muted来解决自动播放问题）
controls	controls	向用户显示播放控件
width	pixels(像素)	设置播放器宽度
height	pixels(像素)	设置播放器高度
loop	loop	播放完是否继续播放该视频，循环播放
preload	auto（预先加载视频） none（不应加载视频）	规定是否预加载视频(如果有了autoplay 就忽略该属性)
src	url	视频url地址
poster	Imgurl	加载等待的画面图片
muted	muted	静音播放

属性很多，有一些属性需要大家重点掌握：

- `autoplay` 自动播放
  - 注意： 在google浏览器上面，默认禁止了自动播放，如果想要自动播放的效果，需要设置 muted属性
- `width` 宽度
- `height` 高度
- `loop` 循环播放
- `src` 播放源
- `muted` 静音播放

```
<video src="media/mi.mp4" autoplay="autoplay" muted="muted"  loop="loop" poster="media/mi9.jpg">
</video>
```

音频标签- audio

基本使用

当前 元素支持三种视频格式： 尽量使用 **mp3格式**

使用语法：

```
<audio src="media/music.mp3"></audio>
```

浏览器	MP3	Wav	Ogg
Internet Explorer	YES	NO	NO
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES

兼容写法

由于各个浏览器的支持情况不同，所以我们会有一种兼容性的写法，这种写法了解一下即可

> 多媒体标签音频标签

```
<!-- <audio src="media/snow.mp3" controls="controls"></audio> -->
    <audio controls="controls">
        <source src="media/snow.mp3" type="audio/mpeg" />
        <source src="media/snow.ogg" type="audio/ogg" />
            您的浏览器不支持播放声音
    </audio>
```

上面这种写法，浏览器会匹配audio标签中的source，如果支持就播放，如果不支持往下匹配，直到没有匹配的格式，就提示文本

audio 常用属性

属性	值	描述
autoplay	autoplay	如果出现该属性，则音频在就绪后马上播放。
controls	controls	如果出现该属性，则向用户显示控件，比如播放按钮。
loop	loop	如果出现该属性，则每当音频结束时重新开始播放。
src	url	要播放的音频的 URL。

示例代码：

```
<audio src="media/music.mp3" autoplay="autoplay" controls="controls"></audio>
```

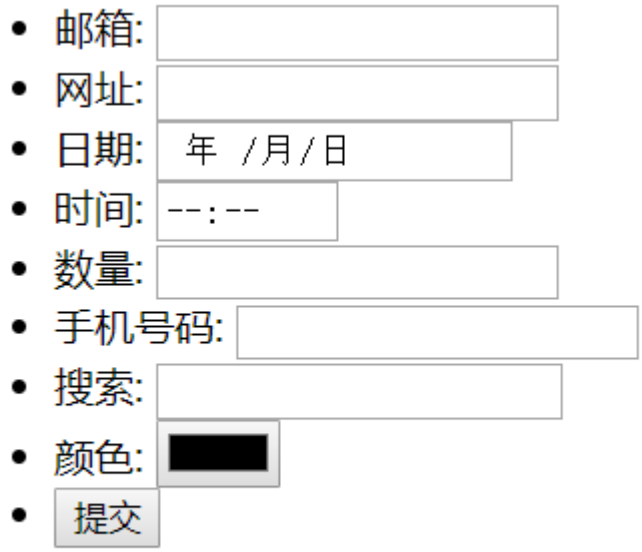
小结

- 音频标签和视频标签使用方式基本一致
- 浏览器支持情况不同
- 谷歌浏览器把音频和视频自动播放禁止了
- 我们可以给视频标签添加 muted 属性来静音播放视频，音频不可以（可以通过JavaScript解决）
- 视频标签是重点，我们经常设置自动播放，不使用 controls 控件，循环和设置大小属性

新增的表单元素（★★）

在H5中，帮我们新增加了很多类型的表单，这样方便了程序员的开发

课堂案例：在这个案例中，熟练了新增表单的用法



> HTML5新增input表单

```
<!-- 我们验证的时候必须添加form表单域 -->
<form action="">
    <ul>
        <li>邮箱: <input type="email" /></li>
        <li>网址: <input type="url" /></li>
        <li>日期: <input type="date" /></li>
        <li>时间: <input type="time" /></li>
        <li>数量: <input type="number" /></li>
        <li>手机号码: <input type="tel" /></li>
        <li>搜索: <input type="search" /></li>
        <li>颜色: <input type="color" /></li>
        <!-- 当我们点击提交按钮就可以验证表单了 -->
        <li> <input type="submit" value="提交"></li>
    </ul>
</form>
```

常见输入类型

```
text password radio checkbox button file hidden submit reset image
```

新的输入类型

属性值	说明
type="email"	限制用户输入必须为Email类型
type="url"	限制用户输入必须为URL类型
type="date"	限制用户输入必须为日期类型
type="time"	限制用户输入必须为时间类型
type="month"	限制用户输入必须为月类型
type="week"	限制用户输入必须为周类型
type="number"	限制用户输入必须为数字类型
type="tel"	手机号码
type="search"	搜索框
type="color"	生成一个颜色选择表单

