2022 最新版前端工程师面试题手册

目录

1 | 前端基础

1.1 | HTTP/HTML/浏览器

- 说一下 http 和 https

参考回答：
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。

- tcp 三次握手，一句话概括

参考回答：
客户端和服务端都需要直到各自可收发，因此需要三次握手。
简化三次握手：
<img width="487" alt="2018-07-10 3 42 11" src="https://user-images.githubusercontent.com/17233651/42496289-1c6d668a-8458-11e8-98b3-65db50f64d48.png">
从图片可以得到三次握手可以简化为：C 发起请求连接 S 确认，也发起连接 C 确认我们再看看每次握手的作用：第一次握手：S 只可以确认 自己可以接受 C 发送的报文段第二次握手：C 可以确认 S 收到了自己发送的报文段，并且可以确认 自己可以接受 S 发送的报文段第三次握手：S 可以确认 C 收到了自己发送的报文段

- TCP 和 UDP 的区别

参考回答：
（1）TCP 是面向连接的，udp 是无连接的即发送数据前不需要先建立链接。
（2）TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP 尽最大努力交付，即不保证可靠交付。 并且因为 tcp 可靠，面向连接，不会丢失数据因此适合大数据量的交换。
（3）TCP 是面向字节流，UDP 面向报文，并且网络出现拥塞不会使得发送速率降低（因此会出现丢包，对实时的应用比如 IP 电话和视频会议等）。
（4）TCP 只能是 1 对 1 的，UDP 支持 1 对 1,1 对多。
（5）TCP 的首部较大为 20 字节，而 UDP 只有 8 字节。
（6）TCP 是面向连接的可靠性传输，而 UDP 是不可靠的。

- WebSocket 的实现和应用

参考回答：

(1)什么是 WebSocket?

WebSocket 是 HTML5 中的协议，支持持久连续，http 协议不支持持久性连接。Http1.0 和 HTTP1.1 都不支持持久性的链接，HTTP1.1 中的 keep-alive，将多个 http 请求合并为 1 个

(2)WebSocket 是什么样的协议，具体有什么优点？

HTTP 的生命周期通过 Request 来界定，也就是 Request 一个 Response，那么在 Http1.0 协议中，这次 Http 请求就结束了。在 Http1.1 中进行了改进，是的有一个 connection：Keep-alive，也就是说，在一个 Http 连接中，可以发送多个 Request，接收多个 Response。但是必须记住，在 Http 中一个 Request 只能对应有一个 Response，而且这个 Response 是被动的，不能主动发起。

WebSocket 是基于 Http 协议的，或者说借用了 Http 协议来完成一部分握手，在握手阶段与 Http 是相同的。我们来看一个 websocket 握手协议的实现，基本是 2 个属性，upgrade，connection。

基本请求如下：

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

多了下面 2 个属性：

```
1Upgrade:webSocket
2Connection:Upgrade
```

告诉服务器发送的是 websocket

```
1Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
2Sec-WebSocket-Protocol: chat, superchat
3Sec-WebSocket-Version:  13
```

- **HTTP 请求的方式，HEAD 方式**

参考回答：

head：类似于 get 请求，只不过返回的响应中没有具体的内容，用户获取报头

options：允许客户端查看服务器的性能，比如说服务器支持的请求方式等等。

- 一个图片 url 访问后直接下载怎样实现？

参考回答：
请求的返回头里面，用于浏览器解析的重要参数就是 OSS 的 API 文档里面的返回 http 头，决定用户下载行为的参数。
下载的情况下：
1. x-oss-object-type:
Normal
2. x-oss-request-id:
598D5ED34F29D01FE2925F41
3. x-oss-storage-class:
Standard

- 说一下 web Quality（无障碍）

参考回答：
能够被残障人士使用的网站才能称得上一个易用的（易访问的）网站。
残障人士指的是那些带有残疾或者身体不健康的用户。
使用 alt 属性：

有时候浏览器会无法显示图像。具体的原因有：
用户关闭了图像显示
浏览器是不支持图形显示的迷你浏览器
浏览器是语音浏览器（供盲人和弱视人群使用）
如果您使用了 alt 属性，那么浏览器至少可以显示或读出有关图像的描述。

- 几个很实用的 BOM 属性对象方法?

