# 前后端交互

1. **Cookie 和 Seeion 区别**

**Cookie：**

1. 存储于客户端电脑上的文本文件中，只要客户端cookie开放且有数据，每一次请求都会自动 添加到http报文中，后台可以实时接收观察获取这些Cookie
2. **属性有：name=value，domain、path、maxAge、expires、secure、httpOnly**
3. **HTTP 是无状态的协议**
4. **Cookie 是不可跨域的**

**Session：**

1. **基于 cookie 实现的，session 存储在服务器端，sessionId 会被存储到客户端的cookie 中（当然也不是必须存储在cookie中）**
2. **客户端第一次访问，服务端记录，并生成唯一的标识sessionId，写回给浏览器；浏览器收到信息后，存储在对应的域下；当浏览器再次发送的时候自动判断是否在该域下携带cookie**

**区别：**

1. Session 比 Cookie 安全，Session 是存储在服务器端的，Cookie 是存储在客户端的
2. Cookie 只支持存字符串数据，想要设置其他类型的数据，需要将其转换成字符串，Session 可以存任意数据类型
3. Cookie 可设置为长时间保持，比如我们经常使用的默认登录功能，Session 一般失效时间较短，客户端关闭（默认情况下）或者 Session 超时都会失效
4. 单个 Cookie 保存的数据不能超过 4K，Session 可存储数据远高于 Cookie，但是当访问量过多，会占用过多的服务器资源
5. **原生的Ajax**
6. 生成XMLHttpRequest对象
7. 绑定响应函数onreadystatechange
8. 设置一些请求头的信息
9. 调用open方法，设置对应的请求方式，请求路径，是否异步
10. 调用send方法，如果包含请求体，请求体数据传入
11. **跨域**

**同源策略：**

是一个重要的安全策略，它用于限制一个[origin](https://developer.mozilla.org/en-US/docs/Glossary/Origin)的文档或者它加载的脚本如何能与另一个源的资源进行交互。它能帮助阻隔恶意文档，减少可能被攻击，协议+端口号+ip地址

跨域的方式：

1. JSONP 的原理很简单，就是利用script标签没有跨域限制的漏洞，但是只能用于get请求
2. CORS需要浏览器和后端同时配合使用；服务端设置 Access-Control-Allow-Origin 就可以开启 CORS。 该属性表示哪些域名可以访问资源， 如果设置通配符则表示所有网站都可以访问资源；Access-Control-Allow-Credentials: true 和 Access-Control-Allow-Origin : ‘\*’ 冲突，浏览器会拦截响应信息，这样服务器无法通过cookie记录客户端
3. Document.domain 该方式只能用于二级域名相同的情况下，比如 a.test.com 和 b.test.com 适用于该方式。 只需要给页面添加 document.domain = 'test.com' 表示二级域名都相同就可以实现跨域了
4. postMessage 通常用于获取嵌入页面中的第三方页面数据。一个页面发送消息，另一个页面判断来源并接收消息。
5. iframe的跨域
6. **浏览器输入url发生了什么**
7. DNS域名解析，找到服务器地址
8. 通过地址和服务器建立TCP连接（如果是https还会去建立TLS连接），开始传输数据
9. 一个GET网络请求，根据URL（统一资源符）描述，返回相应报文和资源
10. 浏览器开始解析文件，如果是 gzip 格式的话会先解压一下，然后通过文件的编码格式知道该如何去解 码文件
11. 解码文件开始浏览器渲染进程开始工作，完成页面展示部分
12. TCP断开连接（四次挥手）
13. **浏览器的存储方式**

**cookie，localStorage，sessionStorage，indexDB 四种：**

* + 1. Cookie 一般由服务器设置决定事件长短，4K，每次请求带上cookie
    2. localStorage 需要手动清除时间不限，5M
    3. sessionStorage 关闭网页就清除，5M
    4. indexDB 需要手动清除时间不限，大小不限，提供查找接口，还能建立索引

它们都有同源策略限制

**localStorage和sessionStorage的区别：**

出于性能考虑，如果没有大量数据存储需求的话，可以使用 localStorage 和 sessionStorage 。对于不怎么改变的数据尽量使用 localStorage 存储，否则可以用 sessionStorage 存储

Cookie如何设置：

Cookie直接设置name和value，如name=value，如果还需要设置其它属性，那么加上“;”号，设置expires=过期时间

1. **浏览器的缓存机制**

缓存可以说是性能优化中简单高效的一种优化方式了，它可以显著减少网络传输所带来的损耗。

**缓存的位置（放在哪）（优先级）：**

1. Service Worker。它的缓存与浏览器其他内建的缓存机制不同，它可以让我们自由控制缓存哪些文件、如何匹配缓存、如何读取缓存，并且缓存是持续性的。 当 Service Worker 没有命中缓存的时候，我们需要去调用 fetch 函数获取数据。也就是说，如 果我们没有在 Service Worker 命中缓存的话，会根据缓存查找优先级去查找数据。但是不管我们 是从 Memory Cache 中还是从网络请求中获取的数据，浏览器都会显示我们是从 Service Worker 中获取的内容
2. Memory Cache也就是内存中的缓存，读取内存中的数据肯定比磁盘快。内存缓存虽然读取高 效，可是缓存持续性很短，会随着进程的释放而释放。 一旦我们关闭 Tab 页面，内存中的缓存也 就被释放了
3. Disk Cache是存储在硬盘中的缓存，读取速度慢点，但是什么都能存储到磁盘中，比之 Memory Cache 胜在容量和存储时效性上。 在所有浏览器缓存中，Disk Cache 覆盖面基本是最大的。它会根据 HTTP Herder 中的字段判断哪 些资源需要缓存，哪些资源可以不请求直接使用，哪些资源已经过期需要重新请求。并且即使在跨 站点的情况下，相同地址的资源一旦被硬盘缓存下来，就不会再次去请求数据
4. Push Cache是 HTTP/2 中的内容，当以上三种缓存都没有命中时，它才会被使用。并且缓存时间 也很短暂，只在会话（Session）中存在，一旦会话结束就被释放
5. 网络请求 如果所有缓存都没有命中的话，那么只能发起请求来获取资源了

**缓存策略：**

1. 强缓存：
   1. Expires 是http1.0内容，Expires受限于本地时间，如果修改了本地时间，可能会造成缓存失效
   2. Cache-control http1.1的内容，优先级比Expires高，可以在请求头或者响应头中设置，并且可以组合使用 多种指令
2. 协商缓存
   1. Last-Modified 和 If-Modified-Since Last-Modified 表示文件最后修改日期， If-Modified-Since 会将 Last-Modified 的值 发送给服务器，询问服务器在该日期后资源是否有更新，有更新的话就会将新的资源发送回 来，否则返回 304 状态码。缺点：1.如果本地打开缓存文件，即使没有对文件进行修改，但还是会造成 Last-Modified 被修改；2. Last-Modified 只能以秒计时，如果在不可感知的时间内修改完成文件，那么服 务端会认为资源还是命中了，不会返回正确的资源
   2. ETag 和 If-None-Match ETag 类似于文件指纹， If-None-Match 会将当前 ETag 发送给服务器，询问该资源 ETag 是否变动，有变动的话就将新的资源发送回来。并且 ETag 优先级比 Last-Modified 高