类型很多，我们现阶段**重点记忆三个**： `number` `tel` `search`

> HTML5新增表单属性

```
    <style>
        input::placeholder {
            color: pink;
        }
    </style>
</head>
<body>
    <form action="">
          <!-- accesskey="s" alt+s -->
        <input type="search" name="search" id="" accesskey="s" required="required"
placeholder="请输入姓名" autofocus="autofocus" autocomplete="off">
        <input type="file" name="" id="" multiple="multiple">
        <input type="submit" value="提交">
    </form>
</body>
```

CSS3新特性

CSS3 的现状

- 新增的CSS3特性有兼容性问题，ie9+才支持
- 移动端支持优于 PC 端
- 不断改进中
- 应用相对广泛
- 现阶段主要学习：新增选择器和盒子模型以及其他特性

CSS3 新增选择器

CSS3 给我们新增了选择器，可以更加便捷，更加自由的选择目标元素。

- 属性选择器
- 结构伪类选择器
- 伪元素选择器

属性选择器（★★）

属性选择器，按照字面意思，都是根据标签中的属性来选择元素

选择符	简介
E[att]	选择具有 att 属性的 E 元素
E[att="val"]	选择具有 att 属性且属性值等于 val 的 E 元素
E[att^="val"]	匹配具有 att 属性且值以 val 开头的 E 元素
E[att$="val"]	匹配具有 att 属性且值以 val 结尾的 E 元素
E[att*="val"]	匹配具有 att 属性且值中含有 val 的 E 元素

> CSS3新增属性选择器

```
<style>
        /* 必须是input 但是同时具有 value这个属性 选择这个元素  [] */
        /* input[value] {
            color:pink;
        } */
        /* 只选择 type =text 文本框的input 选取出来 */
        input[type=text] {
            color: pink;
        }
        /* 选择首先是div 然后 具有class属性 并且属性值 必须是 icon开头的这些元素 */
        div[class^=icon] {
            color: red;
        }
        section[class$=data] {
            color: blue;
        }
        div.icon1 {
            color: skyblue;
        }
        /* 类选择器和属性选择器 伪类选择器 权重都是 10 */
    </style>
</head>
<body>
    <!-- 1. 利用属性选择器就可以不用借助于类或者id选择器 -->
    <!-- <input type="text" value="请输入用户名">
    <input type="text"> -->

    <!-- 2. 属性选择器还可以选择属性=值的某些元素 重点务必掌握的 -->
```

```
    <input type="text" name="" id="">
    <input type="password" name="" id="">
    <!-- 3. 属性选择器可以选择属性值开头的某些元素 -->
    <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>
    <div>我热爱我的生活</div>
    <!-- 4. 属性选择器可以选择属性值结尾的某些元素 -->
    <section class="icon1-data">北京我爱你</section>
    <section class="icon2-data">上海我爱你</section>
    <section class="icon3-ico" aa="hello">真的好吗</section>

</body>
```

- 属性选择器，按照字面意思，都是根据标签中的属性来选择元素
- 属性选择器可以根据元素特定属性的来选择元素。 这样就可以不用借助于类或者id选择器
- 属性选择器也可以选择出来自定义的属性
- **注意：**类选择器、属性选择器、伪类选择器，权重为 10。　　0 0 1 0

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器元素， 常用于根据父级选择器里面的子元素

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	匹配父元素中最后一个 E 元素
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

E:first-child

匹配父元素的第一个子元素E

```
<style>
   ul li:first-child{
     background-color: red;
   }
 </style>

 <ul>
   <li>列表项一</li>
   <li>列表项二</li>
   <li>列表项三</li>
   <li>列表项四</li>
 </ul>
```

E:last-child 则是选择到了最后一个li标签

E:nth-child(n) (★★★)

匹配到父元素的第n个元素

- 匹配到父元素的第2个子元素

  `ul li:nth-child(2){}`
- 匹配到父元素的序号为奇数的子元素

  `ul li:nth-child(odd){}` **odd** 是关键字 奇数的意思（3个字母 ）
- 匹配到父元素的序号为偶数的子元素

  `ul li:nth-child(even){}` **even**（4个字母 ）
- **匹配到父元素的前3个子元素**

  `ul li:nth-child(-n+3){}`

  选择器中的 **n** 是怎么变化的呢？

  因为 n是从 0 ，1，2，3.. 一直递增

  所以 -n+3 就变成了

  - n=0 时 -0+3=3
  - n=1时 -1+3=2
  - n=2时 -2+3=1
  - n=3时 -3+3=0
  - ...

一些常用的公式： 公式不是死的，在这里列举出来让大家能够找寻到这个模式，能够理解代码，这样才能写出满足自己功能需求的代码

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

常用的结构伪类选择器是： `nth-child(n) {...}`

> CSS3新增结构伪类选择器

```
    <style>
        /* 1. 选择ul里面的第一个孩子 小li */
        ul li:first-child {
            background-color: pink;
        }
        /* 2. 选择ul里面的最后一个孩子 小li */
        ul li:last-child {
            background-color: pink;
        }
         /* 3. 选择ul里面的第2个孩子 小li */
         ul li:nth-child(2) {
            background-color: skyblue;
        }
        ul li:nth-child(6) {
            background-color: skyblue;
        }
    </style>
</head>
<body>
    <ul>
        <li>我是第1个孩子</li>
        <li>我是第2个孩子</li>
        <li>我是第3个孩子</li>
        <li>我是第4个孩子</li>
        <li>我是第5个孩子</li>
        <li>我是第6个孩子</li>
        <li>我是第7个孩子</li>
        <li>我是第8个孩子</li>
    </ul>
</body>
```

