一个 Web 页面，一个 APP，想让别人用的爽，也就是所谓的良好的用户体验，我觉得他可能包括但不限于：急速的打开速度、眼前一亮的 UI 设计、酷炫的动画效果、丰富的个性化设置、便捷的操作、贴心的细节、关注残障人士，良好的可访问性...

所谓的用户体验设计，其实是一个比较虚的概念，是秉承着**以用户为中心的思想**的一种设计手段，以用户需求为目标而进行的设计。设计过程注重以用户为中心，用户体验的概念从开发的最早期就开始进入整个流程，并贯穿始终。

良好的用户体验设计，是产品每一个环节共同努力的结果。

除去一些很难一蹴而就的，本文将就**页面展示**、**交互细节**、**可访问性**三个方面入手，罗列一些在实际的开发过程中，积攒的一些有益的经验。通过本文，你将能收获到：

1. 了解到一些小细节是如何影响用户体验的
2. 了解到如何在尽量小的开发改动下，提升页面的用户体验
3. 了解到一些优秀的交互设计细节
4. 了解基本的无障碍功能及页面可访问性的含义
5. 了解基本的提升页面可访问性的方法

## 页面展示

就整个页面的展示，页面内容的呈现而言，有一些小细节是需要我们注意的。

### 整体布局

先来看看一些布局相关的问题。

对于大部分 PC 端的项目，我们首先需要考虑的肯定是最外层的一层包裹。假设就是 .g-app-wrapper。

<div class="g-app-wrapper">

<!-- 内部内容 --></div>

首先，对于 .g-app-wrapper，有几点，是我们在项目开发前必须弄清楚的：

1. 项目是全屏布局还是定宽布局？
2. 对于全屏布局，需要适配的最小的宽度是多少？

对于定宽布局，就比较方便了，假设定宽为 1200px，那么：

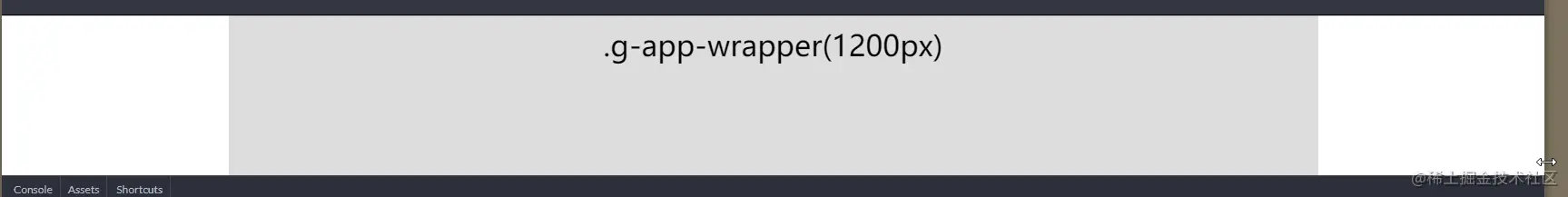
.g-app-wrapper {

width: 1200px;

margin: 0 auto;

}

利用 margin: 0 auto 实现布局的水平居中。在屏幕宽度大于 1200px 时，两侧留白，当然屏幕宽度小于 1200px 时，则出现滚动条，保证内部内容不乱。



对于现代布局，更多的是全屏布局。其实现在也更提倡这种布局，即使用可随用户设备的尺寸和能力而变化的自适应布局。

通常而言是左右两栏，左侧定宽，右侧自适应剩余宽度，当然，会有一个最小的宽度。那么，它的布局应该是这样：

<div class="g-app-wrapper">

<div class="g-sidebar"></div>

<div class="g-main"></div>

</div>

.g-app-wrapper {

display: flex;

min-width: 1200px;

}.g-sidebar {

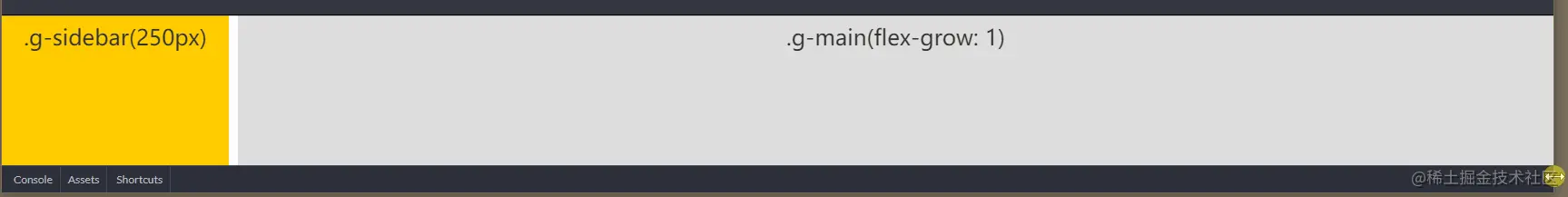
flex-basis: 250px;

margin-right: 10px;

}.g-main {

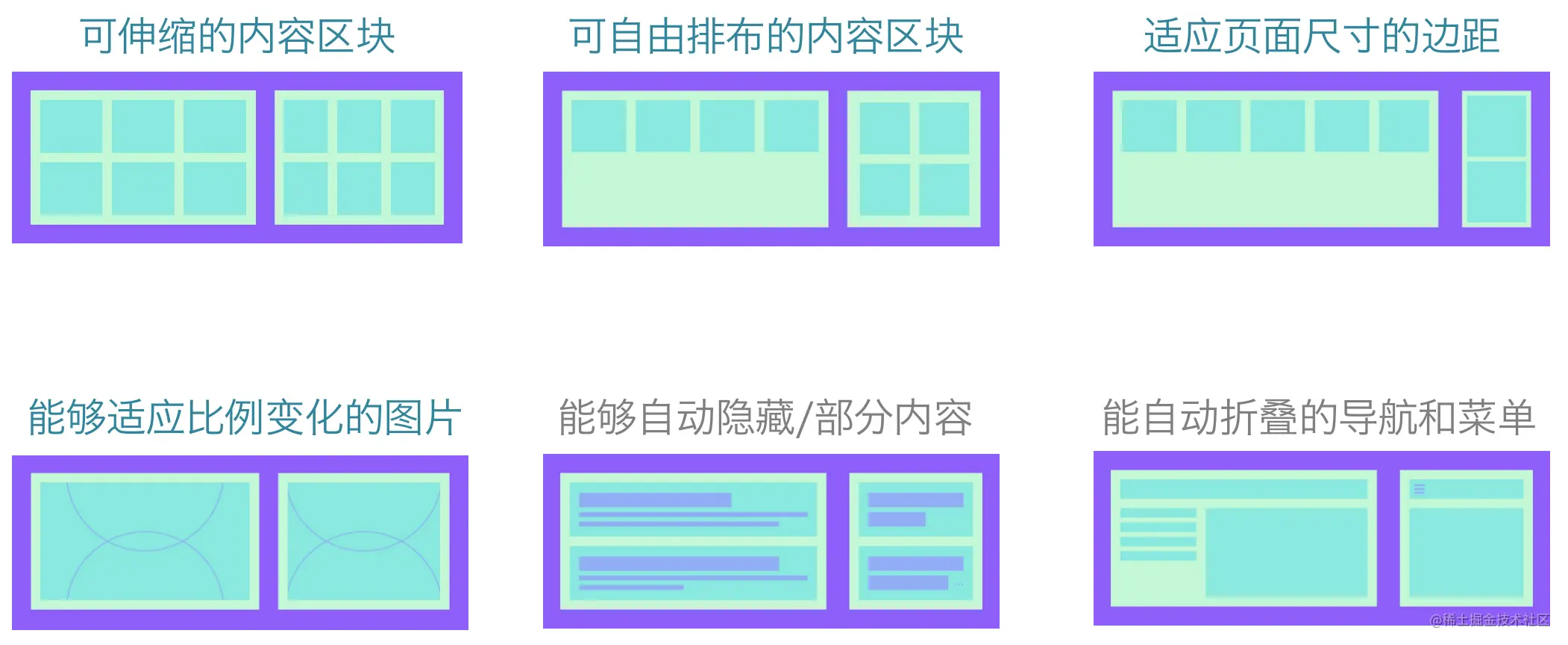
flex-grow: 1;

