

前言

最近查看了几位同事的代码，发现很多CSS编写习惯都是清一色的类而无相应的选择器，层层嵌套的标签都包含至少一个类。有些同学会问，很多文章都说 选择器 有性能问题，为何还需使用 选择器 呢？

是的， 选择器 和 类 对比起来性能上确实没后者那么好，但是如今浏览器对于CSS的解析速度已得到大大的提升，完全可忽略 选择器 那丁点的性能问题。有兴趣的同学可自行百度搜索 CSS选择器性能 的相关问题学习。多一个技巧多一份保障！

本章不细说 选择器 的性能问题，而是细说怎样用好 选择器 。先来对 选择器 做一个功能性的分类。当然熟悉全部CSS选择器是玩转CSS的 最最最最基本功 。

分类

在讲解 选择器 的奇妙用处前，还是先把选择器分类记忆吧。没错，笔者就是喜欢总结。由于 选择器 的标准概念上无作出明确的分类，以下的分类是为了方便记忆而整理的。

基础选择器

选择器	别名	说明	版本
tag	标签选择器	指定类型的 标签	1
#id	ID选择器	指定身份的 标签	1
.class	类选择器	指定类名的 标签	1
*	通配选择器	所有类型的 标签	2

层次选择器

选择器	别名	说明	版本
<code>elemP elemC</code>	后代选择器	元素的 后代元素	1
<code>elemP>elemC</code>	子代选择器	元素的 子代元素	2
<code>elem1+elem2</code>	相邻同胞选择器	元素相邻的 同胞元素	2
<code>elem1~elem2</code>	通用同胞选择器	元素后面的 同胞元素	3

集合选择器

选择器	别名	说明	版本
<code>elem1,elem2</code>	并集选择器	多个指定的 元素	1
<code>elem.class</code>	交集选择器	指定类名的 元素	1

条件选择器

选择器	说明	版本
<code>:lang</code>	指定标记语言的 元素	2
<code>:dir()</code>	指定编写方向的 元素	4
<code>:has</code>	包含指定元素的 元素	4
<code>:is</code>	指定条件的 元素	4
<code>:not</code>	非指定条件的 元素	4
<code>:where</code>	指定条件的 元素	4
<code>:scope</code>	指定 元素 作为参考点	4
<code>:any-link</code>	所有包含 href 的 链接元素	4
<code>:local-link</code>	所有包含 href 且属于绝对地址的 链接元素	4

行为选择器

选择器	说明	版本
<code>:active</code>	鼠标激活的 元素	1
<code>:hover</code>	鼠标悬浮的 元素	1
<code>::selection</code>	鼠标选中的 元素	3

状态选择器

选择器	说明	版本
<code>:target</code>	当前锚点的 元素	3
<code>:link</code>	未访问的 链接元素	1
<code>:visited</code>	已访问的 链接元素	1
<code>:focus</code>	输入聚焦的 表单元素	2
<code>:required</code>	输入必填的 表单元素	3
<code>:valid</code>	输入合法的 表单元素	3
<code>:invalid</code>	输入非法的 表单元素	3
<code>:in-range</code>	输入范围以内的 表单元素	3
<code>:out-of-range</code>	输入范围以外的 表单元素	3
<code>:checked</code>	选项选中的 表单元素	3
<code>:optional</code>	选项可选的 表单元素	3
<code>:enabled</code>	事件启用的 表单元素	3
<code>:disabled</code>	事件禁用的 表单元素	3
选择器	说明	版本
<code>:read-only</code>	只读的 表单元素	3

<code>:read-write</code>	可读可写的 表单元素	3
<code>:target-within</code>	内部锚点元素处于激活状态的 元素	4
<code>:focus-within</code>	内部表单元素处于聚焦状态的 元素	4
<code>:focus-visible</code>	输入聚焦的 表单元素	4
<code>:blank</code>	输入为空的 表单元素	4
<code>:user-invalid</code>	输入合法的 表单元素	4
<code>:indeterminate</code>	选项未定的 表单元素	4
<code>:placeholder-shown</code>	占位显示的 表单元素	4
<code>:current()</code>	浏览中的 元素	4
<code>:past()</code>	已浏览的 元素	4
<code>:future()</code>	未浏览的 元素	4
<code>:playing</code>	开始播放的 媒体元素	4
<code>:paused</code>	暂停播放的 媒体元素	4