> -nth-child

```
 <style>
```

```
        /* 1.把所有的偶数 even的孩子选出来 */
        ul li:nth-child(even) {
            background-color: #ccc;
        }
        /* 2.把所有的奇数 odd的孩子选出来 */
        ul li:nth-child(odd) {
            backgr000..
             ound-color: gray;
        }
        /* 3.nth-child(n) 从0开始 每次加1 往后面计算  这里面必须是n 不能是其他的字母 选择了所有的孩子
*/
        /* ol li:nth-child(n) {
            background-color: pink;
        } */
        /* 4.nth-child(2n)母选择了所有的偶数孩子 等价于 even*/
        /* ol li:nth-child(2n) {
            background-color: pink;
        }
        ol li:nth-child(2n+1) {
            background-color: skyblue;
        } */
        /* ol li:nth-child(n+3) {
            background-color: pink;
        } */
        ol li:nth-child(-n+3) {
            background-color: pink;
        }
    </style>
</head>
<body>
    <ul>
        <li>我是第1个孩子</li>
        <li>我是第2个孩子</li>
        <li>我是第3个孩子</li>
        <li>我是第4个孩子</li>
        <li>我是第5个孩子</li>
        <li>我是第6个孩子</li>
        <li>我是第7个孩子</li>
        <li>我是第8个孩子</li>
    </ul>
    <ol>
        <li>我是第1个孩子</li>
        <li>我是第2个孩子</li>
        <li>我是第3个孩子</li>
        <li>我是第4个孩子</li>
        <li>我是第5个孩子</li>
        <li>我是第6个孩子</li>
        <li>我是第7个孩子</li>
        <li>我是第8个孩子</li>
    </ol>
</body>
```

E:nth-child 与 E:nth-of-type 的区别

这里只讲明 **E:nth-child(n)** 和 **E:nth-of-type(n)** 的区别 剩下的 **E:first-of-type E:last-of-type E:nth-last-of-type(n)** 同理做推导即可

> CSS3新增nth-of-type选择器

```
    <style>
        ul li:first-of-type {
            background-color: pink;
        }
        ul li:last-of-type {
            background-color: pink;
        }
        ul li:nth-of-type(even) {
            background-color: skyblue;
        }
        /* nth-child 会把所有的盒子都排列序号 */
        /* 执行的时候首先看  :nth-child(1) 之后回去看 前面 div */
        section div:nth-child(1) {
            background-color: red;
        }
        /* nth-of-type 会把指定元素的盒子排列序号 */
        /* 执行的时候首先看  div指定的元素  之后回去看 :nth-of-type(1) 第几个孩子 */
        section div:nth-of-type(1) {
            background-color: blue;
        }
    </style>
</head>
<body>
    <ul>
        <li>我是第1个孩子</li>
        <li>我是第2个孩子</li>
        <li>我是第3个孩子</li>
        <li>我是第4个孩子</li>
        <li>我是第5个孩子</li>
        <li>我是第6个孩子</li>
        <li>我是第7个孩子</li>
        <li>我是第8个孩子</li>
    </ul>
    <!-- 区别 -->
    <section>
        <p>深圳</p>
        <div>上海</div>
        <div>北京</div>
    </section>
</body>
```

也就是说：

- `E:nth-child(n)` 匹配父元素的第n个子元素E，也就是说，nth-child 对父元素里面所有孩子排序选择（序号是固定的） 先找到第n个孩子，然后看看是否和E匹配
- `E:nth-of-type(n)` 匹配同类型中的第n个同级兄弟元素E，也就是说，对父元素里面指定子元素进行排序选择。 先去匹配E ，然后再根据E 找第n个孩子

小结

- 结构伪类选择器一般用于选择父级里面的第几个孩子
- nth-child 对父元素里面所有孩子排序选择（序号是固定的）先找到第n个孩子，然后看看是否和E匹配
- nth-of-type 对父元素里面指定子元素进行排序选择。 先去匹配E ，然后再根据E 找第n个孩子
- 关于 nth-child（n）我们要知道 n 是从 0 开始计算的，要记住常用的公式
- 如果是无序列表，我们肯定用 nth-child 更多
- 类选择器、属性选择器、伪类选择器，权重为 10

伪元素选择器（★★★）

伪元素选择器可以帮助我们利用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

```
<style>
    div {
        width: 200px;
        height: 200px;
        background-color: pink;
    }
    /* div::before 权重是2 */
    div::before {
        /* 这个content是必须要写的 */
        content: '我';
    }
    div::after {
        content: '一个好人';
    }
</style>
<body>
    <div>
        是
    </div>
</body>
```