参考回答：
什么是 Bom? Bom 是浏览器对象。有哪些常用的 Bom 属性呢？
(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

- 说一下 HTML5 drag api

参考回答：
dragstart：事件主体是被拖放元素，在开始拖放被拖放元素时触发，。
darg：事件主体是被拖放元素，在正在拖放被拖放元素时触发。
dragenter：事件主体是目标元素，在被拖放元素进入某元素时触发。
dragover：事件主体是目标元素，在被拖放在某元素内移动时触发。
dragleave：事件主体是目标元素，在被拖放元素移出目标元素是触发。
drop：事件主体是目标元素，在目标元素完全接受被拖放元素时触发。
dragend：事件主体是被拖放元素，在整个拖放操作结束时触发

- 说一下 http2.0

参考回答：
首先补充一下，http 和 https 的区别，相比于 http,https 是基于 ssl 加密的 http 协议
简要概括：http2.0 是基于 1999 年发布的 http1.0 之后的首次更新。
提升访问速度（可以对于，请求资源所需时间更少，访问速度更快，相比 http1.0）
允许多路复用：多路复用允许同时通过单一的 HTTP/2 连接发送多重请求-响应信息。
改善了：在 http1.1 中，浏览器客户端在同一时间，针对同一域名下的请求有一定数量限制（连接数量），超过限制会被阻塞。
二进制分帧：HTTP2.0 会将所有的传输信息分割为更小的信息或者帧，并对他们进行二进制编码
首部压缩
服务器端推送

- 补充 400 和 401、403 状态码

参考回答：
(1)400 状态码：请求无效
产生原因：
前端提交数据的字段名称和字段类型与后台的实体没有保持一致
前端提交到后台的数据应该是 json 字符串类型，但是前端没有将对象 JSON.stringify 转化成字符串。
解决方法：
对照字段的名称，保持一致性
将 obj 对象通过 JSON.stringify 实现序列化
(2)401 状态码：当前请求需要用户验证
(3)403 状态码：服务器已经得到请求，但是拒绝执行

- fetch 发送 2 次请求的原因

参考回答：
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。

- Cookie、sessionStorage、localStorage 的区别

参考回答：
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。

- 说一下 web worker

参考回答：
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

- 对 HTML 语义化标签的理解

参考回答：
HTML5 语义化标签是指正确的标签包含了正确的内容，结构良好，便于阅读，比如 nav 表示导航条，类似的还有 article、header、footer 等等标签。

- iframe 是什么？有什么缺点？

参考回答：
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。

- Doctype 作用?严格模式与混杂模式如何区分？它们有何意义?

参考回答：

Doctype 声明于文档最前面，告诉浏览器以何种方式来渲染页面，这里有两种模式，严格模式和混杂模式。
严格模式的排版和 JS 运作模式是 以该浏览器支持的最高标准运行。
混杂模式，向后兼容，模拟老式浏览器，防止浏览器无法兼容页面。

- Cookie 如何防范 XSS 攻击

参考回答：
XSS（跨站脚本攻击）是指攻击者在返回的 HTML 中嵌入 javascript 脚本，为了减轻这些攻击，需要在 HTTP 头部配上，set-cookie：
httponly-这个属性可以防止 XSS,它会禁止 javascript 脚本来访问 cookie。
secure - 这个属性告诉浏览器仅在请求为 https 的时候发送 cookie。
结果应该是这样的：Set-Cookie=<cookie-value>.....

- Cookie 和 session 的区别

参考回答：
HTTP 是一个无状态协议，因此 Cookie 的最大的作用就是存储 sessionId 用来唯一标识用户。

- 一句话概括 RESTFUL

参考回答：
就是用 URL 定位资源，用 HTTP 描述操作。

- 讲讲 viewport 和移动端布局

参考回答：
可以参考这篇文章：
响应式布局的常用解决方案对比(媒体查询、百分比、rem 和 vw/vh）

- click 在 ios 上有 300ms 延迟，原因及如何解决？

参考回答：
(1)粗暴型，禁用缩放
<meta name="viewport" content="width=device-width, user-scalable=no">
(2)利用 FastClick，其原理是：
检测到 touchend 事件后，立刻出发模拟 click 事件，并且把浏览器 300 毫秒之后真正出发的事件给阻断掉

- addEventListener 参数

参考回答：
addEventListener(event, function, useCapture)
其中，event 指定事件名；function 指定要事件触发时执行的函数；useCapture 指定事件是否在捕获或冒泡阶段执行。

- cookie sessionStorage localStorage 区别

参考回答：
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
cookie 数据还有路径（path）的概念，可以限制。cookie 只属于某个路径下
存储大小限制也不同，cookie 数据不能超过 4K，同时因为每次 http 请求都会携带 cookie，所以 cookie 只适合保存很小的数据，如回话标识。
webStorage 虽然也有存储大小的限制，但是比 cookie 大得多，可以达到 5M 或更大
数据的有效期不同 sessionStorage：仅在当前的浏览器窗口关闭有效；
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，即使窗口和浏览器关闭
作用域不同 sessionStorage：不在不同的浏览器窗口中共享，即使是同一个页面；
localStorage：在所有同源窗口都是共享的；cookie：也是在所有同源窗口中共享的

