Frontend Basic

Q: Http v.s Https

https 的SSL 加密是在传输层实现的。
(1)http 和https 的基本概念
http: 超文本传输协议，是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW 服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。
https: 是以安全为目标的HTTP 通道，简单讲是HTTP 的安全版，即HTTP 下加入SSL 层，HTTPS 的安全基础是SSL，因此加密的详细内容就需要SSL。
https 协议的主要作用是：建立一个信息安全通道，来确保数组的传输，确保网站的真实性。

(2)http 和https 的区别？
http 传输的数据都是未加密的，也就是明文的，网景公司设置了SSL 协议来对http 协议传输的数据进行加密处理，简单来说https 协议是由http 和ssl 协议构建的可进行加密传输和身份认证的网络协议，比http 协议的安全性更高。
主要的区别如下：
Https 协议需要ca 证书，费用较高。
http 是超文本传输协议，信息是明文传输，https 则是具有安全性的ssl 加密传输协议。
使用不同的链接方式，端口也不同，一般而言，http 协议的端口为80，https 的端口为443
http 的连接很简单，是无状态的；HTTPS 协议是由SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比http 协议安全。

(3)https 协议的工作原理
客户端在使用HTTPS 方式与Web 服务器通信时有以下几个步骤，如图所示。
客户使用https url 访问服务器，则要求web 服务器建立ssl 链接。
web 服务器接收到客户端的请求之后，会将网站的证书（证书中包含了公钥），返回或者说传输给客户端。
客户端和web 服务器端开始协商SSL 链接的安全等级，也就是加密等级。
客户端浏览器通过双方协商一致的安全等级，建立会话密钥，然后通过网站的公钥来加密会话密钥，并传送给网站。
web 服务器通过自己的私钥解密出会话密钥。
web 服务器通过会话密钥加密与客户端之间的通信。

(4)https 协议的优点
使用HTTPS 协议可认证用户和服务器，确保数据发送到正确的客户机和服务器；
HTTPS 协议是由SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，要比http 协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。
HTTPS 是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。
谷歌曾在2014 年8 月份调整搜索引擎算法，并称“比起同等HTTP 网站，采用HTTPS 加密的网站在搜索结果中的排名将会更高”。

(5)https 协议的缺点
https 握手阶段比较费时，会使页面加载时间延长50%，增加10%~20%的耗电。
https 缓存不如http 高效，会增加数据开销。

SSL 证书也需要钱，功能越强大的证书费用越高。
SSL 证书需要绑定IP，不能再同一个ip 上绑定多个域名，ipv4 资源支持不了这种消耗。

Q: BOM, DOM

DOM 指的是文档对象模型，它指的是把文档当做一个对象来对待，这个对象主要定义了处理网页内容的方法和接口。

```
const parent = document.getElementById('parent');
parent.appendChild(newDiv);

const element = document.querySelector('.myClass');
```

BOM 指的是浏览器对象模型，它指的是把浏览器当做一个对象来对待，这个对象主要定义了与浏览器进行交互的方法和接口。
BOM的核心是 window，而 window 对象具有双重角色，它既是通过 js 访问浏览器窗口的一个接口，又是一个 Global（全局）
对象。这意味着在网页中定义的任何对象，变量和函数，都作为全局对象的一个属性或者方法存在。
window 对象含有 locati
on 对象、navigator 对象、screen 对象、history对象等子对象，并且 DOM 的最根本的对象 document 对象也是 BOM 的 window 对
象的子对象。

```
(1)location 对象
location.href-- 返回或设置当前文档的URL
location.search -- 返回URL 中的查询字符串部分。例如
http://www.dreamdu.com/dreamdu.php?id=5&name=dreamdu 返回包括(?)后面的内容?
id=5&name=dreamdu
location.hash -- 返回URL#后面的内容，如果没有#，返回空
location.host -- 返回URL 中的域名部分，例如www.dreamdu.com
location.hostname -- 返回URL 中的主域名部分，例如dreamdu.com
location.pathname -- 返回URL 的域名后的部分。例如http://www.dreamdu.com/xhtml/
返回/xhtml/
location.port -- 返回URL 中的端口部分。例如http://www.dreamdu.com:8080/xhtml/
返回8080
location.protocol -- 返回URL 中的协议部分。例如
http://www.dreamdu.com:8080/xhtml/ 返回(//)前面的内容http:
location.assign -- 设置当前文档的URL
location.replace() -- 设置当前文档的URL，并且在history 对象的地址列表中移除这个URL
location.replace(url);
location.reload() -- 重载当前页面

(2)history 对象
history.go() -- 前进或后退指定的页面数history.go(num);
history.back() -- 后退一页
history.forward() -- 前进一页

(3)Navigator 对象
navigator.userAgent -- 返回用户代理头的字符串表示(就是包括浏览器版本信息等的字符串)
navigator.cookieEnabled -- 返回浏览器是否支持(启用)cookie
```