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

应用场景一： 字体图标

在实际工作中，字体图标基本上都是用伪元素来实现的，好处在于我们不需要在结构中额外去定义字体图标的标签，通过content属性来设置字体图标的 编码

步骤：

- 结构中定义div盒子
- 在style中先申明字体 @font-face
- 在style中定义after伪元素 div::after{...}
- 在after伪元素中 设置content属性，属性的值就是字体编码
- 在after伪元素中 设置font-family的属性
- 利用定位的方式，让伪元素定位到相应的位置；记住定位口诀：子绝父相

```
<head>
    ...
    <style>
        @font-face {
            font-family: 'icomoon';
            src: url('fonts/icomoon.eot?1lv3na');
            src: url('fonts/icomoon.eot?1lv3na#iefix') format('embedded-opentype'),
                url('fonts/icomoon.ttf?1lv3na') format('truetype'),
                url('fonts/icomoon.woff?1lv3na') format('woff'),
                url('fonts/icomoon.svg?1lv3na#icomoon') format('svg');
            font-weight: normal;
            font-style: normal;
            font-display: block;
        }
        div {
            position: relative;
            width: 200px;
            height: 35px;
            border: 1px solid red;
        }

        div::after {
            position: absolute;
            top: 10px;
            right: 10px;
            font-family: 'icomoon';
            /* content: '□'; */
            content: '\e91e';
            color: red;
            font-size: 18px;
        }
    </style>
```

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

应用场景二： 仿土豆效果

把之前的代码进行了改善

步骤：

- 找到之前写过的仿土豆的结构和样式，拷贝到自己的页面中
- 删除之前的mask遮罩
- 在style中，给大的div盒子（类名叫tudou的），设置 before伪元素
- 这个伪元素充当的是遮罩的角色，所以我们不用设置内容，但是需要设置content属性，属性的值为空字符串
- 给这个遮罩设置宽高，背景颜色，默认是隐藏的
- 当鼠标移入到 div盒子时候，让遮罩显示，利用 hover 来实现

```
<head>
    ...
    <style>
        .tudou {
            position: relative;
            width: 444px;
            height: 320px;
            background-color: pink;
            margin: 30px auto;
        }

        .tudou img {
            width: 100%;
            height: 100%;
        }

        .tudou::before {
            content: '';
            /* 隐藏遮罩层 */
            display: none;
            position: absolute;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            background: rgba(0, 0, 0, .4) url(images/arr.png) no-repeat center;
        }

        /* 当我们鼠标经过了 土豆这个盒子，就让里面before遮罩层显示出来 */
        .tudou:hover::before {
            /* 而是显示元素 */
            display: block;

        }
```

```
    </style>
</head>

<body>
    <div class="tudou">
        <img src="images/tudou.jpg" alt="">
    </div>
    <div class="tudou">
        <img src="images/tudou.jpg" alt="">
    </div>
    <div class="tudou">
        <img src="images/tudou.jpg" alt="">
    </div>
    <div class="tudou">
        <img src="images/tudou.jpg" alt="">
    </div>
</body>
```

应用场景三： 清除浮动

回忆一下清除浮动的方式：

- 额外标签法也称为隔墙法，是 W3C 推荐的做法。
- 父级添加 overflow 属性
- 父级添加after伪元素
- 父级添加双伪元素

额外标签法也称为隔墙法，是 W3C 推荐的做法

注意： 要求这个新的空标签必须是块级元素

后面两种伪元素清除浮动算是第一种额外标签法的一个**升级**和**优化**

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

伪元素必须写的属性

插入的元素必须是块级

不要看见这个元素

核心代码清除浮动

不要看见这个元素

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

转换为块级元素并且一行显示

盒子模型（★★★）

CSS3 中可以通过 box-sizing 来指定盒模型，有2个值：即可指定为 content-box、border-box，这样我们计算盒子大小的方式就发生了改变

可以分成两种情况：

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

如果盒子模型我们改为了box-sizing: border-box ， 那padding和border就不会撑大盒子了（前提padding和border不会超过width宽度）

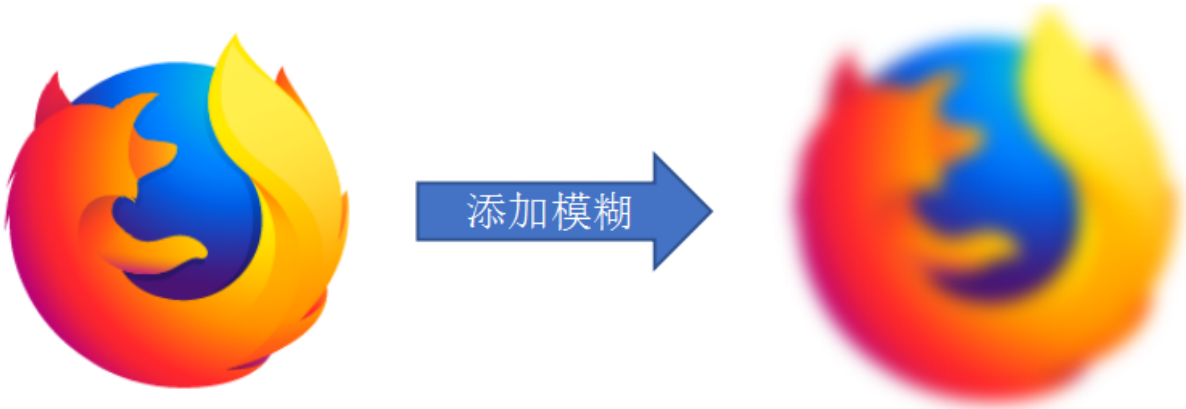
其他特性（★）

图标变模糊 -- CSS3滤镜filter

filter CSS属性将模糊或颜色偏移等图形效果应用于元素

语法：

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



计算盒子宽度 -- calc 函数

calc() 此CSS函数让你在声明CSS属性值时执行一些计算

语法：

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

括号里面可以使用 + - * / 来进行计算

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 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： 是从一个状态 渐渐的过渡到另外一个状态