- cookie session 区别

参考回答：
1. cookie 数据存放在客户的浏览器上，session 数据放在服务器上。
2. cookie 不是很安全，别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗
考虑到安全应当使用 session。
3. session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能
考虑到减轻服务器性能方面，应当使用 COOKIE。
4. 单个 cookie 保存的数据不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie。

- 介绍知道的 http 返回的状态码

参考回答：
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
503 Service Unavailable 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的 Retry-After 头信息中
504 Gateway Time-out 充当网关或代理的服务器，未及时从远端服务器获取请求
505 HTTP Version not supported 服务器不支持请求的 HTTP 协议的版本，无法完成处理

- **http 常用请求头**

参考回答：

协议头	说明
Accept	可接受的响应内容类型（Content-Types）。
Accept-Charset	可接受的字符集
Accept-Encoding	可接受的响应内容的编码方式。
Accept-Language	可接受的响应内容语言列表。

Accept-Datetime	可接受的按照时间来表示的响应内容版本
Authorization	用于表示 HTTP 协议中需要认证资源的认证信息
Cache-Control	用来指定当前的请求/回复中的，是否使用缓存机制。
Connection	客户端（浏览器）想要优先使用的连接类型
Cookie	由之前服务器通过 Set-Cookie（见下文）设置的一个 HTTP 协议 Cookie
Content-Length	以 8 进制表示的请求体的长度
Content-MD5	请求体的内容的二进制 MD5 散列值（数字签名），以 Base64 编码的结果
Content-Type	请求体的 MIME 类型 （用于 POST 和 PUT 请求中）
Date	发送该消息的日期和时间（以 RFC 7231 中定义的"HTTP 日期"格式来发送）
Expect	表示客户端要求服务器做出特定的行为
From	发起此请求的用户的邮件地址

Host	表示服务器的域名以及服务器所监听的端口号。如果所请求的端口是对应的服务的标准端口（80），则端口号可以省略。
If-Match	仅当客户端提供的实体与服务器上对应的实体相匹配时，才进行对应的操作。主要用于像 PUT 这样的方法中，仅当从用户上次更新某个资源后，该资源未被修改的情况下，才更新该资源。
If-Modified-Since	允许在对应的资源未被修改的情况下返回 304 未修改
If-None-Match	允许在对应的内容未被修改的情况下返回 304 未修改（ 304 Not Modified ），参考 超文本传输协议 的实体标记
If-Range	如果该实体未被修改过，则向返回所缺少的那一个或多个部分。否则，返回整个新的实体
If-Unmodified-Since	仅当该实体自某个特定时间以来未被修改的情况下，才发送回应。
Max-Forwards	限制该消息可被代理及网关转发的次数。
Origin	发起一个针对跨域资源共享的请求（该请求要求服务器在响应中加入一个 Access-Control-Allow-Origin 的消息头，表示访问控制所允许的来源）。
Pragma	与具体的实现相关，这些字段可能在请求/回应链中的任何时候产生。

Proxy-Authorization	用于向代理进行认证的认证信息。
Range	表示请求某个实体的一部分，字节偏移以 0 开始。
Referer	表示浏览器所访问的前一个页面，可以认为是之前访问页面的链接将浏览器带到了当前页面。Referer 其实是 Referrer 这个单词，但 RFC 制作标准时给拼错了，后来也就将错就错使用 Referer 了。
TE	浏览器预期接受的传输时的编码方式：可使用回应协议头 Transfer-Encoding 中的值（还可以使用"trailers"表示数据传输时的分块方式）用来表示浏览器希望在最后一个大小为 0 的块之后还接收到一些额外的字段。
User-Agent	浏览器的身份标识字符串
Upgrade	要求服务器升级到一个高版本协议。
Via	告诉服务器，这个请求是由哪些代理发出的。
Warning	一个一般性的警告，表示在实体内容体中可能存在错误。

