h5 & css

1. 水平垂直居中方案

- flex布局

```
.father {
    display: flex;
    justify-content: center;
    align-items: center;
}
.son {
    ...
}

// 方法二
.box {
  display: flex;
}

.item {
  margin: auto;
}
```

- margin:auto + 绝对定位

```
.father {
    position: relative;
}
.son {
    position: absolute;
    top: 0;
    left: 0;
    bottom: 0;
    right: 0;
    margin: auto;
}
```

- 绝对定位配合transform实现

```
.father {
    position: relative;
}
.son {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}
```

- Grid布局

```
.father {
 height: 100%;
 display: grid;
}
.son { /* thing to center */
 margin: auto;
}

// 一维布局
// justify-content||justify-items && align-content||align-items
// place-content||place-items
```

2. 图片懒加载

> 1.如何判断图片出现在了当前视口 （即如何判断我们能够看到图片）
> 2.如何控制图片的加载

- 方案一：位置计算 + 滚动事件 (Scroll) + DataSet API
  - 判断在当前视口: `clientTop`， `offsetTop`， `clientHeight` 以及 `scrollTop` 各种关于图片的高度作比对
  - 监听 `window.scroll` 事件
  - `DataSet API` 控制图片
- 方案二 [improve]：getBoundingClientRect API + Scroll with Throttle + DataSet API
  - `Element.getBoundingClientRect()` 方法返回元素的大小及其相对于视口的位置
  - 加入节流 优化监听时间

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>图片懒加载</title>
    <style>
      img {
        width: 100%;
        height: 600px;
      }
    </style>
  </head>
  <body>
    <img
      src="test01.jpg"
      alt="1"
    />
    <img
      src="test02.jpg"
      alt="2"
    />
    <img
      data-src="test03.jpg"
      alt="3"
    />
    <img
      data-src="test04jpg"
      alt="4"
    />
    <img
      data-src="test05.jpg"
      alt="5"
    />
    <img
      data-src="test06.jpg"
      alt="6"
    />
    <script src="https://cdn.bootcdn.net/ajax/libs/lodash.js/4.17.20/lodash.js"></scrip
    <script>
      const images = document.querySelectorAll("img");
      const lazyLoad = () => {
        images.forEach((item) => {
          // 触发条件为img元素的CSSOM对象到视口顶部的距离 < 100px + 视口高度，+100px为了提前触发图
          if (
            item.getBoundingClientRect().top <
            document.documentElement.clientHeight + 100
          ) {
            if ("src" in item.dataset) {
              item.src = item.dataset.src;
```

```
            }
          }
        });
      };
      document.addEventListener("scroll", _.throttle(lazyLoad, 200));
    </script>
  </body>
</html>
```

- 方案三： `IntersectionObserver` API + DataSet API
  - `IntersectionObserver` API，一个能够监听元素是否到了当前视口的事件，一步到位！
  - `IntersectionObserver` 除了给图片做懒加载外，还可以对单页应用资源做预加载
- 方案四： `LazyLoading` 属性 浏览器属性