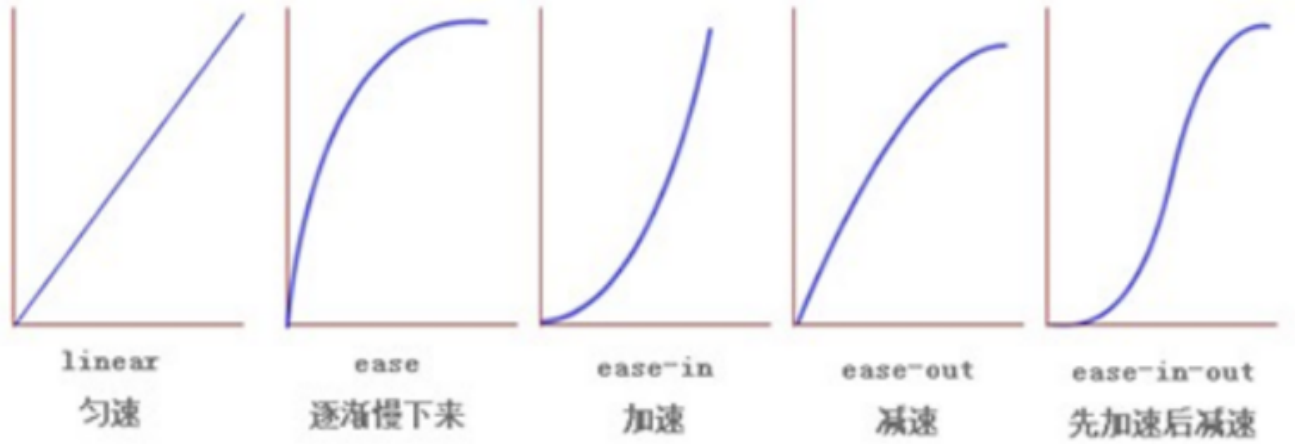
可以让我们页面更好看，更动感十足，虽然 低版本浏览器不支持（ie9以下版本） 但是不会影响页面布局。

我们现在经常和 :hover 一起 搭配使用。

语法：

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

- 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渡， 写一个all 就可以
- 花费时间： 单位是 秒（必须写单位） 比如 0.5s
- 运动曲线： 默认是 ease （可以省略）
- 何时开始：单位是 秒（必须写单位）可以设置延迟触发时间 默认是 0s （可以省略）
- **后面两个属性可以省略**
- **记住过渡的使用口诀： 谁做过渡给谁加**



过渡练习

步骤：

- 创建两个div的盒子，属于的嵌套关系，外层类名叫 bar，里层类名叫 bar_in
- 给外层的bar 这个盒子设置边框，宽高，圆角边框
- 给里层的bar_in 设置 初试的宽度，背景颜色，过渡效果
- 给外层的 bar 添加 hover事件，当触发了hover事件 让里层的bar_in 来进行宽度的变化

代码：

```
<head>
    ...
    <style>
        .bar {
            width: 150px;
            height: 15px;
            border: 1px solid red;
            border-radius: 7px;
            padding: 1px;
        }
        .bar_in {
            width: 50%;
            height: 100%;
            background-color: red;
            /* 谁做过渡给谁加 */
            transition: all .7s;
        }
        .bar:hover .bar_in {
            width: 100%;
        }
    </style>
</head>
<body>
    <div class="bar">
        <div class="bar_in"></div>

    </div>
```

```
</body>
```

transform变换

transform`转换之`translate`

1. `2D` 转换
   - `2D` 转换是改变标签在二维平面上的位置和形状
   - 移动： `translate`
   - 旋转： `rotate`
   - 缩放： `scale`
2. `translate` 语法
   - x 就是 x 轴上水平移动
   - y 就是 y 轴上水平移动

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

3. 重点知识点
   - `2D` 的移动主要是指 水平、垂直方向上的移动
   - `translate` 最大的优点就是不影响其他元素的位置
   - `translate` 中的100%单位，是相对于本身的宽度和高度来进行计算的
   - 行内标签没有效果

> 2D转换之位移translate

```
div {
  background-color:purple;
  width: 200px;
  height: 100px;
  /* 平移 */
  /* 水平垂直移动 100px */
  /* transform: translate(100px, 100px); */

  /* 水平移动 100px */
  /* transform: translate(100px, 0) */

  /* 垂直移动 100px */
  /* transform: translate(0, 100px) */

  /* 水平移动 100px */
  /* transform: translateX(100px); */

  /* 垂直移动 100px */
  transform: translateY(100px)
}
```