}



利用了 flex 布局下的 flex-grow: 1，让 .main 进行伸缩，占满剩余空间，利用 min-width 保证了整个容器的最小宽度。

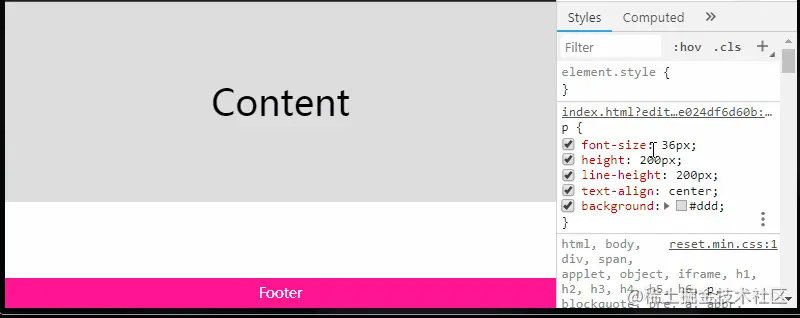
当然，这是最基本的自适应布局。对于现代布局，我们应该尽可能的考虑更多的场景。做到：



### 底部 footer

下面一种情形也是非常常见的一个情景。页面存在一个 footer 页脚部分，如果整个页面的内容高度小于视窗的高度，则 footer 固定在视窗底部，如果整个页面的内容高度大于视窗的高度，则 footer 正常流排布（也就是需要滚动到底部才能看到 footer）。

看看效果：



嗯，这个需求如果能够使用 flex 的话，使用 justify-content: space-between 可以很好的解决，同理使用 margin-top: auto 也非常容易完成：

<div class="g-container">

<div class="g-real-box">

...

</div>

<div class="g-footer"></div>

</div>

.g-container {

height: 100vh;

display: flex;

flex-direction: column;

}

.g-footer {

margin-top: auto;

flex-shrink: 0;

height: 30px;

background: deeppink;

}

[Codepen Demo -- sticky footer by flex margin auto](https://link.juejin.cn?target=https://codepen.io/Chokcoco/pen/pmrbWX" \o "https://codepen.io/Chokcoco/pen/pmrbWX" \t "_blank)

当然，实现它的方法有很多，这里仅给出一种推荐的解法。

### 处理动态内容 - 文本超长

对于所有接收后端接口字段的文本展示类的界面。都需要考虑全面（防御性编程：所有的外部数据都是不可信的），正常情况如下，是没有问题的。



但是我们是否考虑到了文本会超长？超长了会折行还是换行？



对于**单行文本**，使用单行省略：

{

width: 200px;

white-space: nowrap;

overflow: hidden;

text-overflow: ellipsis;

}



当然，目前对于**多行文本**的超长省略，兼容性也已经非常好了：

{

width: 200px;

overflow : hidden;

text-overflow: ellipsis;

display: -webkit-box;

-webkit-line-clamp: 2;

-webkit-box-orient: vertical;

}



### 处理动态内容 - 保护边界

对于一些动态内容，我们经常使用 min/max-width 或 min/max-height 对容器的高宽限度进行合理的控制。

在使用它们的时候，也有一些细节需要考虑到。

譬如经常会使用 min-width 控制按钮的最小宽度：

.btn {

...

min-width: 120px;

}

IMG_264

当内容比较少的时候是没问题的，但是当内容比较长，就容易出现问题。使用了 min-width 却没考虑到按钮的过长的情况：

IMG_265

这里就需要配合 padding 一起：

.btn {

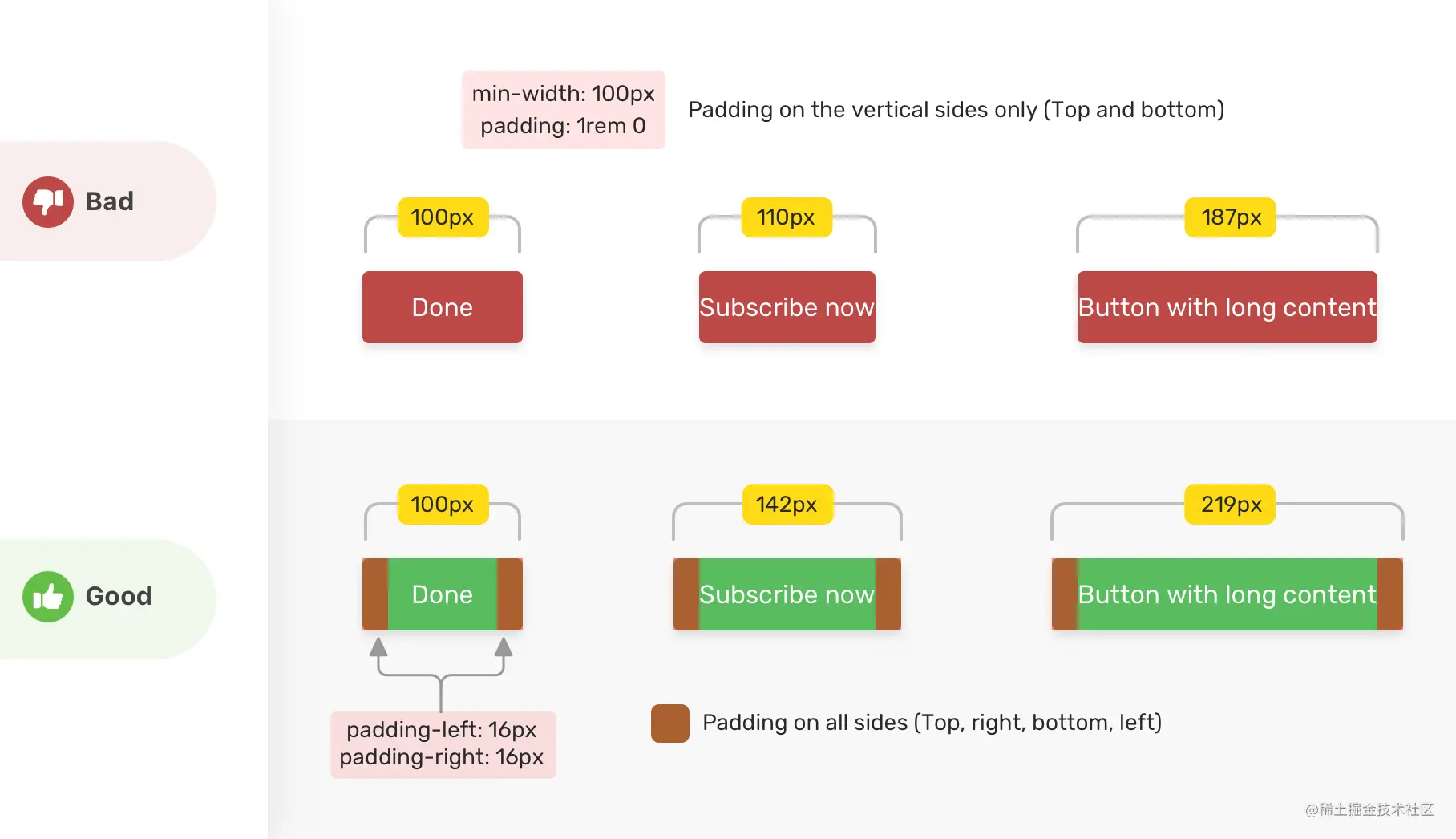
...