结构选择器

选择器	说明	版本
<code>:root</code>	文档的 根元素	3
<code>:empty</code>	无子元素的 元素	3
<code>:first-letter</code>	元素的 首字母	1
<code>:first-line</code>	元素的 首行	1
<code>:nth-child(n)</code>	元素中指定顺序索引的 元素	3
<code>:nth-last-child(n)</code>	元素中指定逆序索引的 元素	3
<code>:first-child</code>	元素中为首的元素	2
<code>:last-child</code>	元素中为尾的元素	3
<code>:only-child</code>	父元素仅有该元素的 元素	3
<code>:nth-of-type(n)</code>	标签中指定顺序索引的 标签	3
<code>:nth-last-of-type(n)</code>	标签中指定逆序索引的 标签	3
<code>:first-of-type</code>	标签中为首 的 标签	3
<code>:last-of-type</code>	标签中为尾 的 标签	3
<code>:only-of-type</code>	父元素仅有该标签的 标签	3

属性选择器

选择器	说明	版本
<code>[attr]</code>	指定属性的 元素	2
<code>[attr=val]</code>	属性等于指定值的 元素	2
<code>[attr*=val]</code>	属性包含指定值的 元素	3
<code>[attr^=val]</code>	属性以指定值开头的 元素	3
<code>[attr\$=val]</code>	属性以指定值结尾的 元素	3
<code>[attr~=val]</code>	属性包含指定值(完整单词)的 元素 (不推荐使用)	2
<code>[attr\ =val]</code>	属性以指定值(完整单词)开头的 元素 (不推荐使用)	2

伪元素

选择器	说明	版本
<code>::before</code>	在元素前插入的内容	2
<code>::after</code>	在元素后插入的内容	2

优势

话说选择器若无用处，那 **W3C** 还干嘛把它纳入到标准里呢？**选择器** 的劣势就不啰嗦了，使用不当可能会引起 **解析性能问题**，这个对于现代浏览器来说几乎可忽略，除非你还是 **IE Explorer** 的忠实粉丝。使用选择器有什么好处呢？笔者给各位同学总结一下。

- 对于那些结构与行为分离的写法，使用 **sass/less** 编写属性时结构会更清晰易读
- 减少很多无用或少用的类，保持 **css文件** 的整洁性和观赏性，代码也是一门艺术
- 减少修改类而有可能导致样式失效的问题，有时修改类但无确保HTML中和CSS中的一致而导致样式失效
- 减少无实质性使用的类，例如很多层嵌套的标签，这些标签可能只使用到一个CSS属性，就没必要建个类关联
- 使用选择器可实现一些看似只能由JS才能实现的效果，既可减少代码量也可减少JS对DOM的操作，使得交互效果更流畅

场景

由于选择器太多，笔者选择几个最具代表性的耍耍，通过选择器的妙用实现一些看似只能由JS才能实现的效果。未提到的选择器可能在其他地方穿插着讲解，请各位同学放心学习。

+和~

+/~ 都是作用于当前节点后的同胞节点，但是两者有一个明显的区别，**+** 是针对紧随该节点的节点，而 **~** 是针对后面所有的节点，包括紧随该节点的节点。**~** 还可针对一些特定类和选择器的节点，所以其使用性更广泛。

另外，**+/~** 通常都会结合 **:checked** 完成一些高难度的纯CSS效果，当 **<input>** 触发了 **:checked** 选中状态后可通过 **+/~** 带动后面指定的节点声明一些特别属性。