> 让盒子实现水平和垂直居中

```
<style>
        div {
            position: relative;
            width: 500px;
            height: 500px;
            background-color: pink;
            /* 1. 我们tranlate里面的参数是可以用 % */
            /* 2. 如果里面的参数是 % 移动的距离是 盒子自身的宽度或者高度来对比的 */
            /* 这里的 50% 就是 50px 因为盒子的宽度是 100px */
            /* transform: translateX(50%); */
        }

        p {
            position: absolute;
            top: 50%;
            left: 50%;
            width: 200px;
            height: 200px;
            background-color: purple;
            /* margin-top: -100px;
            margin-left: -100px; */
            /* translate(-50%, -50%)  盒子往上走自己高度的一半   */
            transform: translate(-50%, -50%);
        }

        span {
            /* translate 对于行内元素是无效的 */
            transform: translate(300px, 300px);
        }
    </style>
</head>

<body>
    <div>
        <p></p>
    </div>
    <span>123</span>
</body>
```

十六、`2D 转换 rotate`

1. rotate 旋转
    - `2D` 旋转指的是让元素在二维平面内顺时针或者逆时针旋转
2. `rotate` 语法

```
/* 单位是：deg */
transform: rotate(度数)
```

3. 重点知识点

- `rotate` 里面跟度数，单位是 `deg`
- 角度为正时，顺时针，角度为负时，逆时针
- 默认旋转的中心点是元素的中心点

4. 代码演示

```
img:hover {
  transform: rotate(360deg)
}
```

> 2d旋转指的是让元素在2维平面内顺时针旋转或者逆时针旋转

使用步骤：

1. 给元素添加转换属性 `transform`
2. 属性值为 `rotate(角度)` 如 `transform:rotate(30deg)` 顺时针方向旋转**30度**

```
div{
      transform: rotate(0deg);
}
```

> 2D转换之旋转rotate

```
 <style>
        img {
            width: 150px;
            /* 顺时针旋转45度 */
            /* transform: rotate(45deg); */
            border-radius: 50%;
            border: 5px solid pink;
            /* 过渡写到本身上，谁做动画给谁加 */
            transition: all 0.3s;
        }

        img:hover {
            transform: rotate(360deg);
        }
    </style>
</head>

<body>
    <img src="media/pic.jpg" alt="">
</body>
```

> CSS3书写三角

```
<style>
        div {
            position: relative;

            width: 249px;
```

```
            height: 35px;
            border: 1px solid #000;
        }

        div::after {
            content: "";
            position: absolute;
            top: 8px;
            right: 15px;
            width: 10px;
            height: 10px;
            border-right: 1px solid #000;
            border-bottom: 1px solid #000;
            transform: rotate(45deg);
            transition: all 0.2s;
        }
        /* 鼠标经过div  里面的三角旋转 */

        div:hover::after {
            transform: rotate(225deg);
        }
    </style>
</head>

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

二、设置元素旋转中心点(transform-origin)

1. `transform-origin` 基础语法

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

2. 重要知识点
   - 注意后面的参数 x 和 y 用空格隔开
   - x y 默认旋转的中心点是元素的中心 (50% 50%)，等价于 `center` `center`
   - 还可以给 x y 设置像素或者方位名词(`top` 、 `bottom` 、 `left` 、 `right` 、 `center`)

> **设置元素转换中心点**

```
<style>
        div {
            width: 200px;
            height: 200px;
            background-color: pink;
            margin: 100px auto;
            transition: all 1s;
            /* 1.可以跟方位名词 */

            /* transform-origin: left bottom; */
```

```
            /* 2. 默认的是 50%  50%  等价于 center  center */
            /* 3. 可以是px 像素 */
            transform-origin: 50px 50px;
        }

        div:hover {
            transform: rotate(360deg);
        }
    </style>
</head>

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

> 旋转中心点案例

```
 <style>
        div {
            overflow: hidden;
            width: 200px;
            height: 200px;
            border: 1px solid pink;
            margin: 10px;
            float: left;
        }
        div::before {
            content: "我喜欢上海";
            display: block;
            width: 100%;
            height: 100%;
            background-color: hotpink;
            transform: rotate(180deg);
            transform-origin: left bottom;
            transition: all 0.4s;
        }
        /* 鼠标经过div 里面的before 复原 */
        div:hover::before {
            transform: rotate(0deg);
        }
    </style>
</head>

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

四、2D 转换之 scale

1. `scale` 的作用
    - 用来控制元素的放大与缩小
2. 语法

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

3. 知识要点
    - 注意，x 与 y 之间使用逗号进行分隔
    - `transform: scale(1, 1)`: 宽高都放大一倍，相当于没有放大
    - `transform: scale(2, 2)`: 宽和高都放大了二倍
    - `transform: scale(2)`: 如果只写了一个参数，第二个参数就和第一个参数一致
    - `transform:scale(0.5, 0.5)`: 缩小
    - `scale` 最大的优势：可以设置转换中心点缩放，默认以中心点缩放，而且不影响其他盒子

```
    div:hover {
        /* 注意，数字是倍数的含义，所以不需要加单位 */
        /* transform: scale(2, 2) */

