1. 说一说Vue3和Vue2

Vue3结合ES6去改进解决架构问题

1. 性能方面的优化，速度更快、体积减少、更易维护、更接近原生、更易使用

（2）响应时API的改变  
 Vue3用Proxy替换Vue2的defineProperty。

**defineProperty**直接在一个对象上定义一个新属性，或者修改一个对象的现有属性（通过getter/setter），并返回此对象。虽然能够劫持对象的属性，但需要对对象的每个属性进行遍历劫持；如果对象上有新增的属性，则需要对新增的属性再次进行劫持（通过$set）；如果属性是对象，还需要深度遍历。

**Proxy**是代理整个对象，通过监听代理对象上属性的变化来获取目标对象属性的变化。Proxy本身是一个构造函数，通过new Proxy生成拦截的实例对象，让外界进行过滤和访问。Proxy不仅能够监听到属性的增加，还能监听属性的删除。 使用 defineProperty 只能重定义属性的读取（get）和设置（set）行为，Proxy能支持更多行为操作，比defineProperty的功能更为强大。

（3）为typescript提供更好的支持

（4) 在diff算法做了优化（不必像Vue2一样将全部dom进行比较）

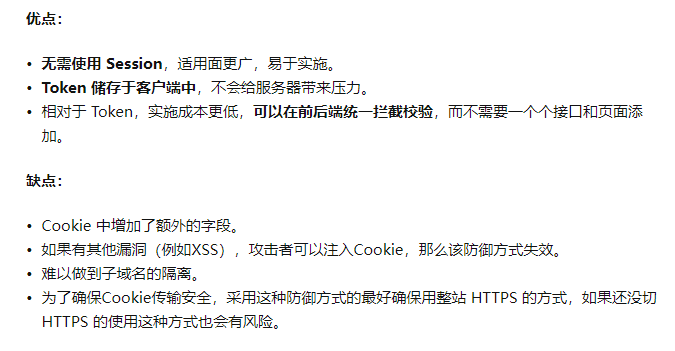
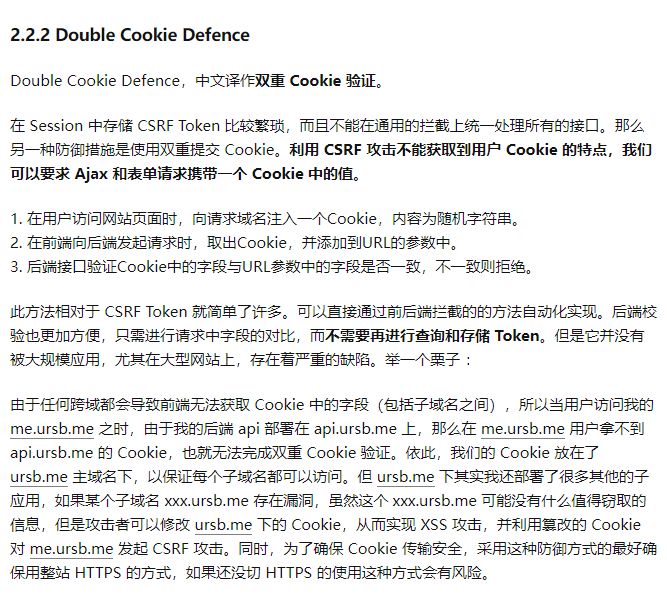
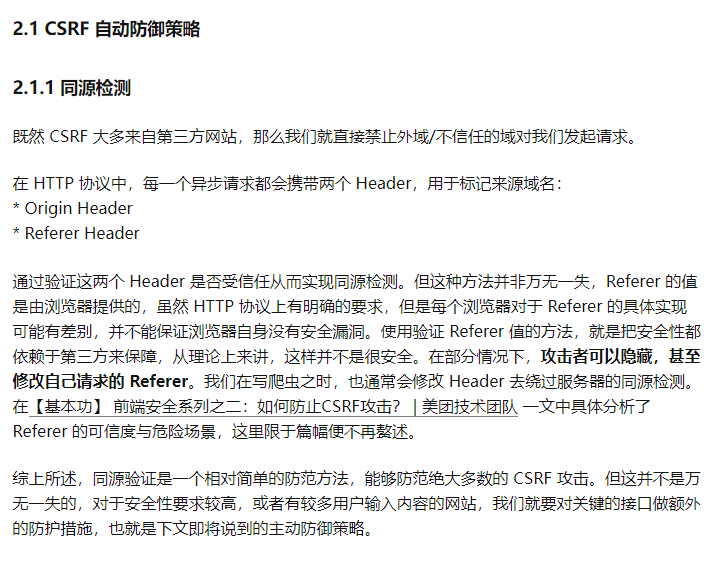
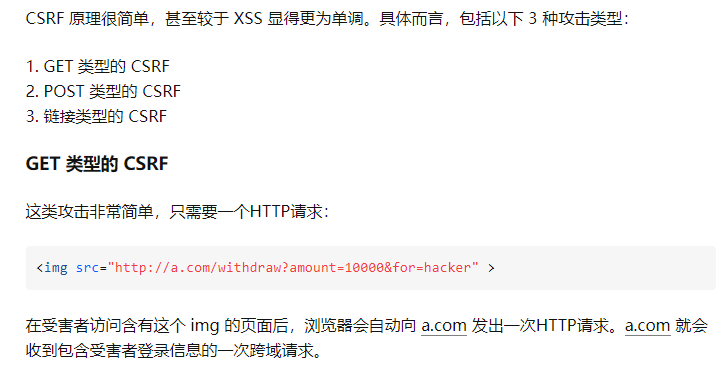
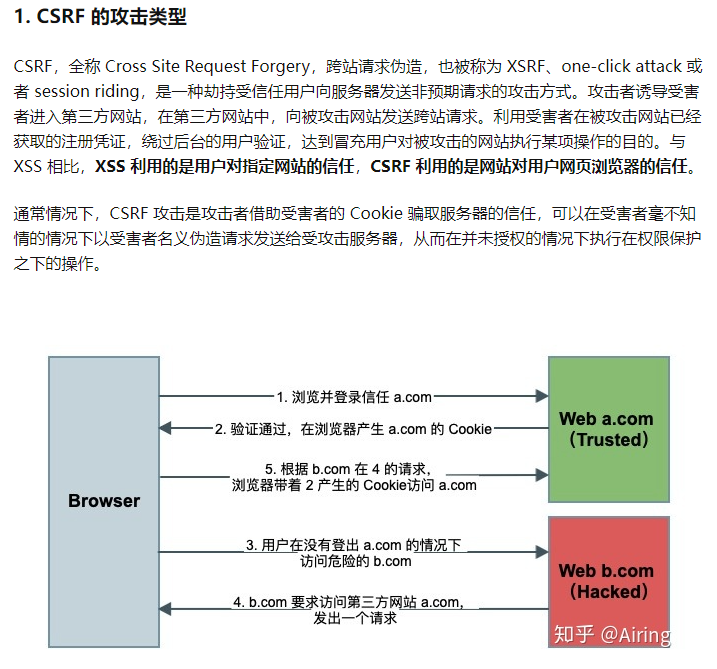
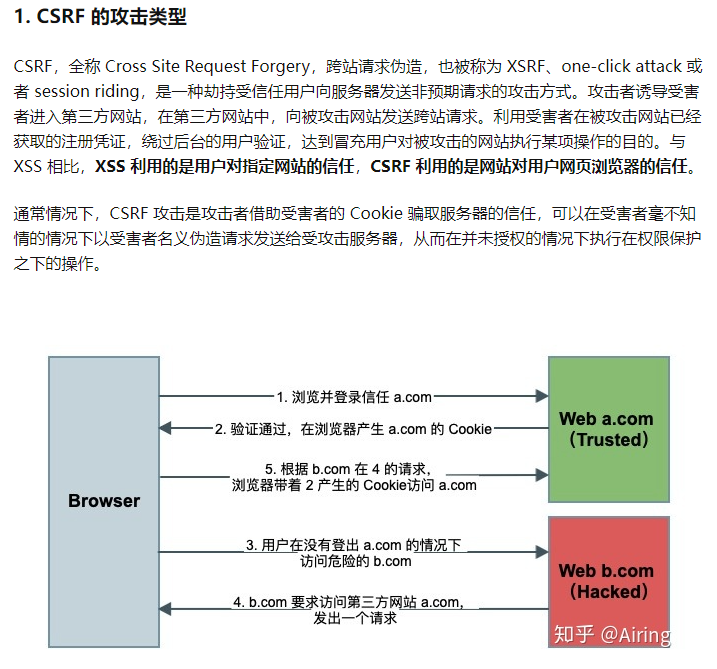