通常其CSS代码形式如下。

```
input:checked + div {}  
input:checked ~ div {}
```

+/~ 的用途很广，静态效果和动态效果都能用上它，是两个很关键的选择器。以下通过动静结合的方式展示 **+/~** 的用途。


```
<div class="specify-selector">
  <ul class="list">
    <li>同胞元素</li>
    <li class="next">当前元素</li>
    <li>同胞元素</li>
    <li>同胞元素</li>
    <li>同胞元素</li>
  </ul>
  <ul class="list">
    <li>同胞元素</li>
    <li class="next-all">当前元素</li>
    <li>同胞元素</li>
    <li>同胞元素</li>
    <li>同胞元素</li>
  </ul>
  <ul class="list">
    <li>同胞元素</li>
```

```
<li class="next-filter">当前元素</li>
<li>同胞元素</li>
<li class="filter">同胞元素</li>
<li>同胞元素</li>
</ul>
</div>
<div class="specify-selector">
  <div class="button">
    <input id="btn1" type="radio" name="btns" hidden>
    <label for="btn1">点击我切换样式</label>
  </div>
  <div class="button">
    <input id="btn2" type="radio" name="btns" hidden>
    <label for="btn2">点击我切换样式</label>
  </div>
  <div class="button">
    <input id="btn3" type="radio" name="btns" hidden>
    <label for="btn3">点击我切换样式</label>
  </div>
</div>
```

```
.specify-selector {
  display: flex;
  & + .specify-selector {
    margin-top: 20px;
  }
  .list {
    border: 1px solid #f66;
    width: 200px;
    line-height: 2;
    font-weight: bold;
    font-size: 20px;
    color: #f66;
    & + .list {
      margin-left: 20px;
    }
  }
}
```

```
li {
    padding: 0 10px;
}
.next {
    background-color: #66f;
    color: #fff;
    & + li {
        background-color: #f90;
        color: #fff;
    }
}
.next-all {
    background-color: #66f;
    color: #fff;
    & ~ li {
        background-color: #09f;
        color: #fff;
    }
}
.next-filter {
    background-color: #66f;
    color: #fff;
    & ~ .filter {
        background-color: #09f;
        color: #fff;
    }
}
.button {
    & + .button {
        margin-left: 20px;
    }
    label {
        display: block;
        padding: 0 10px;
        height: 40px;
        background-color: #3c9;
```

```

        cursor: pointer;
        line-height: 40px;
        font-size: 16px;
        color: #fff;
        transition: all 300ms;
    }
    input:checked + label {
        padding: 0 20px;
        border-radius: 20px;
        background-color: #f66;
    }
}
}

```

☑ 在线演示: [Here](#)

☑ 在线源码: [Here](#)

:hover

:hover 作用于鼠标悬浮的节点，是一个很好用的选择器。在特定场景可代替 **mouseenter** 和 **mouseleave** 两个鼠标事件，加上 **transition** 让节点的动画更丝滑。

结合 **attr()** 有一个很好用的场景，就是鼠标悬浮在某个节点上显示提示浮层，提示浮层里包含着该动作的文本。

- 给节点标记一个用户属性 **data-***
- 当鼠标悬浮在该节点上触发 **:hover**
- 通过 **attr()** 获取 **data-*** 的内容
- 将 **data-*** 的内容赋值到伪元素的 **content** 上



```

<ul class="hover-tips">
  <li data-name="姨妈红"></li>

```

```
<li data-name="基佬紫"></li>
<li data-name="笋底橙"></li>
<li data-name="姣婆蓝"></li>
<li data-name="大粪青"></li>
<li data-name="原谅绿"></li>
</ul>
```

```
$color-list: #f66 #66f #f90 #09f #9c3 #3c9;
.hover-tips {
  display: flex;
  justify-content: space-between;
  width: 200px;
  li {
    position: relative;
    padding: 2px;
    border: 2px solid transparent;
    border-radius: 100%;
    width: 24px;
    height: 24px;
    background-clip: content-box;
    cursor: pointer;
    transition: all 300ms;
    &::before,
    &::after {
      position: absolute;
      left: 50%;
      bottom: 100%;
      opacity: 0;
      transform: translate3d(0, -30px, 0);
      transition: all 300ms;
    }
    &::before {
      margin: 0 0 12px -35px;
      border-radius: 5px;
      width: 70px;
      height: 30px;
```

```
background-color: rgba(#000, .5);
line-height: 30px;
text-align: center;
color: #fff;
content: attr(data-name);
}
&::after {
margin-left: -6px;
border: 6px solid transparent;
border-top-color: rgba(#000, .5);
width: 0;
height: 0;
content: ""';
}
@each $color in $color-list {
  $index: index($color-list, $color);
  &:nth-child(#{ $index }) {
    background-color: $color;
    &:hover {
      border-color: $color;
    }
  }
}
&:hover {
  &::before,
  &::after {
    opacity: 1;
    transform: translate3d(0, 0, 0);
  }
}
}
```