Q: Virtual dom

用JavaScript 对象结构表示DOM 树的结构；然后用这个树构建一个真正的DOM 树，插到文档当中当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异把所记录的差异应用到所构建的真正的DOM 树上，视图就更新了。Virtual DOM 本质上就是在JS 和DOM 之间做了一个缓存。

Q: HTTP Status Code

100 Continue 继续。客户端应继续其请求
101 Switching Protocols 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP 的新版本协议
200 OK 请求成功。一般用于GET 与POST 请求
201 Created 已创建。成功请求并创建了新的资源
202 Accepted 已接受。已经接受请求，但未处理完成
203 Non-Authoritative Information 非授权信息。请求成功。但返回的meta 信息不在原始的服务器，而是一个副本
204 No Content 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
205 Reset Content 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
206 Partial Content 部分内容。服务器成功处理了部分GET 请求
300 Multiple Choices 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301 Moved Permanently 永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI 代替
302 Found 临时移动。与301 类似。但资源只是临时被移动。客户端应继续使用原有URI
303 See Other 查看其它地址。与301 类似。使用GET 和POST 请求查看
304 Not Modified 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
305 Use Proxy 使用代理。所请求的资源必须通过代理访问
306 Unused 已经被废弃的HTTP 状态码
307 Temporary Redirect 临时重定向。与302 类似。使用GET 请求重定向
400 Bad Request 客户端请求的语法错误，服务器无法理解
401 Unauthorized 请求要求用户的身份认证
402 Payment Required 保留，将来使用
403 Forbidden 服务器理解请求客户端的请求，但是拒绝执行此请求
404 Not Found 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面
405 Method Not Allowed 客户端请求中的方法被禁止
406 Not Acceptable 服务器无法根据客户端请求的内容特性完成请求
407 Proxy Authentication Required 请求要求代理的身份认证，与401 类似，但请求者应当使用代理进行授权
408 Request Time-out 服务器等待客户端发送的请求时间过长，超时
409 Conflict 服务器完成客户端的PUT 请求是可能返回此代码，服务器处理请求时发生了冲突
410 Gone 客户端请求的资源已经不存在。410 不同于404，如果资源以前有现在被永久删除了可使用410 代码，网站设计人员可通过301 代码指定资源的新位置
411 Length Required 服务器无法处理客户端发送的不带Content-Length 的请求信息

412 Precondition Failed 客户端请求信息的先决条件错误
413 Request Entity Too Large 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After 的响应信息
414 Request-URI Too Large 请求的URI 过长（URI 通常为网址），服务器无法处理
415 Unsupported Media Type 服务器无法处理请求附带的媒体格式
416 Requested range not satisfiable 客户端请求的范围无效
417 Expectation Failed 服务器无法满足Expect 的请求头信息
500 Internal Server Error 服务器内部错误，无法完成请求
501 Not Implemented 服务器不支持请求的功能，无法完成请求
502 Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应

Q: Why fetch request will send twice?

fetch 发送post 请求的时候，总是发送2 次，第一次状态码是204，第二次才成功？
原因很简单，因为你用fetch 的post 请求的时候，导致fetch 第一次发送了一个Options 请求，询问服务器是否支持修改的请求头，如果服务器支持，则在第二次中发送真正的请求