min-width: 88px;

padding: 0 16px

}

借用[Min and Max Width/Height in CSS](https://link.juejin.cn?target=https://ishadeed.com/article/min-max-css/" \o "https://ishadeed.com/article/min-max-css/" \t "_blank)中一张非常好的图，作为释义：



### 0 内容展示

这个也是一个常常被忽略的地方。

页面经常会有列表搜索，列表展示。那么，既然存在有数据的正常情况，当然也会存在搜索不到结果或者列表无内容可展示的情形。

对于这种情况，一定要注意 0 结果页面的设计，同时也要知道，这也是引导用户的好地方。对于 0 结果页面，分清楚：

* **数据为空**：其中又可能包括了用户无权限、搜索无结果、筛选无结果、页面无数据
* **异常状态**：其中又可能包括了网络异常、服务器异常、加载失败等待

不同的情况可能对应不同的 0 结果页面，附带不同的操作引导。

譬如网络异常：



或者确实是 0 结果：



关于 0 结果页面设计，可以详细看看这篇文章：[如何设计产品的空白页面？](https://link.juejin.cn?target=http://www.woshipm.com/pd/3742114.html" \o "http://www.woshipm.com/pd/3742114.html" \t "_blank)

小小总结一下，上述比较长的篇幅一直都在阐述一个道理，**开发时，不能仅仅关注正常现象，要多考虑各种异常情况，思考全面。做好各种可能情况的处理**。

### 图片相关

图片在我们的业务中应该是非常的常见了。有一些小细节是需要注意的。

#### 给图片同时设置高宽

有的时候和产品、设计会商定，只能使用固定尺寸大小的图片，我们的布局可能是这样：



对应的布局：

<ul class="g-container">

<li>

<img src="http://placehold.it/150x100">

<p>图片描述</p>

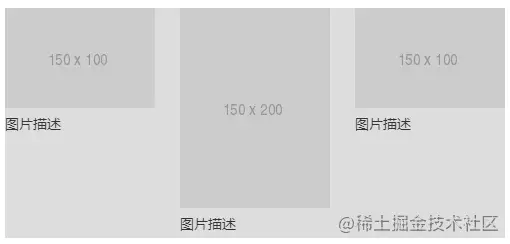
</li></ul>

ul li img {

width: 150px;

}

当然，万一假设后端接口出现一张非正常大小的图片，上述不加保护的布局就会出问题：



所以对于图片，我们总是建议同时写上高和宽，避免因为图片尺寸错误带来的布局问题：

ul li img {

width: 150px;

height: 100px;

}

同时，给 <img> 标签同时写上高宽，可以在图片未加载之前提前占住位置，避免图片从未加载状态到渲染完成状态高宽变化引起的重排问题。

#### object-fit

当然，限制高宽也会出现问题，譬如图片被拉伸了，非常的难看：



这个时候，我们可以借助 object-fit，它能够指定可替换元素的内容（也就是图片）该如何适应它的父容器的高宽。

ul li img {

width: 150px;

height: 100px;

object-fit: cover;

}

利用 object-fit: cover，使图片内容在保持其宽高比的同时填充元素的整个内容框。



object-fit 还有一个配套属性 object-position，它可以控制图片在其内容框中的位置。（类似于 background-position），m默认是 object-position: 50% 50%，如果你不希望图片居中展示，可以使用它去改变图片实际展示的 position 。

ul li img {

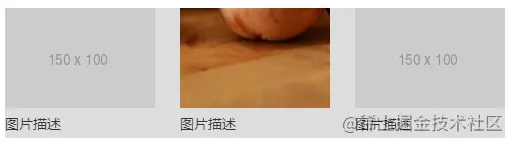
width: 150px;

height: 100px;

object-fit: cover;

object-position: 50% 100%;

}



像是这样，object-position: 100% 50% 指明从底部开始展示图片。这里有一个很好的 Demo 可以帮助你理解 object-position。

[CodePen Demo -- Object position](https://link.juejin.cn?target=https://codepen.io/robinrendle/pen/raGOOJ" \o "https://codepen.io/robinrendle/pen/raGOOJ" \t "_blank)

#### 考虑屏幕 dpr -- 响应式图片

正常情况下，图片的展示应该没有什么问题了。但是对于有图片可展示的情况下，我们还可以做的更好。

在移动端或者一些高清的 PC 屏幕（苹果的 MAC Book），屏幕的 dpr 可能大于 1。这种时候，我们可能还需要考虑利用多倍图去适配不同 dpr 的屏幕。

正好，<img> 标签是有提供相应的属性 srcset 让我们进行操作的。

<img src='photo@1x.png'

srcset='photo@1x.png 1x,

photo@2x.png 2x,

photo@3x.png 3x'

/>

当然，这是比较旧的写法，srcset 新增了新的 w 宽度描述符，需要配合 sizes 一起使用，所以更好的写法是：

<img

src = "photo.png"

sizes = “(min-width: 600px) 600px, 300px"

srcset = “photo@1x.png 300w,

photo@2x.png 600w,

photo@3x.png 1200w,

>

利用 srcset，我们可以给不同 dpr 的屏幕，提供最适合的图片。

上述出现了一些概念，dpr，图片的 srcset ，sizes 属性，不太了解的可以移步 [前端基础知识概述](https://link.juejin.cn?target=https://github.com/chokcoco/cnblogsArticle/issues/25" \o "https://github.com/chokcoco/cnblogsArticle/issues/25" \t "_blank)

#### 图片丢失

好了，当图片链接没问题时，已经处理好了。接下来还需要考虑，当图片链接挂了，应该如何处理。

处理的方式有很多种。最好的处理方式，是我最近在张鑫旭老师的这篇文章中 -- [图片加载失败后CSS样式处理最佳实践](https://link.juejin.cn?target=https://www.zhangxinxu.com/wordpress/2020/10/css-style-image-load-fail/" \o "https://www.zhangxinxu.com/wordpress/2020/10/css-style-image-load-fail/" \t "_blank) 看到的。这里简单讲下：

1. 利用图片加载失败，触发 <img> 元素的 onerror 事件，给加载失败的 <img> 元素新增一个样式类
2. 利用新增的样式类，配合 <img> 元素的伪元素，展示默认兜底图的同时，还能一起展示 <img> 元素的 alt 信息

<img src="test.png" alt="图片描述" onerror="this.classList.add('error');">

img.error {

position: relative;

display: inline-block;

}

img.error::before {

content: "";

/\*\* 定位代码 \*\*/

background: url(error-default.png);

}

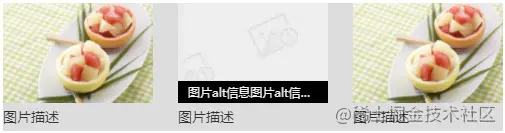
img.error::after {

content: attr(alt);

/\*\* 定位代码 \*\*/

}

我们利用伪元素 before ，加载默认错误兜底图，利用伪元素 after，展示图片的 alt 信息：



OK，到此，完整的对图片的处理就算完成了，完整的 Demo 你可以戳这里看看：

[CodePen Demo -- 图片处理](https://link.juejin.cn?target=https://codepen.io/Chokcoco/pen/WNGgNqv?editors=1100" \o "https://codepen.io/Chokcoco/pen/WNGgNqv?editors=1100" \t "_blank)