:valid和:invalid

很多同学可能还会使用JS去判断表单输入内容是否合法，其实HTML5发布后，可用纯CSS完成这些工作，正确搭配一些属性能大大减少校验表单的代码量。

完成一个完整的表单验证，需以下HTML属性和选择器搭配。

- **placeholder**：占位，在未输入内容时显示提示文本
- **pattern**：正则，在输入内容时触发正则验证
- **:valid**：作用于输入合法的表单节点
- **:invalid**：作用于输入非法的表单节点

```
<input type="text" placeholder="" pattern="">
```

```
input:valid {}
```

```
input:invalid {}
```

这个 **pattern** 与JS正则有点不同，JS的正则形式是 `/regexp/`，而 **pattern** 的正则形式只需 `/regexp/` 里的 **regexp**。这个校验过程是动态触发的，监听了 **input** 这个键盘事件，当输入内容合法时触发 **:valid**，当输入内容非法时触发 **:invalid**。

```
<form class="form-validation">
  <div>
    <label>名字</label>
    <input type="text" placeholder="请输入你的名字(1到10个中文)" pattern="^[a-zA-Z0-9]{1,10}" />
  </div>
  <div>
    <label>手机</label>
    <input type="text" placeholder="请输入你的手机" pattern="^[1-9]{1}[3-9]{1}[0-9]{9}" />
  </div>
  <div>
    <label>简介</label>
    <textarea required></textarea>
  </div>
</form>
```



```
.form-validation {  
  width: 500px;  
  div + div {  
    margin-top: 10px;  
  }  
  label {  
    display: block;  
    padding-bottom: 5px;  
    font-weight: bold;  
    font-size: 16px;  
  }  
  input,  
  textarea {  
    display: block;  
    padding: 0 20px;  
    border: 1px solid #ccc;  
    width: 100%;  
    height: 40px;  
    outline: none;  
    caret-color: #09f;  
    transition: all 300ms;  
    &:valid {  
      border-color: #3c9;  
    }  
    &:invalid {  
      border-color: #f66;  
    }  
  }  
  textarea {  
    height: 122px;  
    resize: none;  
    line-height: 30px;  
    font-size: 16px;  
  }  
}
```

- ☑ 在线演示: [Here](#)
- ☑ 在线源码: [Here](#)

:checked

:checked 作用于选项选中的表单节点, 当 `<input>` 的 `type` 设置成 `radio` 和 `checkbox` 时可用。在**CSS神操作骚技巧**中是一个很重要的技巧, 主要是用于模拟鼠标点击事件。



```
<input class="ios-switch" type="checkbox">
```

```
.btn {  
  border-radius: 31px;  
  width: 102px;  
  height: 62px;  
  background-color: #e9e9eb;  
}  
  
.ios-switch {  
  position: relative;  
  appearance: none;  
  cursor: pointer;  
  transition: all 100ms;  
  @extend .btn;  
  &::before {  
    position: absolute;  
    content: "";  
    transition: all 300ms cubic-bezier(.45, 1, .4, 1);  
    @extend .btn;  
  }  
  &::after {
```

```
position: absolute;
left: 4px;
top: 4px;
border-radius: 27px;
width: 54px;
height: 54px;
background-color: #fff;
box-shadow: 1px 1px 5px rgba(#000, .3);
content: "";
transition: all 300ms cubic-bezier(.4, .4, .25, 1.35);
}
&:checked {
  background-color: #5eb662;
  &::before {
    transform: scale(0);
  }
  &::after {
    transform: translateX(40px);
  }
}
}
```

☑ 在线演示: [Here](#)

☑ 在线源码: [Here](#)

<input> 与 <label> 的巧妙搭配

上述有提到与 `+/~` 的搭配使用, 在此还有一个很重要的技巧, 就是结合 `<label>` 使用。为何要结合 `<label>` 呢? 因为要让 `input:checked + div {}` 或 `input:checked ~ div {}` 起效, 其HTML结构必须像以下那样。

```
<input type="radio">
<div></div>
```