Q: Cookie, sessionStorage, LocalStorage

共同点：都是保存在浏览器端，并且是同源的

Cookie：cookie 数据始终在同源的http 请求中携带（即使不需要），即cookie 在浏览器和服务器间来回传递。而sessionStorage 和localStorage 不会自动把数据发给服务器，仅在本地保存。cookie 数据还有路径（path）的概念，可以限制cookie 只属于某个路径下，存储的大小很小只有4K 左右。（key：可以在浏览器和服务器端来回传递，存储容量小，只有大约4K 左右）

sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持，localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie 只在设置的cookie 过期时间之前一直有效，即使窗口或浏览器关闭。（key：本身就是一个回话过程，关闭浏览器后消失，session 为一个回话，当页面不同即使是同一页面打开两次，也被视为同一次回话）

localStorage：localStorage 在所有同源窗口中都是共享的；cookie 也是在所有同源窗口中都是共享的。（key：同源窗口都会共享，并且不会失效，不管窗口或者浏览器关闭与否都会始终生效）

补充说明一下cookie 的作用：
保存用户登录状态。例如将用户id 存储于一个cookie 内，这样当用户下次访问该页面时就不需要重新登录了，现在很多论坛和社区都提供这样的功能。cookie 还可以设置过期时间，当超过时间期限后，cookie 就会自动消失。因此，系统往往可以提示用户保持登录状态的时间：常见选项有一个月、三个月、一年等。

跟踪用户行为。例如一个天气预报网站，能够根据用户选择的地区显示当地的天气情况。如果每次都需要选择所在地是烦琐的，当利用了cookie 后就会显得很人性化了，系统能够记住上一次访问的地区，当下次再打开该页面时，它就会自动显示上次用户所在地区

```
的天气情况。因为一切都是在后台完成，所以这样的页面就像为某个用户所定制的一
样，使用起来非常方便定制页面。如果网站提供了换肤或更换布局的功能，那么可以使
用cookie 来记录用户的选项，例如：背景色、分辨率等。当用户下次访问时，仍然可以
保存上一次访问的界面风格。

存储大小：
  cookie 数据大小不能超过4 k 。
  sessionStorage 和 localStorage 虽然也有存储大小的限制，但比 cookie 大得多，可以
达到 5M 或更大。

有期时间：
  localStorage    存储持久数据，浏览器关闭后数据不丢失除非主动删除数据。
  sessionStorage  数据在页面会话结束时会被清除。页面会话在浏览器打开期间一直保持，并且
重新加载或恢复页面仍会
                    保持原来的页面会话。在新标签或窗口打开一个页面时会在顶级浏览上下文
中初始化一个新的会话。
  cookie          设置的 cookie 过期时间之前一直有效，即使窗口或浏览器关闭。

作用域：
    sessionStorage  只在同源的同窗口（或标签页）中共享数据，也就是只在当前会话中共享。
    localStorage    在所有同源窗口中都是共享的。
    cookie          在所有同源窗口中都是共享的。
```

Q: Web Worker

```
在HTML 页面中，如果在执行脚本时，页面的状态是不可相应的，直到脚本执行完成后，
页面才变成可相应。web worker 是运行在后台的js，独立于其他脚本，不会影响页面你
的性能。并且通过postMessage 将结果回传到主线程。这样在进行复杂操作的时候，就
不会阻塞主线程了。

如何创建web worker：
检测浏览器对于web worker 的支持性
创建web worker 文件（js，回传函数等）
创建web worker 对象
```

Q: iframe

```
定义：iframe 元素会创建包含另一个文档的内联框架
提示：可以将提示文字放在<iframe></iframe>之间，来提示某些不支持iframe 的浏览器

缺点：
会阻塞主页面的onload 事件
搜索引擎无法解读这种页面，不利于SEO
iframe 和主页面共享连接池，而浏览器对相同区域有限制所以会影响性能。
```

Q: Doctype?

```
Tell browser use which method to read the document, there are 2 types:
stands-mode, quirks-mode
```