- **强，协商缓存**

参考回答：
缓存分为两种：强缓存和协商缓存，根据响应的 header 内容来决定。

	获取资源形式	状态码	发送请求到服务器

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

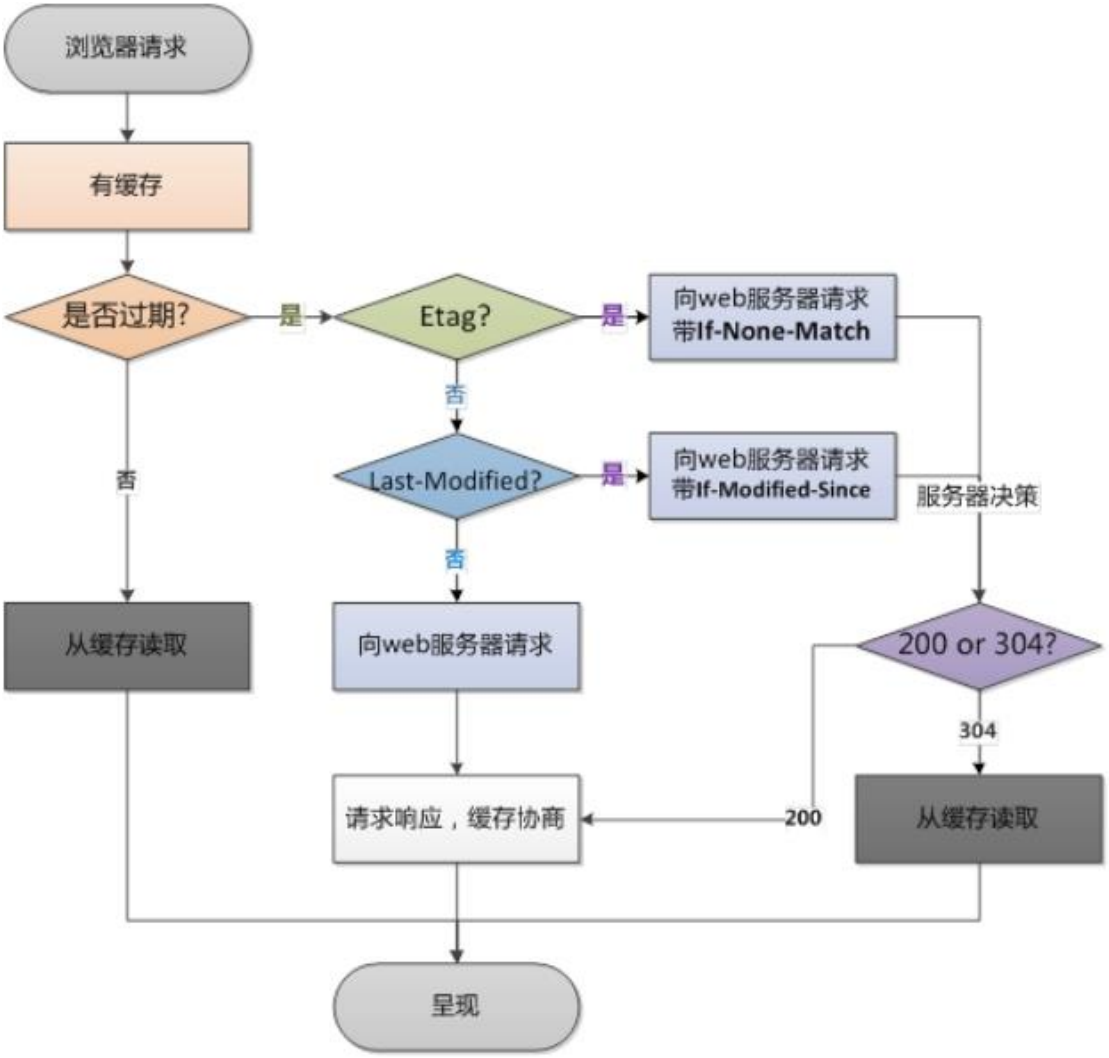
强缓存相关字段有 expires，cache-control。如果 cache-control 与 expires 同时存在的话，cache-control 的优先级高于 expires。
协商缓存相关字段有 Last-Modified/If-Modified-Since，Etag/If-None-Match

- 讲讲 304

参考回答：
304：如果客户端发送了一个带条件的 GET 请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个 304 状态码。

- 强缓存、协商缓存什么时候用哪个

参考回答：
因为服务器上的资源不是一直固定不变的，大多数情况下它会更新，这个时候如果我们还访问本地缓存，那么对用户来说，那就相当于资源没有更新，用户看到的还是旧的资源；所以我们希望服务器上的资源更新了浏览器就请求新的资源，没有更新就使用本地的缓存，以最大程度的减少因网络请求而产生的资源浪费。