```
<img src="test.jpg" loading="lazy" />
```

3. CSRF

> 跨站请求伪造（英语：Cross-site request forgery），也被称为 one-click attack 或者 session riding，通常缩写为 CSRF 或者 XSRF，
> 是一种挟制用户在当前已登录的 Web 应用程序上执行非本意的操作的攻击方法。
> 跟跨网站脚本（XSS）相比，XSS 利用的是用户对指定网站的信任，CSRF 利用的是网站对用户网页浏览器的信任。

- 使用 JSON API。当进行 CSRF 攻击时，请求体通过 `<form>` 构建，请求头为 application/www-form-urlencoded。它难以发送 JSON 数据被服务器所理解。
- CSRF Token。生成一个随机的 token，切勿放在 cookie 中，每次请求手动携带该 token 进行校验。
- SameSite Cookie。设置为 Lax 或者 Strict，禁止发送第三方 Cookie。

4. 响应式布局

- rem: 根据根元素(即 html)的 font-size
- em: 根据自身元素的 font-size
- vw: viewport width
- vh: viewport height

5. BFC

> 块格式化上下文（Block Formatting Context，BFC）是Web页面的可视化CSS渲染的一部分，是布局过程中生成块级盒子的区域，也是浮动元素与其他元素的交互限定区域。

> - BFC 是一个独立的布局环境,可以理解为一个容器,在这个容器中按照一定规则进行物品摆放,并且不会影响其它环境中的物品。

> - 如果一个元素符合触发 BFC 的条件，则 BFC 中的元素布局不受外部影响。
> - 浮动元素会创建 BFC，则浮动元素内部子元素主要受该浮动元素影响，所以两个浮动元素之间是互不影响的。

- 创建条件
  - 根元素或包含根元素的元素
  - 浮动元素 float ＝ left | right 或 inherit（≠ none）
  - 绝对定位元素 position ＝ absolute 或 fixed
  - display ＝ inline-block | flex | inline-flex | table-cell 或 table-caption
  - overflow ＝ hidden | auto 或 scroll (≠ visible)
- 消除浮动 解决高度塌陷问题（脱离文档流，无法计算准确高度）
  - 解决方法：
    - 在容器中创建BFC
    - `overflow: hidden` ,会造成溢出隐藏，影响与JS交互效果
    - clearfix

浏览器相关

1. cookie相关

什么是cookie?

> HTTP Cookie（也叫 Web Cookie 或浏览器 Cookie）是服务器发送到用户浏览器并保存在本地的一小块数据，它会在浏览器下次向同一服务器再发起请求时被携带并发送到服务器上。通常，它用于告知服务端两个请求是否来自同一浏览器，如保持用户的登录状态。Cookie 使基于无状态的 HTTP 协议记录稳定的状态信息成为了可能。

如何设置一个cookie

> 服务端：通过 setCookie 的响应头来设置 cookie 的，要设置多个 cookie 时，得多写几个 setCookie。服务器如果希望在浏览器保存 Cookie，就要在 HTTP 回应的头信息里面，放置一个 Set-Cookie 字段。
> 前端：使用 `document.cookie` 属性来读写当前网页的 Cookie。写入的时候，Cookie 的值必须写成 key=value 的形式
> Cookie 曾一度用于客户端数据的存储，因当时并没有其它合适的存储办法而作为唯一的存储手段，但现在随着现代浏览器开始支持各种各样的存储方式，Cookie 渐渐被淘汰。由于服务器指定 Cookie 后，浏览器的每次请求都会携带 Cookie 数据，会带来额外的性能开销（尤其是在移动环境下）。新的浏览器 API 已经允许开发者直接将数据存储到本地，如使用 Web storage API （本地存储和会话存储）或 IndexedDB

如何删除一个cookie

> 前端一般不操作，服务端操作
> 通过把该 cookie 的过期时间改为过去时即可删除成功，具体操作的话可以通过操作两个字段来完成
>
> - max-age: 将要过期的最大秒数，设置为 -1 即可删除
> - expires: 将要过期的绝对时间，存储到 cookies 中需要通过 date.toUTCString() 处理，设置为过期时间即可删除

```
// max-age 设置为 -1 即可成功
document.cookie = "a=3; max-age=-1";
```

当 cookie 没有设置 maxage 时，cookie 会存在多久

> 不设置 max-age 和 expires，此 cookie 就是会话级别的，浏览器关闭就没了

浏览器cookie有哪些字段

> - Domain
> - Path
> - Expire/MaxAge
> - HttpOnly
> - Secure
> - SameSite

2. 关于 sessionStorage 与 localStorage

有何区别

> localStorage 生命周期是永久除非自主清除
> sessionStorage 生命周期为当前窗口或标签页，关闭窗口或标签页则会清除数据

> 均只能存储字符串类型的对象

- 不同浏览器无法共享 localStorage 或 sessionStorage 中的信息。
- 相同浏览器的不同页面间可以共享相同的 localStorage（页面属于相同域名和端口），但是不同页面或标签页间无法共享 sessionStorage 的信息。
- 这里需要注意的是，页面及标签页仅指顶级窗口，如果一个标签页包含多个 iframe 标签且他们属于同源页面，那么他们之间是可以共享 sessionStorage 的。

3. 关于跨域

什么是跨域

协议，域名，端口，三者有一不一样，就是跨域
例子🌰：www.baidu.com 与 zhidao.baidu.com 是跨域

如何解决跨域

目前有两种最常见的解决方案：

- CORS，在服务器端设置几个响应头，如 Access-Control-Allow-Origin: *
- Reverse Proxy，在 nginx/traefik/haproxy 等反向代理服务器中设置为同一域名
- JSONP，详解见 JSONP 的原理是什么，如何实现
  - 只能处理 GET 跨域，虽然现在基本上都使用 CORS 跨域
  - 基于两个原理:
    - 动态创建 script，使用 script.src 加载请求跨过跨域
    - script.src 加载的脚本内容为 JSONP: 即 PADDING(JSON) 格式
  - 使用 JSONP 跨域同样需要服务端的支持
  - `$ curl https://shanyue.tech/api/user?id=100&callback=padding`

4. 如何取消请求的发送

- XHR 使用 `xhr.abort()`
- fetch 使用 `AbortController`
  - 发送请求时使用一个 signal 选项控制 fetch 请求
  - `control.abort()` 用以取消请求发送
  - 取消请求发送之后会得到异常 `AbortError`
- Axios: `xhr` 与 `http/https`
  - `Axios` 中通过 `cancelToken` 取消请求发送

5. 如何统计当前页面出现的所有标签

- `document.querySelectorAll('*')`
- `document.getElementsByTagName('*')`
- `$$('*')`，可在浏览器控制台使用
- `document.all`，已废弃，不建议使用