Q: XSS attack?

```
Cross Site Scripting
XSS 攻击指的是跨站脚本攻击，是一种代码注入攻击。攻击者通过在网站注入恶意脚本，使之在用户的浏览器上运行，从而盗取用户的信息如 cookie 等。

XSS 的本质是因为网站没有对恶意代码进行过滤，与正常的代码混合在一起了，浏览器没有办法分辨哪些脚本是可信的，从而导致了恶意代码的执行。

XSS 一般分为存储型、反射型和 DOM 型。

Stored XSS
存储型指的是恶意代码提交到了网站的数据库中，当用户请求数据的时候，服务器将其拼接为 HTML 后返回给了用户，从而导致了恶意代码的执行。

Reflected XSS
反射型指的是攻击者构建了特殊的 URL，当服务器接收到请求后，从 URL 中获取数据，拼接到 HTML 后返回，从而导致了恶意代码的执行。

DOM Based XSS
DOM 型指的是攻击者构建了特殊的 URL，用户打开网站后，js 脚本从 URL 中获取数据，从而导致了恶意代码的执行。

XSS 攻击的预防可以从两个方面入手，一个是恶意代码提交的时候，一个是浏览器执行恶意代码的时候。

对于第一个方面，如果我们对存入数据库的数据都进行的转义处理，但是一个数据可能在多个地方使用，有的地方可能不需要转义，由于我们没有办法判断数据最后的使用场景，所以直接在输入端进行恶意代码的处理，其实是不太可靠的。

因此我们可以从浏览器的执行来进行预防，一种是使用纯前端的方式，不用服务器端拼接后返回。另一种是对需要插入到 HTML 中的代码做好充分的转义。对于 DOM 型的攻击，主要是前端脚本的不可靠而造成的，我们对于数据获取渲染和字符串拼接的时候应该对可能出现的恶意代码情况进行判断。

还有一些方式，比如使用 CSP ，CSP 的本质是建立一个白名单，告诉浏览器哪些外部资源可以加载和执行，从而防止恶意代码的注入攻击。

还可以对一些敏感信息进行保护，比如 cookie 使用 http-only ，使得脚本无法获取。也可以使用验证码，避免脚本伪装成用户执行一些操作。
```

Q: CSP?

```
Content-Security-Policy
CSP 指的是内容安全策略，它的本质是建立一个白名单，告诉浏览器哪些外部资源可以加载和执行。
```

我们只需要配置规则，如何拦截由浏览器自己来实现。

通常有两种方式来开启 CSP，一种是设置 HTTP 首部中的 Content-Security-Policy，
一种是设置 meta 标签的方式 <metahttp-equiv="Content-Security-Policy">

Q: CSRF attack?

Cross-site request forgery
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

CSRF 攻击指的是跨站请求伪造攻击，攻击者诱导用户进入一个第三方网站，然后该网站向被攻击网站发送跨站请求。
如果用户在被攻击网站中保存了登录状态，那么攻击者就可以利用这个登录状态，绕过后台的用户验证，冒充用户向服务器执行一些操作。

CSRF 攻击的本质是利用了 cookie 会在同源请求中携带发送给服务器的特点，以此来实现用户的冒充。

一般的 CSRF 攻击类型有三种：

第一种是 GET 类型的 CSRF 攻击，比如在网站中的一个 img 标签里构建一个请求，当用户打开这个网站的时候就会自动发起提
交。

第二种是 POST 类型的 CSRF 攻击，比如说构建一个表单，然后隐藏它，当用户进入页面时，自动提交这个表单。

第三种是链接类型的 CSRF 攻击，比如说在 a 标签的 href 属性里构建一个请求，然后诱导用户去点击。

CSRF 可以用下面几种方法来防护：

第一种是同源检测的方法，服务器根据 http 请求头中 origin 或者 referer 信息来判断请求是否为允许访问的站点，从而对请求进行过滤。当 origin 或者 referer 信息都不存在的时候，直接阻止。这种方式的缺点是有些情况下 referer 可以被伪造。还有就是我们这种方法同时把搜索引擎的链接也给屏蔽了，所以一般网站会允许搜索引擎的页面请求，但是相应的页面请求这种请求方式也可能被攻击者给利用。

第二种方法是使用 CSRF Token 来进行验证，服务器向用户返回一个随机数 Token ，当网站再次发起请求时，在请求参数中加入服务器端返回的 token ，然后服务器对这个 token 进行验证。这种方法解决了使用 cookie 单一验证方式时，可能会被冒用的问题，但是这种方法存在一个缺点就是，我们需要给网站中的所有请求都添加上这个 token，操作比较繁琐。还有一个问题是一般不会只有一台网站服务器，如果我们的请求经过负载平衡转移到了其他的服务器，但是这个服务器的 session 中没有保留这个 token 的话，就没有办法验证了。这种情况我们可以通过改变 token 的构建方式来解决。