## 交互设计优化

接下来一个大环节是关于一些交互的细节。对于交互设计，一些比较通用的准则：

* Don’t make me think
* 符合用户的习惯与预期
* 操作便利
* 做适当的提醒
* 不强迫用户

### 过渡与动画

在我们的交互过程中，**适当的增加过渡与动画，能够很好的让用户感知到页面的变化**。

譬如我们页面上随处可见 loading 效果，其实就是这样一种作用，让用户感知页面正在加载，或者正在处理某些事务。



### 滚动优化

滚动也是操作网页中非常重要的一环。看看有哪些可以优化的点：

#### 滚动平滑：使用 scroll-behavior: smooth 让滚动丝滑

使用 scroll-behavior: smooth，可以让滚动框实现平稳的滚动，而不是突兀的跳动。看看效果，假设如下结构：

<div class="g-container">

<nav>

<a href="#1">1</a>

<a href="#2">2</a>

<a href="#3">3</a>

</nav>

<div class="scrolling-box">

<section id="1">First section</section>

<section id="2">Second section</section>

<section id="3">Third section</section>

</div></div>

不使用 scroll-behavior: smooth，是突兀的跳动切换：



给可滚动容器添加 scroll-behavior: smooth，实现平滑滚动：

{

scroll-behavior: smooth;

}



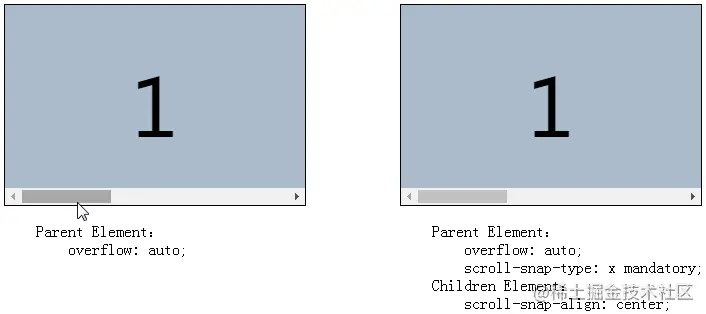
#### 使用 scroll-snap-type 优化滚动效果

sroll-snap-type 可能算得上是新的滚动规范里面最核心的一个属性样式。

[scroll-snap-type](https://link.juejin.cn?target=https://developer.mozilla.org/zh-CN/docs/Web/CSS/scroll-snap-type" \o "https://developer.mozilla.org/zh-CN/docs/Web/CSS/scroll-snap-type" \t "_blank)：属性定义在滚动容器中的一个临时点（snap point）如何被严格的执行。

光看定义有点难理解，简单而言，这个属性规定了一个容器是否对内部滚动动作进行捕捉，并且规定了如何去处理滚动结束状态。让滚动操作结束后，元素停止在适合的位置。

看个简单示例：



当然，scroll-snap-type 用法非常多，可控制优化的点很多，限于篇幅无法一一展开，具体更详细的用法可以看看我的另外一篇文章 -- [使用 sroll-snap-type 优化滚动](https://link.juejin.cn?target=https://github.com/chokcoco/iCSS/issues/74" \o "https://github.com/chokcoco/iCSS/issues/74" \t "_blank)

#### 控制滚动层级，避免页面大量重排

这个优化可能稍微有一点难理解。需要了解 CSS 渲染优化的相关知识。

先说结论，控制滚动层级的意思是**尽量让需要进行 CSS 动画（可以是元素的动画，也可以是容器的滚动）的元素的 z-index 保持在页面最上方，避免浏览器创建不必要的图形层（GraphicsLayer），能够很好的提升渲染性能**。

这一点怎么理解呢，一个元素触发创建一个 Graphics Layer 层的其中一个因素是：

* 元素有一个 z-index 较低且包含一个复合层的兄弟元素

根据上述这点，我们对滚动性能进行优化的时候，需要注意两点：

1. 通过生成独立的 GraphicsLayer，利用 GPU 加速，提升滚动的性能
2. 如果本身滚动没有性能问题，不需要独立的 GraphicsLayer，也要注意滚动容器的层级，避免因为层级过高而被其他创建了 GraphicsLayer 的元素合并，被动的生成一个 Graphics Layer ，影响页面整体的渲染性能

如果你对这点还有点懵，可以看看这篇文章 -- [你所不知道的 CSS 动画技巧与细节](https://link.juejin.cn?target=https://github.com/chokcoco/iCSS/issues/27" \o "https://github.com/chokcoco/iCSS/issues/27" \t "_blank)

### 点击交互优化

在用户点击交互方面，也有一些有意思的小细节。

#### 优化手势 -- 不同场景应用不同 cursor

对于不同的内容，最好给与不同的 cursor 样式，CSS 原生提供非常多种常用的手势。

在不同的场景使用不同的鼠标手势，**符合用户的习惯与预期**，可以很好的提升用户的交互体验。

首先对于按钮，就至少会有 3 种不同的 cursor，分别是可点击，不可点击，等待中：

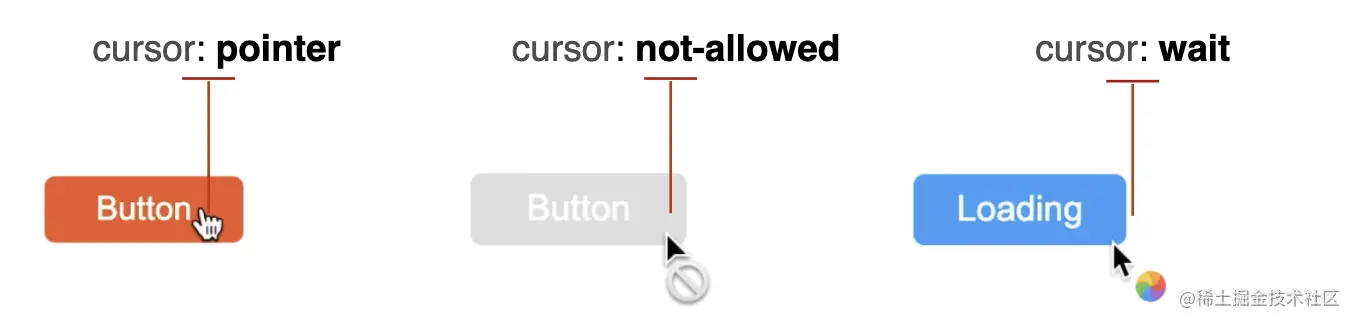
{

cursor: pointer; // 可点击

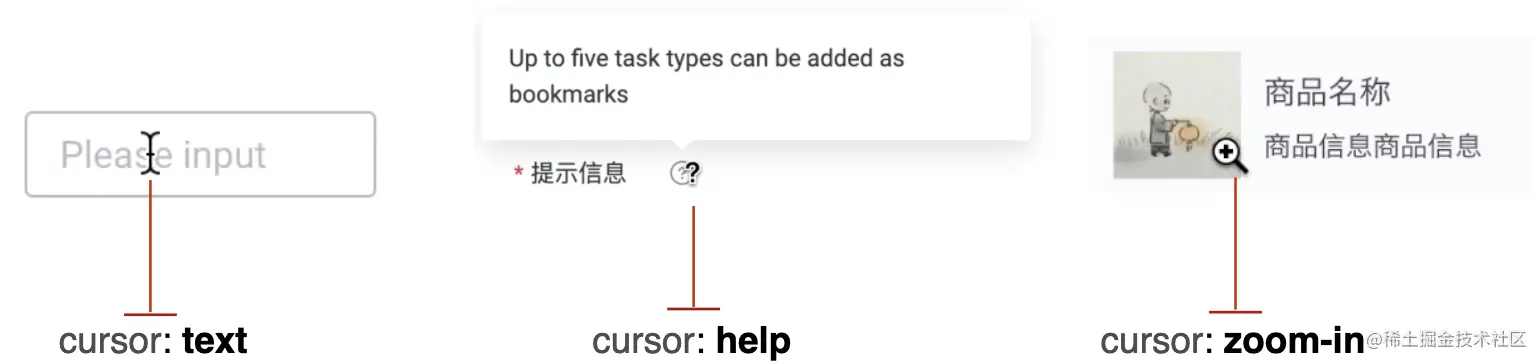
cursor: not-allowed; // 不可点击

cursor: wait; // loading

}



除此之外，还有一些常见的，对于一些可输入的 Input 框，使用 cursor: text，对于提示 Tips 类使用 cursor: help，放大缩小图片 zoom-in、zoom-out 等等：