        /* 实现等比缩放，同时修改宽与高 */
        /* transform: scale(2) */

        /* 小于 1 就等于缩放*/
        transform: scale(0.5, 0.5)
    }
```

> 2D转换之缩放scale

```
 <style>
        div {
            width: 200px;
            height: 200px;
            background-color: pink;
            margin: 100px auto;
            /* transform-origin: left bottom; */
        }
        div:hover {
            /* 1. 里面写的数字不跟单位 就是倍数的意思 1 就是1倍  2就是 2倍 */
            /* transform: scale(x, y); */
            /* transform: scale(2, 2); */
            /* 2. 修改了宽度为原来的2倍  高度 不变 */
            /* transform: scale(2, 1); */
            /* 3. 等比例缩放 同时修改宽度和高度，我们有简单的写法  以下是 宽度修改了2倍，高度默认和第一个参数一样*/
            /* transform: scale(2); */
            /* 4. 我们可以进行缩小  小于1 就是缩放 */
            /* transform: scale(0.5, 0.5); */
            /* transform: scale(0.5); */

            /* 5. scale 的优势之处： 不会影响其他的盒子 而且可以设置缩放的中心点*/
```

```
            /* width: 300px;
            height: 300px; */
            transform: scale(2);
        }
    </style>
</head>
<body>
    <div></div>
    123123
</body>
```

> 图片放大案例

```
<style>
        div {
            overflow: hidden;
            float: left;
            margin: 10px;
        }
        div img {
            transition: all .4s;
        }
        div img:hover {
            transform: scale(1.1);
        }
    </style>
</head>
<body>
    <div>
        <a href="#"><img src="media/scale.jpg" alt=""></a>
    </div>
    <div>
        <a href="#"><img src="media/scale.jpg" alt=""></a>
    </div>
    <div>
        <a href="#"><img src="media/scale.jpg" alt=""></a>
    </div>
</body>
```

> 分页按钮案例

```
<style>
        li {
            float: left;
            width: 30px;
            height: 30px;
            border: 1px solid pink;
            margin: 10px;
            text-align: center;
            line-height: 30px;
            list-style: none;

            border-radius: 50%;
```

```
            cursor: pointer;
            transition: all .4s;
        }

        li:hover {
            transform: scale(1.2);
        }
    </style>
</head>

<body>
    <ul>
        <li>1</li>
        <li>2</li>
        <li>3</li>
        <li>4</li>
        <li>5</li>
        <li>6</li>
        <li>7</li>
    </ul>
</body>
```

2D转换综合写法顺序

```
 <style>
        div {
            width: 200px;
            height: 200px;
            background-color: pink;
            transition: all .5s;
        }
        div:hover {
            /* transform: rotate(180deg) translate(150px, 50px); */
            /* 我们同时有位移和其他属性，我们需要把位移放到最前面 */
            transform: translate(150px, 50px) rotate(180deg) scale(1.2);
        }
    </style>
</head>
<body>
    <div></div>
</body>
```

七、 2D 转换综合写法以及顺序问题

1. 知识要点
    - 同时使用多个转换，其格式为 `transform: translate() rotate() scale()`
    - 顺序会影响到转换的效果(先旋转会改变坐标轴方向)
    - 但我们同时有位置或者其他属性的时候，要将位移放到最前面

```
div:hover {
  transform: translate(200px, 0) rotate(360deg) scale(1.2)
}
```

八、 动画(animation)

1. 什么是动画
   - 动画是 `CSS3` 中最具颠覆性的特征之一，可通过设置多个节点来精确的控制一个或者一组动画，从而实现复杂的动画效果
2. 动画的基本使用
   - 先定义动画
   - 在调用定义好的动画
3. 语法格式(定义动画)

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

1. 语法格式(使用动画)

```
div {
        /* 调用动画 */
    animation-name: 动画名称;
        /* 持续时间 */
        animation-duration: 持续时间;
}
```

1. 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
   - 在 @keyframs 中规定某项 CSS 样式，就由创建当前样式逐渐改为新样式的动画效果
   - 动画是使元素从一个样式逐渐变化为另一个样式的效果，可以改变任意多的样式任意多的次数
   - 用百分比来规定变化发生的时间，或用 `from` 和 `to` ，等同于 0% 和 100%

> CSS3动画基本使用

```
<style>
        /* 我们想页面一打开，一个盒子就从左边走到右边 */
        /* 1. 定义动画 */
        @keyframes move {
            /* 开始状态 */
            0% {
                transform: translateX(0px);
            }
            /* 结束状态 */
```

```
            100% {
                transform: translateX(1000px);
            }
        }
        div {
            width: 200px;
            height: 200px;
            background-color: pink;
            /* 2. 调用动画 */
            /* 动画名称 */
            animation-name: move;
            /* 持续时间 */
            animation-duration: 2s;
        }
    </style>
</head>
<body>
    <div></div>
</body>
```

> 动画序列

```
<style>
    /* from to 等价于  0% 和  100% */
    /* @keyframes move {
        from {
            transform: translate(0, 0);
        }
        to {
            transform: translate(1000px, 0);
        }
    } */
    /* 动画序列 */
    /* 1. 可以做多个状态的变化 keyframe 关键帧 */
    /* 2. 里面的百分比要是整数 */
    /* 3. 里面的百分比就是 总的时间（我们这个案例10s）的划分 25% * 10  =  2.5s */

    @keyframes move {
        0% {
            transform: translate(0, 0);
        }
        25% {
            transform: translate(1000px, 0)
        }
        50% {
            transform: translate(1000px, 500px);
        }
        75% {
            transform: translate(0, 500px);
        }
        100% {
            transform: translate(0, 0);
        }
```