第三种方式使用双重 Cookie 验证的办法，服务器在用户访问网站页面时，向请求域名注入一个 Cookie，内容为随机字符串，然后当用户再次向服务器发送请求的时候，从 cookie 中取出这个字符串，添加到 URL 参数中，然后服务器通过对 cookie 中的数据和参数中的数据进行比较，来进行验证。使用这种方式是利用了攻击者只能利用 cookie，但是不能访问获取 cookie 的特点。并且这

种方法比 CSRF Token 的方法更加方便，并且不涉及到分布式访问的问题。这种方法的缺点是如果网站存在 XSS 漏洞的，那么这种方式会失效。同时这种方式不能做到子域名的隔离。

第四种方式是使用在设置 cookie 属性的时候设置 Samesite ，限制 cookie 不能作为被第三方使用，从而可以避免被攻击者利用。Samesite 一共有两种模式，一种是严格模式，在严格模式下 cookie 在任何情况下都不可能作为第三方 Cookie 使用，在宽松模式下，cookie 可以被请求是 GET 请求，且会发生页面跳转的请求所使用。

Q: Samesite cookie?

Samesite Cookie 表示同站 cookie，避免 cookie 被第三方所利用。

将 Samesite 设为 strict ，这种称为严格模式，表示这个 cookie 在任何情况下都不可能作为第三方 cookie。

将 Samesite 设为 Lax ，这种模式称为宽松模式，如果这个请求是个 GET 请求 ，并且这个请求改变了当前页面或者打开了新的页面，那么这个 cookie 可以作为第三方 cookie，其余情况下都不能作为第三方 cookie。

使用这种方法的缺点是，因为它不支持子域，所以子域没有办法与主域共享登录信息，每次转入子域的网站，都回重新登录。
还有一个问题就是它的兼容性不够好。

Q. 强缓存 协商缓存

强缓存 从缓存取 200（from cache） 直接从缓存取
协商缓存 从缓存取 304（not modified） 通过服务器来告知缓存是否可用

强缓存相关字段有expires，cache-control。如果cache-control 与expires 同时存在的话，
cache-control 的优先级高于expires。

协商缓存相关字段有Last-Modified/If-Modified-Since，Etag/If-None-Match

Q. GET vs. POST

get 参数通过url 传递，post 放在request body 中。

get 请求在url 中传递的参数是有长度限制的，而post 没有。

get 比post 更不安全，因为参数直接暴露在url 中，所以不能用来传递敏感信息。

get 请求只能进行url 编码，而post 支持多种编码方式

get 请求会浏览器主动cache，而post 支持多种编码方式。

```
get 请求参数会被完整保留在浏览历史记录里，而post 中的参数不会被保留。

GET 和POST 本质上就是TCP 链接，并无差别。但是由于HTTP 的规定和浏览器/服务器
的限制，导致他们在应用过程中体现出一些不同。

GET 产生一个TCP 数据包；POST 产生两个TCP 数据包。
对于GET 方式的请求，浏览器会把http header 和data 一并发送出去，服务器响应200
（返回数据）；
而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服
务器响应200 ok（返回数据）
```

Q. HTTP methods?

```
GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE, CONNECT
```

Q. What will happen when you input url in the browser?

```
1. Find DNS
 - can from browser cache, system cache, router cache and etc.

2. Go to the ip to get content

3. Use return html to build DOM, CSSOM

4. Show to user

DNS 解析
TCP 连接
发送HTTP 请求
服务器处理请求并返回HTTP 报文
浏览器解析渲染页面
连接结束
```

Q. web 性能优化

```
降低请求量：合并资源，减少HTTP 请求数，minify / gzip 压缩，webP，lazyLoad。

加快请求速度：预解析DNS，减少域名数，并行加载，CDN 分发。

缓存：HTTP 协议缓存请求，离线缓存manifest，离线数据缓存localStorage。

渲染：JS/CSS 优化，加载顺序，服务端渲染，pipeline。
```

PWA SPA SSR

在现代前端开发中，PWA（Progressive Web App）、SPA（Single Page App）和 SSR（服务器端渲染）是三种常见的技术概念，它们各自关注不同的方面，但也可以相互结合，以提升用户体验和应用性能。

单页应用（SPA）

SPA 是一种通过动态重写当前页面与用户交互的 Web 应用程序，避免了传统的页面重新加载。

特点：

仅在初次加载时请求完整页面，后续通过 JavaScript 动态更新内容。
提供更流畅的用户体验，类似于桌面应用程序。
常用框架包括 React、Vue.js 和 Angular。