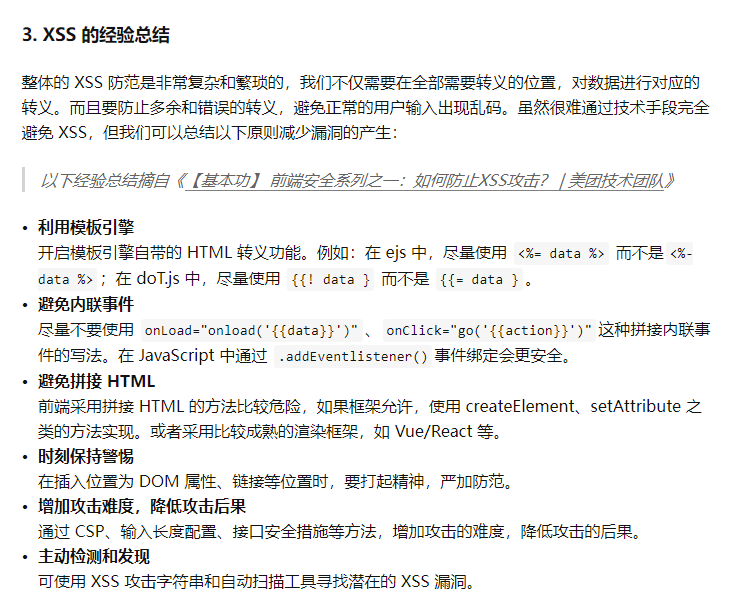
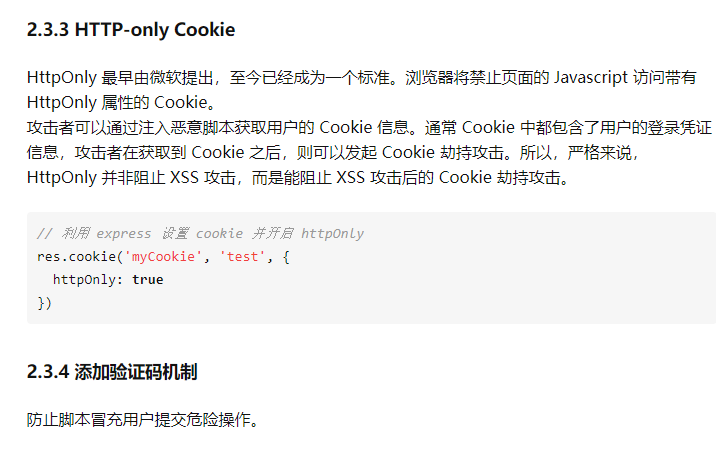
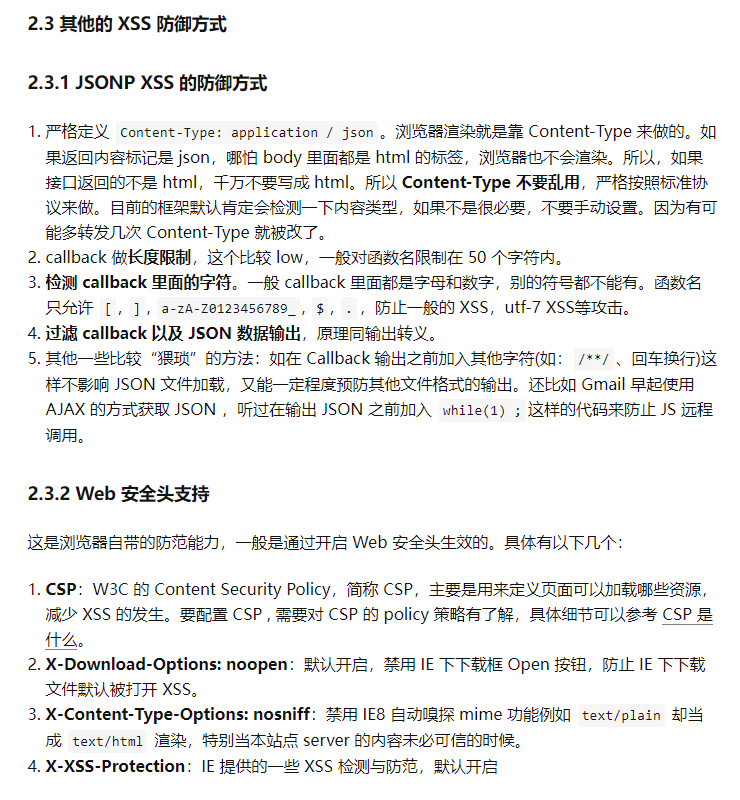
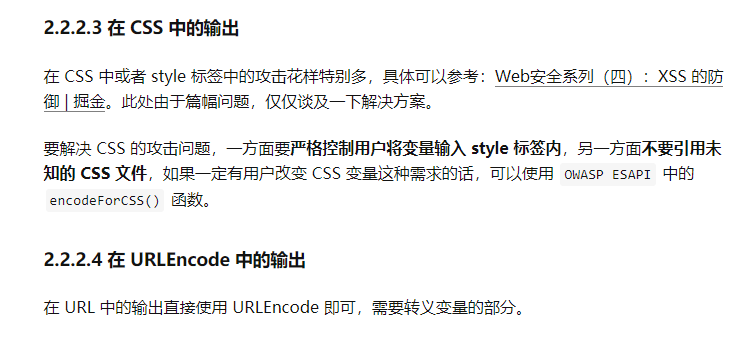
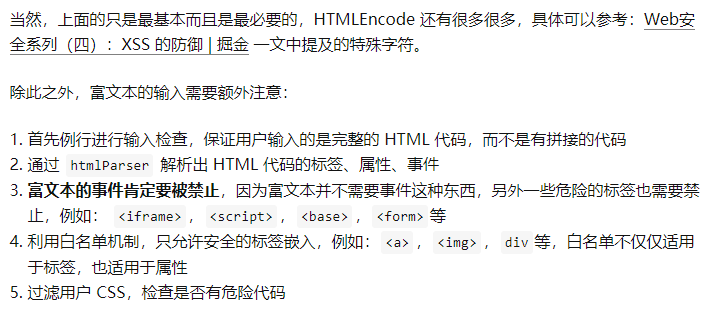
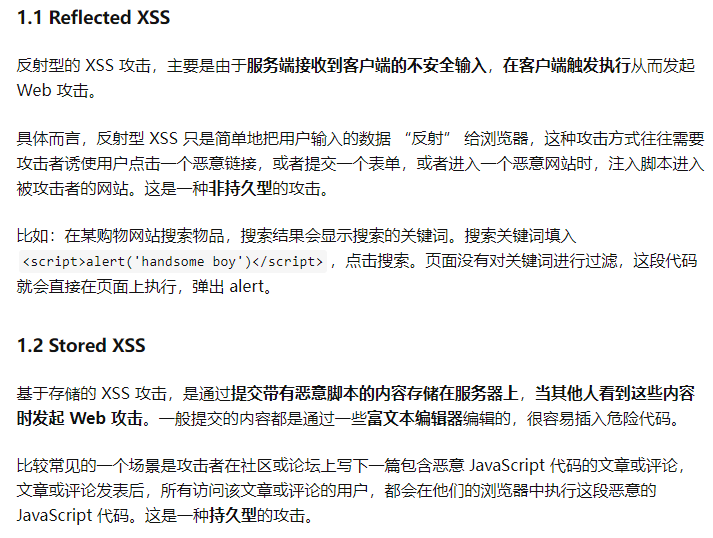
（5）Vue2使用Options API，Vue3使用Composition API