一些常用的简单列一列：

* 按钮可点击: cursor: pointer
* 按钮禁止点击：cursor: not-allowed
* 等待 Loading 状态：cursor: wait
* 输入框：cursor: text;
* 图片查看器可放大可缩小：cursor: zoom-in/ zoom-out
* 提示：cursor: help;

当然，实际 cursor 还支持非常多种，可以在 [MDN](https://link.juejin.cn?target=https://developer.mozilla.org/en-US/docs/Web/CSS/cursor" \o "https://developer.mozilla.org/en-US/docs/Web/CSS/cursor" \t "_blank) 或者下面这个 CodePen Demo 中查看这里看完整的列表：

[CodePen Demo -- Cursor Demo](https://link.juejin.cn?target=https://codepen.io/Chokcoco/pen/poEBLqr" \o "https://codepen.io/Chokcoco/pen/poEBLqr" \t "_blank)

### 点击区域优化 -- 伪元素扩大点击区域

按钮是我们网页设计中十分重要的一环，而按钮的设计也与用户体验息息相关。

考虑这样一个场景，在摇晃的车厢上或者是单手操作着屏幕，有的时候一个按钮，死活也点不到。

让用户更容易的点击到按钮无疑能很好的增加用户体验及可提升页面的访问性，尤其是在移动端，按钮通常都很小，但是受限于设计稿或者整体 UI 风格，我们不能直接去改变按钮元素的高宽。

那么这个时候有什么办法在不改变按钮原本大小的情况下去增加他的点击热区呢？

这里，伪元素也是可以代表其宿主元素来响应的鼠标交互事件的。借助伪元素可以轻松帮我们实现，我们可以这样写：

.btn::befoer{

content:"";

position:absolute;

top:-10px;

right:-10px;

bottom:-10px;

left:-10px;

}

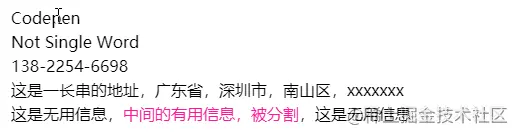
当然，在 PC 端下这样子看起来有点奇怪，但是合理的用在点击区域较小的移动端则能取到十分好的效果，效果如下：



在按钮的伪元素没有其它用途的时候，这个方法确实是个很好的提升用户体验的点。

### 快速选择优化 -- user-select: all

操作系统或者浏览器通常会提供一些快速选取文本的功能，看看下面的示意图：



快速单击两次，可以选中单个单词，快速单击三次，可以选中一整行内容。但是如果有的时候我们的核心内容，被分隔符分割，或者潜藏在一整行中的一部分，这个时候选取起来就比较麻烦。

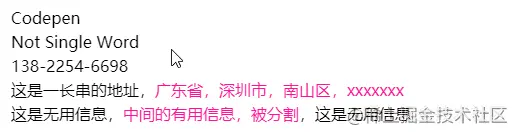
利用 user-select: all，可以将需要一次选中的内容进行包裹，用户只需要点击一次，就可以选中该段信息：

.g-select-all {

user-select: all

}

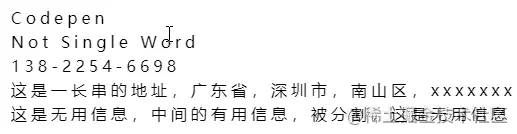
给需要一次选中的信息，加上这个样式后的效果，这个细节作用在一些需要复制粘贴的场景，非常好用：



[CodePen -- user-select: all 示例](https://link.juejin.cn?target=https://codepen.io/Chokcoco/pen/LYRBmEJ" \o "https://codepen.io/Chokcoco/pen/LYRBmEJ" \t "_blank)

### 选中样式优化 -- ::selection

当然，如果你想更进一步，CSS 还有提供一个 ::selection 伪类，可以控制选中的文本的样式（只能控制color, background, text-shadow），进一步加深效果。



[CodePen -- user-select: all && ::selection 控制选中样式](https://link.juejin.cn?target=https://codepen.io/Chokcoco/pen/RwGYWNj" \o "https://codepen.io/Chokcoco/pen/RwGYWNj" \t "_blank)

### 添加禁止选择 -- user-select: none

有快速选择，也就会有它的对立面 -- 禁止选择。

对于一些可能频繁操作的按钮，可能出现如下尴尬的场景：

* 文本按钮的快速点击，触发了浏览器的双击快速选择，导致文本被选中：



* 翻页按钮的快速点击，触发了浏览器的双击快速选择：

对于这种场景，我们需要把不可被选中元素设置为不可被选中，利用 CSS 可以快速的实现这一点：

{

-webkit-user-select: none; /\* Safari \*/

-ms-user-select: none; /\* IE 10 and IE 11 \*/

user-select: none; /\* Standard syntax \*/

}

这样，无论点击的频率多快，都不会出现尴尬的内容选中：



### 跳转优化

现阶段，单页应用（Single Page Application）的应用非常广泛，Vue 、React 等框架大行其道。但是一些常见的写法，也容易衍生一些小问题。

譬如，点击按钮、文本进行路由跳转。譬如，经常会出现这种代码：

<template>

...

<button @click="gotoDetail">

Detail

</button>

...

<template>

...

gotoDetail() {

this.$router.push({

name: 'xxxxx',

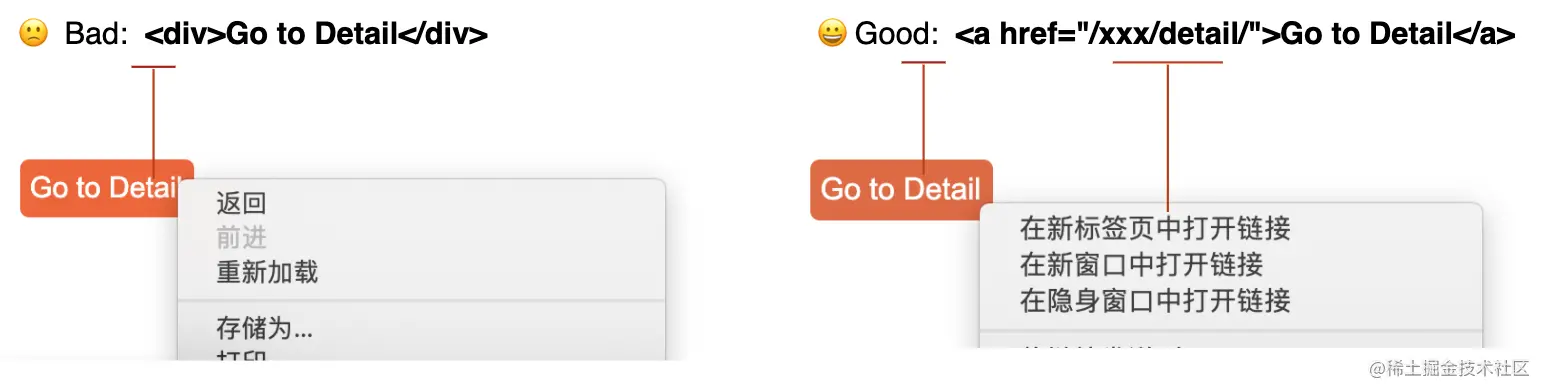
});