这样就无法分离结构与行为了，导致CSS必须跟着HTML走，只能使用绝对定位将 `<input>` 固定到指定位置。使用 `<label>` 绑定 `<input>`，可将 `<input>` 的鼠标选择事件转移到 `<label>` 上，由 `<label>` 控制选中状态。那么HTML结构可改为以下那样，此时的 `<input>` 可设置 `hidden` 隐藏起来，不参与任何排版。

```
<input type="radio" id="btn" hidden>
<div>
  <label for="btn">
</div>
```

`<input>` 使用 `id` 与 `<label>` 使用 `for` 关联起来，而 `hidden` 使 `<input>` 隐藏起来，不占用页面任何位置，此时 `<label>` 放置在页面任何位置都行。

```
input:checked + div {}
input:checked ~ div {}
```

笔者使用纯CSS实现的标签导航是一个很好的学习用例，在第8章变量计算有提及。

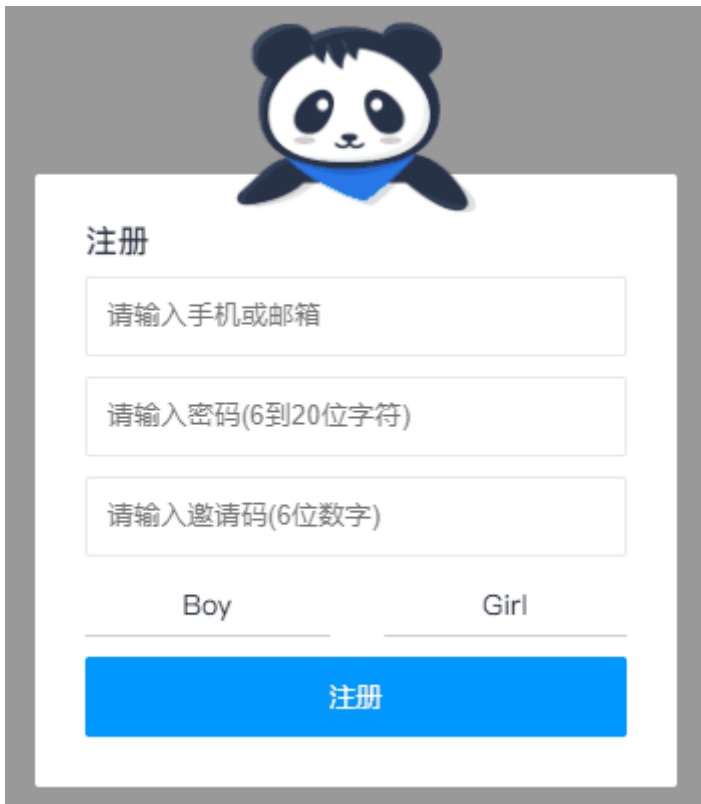


- ☑ 在线演示: [Here](#)
- ☑ 在线源码: [Here](#)