Vue2将data，computed，watch，methods等属性方法以及mounted等生命周期钩子分割放置，看起来层次分明，但随着需求变得复杂和组件功能的增多,组件的阅读和理解难度会增加，而且文件会越来越大，查询和修改起来会比较麻烦，且逻辑过多时调用使用 this 会出现指向混乱等问题。

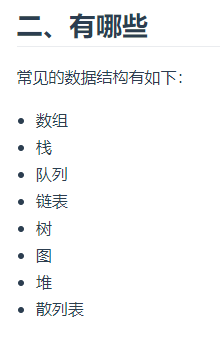
Vue3把零散在各个data，methods的代码重新组合，用setup()和return来存放和提供调用，单独一个功能的代码放在一起维护（高内聚低耦合），并可将代码单独拆分成函数。要查找修改某些功能时能比较清楚完整的看到整个功能的详细代码，不像Vue2比较散的一截截断断续续的代码。代码重用性更好。也没有对this的使用，填补了Vue2的漏洞。Vue3可以通过引入生命周期钩子加个on来访问生命周期

1. 提供Tree Shaking 支持（一种通过消除最终文件中未使用的代码来优化体积，支持按需编译的方法）
2. 提供一些新的API、新的组件（Fragment《不必特意用一个新的元素节点（如div）包裹隔离才能形成组件》、Teleport（类似于React的Portal）、Suspense）以及一些语法的改写
3. 常见网络安全攻击及其防御

|  |  |  |
| --- | --- | --- |
|  | 定义 | 解决方法 |
| XSS | 最常见的 Web 攻击，是一种代码注入攻击。攻击者通过在目标网站上注入恶意脚本，使之在用户的浏览器上运行。利用这些恶意脚本，攻击者可获取用户的敏感信息如 Cookie、SessionID 等，进而危害数据安全。  XSS攻击实质是两种情况：  1.攻击者提交恶意代码。  2.浏览器执行恶意代码。  XSS 攻击一般存在以下几类：  （1）反射型 XSS 攻击：主要是由于**服务端接收到客户端的不安全输入**，**在客户端触发执行**从而发起 Web 攻击。比如在图片或者链接中嵌入恶意脚本  （2）存储型（持久型） XSS 攻击：  通过**提交带有恶意脚本的内容存储在服务器上**，**当其他人看到这些内容时发起 Web 攻击。比如**攻击者在社区或论坛上写下一篇包含恶意JS代码的文章或评论，所有访问该文章或评论的用户的浏览器中都会执行这段恶意的 JS代码  （3）DOM XSS攻击：通过恶意脚本修改页面的 DOM 结构，是**纯粹发生在客户端的攻击，DOM 型 XSS 攻击中，取出和执行恶意代码由浏览器端完成，属于前端 JS 自身的安全漏洞，而上面两种 XSS 都属于服务端的安全漏洞，本质上是前端 JS 代码本身不够严谨，把不可信的数据当作代码执行了**  （4）JSONP XSS攻击：**JSONP 的 callback 参数非常危险，一是callback 参数意外截断 JS代码，特殊字符单引号双引号，换行符均存在风险。二是callback 参数恶意添加标签，造成 XSS 漏洞** | 1. 设置HttpOnly属性,浏览器将禁止页面的 Javascript 访问带有 HttpOnly 属性的 Cookie。   攻击者可以通过注入恶意脚本获取用户的 Cookie 信息。借此发起 Cookie 劫持攻击  HttpOnly 并非阻止 XSS 攻击，而是阻止 XSS 攻击后的 Cookie 劫持攻击   1. 从输入到输出都需要过滤、转义 2. 添加验证码 |
| CSRF | 跨站请求伪造，是一种劫持受信任用户向服务器发送非预期请求的攻击方式。攻击者诱导受害者进入第三方网站并窃取其登录注册等敏感信息，利用这些信息冒充用户对被攻击的网站执行某项操作。**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 的两个特点：  1.CSRF（通常）发生在第三方域名。  2.CSRF 攻击者不能获取到 Cookie 等信息，只是使用。  （1）GET 类型的 CSRF：如通过设置<img>标签，通过在标签中嵌入恶意脚本，受害者一访问含有这个图片的页面后，浏览器就会自动向攻击网站发出一次HTTP请求。攻击网站就会收到包含受害者登录信息的一次跨域请求  （2）POST 类型的 CSRF：设置一种嵌入了恶意脚本的表单，访问该页面后，表单会自动提交，相当于模拟用户完成了一次 POST 操作  （3）链接类型的 CSRF，常见的往链接嵌入恶意代码，比如常见的网站里面一些美女服务、一夜暴富，点进去就中招了 | 1. Referer请求头，http请求里有一个referer请求头，带有这个请求的来源，验证这个请求是不是从自己的页面里来的，如果是的话才执行，否则就不要执行 2. 随机Tokens：通过响应页面时将 token 渲染到页面上，在 form 表单提交的时候通过隐藏域提交上来。 3. Cookie防御：利用 CSRF 攻击不能获取到用户 Cookie 的特点，我们可以要求 Ajax 和表单请求携带一个 Cookie 中的值   如将 token 设置在 Cookie 中，在提交 POST 请求的时候提交 Cookie，并通过 header 或者 body 带上 Cookie 中的 token，服务端进行对比校验。 |
| sql注入攻击、上传文件攻击、DDos攻击 |  |  |