优点：

减少页面刷新，提升响应速度。
前后端分离，开发维护更灵活。

缺点：

首次加载时间可能较长。
对 SEO 不友好，需要额外处理。
服务器端渲染（SSR）

SSR 是指在服务器上生成 HTML 内容，然后将生成的页面发送给客户端。

特点：

首屏加载速度快，因为 HTML 是由服务器生成的，并且直接返回给客户端。
对 SEO 友好，因为爬虫可以立即访问完整的 HTML 内容。
可以通过框架（如 Next.js for React, Nuxt.js for Vue）实现，既能保持客户端交互性，也能享受服务端渲染的性能优势。

优点：

首屏渲染快，提升用户体验。
有利于搜索引擎优化。

缺点：

服务器压力大，复杂的应用可能需要更强的服务器资源。
开发和部署相对复杂。

渐进式 Web 应用（PWA）

PWA 是利用现代 Web 技术构建的应用程序，旨在提供类似原生应用的用户体验。
可以在各种设备和平台上提供一致的用户体验。

特点：

可离线访问，通过 Service Worker 缓存资源。
可安装到主屏幕，提供全屏体验。
支持推送通知等功能。(Push notification)

优点：

提升用户体验，增强用户粘性。
降低开发成本，无需为不同平台开发独立的应用。

缺点：

部分功能在某些浏览器中可能不受支持。
需要 HTTPS 支持，增加部署复杂度。

SPA 专注于页面的动态加载和交互，SSR 专注于提升首屏渲染速度和 SEO，PWA 专注于提升 Web 应用的可靠性、性能和用户体验。
它们并非互相包含的关系，但可以相互结合。例如，SPA 可以通过引入 PWA 的特性（如 Service Worker、Web App Manifest）来增强功能，提供更好的用户体验。
同时，SSR 可以与 PWA 结合，既提升首屏加载速度，又提供离线访问等功能。

什么是浏览器的同源政策？

我对浏览器的同源政策的理解是，一个域下的 js 脚本在未经允许的情况下，不能够访问另一个域的内容。这里的同源的指的是两个
域的协议、域名、端口号必须相同，否则则不属于同一个域。

同源政策主要限制了三个方面

第一个是当前域下的 js 脚本不能够访问其他域下的 cookie、localStorage 和 indexDB。

第二个是当前域下的 js 脚本不能够操作访问操作其他域下的 DOM。

第三个是当前域下 ajax 无法发送跨域请求。

同源政策的目的主要是为了保证用户的信息安全，它只是对 js 脚本的一种限制，并不是对浏览器的限制，对于一般的 img、或者
script 脚本请求都不会有跨域的限制，这是因为这些操作都不会通过响应结果来进行可能出现安全问题的操作。

节流(Throttle)与防抖（Debounce）

都是控制函数执行频率的技术，但有所区别：
节流： 在连续触发事件时，保证在指定时间间隔内至少执行一次函数。
防抖： 在连续触发事件时，只有在事件停止触发后的指定时间内才执行函数。

What's browser?

简单来说浏览器可以分为两部分，shell 和 内核。

其中 shell 的种类相对比较多，内核则比较少。shell 是指浏览器的外壳：例如菜单，工具栏等。主要是提供给用户界面操作，
参数设置等等。它是调用内核来实现各种功能的。内核才是浏览器的核心。内核是基于标记语言显示内容的程序或模块。也有一些
浏览器并不区分外壳和内核。从 Mozilla 将 Gecko 独立出来后，才有了外壳和内核的明确划分。

```
shell + rendering engine + js engine

* 用户界面
  * 主进程
  * 内核
      * 渲染引擎
      * JS 引擎
          * 执行栈
      * 事件触发线程
          * 消息队列
              * 微任务
              * 宏任务
      * 网络异步线程
      * 定时器线程
```