:focus-within

`:focus-within` 作用于内部表单节点处于聚焦状态的节点。它监听当前节点里是否有表单节点，且该表单节点是否处于聚焦状态。

有些同学听上去可能觉得拗口，其实它是一个简单易用的属性。表单控件触发 `focus` 和 `blur` 两个鼠标事件后往祖先节点冒泡，在祖先节点上通过 `:focus-within` 捕获该冒泡事件声明样式。



```
<form class="bubble-distribution">
  <h3>注册</h3>
  <div class="accout">
    <input type="text" placeholder="请输入手机或邮箱" pattern="^[13456
    
    <input type="password" placeholder="请输入密码(6到20位字符)" patter
    
    <input type="text" placeholder="请输入邀请码(6位数字)" pattern="^[
    <button type="button">查询</button>
    
<ul>
  <li>
    <input id="male" type="radio" name="sex">
    <label for="male">Boy</label>
  </li>
  <li>
    <input id="female" type="radio" name="sex">
    <label for="female">Girl</label>
  </li>
</ul>
<button type="button">注册</button>
</form>
```

```
.bubble-distribution {
  position: relative;
  margin-top: 50px;
  padding: 25px;
  border-radius: 2px;
  width: 320px;
  background-color: #fff;
  h3 {
    font-size: 16px;
    color: #333;
  }
  div {
    margin-top: 10px;
  }
  img {
    position: absolute;
    left: 50%;
    bottom: 100%;
    margin: 0 0 -20px -60px;
    width: 120px;
  }
  ul {
```

```
display: flex;
justify-content: space-between;
align-items: center;
margin-top: 10px;
height: 30px;
line-height: 30px;
}
li {
  position: relative;
  width: 45%;
  transition: all 300ms;
  &:focus-within {
    background: linear-gradient(90deg, #09f 50%, transparent 0)
      linear-gradient(90deg, #09f 50%, transparent 0) repeat-x,
      linear-gradient(0deg, #09f 50%, transparent 0) repeat-y,
      linear-gradient(0deg, #09f 50%, transparent 0) repeat-y;
    background-position: 0 0, 0 100%, 0 0, 100% 0;
    background-size: 8px 1px, 8px 1px, 1px 8px, 1px 8px;
    animation: move 500ms infinite linear;
  }
}
input[type=text],
input[type=password] {
  padding: 10px;
  border: 1px solid #e9e9e9;
  border-radius: 2px;
  width: 100%;
  height: 40px;
  outline: none;
  transition: all 300ms;
  &:focus:valid {
    border-color: #09f;
  }
  &:focus:invalid {
    border-color: #f66;
  }
}
```

```
input[type=radio] {
  position: absolute;
  width: 0;
  height: 0;
  &:checked + label {
    border: 3px solid transparent;
    background-color: #09f;
    color: #fff;
  }
}

label {
  display: block;
  border-bottom: 1px solid #ccc;
  width: 100%;
  background-clip: padding-box;
  cursor: pointer;
  text-align: center;
  transition: all 300ms;
}

button {
  overflow: hidden;
  margin-top: 10px;
  border: none;
  border-radius: 2px;
  width: 100%;
  height: 40px;
  outline: none;
  background-color: #09f;
  cursor: pointer;
  color: #fff;
  transition: all 300ms;
}

.accout,
.password,
.code {
  img {
    display: none;
  }
}
```



```
        margin-bottom: -27px;
    }
    &:focus-within {
        img {
            display: block;
        }
        & ~ img {
            display: none;
        }
    }
}

.code {
    display: flex;
    justify-content: space-between;
    button {
        margin-top: 0;
    }
    input {
        &:not(:placeholder-shown) {
            width: 70%;
            & + button {
                width: 25%;
            }
        }
        &:placeholder-shown {
            width: 100%;
            & + button {
                width: 0;
                opacity: 0;
            }
        }
    }
}

}

@keyframes move {
    to {
        background-position: 6% 0, -6% 100%, 0 -6%, 100% 6%;
    }
}
```

```
}  
  
}
```

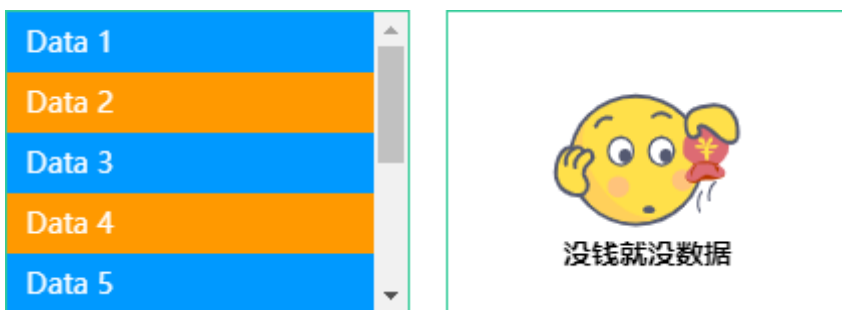
- ☑ 在线演示: [Here](#)
- ☑ 在线源码: [Here](#)

:empty

还有使用JS判断列表集合为空时显示占位符吗？相信很多使用MVVM框架开发的同学都会使用条件判断的方式渲染虚拟DOM，若列表长度不为0则渲染列表，否则渲染占位符。然而CSS提供了一个空判断的选择器 **:empty**，这应该很少同学会注意到。

:empty 作用于无子节点的节点，这个子节点也包括行内匿名盒(**单独的文本内容**)，匿名盒在第4章**盒模型**有提及。以下三种情况均为非空状态，若不出现这三种状态则为空状态，此时 **:empty** 才会触发。