1. 说一下你了解的数据结构、区别？





1. set、map、weakset、weakmap

set是一种类似于数组的叫做集合的数据结构，map是一种叫做字典的数据结构，二者都按顺序读取存储不重复值。

**Set**以 [value, value]的形式储存任意类型的值。向 Set 加入值的时候，不会发生类型转换，所以整型的1和字符串的“1”是不同的值，类似于严格等于“===”。**方法：**Add、delete、has、clear以及遍历迭代器keys()、values()、entries()等，不过键值与键名是一致的（或者说只有键值，没有键名），所以keys（）和values（）返回结果相同。**应用场景**：数组去重、遍历过滤、获取一些并集交集等

**Map**以[key, value] 的形式储存任何类型作为键，提供了“键—值”的一一对应关系。（1）如果对同一个键多次赋值，后面的值将覆盖前面的值，比如先给键1赋值A，再赋值B，查询键1结果会是B（2）如果读取一个未知的键，则返回 undefined（3）同样的值的两个实例，在 Map 结构中被视为两个键。即值都是A的情况下，对应的两个键在Map看来是不同的。

**WeakSet**将**弱引用对象**储存在一个集合中，但不能存放值。方法：add、has、delete;

**WeakMap** 是一组键值对的集合，其中的键是弱引用对象，而值可以是任意。WeakMap 弱引用的只是键名，而不是键值，键值依然是正常引用，但由于是弱引用，key 是不可枚举的。方法：set、has、get、delete

**二者都不能遍历。**

1. 弱引用、强引用、软引用、虚引用

**弱引用（Weak Reference）**：弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描到它所占据的内存区域时，发现只具有弱引用的对象，不管当前内存空间是否满了，都会回收它的内存。也就是说垃圾回收机制不考虑上面提到的 WeakSet和WeakMap 对对象的引用，如果没有其他的变量或属性引用这个对象值，则这个对象将会被垃圾回收掉。所以，WeakSet 对象里有多少个成员元素，取决于垃圾回收机制有没有运行，运行前后成员个数可能不一致，遍历结束之后，有的成员可能取不到了（被垃圾回收了），WeakSet 对象是无法被遍历的（ES6 规定 WeakSet 不可遍历），也没有办法拿到它包含的所有元素。

**强引用（Strong Reference）**：通常我们通过new来创建一个新对象时返回的引用就是一个强引用，若一个对象通过一系列强引用可到达，它就是强可达的(strongly reachable)，那么它就不被回收。

**软引用（Soft Reference）**：软引用和弱引用的区别在于，若一个对象是弱引用可达，无论当前内存是否充足它都会被回收，而软引用可达的对象在内存不充足时才会被回收，因此软引用要比弱引用“强”一些

**虚引用（Phantom Reference）**：虚引用是Java中最弱的引用，弱到以至于我们通过虚引用甚至无法获取到被引用的对象，虚引用存在的唯一作用就是当它指向的对象被回收后，虚引用本身会被加入到引用队列中，用作记录它指向的对象已被回收。