浏览器内核(core)的理解？

主要分成两部分：渲染引擎(Rendering Engine)和 JS引擎(V8)。

渲染引擎的职责就是渲染，即在浏览器窗口中显示所请求的内容。
默认情况下，渲染引擎可以显示 html、xml 文档及图片，它也可以借助插件（一种浏览器扩展）显示其他类型数据，
例如使用 PDF 阅读器插件，可以显示 PDF 格式。
Google Chrome：使用 Blink 内核。
Mozilla Firefox：使用 Gecko 内核。
Apple Safari：使用 WebKit 内核。
Microsoft Edge：早期版本使用 EdgeHTML 内核，后续版本转为使用 Blink 内核。
Opera：早期版本使用自研的 Presto 内核，后转为使用 Blink 内核。

JS 引擎：解析和执行 javascript 来实现网页的动态效果。
avaScript 引擎是用于解析和执行 JavaScript 代码的程序或解释器，通常嵌入在浏览器中，也可用于其他运行环境。它们将 JavaScript 源代码转换为机器码，以便计算机能够执行。

主要 JavaScript 引擎：
V8：V8 是 Google Chrome 和 Node.js 的 JavaScript 引擎。
NODE.JS 中文网
SpiderMonkey：由 Mozilla 开发，是第一个 JavaScript 引擎，最初用于 Netscape Navigator 浏览器，现在用于 Firefox 浏览器。

JavaScriptCore（Nitro）：由 Apple 开发，用于 Safari 浏览器。
Chakra：由微软开发，最初用于 Internet Explorer 浏览器，后续版本用于 Microsoft Edge 浏览器。
Hermes：由 Facebook 开发，专为 React Native 优化的 JavaScript 引擎，旨在提高移动应用的性能。

最开始渲染引擎和 JS 引擎并没有区分的很明确，后来 JS 引擎越来越独立，内核就倾向于只指渲染引擎。

前端需要注意哪些 SEO ？

（1）合理的 title、description、keywords：搜索对这三项的权重逐个减小，title 值强调重点即可，重要关键词出现不要超
过2次，而且要靠前，不同页面 title 要有所不同；description 把页面内容高度概括，长度合适，不可过分堆砌关键词，不
同页面 description 有所不同；keywords 列举出重要关键词即可。

（2）语义化的 HTML 代码，符合 W3C 规范：语义化代码让搜索引擎容易理解网页。

（3）重要内容 HTML 代码放在最前：搜索引擎抓取 HTML 顺序是从上到下，有的搜索引擎对抓取长度有限制，保证重要内容肯定被
抓取。

（4）重要内容不要用 js 输出：爬虫不会执行 js 获取内容

（5）少用 iframe：搜索引擎不会抓取 iframe 中的内容

（6）非装饰性图片必须加 alt

（7）提高网站速度：网站速度是搜索引擎排序的一个重要指标

如何实现浏览器内多个标签页之间的通信?

（1）使用 WebSocket，通信的标签页连接同一个服务器，发送消息到服务器后，服务器推送消息给所有连接的客户端。

（2）使用 SharedWorker （只在 chrome 浏览器实现了），两个页面共享同一个线程，通过向线程发送数据和接收数据来实现标
签页之间的双向通行。

（3）可以调用 localStorage、cookies 等本地存储方式，localStorge 另一个浏览上下文里被添加、修改或删除时，它都会触
发一个 storage 事件，我们通过监听 storage 事件，控制它的值来进行页面信息通信；

（4）如果我们能够获得对应标签页的引用，通过 postMessage 方法也是可以实现多个标签页通信的。

Chrome waterfall

```
In web development, the Waterfall view in Chrome's Developer Tools is a
crucial feature for analyzing the performance of network requests.
此时可以使用Chrome提供的开发者工具在前端页面上具体看到具体那里花费时间，方便定位问题。
```

web vital

```
Web Vitals is an initiative by Google that provides unified guidance for
quality signals
essential to delivering a great user experience on the web.

1. Largest Contentful Paint (LCP):
   Measures loading performance. To provide a good user experience, LCP
should occur within 2.5 seconds of when the page first starts loading.

2. Interaction to Next Paint (INP):
  Measures responsiveness. To provide a good user experience, pages should
have an INP of 200 milliseconds or less.

3. Cumulative Layout Shift (CLS):
  Measures visual stability. To provide a good user experience, pages
should maintain a CLS of 0.1 or less.


To measure and report these metrics, developers can use various tools and
libraries.
For instance, the `web-vitals` JavaScript library is a small, production-
ready wrapper around the underlying web APIs that measures each metric
in a way that accurately matches how they're reported by all the Google
tools.
```