}

大致逻辑就是给按钮添加一个事件，点击之后，跳转到另外一个路由。当然，本身这个功能是没有任何问题的，但是没有考虑到用户实际使用的场景。

实际使用的时候，由于是一个页面跳转，很多时候，用户希望能够保留当前页面的内容，同时打开一个新的窗口，这个时候，他会尝试下的鼠标右键，选择**在新标签页中打开页面**，遗憾的是，上述的写法是不支持鼠标右键打开新页面的。

原因在于浏览器是通过读取 <a> 标签的 href 属性，来展示类似**在新标签页中打开页面**这种选项，对于上述的写法，浏览器是无法识别它是一个可以跳转的链接。简单的示意图如下：



所以，对于所有路由跳转按钮，建议都使用 <a> 标签，并且内置 href 属性，填写跳转的路由地址。实际渲染出来的 DOM 可能是需要类似这样：

<a href="/xx/detail">Detail</a>

### 易用性

易用性也是交互设计中需要考虑的一个非常重要的环节，能做的有非常多。简单的罗列一下：

* 注意界面元素的一致性，降低用户学习成本
* 延续用户日常的使用习惯，而不是重新创造
* 给下拉框增加一些预设值，降低用户填写成本
* 同类的操作合并在一起，降低用户的认知成本
* 任何操作之后都要给出反馈，让用户知道操作已经生效

### 先探索，后表态

这一点非常的有意思，什么叫先探索后表态呢？就是我们不要一上来就强迫用户去做一些事情，譬如登录。

想一想一些常用网站的例子：

* 类似虎牙、Bilibili 等视频网站，可以先观看体验，一定观看时间后才会要求登录（登录享受蓝光）
* 电商网站，只有到付款的时候，才需要登录

上述**易用性**和**先探索，后表态**的内容，部分来源于：[Learn From What Leading Companies A/B Test](https://link.juejin.cn?target=https://goodui.org/" \o "https://goodui.org/" \t "_blank)，可以好好读一读。

## 字体优化

字体的选择与使用其实是非常有讲究的。

如果网站没有强制必须使用某些字体。最新的规范建议我们更多的去使用系统默认字体。也就是 [CSS Fonts Module Level 4 -- Generic font families](https://link.juejin.cn?target=https://www.w3.org/TR/css-fonts-4/%23generic-font-families" \o "https://www.w3.org/TR/css-fonts-4/#generic-font-families" \t "_blank) 中新增的 font-family: system-ui 关键字。

font-family: system-ui 能够自动选择本操作系统下的默认系统字体。

默认使用特定操作系统的系统字体可以提高性能，因为浏览器或者 webview 不必去下载任何字体文件，而是使用已有的字体文件。 font-family: system-ui 字体设置的优势之处在于它与当前操作系统使用的字体相匹配，对于文本内容而言，它可以得到最恰当的展示。

举两个例子，天猫的字体定义与 Github 的字体定义：

* [天猫](https://link.juejin.cn?target=https://www.tmall.com/" \o "https://www.tmall.com/" \t "_blank)：font-family: "PingFang SC",miui,system-ui,-apple-system,BlinkMacSystemFont,Helvetica Neue,Helvetica,sans-serif;
* [Github](https://link.juejin.cn?target=https://github.com" \o "https://github.com" \t "_blank)：font-family: -apple-system,BlinkMacSystemFont,Segoe UI,Helvetica,Arial,sans-serif,Apple Color Emoji,Segoe UI Emoji,Segoe UI Symbol;

简单而言，它们总体遵循了这样一个基本原则：

#### 1、尽量使用系统默认字体

使用系统默认字体的主要原因是性能，并且系统字体的优点在于它与当前操作系统使用的相匹配，因此它的文本展示必然也是一个让人舒适展示效果。

#### 2、兼顾中西，西文在前，中文在后

中文或者西文（英文）都要考虑到。由于大部分中文字体也是带有英文部分的，但是英文部分又不怎么好看，但是英文字体中大多不包含中文。通常会先进行英文字体的声明，选择最优的英文字体，这样不会影响到中文字体的选择，中文字体声明则紧随其次。

#### 3、兼顾多操作系统

选择字体的时候要考虑多操作系统。例如 MAC OS 下的很多中文字体在 Windows 都没有预装，为了保证 MAC 用户的体验，在定义中文字体的时候，先定义 MAC 用户的中文字体，再定义 Windows 用户的中文字体；

#### 4、兼顾旧操作系统，以字体族系列 serif 和 sans-serif 结尾

当使用一些非常新的字体时，要考虑向下兼容，兼顾到一些极旧的操作系统，使用字体族系列 serif 和 sans-serif 结尾总归是不错的选择。

对于上述的一些字体可能会有些懵，譬如 -apple-system, BlinkMacSystemFont，这是因为不同浏览器厂商对规范的实现有所不同，对于字体定义更多的相关细节，可以再看看这篇文章 -- [Web 字体 font-family 再探秘](https://link.juejin.cn?target=https://github.com/chokcoco/iCSS/issues/69" \o "https://github.com/chokcoco/iCSS/issues/69" \t "_blank)

## 可访问性（A11Y）

可访问性，在我们的网站中，属于非常重要的一环，但是大部分前端（其实应该是设计、前端、产品）同学都会忽视它。

我潜伏在一个叫**无障碍设计小组**的群里，其中包含了很多无障碍设计师以及患有一定程度视觉、听力、行动障碍的用户，他们在群里经常会表达出一个观点，就是国内的大部分 Web 网站及 APP 基本没有考虑过残障人士的使用（或者可访问性做的非常差），非常的令人揪心。

尤其在我们一些重交互、重逻辑的网站中，我们需要考虑用户的使用习惯、使用场景，从高可访问性的角度考虑，譬如假设用户没有鼠标，仅仅使用键盘，能否顺畅的使用我们的网站？

假设用户没有鼠标，这个真不一定是针对残障人士，很多情况下，用户拿鼠标的手可能在干其他事情，比如在吃东西，又或者在 TO B 类的业务，如超市收银、仓库收货，很可能用户拿鼠标的手操作着其他设备（扫码枪）等等。

本文不会专门阐述无障碍设计的方方面面，只是从一些我觉得前端工程师需要关注的，并且仅需要花费少量代价就能做好的一些无障碍设计细节。记住，**无障碍设计对所有人都更友善**。

### 色彩对比度

颜色，也是我们天天需要打交道的属性。对于大部分视觉正常的用户，可能对页面的颜色敏感度还没那么高。但是对于一小部分色弱、色盲用户，他们对于网站的颜色会更加敏感，不好的设计会给他们访问网站带来极大的不便。

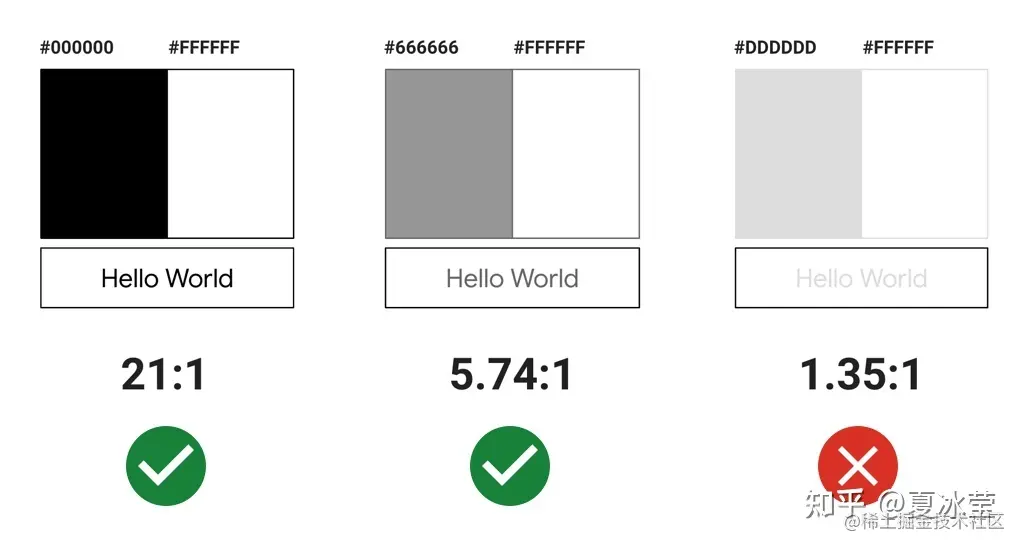
#### 什么是色彩对比度

是否曾关心过页面内容的展示，使用的颜色是否恰当？色弱、色盲用户能否正常看清内容？良好的色彩使用，在任何时候都是有益的，而且不仅仅局限于对于色弱、色盲用户。在户外用手机、阳光很强看不清，符合无障碍标准的高清晰度、高对比度文字就更容易阅读。