```
        }
        div {
            width: 100px;
            height: 100px;
            background-color: pink;
            animation-name: move;
            animation-duration: 10s;
        }
    </style>
</head>
<body>
    <div>
    </div>
</body>
```

十、动画常见属性

1. 常见的属性

属性	描述
@keyframes	规定动画。
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	规定动画是否正在运行或暂停。默认是"running",还有"paused"。
animation-fill-mode	规定动画结束后状态，保持forwards回到起始backwards

> 动画属性

```
<style>
        @keyframes move {
            0% {
                transform: translate(0, 0);
            }
            100% {
                transform: translate(1000px, 0);
            }

        }
```

```
        div {
            width: 100px;
            height: 100px;
            background-color: pink;
            /* 动画名称 */
            animation-name: move;
            /* 持续时间 */
            /* animation-duration: 2s; */
            /* 运动曲线 */
            /* animation-timing-function: ease; */
            /* 何时开始 */
            animation-delay: 1s;
            /* 重复次数  iteration 重复的 conut 次数  infinite  无限 */
            /* animation-iteration-count: infinite; */
            /* 是否反方向播放 默认的是 normal  如果想要反方向 就写 alternate */
            /* animation-direction: alternate; */
            /* 动画结束后的状态 默认的是 backwards  回到起始状态 我们可以让他停留在结束状态
forwards */
            /* animation-fill-mode: forwards; */
            /* animation: name duration timing-function delay iteration-count direction
fill-mode; */
            /* animation: move 2s linear 0s 1 alternate forwards; */
            /* 前面2个属性 name  duration 一定要写 */
            /* animation: move 2s linear  alternate forwards; */
        }
        div:hover {
            /* 鼠标经过div 让这个div 停止动画，鼠标离开就继续动画 */
            animation-play-state: paused;
        }
    </style>
</head>
<body>
    <div>
    </div>
</body>
```

十一、 动画简写方式

1. 动画简写方式

```
/* 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 起始与结束状态 */
animation: name duration timing-function delay iteration-count direction fill-mode
```

2. 知识要点
   - 简写属性里面不包含 `animation-paly-state`
   - 暂停动画 `animation-paly-state: paused`；经常和鼠标经过等其他配合使用
   - 要想动画走回来，而不是直接调回来：`animation-direction: alternate`
   - 盒子动画结束后，停在结束位置：`animation-fill-mode: forwards`
3. 代码演示

```
animation: move 2s linear 1s infinite alternate forwards;
```

> 大数据热点图

```
<style>
      body {
          background-color: #333;
      }
      .map {
          position: relative;
          width: 747px;
          height: 616px;
          background: url(media/map.png) no-repeat;
          margin: 0 auto;
      }
      .city {
          position: absolute;
          top: 227px;
          right: 193px;
          color: #fff;
      }
      .tb {
          top: 500px;
          right: 80px;
      }
      .dotted {
          width: 8px;
          height: 8px;
          background-color: #09f;
          border-radius: 50%;
      }
      .city div[class^="pulse"] {
          /* 保证我们小波纹在父盒子里面水平垂直居中 放大之后就会中心向四周发散 */
          position: absolute;
          top: 50%;
          left: 50%;
          transform: translate(-50%, -50%);
          width: 8px;
          height: 8px;
          box-shadow: 0 0 12px #009dfd;
          border-radius: 50%;
          animation: pulse 1.2s linear infinite;
      }
      .city div.pulse2 {
          animation-delay: 0.4s;
      }
      .city div.pulse3 {
          animation-delay: 0.8s;
      }
      @keyframes pulse {
          0% {}
          70% {
              /* transform: scale(5);  我们不要用scale 因为他会让 阴影变大*/
              width: 40px;
```

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

十二、速度曲线细节

1. 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）

> 速度曲线步长

```
<style>
```

```
        div {
            overflow: hidden;
            font-size: 20px;
            width: 0;
            height: 30px;
            background-color: pink;
            /* 让我们的文字强制一行内显示 */
            white-space: nowrap;
            /* steps 就是分几步来完成我们的动画 有了steps 就不要在写 ease 或者linear 了 */
            animation: w 4s steps(10) forwards;
        }
        @keyframes w {
            0% {
                width: 0;
            }
            100% {
                width: 200px;
            }
        }
    </style>
</head>
<body>
    <div>中国我爱你你是我全部</div>
</body>
```

十三、奔跑的熊大

```
 <style>
        body {
            background-color: #ccc;
        }
        div {
            position: absolute;
            width: 200px;
            height: 100px;
            background: url(media/bear.png) no-repeat;
            /* 我们元素可以添加多个动画， 用逗号分隔 */
            animation: bear .4s steps(8) infinite, move 3s forwards;
        }
        @keyframes bear {
            0% {
                background-position: 0 0;
            }
            100% {
                background-position: -1600px 0;
            }
        }
        @keyframes move {
            0% {
                left: 0;
            }
            100% {
                left: 50%;
```

```
                /* margin-left: -100px; */
                transform: translateX(-50%);
            }
        }
    </style>
</head>
<body>
    <div></div>
</body>
```