- 仅存在节点: `<div><p>CSS</p></div>`
- 仅存在文本: `<div>CSS</div>`
- 同时存在节点和文本: `<div>Hello <p>CSS</p></div>`



```
<ul class="empty-list">  
  <li v-for="v in 10" :key="v">Data {{v}}</li>  
</ul>  
<ul class="empty-list"></ul>
```

```
$empty: "https://yangzw.vip/img/empty.svg";  
.empty-list {  
  overflow: auto;  
  width: 200px;  
  height: 150px;
```

```
outline: 1px solid #3c9;
&:empty {
  display: flex;
  justify-content: center;
  align-items: center;
  background: url($empty) no-repeat center/100px auto;
  &::after {
    margin-top: 90px;
    font-weight: bold;
    content: "没钱就没数据";
  }
}
& + .empty-list {
  margin-left: 20px;
}
li {
  padding: 0 10px;
  height: 30px;
  background-color: #09f;
  line-height: 30px;
  color: #fff;
  &:nth-child(even) {
    background-color: #f90;
  }
}
```

☑ 在线演示: [Here](#)

☑ 在线源码: [Here](#)

::before和::after

有时为了实现某个效果而往页面里反复添加标签变得很繁琐，添加太多标签反而不好处理而变得难以维护。此时会引入 **伪元素** 这个概念解决上述问题。

伪元素指页面里不存在的元素。**伪元素**在HTML代码里未声明，却能正常显示，在页面渲染时看到这些本来不存在的元素发挥着重要作用。**:before** 和 **:after** 是两个很重要的伪元素，早在CSS2就出现了。

起初 **伪元素** 的前缀使用**单冒号语法**。随着CSS改革，伪元素的前缀被修改成**双冒号语法**，**:before/:after** 从此变成 **::before/::after**，用来区分 **伪类**。若兼容低版本浏览器，还需使用 **:before** 和 **:after**，但是本小册均以 **::before/::after** 编写CSS代码。

伪元素 和 **伪类** 虽然都是选择器，但是它们还是存在一丝丝的差别。

- **伪元素** 通常是一些实体选择器，选择满足指定条件的DOM，例如 **::selection**、**:nth-child(n)** 和 **:first-child**
- **伪类** 通常是一些状态选择器，选择处于特定状态的DOM，例如 **:hover**、**:focus** 和 **:checked**

::before/::after 必须结合 **content** 使用，通常用作修饰节点，为节点插入一些多余的东西，但又不想内嵌一些其他标签。若插入2个以下(包含2个)的修饰，建议使用 **::before/::after**。

以下两个HTML结构是等效的

```
<p>
  <span>:before</span>
  CSS
  <span>:after</span>
</p>
```

```
<p>CSS</p>
```

// 接上一个HTML结构

```
p {
  &::before {
    content: ":before";
  }
  &::after {
    content: ":after";
  }
}
```

::before/::after 最常用的场景就是气泡对话框，圆滚滚的身子带上一个三角形的尾巴。像以下第二个挖空的气泡对话框，其实使用白色填充背景颜色，而小尾巴使用白色的 **::after** 叠加橙色的 **::before** 形成障眼法。



```
<div class="bubble-box">iCSS</div>
<div class="bubble-empty-box">iCSS</div>
```

```
.bubble-box {
  position: relative;
  border-radius: 5px;
  width: 200px;
  height: 50px;
  background-color: #f90;
  line-height: 50px;
  text-align: center;
  font-size: 20px;
  color: #fff;
  &::after {
    position: absolute;
    left: 100%;
    top: 50%;
    margin-top: -5px;
    border: 5px solid transparent;
    border-left-color: #f90;
    content: "";
  }
}

.bubble-empty-box {
  position: relative;
  margin-top: 10px;
  border: 2px solid #f90;
  border-radius: 5px;
  width: 200px;
  height: 50px;
  line-height: 46px;
```

```
text-align: center;
font-size: 20px;
color: #f90;
&::before {
    position: absolute;
    left: 100%;
    top: 50%;
    margin: -5px 0 0 2px;
    border: 5px solid transparent;
    border-left-color: #f90;
    content: "''";
}
&::after {
    position: absolute;
    left: 100%;
    top: 50%;
    margin-top: -4px;
    border: 4px solid transparent;
    border-left-color: #fff;
    content: "''";
}
}
```

☑ 在线演示: [Here](#)

☑ 在线源码: [Here](#)