这里就有一个概念 -- **颜色对比度**，简单地说，描述就是两种颜色在亮度（Brightness）上的差别。运用到我们的页面上，大多数的情况就是背景色（background-color）与内容颜色（color）的对比差异。

最权威的互联网无障碍规范 —— [WCAG AA](https://link.juejin.cn?target=https://www.w3.org/Translations/WCAG21-zh/" \o "https://www.w3.org/Translations/WCAG21-zh/" \t "_blank)规范规定，所有重要内容的色彩对比度需要达到 4.5:1 或以上（字号大于18号时达到 3:1 或以上），才算拥有较好的可读性。

借用一张图 -- [知乎 -- 助你轻松做好无障碍的15个UI设计工具推荐](https://link.juejin.cn?target=https://zhuanlan.zhihu.com/p/349761993" \o "https://zhuanlan.zhihu.com/p/349761993" \t "_blank)：



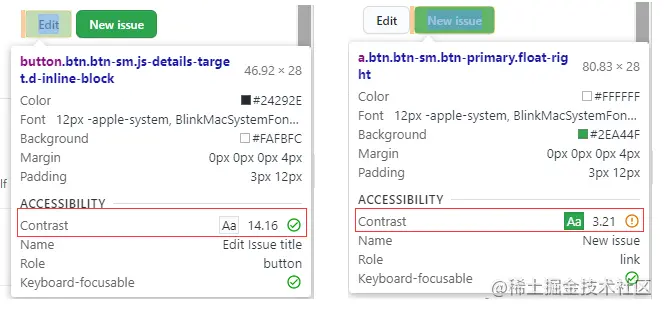
很明显，上述最后一个例子，文字已经非常的不清晰了，正常用户都已经很难看得清了。

#### 检查色彩对比度的工具

Chrome 浏览器从很早开始，就已经支持检查元素的色彩对比度了。以我当前正在写作的页面为例子，Github Issues 编辑页面的两个按钮：

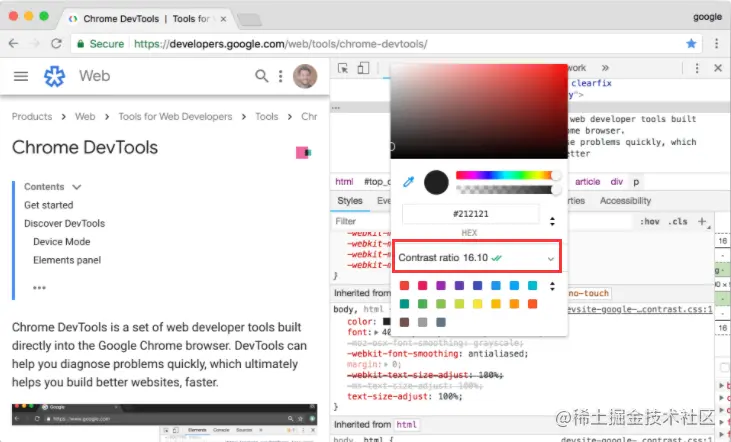
IMG_289

审查元素，分别可以看到两个按钮的色彩对比度：



可以看到，绿底白字按钮的色彩对比度是没有达到标准的，也被用黄色的叹号标识了出来。

除此之外，在审查元素的 Style 界面的取色器，改变颜色，也能直观的看到当前的色彩对比度：



### 焦点响应

类似百度、谷歌的首页，进入页面后会默认让输入框获得焦点：



并非所有的有输入框的页面，都需要进入页面后进行聚焦，但是焦点能够让用户非常明确的知道，当前自己在哪，需要做些什么。尤其是对于无法操作鼠标的用户。

页面上可以聚焦的元素，称为**可聚焦元素**，获得焦点的元素，则会触发该元素的 focus 事件，对应的，也就会触发该元素的 :focus 伪类。

浏览器通常会使用元素的 :focus 伪类，给元素添加一层边框，告诉用户，当前的获焦元素在哪里。

我们可以通过键盘的 Tab 键，进行焦点的切换，而获焦元素则可以通过元素的 :focus 伪类的样式，告诉用户当前焦点位置。

当然，除了 Tab 键之外，对于一些多输入框、选择框的表单页面，我们也应该想着如何简化用户的操作，譬如用户按回车键时自动前进到下一字段。一般而言，用户必须执行的触按越少，体验越佳。:thumbsup:

**下面的截图，完全由键盘操作完成**：



通过元素的 :focus 伪类以及键盘 Tab 键切换焦点，用户可以非常顺畅的在脱离鼠标的情况下，对页面的焦点切换及操作。

然而，在许多 reset.css 中，经常能看到这样一句 CSS 样式代码，为了样式的统一，消除了可聚焦元素的 :focus 伪类：

:focus {

outline: 0;

}

我们给上述操作的代码。也加上这样一句代码，**全程再用键盘操作一下**：



除了在 input 框有光标提示，当使用 Tab 进行焦点切换到 select 或者到 button 时，由于没有了 :focus 样式，用户将完全懵逼，不知道页面的焦点现在处于何处。

#### 保证非鼠标用户体验，合理运用 :focus-visible

当然，造成上述结果很重要的一个原因在于。:focus 伪类不论用户在使用鼠标还是使用键盘，只要元素获焦，就会触发。

而其本身的默认样式又不太能被产品或者设计接受，导致了很多人会在焦点元素触发 :focus 伪类时，通过改变 border 的颜色或者其他一些方式替代或者直接禁用。而这样做，从可访问性的角度来看，对于非鼠标用户，无疑是灾难性的。

基于此，在[W3 CSS selectors-4 规范](https://link.juejin.cn?target=https://drafts.csswg.org/selectors-4/%23the-focus-visible-pseudo" \o "https://drafts.csswg.org/selectors-4/#the-focus-visible-pseudo" \t "_blank) 中，新增了一个非常有意思的 :focus-visible 伪类。

:focus-visible：这个选择器可以有效地根据用户的输入方式(鼠标 vs 键盘)展示不同形式的焦点。

有了这个伪类，就可以做到，当用户使用鼠标操作可聚焦元素时，不展示 :focus 样式或者让其表现较弱，而当用户使用键盘操作焦点时，利用 :focus-visible，让可获焦元素获得一个较强的表现样式。

看个简单的 Demo：

<button>Test 1</button>

button:active {

background: #eee;

}button:focus {

outline: 2px solid red;

}

使用鼠标点击：

IMG_295

可以看到，使用鼠标点击的时候，触发了元素的 :active 伪类，也触发了 :focus伪类，不太美观。但是如果设置了 outline: none 又会使键盘用户的体验非常糟糕。尝试使用 :focus-visible 伪类改造一下：

button:active {

background: #eee;

}button:focus {

outline: 2px solid red;

}button:focus:not(:focus-visible) {

outline: none;

}

看看效果，分别是在鼠标点击 Button 和使用键盘控制焦点点击 Button：

IMG_296

[CodePen Demo -- :focus-visible example](https://link.juejin.cn?target=https://codepen.io/Chokcoco/pen/abBbPrE" \o "https://codepen.io/Chokcoco/pen/abBbPrE" \t "_blank)

可以看到，使用鼠标点击，不会触发 :foucs，只有当键盘操作聚焦元素，使用 Tab 切换焦点时，outline: 2px solid red 这段代码才会生效。

这样，我们就既保证了正常用户的点击体验，也保证了一批无法使用鼠标的用户的焦点管理体验。

值得注意的是，有同学会疑惑，这里为什么使用了 :not 这么绕的写法而不是直接这样写呢：

button:focus {

outline: unset;

}button:focus-visible {

outline: 2px solid red;

}

为的是兼容不支持 :focus-visible 的浏览器，当 :focus-visible 不兼容时，还是需要有 :focus 伪类的存在。

### 使用 WAI-ARIA 规范增强语义 -- div 等非可获焦元素模拟获焦元素

还有一个非常需要注意的点。

现在很多前端同学在前端开发的过程中，喜欢使用非可获焦元素模拟获焦元素，譬如：

* 使用 div 模拟 button 元素
* 使用 ul 模拟下拉列表 select 等等

当下很多组件库都是这样做的，譬如 element-ui 和 ant-design。

在使用非可获焦元素模拟获焦元素的时候，一定要注意，不仅仅只是外观长得像就完事了，其行为表现也需要符合原本的 button、select 等可聚焦元素的性质，能够体现元素的语义，能够被聚焦，能够通过 Tab 切换等等。

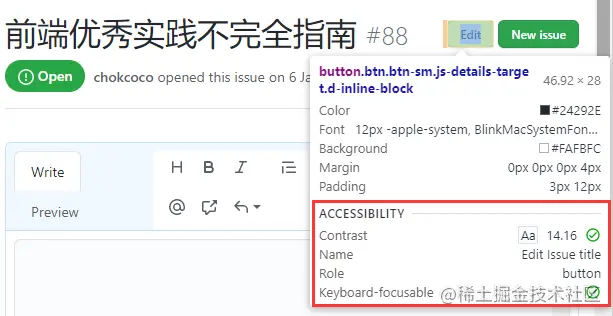
基于大量类似的场景，有了 [WAI-ARIA 标准](https://link.juejin.cn?target=https://www.w3.org/TR/wai-aria-1.1/" \o "https://www.w3.org/TR/wai-aria-1.1/" \t "_blank)，WAI-ARIA是一个为残疾人士等提供无障碍访问动态、可交互Web内容的技术规范。

简单来说，它提供了一些属性，增强标签的语义及行为：

* 可以使用 tabindex 属性控制元素是否可以聚焦，以及它是否/在何处参与顺序键盘导航
* 可以使用 role 属性，来标识元素的语义及作用，譬如使用 <div id="saveChanges" tabindex="0" role="button">Save</div> 来模拟一个按钮
* 还有大量的 aria-\* 属性，表示元素的属性或状态，帮助我们进一步地识别以及实现元素的语义化，优化无障碍体验

#### 使用工具查看标签的语义

我们来看看 Github 页面是如何定义一个按钮的，以 Github Issues 页面的 Edit 按钮为例子：



这一块，清晰的描述了这个按钮在可访问性相关的一些特性，譬如 Contrast 色彩对比度，按钮的描述，也就是 Name，是给屏幕阅读器看到的，Role 标识是这个元素的属性，它是一个按钮，Keyboard focusable 则表明他能否被键盘的 Tab 按钮给捕获。

### 分析使用非可聚焦元素模拟的按钮

这里，我随便选取了我们业务中一个使用 span 模拟按钮的场景，是一个面包屑导航，点击可进行跳转，发现惨不忍睹：

IMG_298

HTML 代码：

<span class="ssc-breadcrumb-item-link"> Inbound </span>

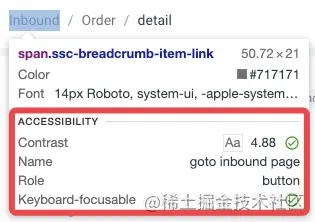


基本上可访问性为 0，作为一个按钮，它不可被聚焦，无法被键盘用户选中，没有具体的语义，色彩对比度太低，可能视障用户无法看清。并且，作为一个能进行页面跳转的按钮，它没有不是 a 标签，没有 href 属性。

即便对于面包屑导航，我们可以不将它改造成 <a> 标签，也需要做到**最基本**的一些可访问性改造：

<span role="button" aria-label="goto inbound page" tabindex="0" class="ssc-breadcrumb-item-link"> Inbound </span>

不要忘了再改一下颜色，达到最低色彩对比度以上，再看看：



OK，这样，一个最最最基本的，满足最低可访问性需求的按钮算是勉强达标，当然，这个按钮可以再更进一步进行改造，涉及了更深入的可访问性知识，本文不深入展开。

### 分析组件库的 A11Y

最后，在我们比较常用的 Vue - [element-ui](https://link.juejin.cn?target=https://element.eleme.io/%23/zh-CN/component/select" \o "https://element.eleme.io/#/zh-CN/component/select" \t "_blank)、React - [ant-design](https://link.juejin.cn?target=https://ant.design/components/select-cn/" \o "https://ant.design/components/select-cn/" \t "_blank) 中，我们来看看 ant-design 在提升可访问性相关的一些功能。

以 Select 选择框组件为例，ant-design 利用了大量的 WAI-ARIA 属性，使得用 div 模拟的下拉框不仅仅在表现上符合一个下拉框，在语义、行为上都符合一个下拉框，简单的一个例子：



看看使用 div 模拟下拉框的 DOM 部分：



再看看在交互体验上：



上述操作全是在键盘下完成，看着平平无奇，实际上组件库在正常响应可获焦元素切换的同时，给用 div 模拟的 select 加了很多键盘事件的响应，可以利用回车，上下键等对可选项进行选择。其实是下了很多功夫。

对于 A11Y 相关的内容，篇幅及内容非常之多，本文无法一一展开，感兴趣的可以通读下下列文章：

* [WAI-ARIA basics](https://link.juejin.cn?target=https://developer.mozilla.org/zh-CN/docs/learn/Accessibility/WAI-ARIA_basics" \o "https://developer.mozilla.org/zh-CN/docs/learn/Accessibility/WAI-ARIA_basics" \t "_blank)
* [WAI-ARIA 1.1](https://link.juejin.cn?target=https://www.w3.org/TR/wai-aria-1.1/" \o "https://www.w3.org/TR/wai-aria-1.1/" \t "_blank)
* [Web中的焦点管理](https://link.juejin.cn?target=https://www.w3cplus.com/a11y/focus-for-web.html" \o "https://www.w3cplus.com/a11y/focus-for-web.html" \t "_blank)
* [无障碍功能](https://link.juejin.cn?target=https://developers.google.com/web/fundamentals/accessibility?hl=zh-cn" \o "https://developers.google.com/web/fundamentals/accessibility?hl=zh-cn" \t "_blank)
* [提升Web用户体验的71个设计要点](https://link.juejin.cn?target=https://www.shejidaren.com/71-good-ui-tips.html" \o "https://www.shejidaren.com/71-good-ui-tips.html" \t "_blank)
* [公众号 -- 无障碍设计小组](https://link.juejin.cn?target=https://werss.app/accounts/NTI1NTUwNTg1ODU0NTM=" \o "https://werss.app/accounts/NTI1NTUwNTg1ODU0NTM=" \t "_blank)