参考 https://segmentfault.com/a/1190000008956069

- 前端优化

参考回答：
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。

- GET 和 POST 的区别

参考回答：
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。

- 301 和 302 的区别

参考回答：
301 Moved Permanently 被请求的资源已永久移动到新位置，并且将来任何对此资源的引用都应该使用本响应返回的若干个 URI 之一。如果可能，拥有链接编辑功能的客户端应当自动把请求的地址修改为从服务器反馈回来的地址。除非额外指定，否则这个响应也是可缓存的。
302 Found 请求的资源现在临时从不同的 URI 响应请求。由于这样的重定向是临时的，客户端应当继续向原有地址发送以后的请求。只有在 Cache-Control 或 Expires 中进行了指定的情况下，这个响应才是可缓存的。
字面上的区别就是 301 是永久重定向，而 302 是临时重定向。
301 比较常用的场景是使用域名跳转。302 用来做临时跳转 比如未登陆的用户访问用户中心重定向到登录页面。

- HTTP 支持的方法

参考回答：

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

- 如何画一个三角形

参考回答：
三角形原理：边框的均分原理

```
div {
width:0px;
height:0px;
border-top:10px solid red;
border-right:10px solid transparent;
border-bottom:10px solid transparent;
border-left:10px solid transparent;
}
```

- 状态码 304 和 200

参考回答：
状态码 200：请求已成功，请求所希望的响应头或数据体将随此响应返回。即返回的数据为全量的数据，如果文件不通过 GZIP 压缩的话，文件是多大，则要有多大传输量。
状态码 304：如果客户端发送了一个带条件的 GET 请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个状态码。即客户端和服务器端只需要传输很少的数据量来做文件的校验，如果文件没有修改过，则不需要返回全量的数据。

- **说一下浏览器缓存**

参考回答：
缓存分为两种：强缓存和协商缓存，根据响应的 header 内容来决定。
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

- **HTML5 新增的元素**

参考回答：
首先 html5 为了更好的实践 web 语义化，增加了 header，footer，nav,aside,section 等语义化标签，在表单方面，为了增强表单，为 input 增加了 color，emial,data ,range 等类型，在存储方面，提供了 sessionStorage，localStorage,和离线存储，通过这些存储方式方便数据在客户端的存储和获取，在多媒体方面规定了音频和视频元素 audio 和 vedio，另外还有地理定位，canvas 画布，拖放，多线程编程的 web worker 和 websocket 协议。

- **在地址栏里输入一个 URL,到这个页面呈现出来，中间会发生什么？**

参考回答：
这是一个必考的面试问题，
输入 url 后，首先需要找到这个 url 域名的服务器 ip,为了寻找这个 ip，浏览器首先会寻找缓存，查看缓存中是否有记录，缓存的查找记录为：浏览器缓存-》系统缓存-》路由器缓存，缓存中没有则查找系统的 hosts 文件中是否有记录，如果没有则查询 DNS 服务器，得到服务器的 ip 地址后，浏览器根据这个 ip 以及相应的端口号，构造一个 http 请求，这个请求报文会包括这次请求的信息，主要是请求方法，请求说明和请求附带的数据，并将这个 http 请求封装在一个 tcp 包中，这个 tcp 包会依次经过传输层，网络层，数据链路层，物理层到达服务器，服务器解析这个请求来作出响应，返回相应的 html 给浏览器，因为 html 是一个树形结构，浏览器根据这个 html 来构建 DOM 树，在 dom 树的构建过程中如果遇到 JS 脚本和外部 JS 连接，则会停止构建 DOM 树来执行和下载相应的代码，这会造成阻塞，这就是为什么推荐 JS 代码应该放在 html 代码的后面，之后根据外部央视，内部央视，内联样式构建一个 CSS 对象模型树 CSSOM 树，构建完成后和 DOM 树合并为渲染树，这里主要做的是排除非视觉节点，比如

script，meta 标签和排除 display 为 none 的节点，之后进行布局，布局主要是确定各个元素的位置和尺寸，之后是渲染页面，因为 html 文件中会含有图片，视频，音频等资源，在解析 DOM 的过程中，遇到这些都会进行并行下载，浏览器对每个域的并行下载数量有一定的限制，一般是 4-6 个，当然在这些所有的请求中我们还需要关注的就是缓存，缓存一般通过 Cache-Control、Last-Modify、Expires 等首部字段控制。Cache-Control 和 Expires 的区别在于 Cache-Control 使用相对时间，Expires 使用的是基于服务器 端的绝对时间，因为存在时差问题，一般采用 Cache-Control，在请求这些有设置了缓存的数据时，会先 查看是否过期，如果没有过期则直接使用本地缓存，过期则请求并在服务器校验文件是否修改，如果上一次 响应设置了 ETag 值会在这次请求的时候作为 If-None-Match 的值交给服务器校验，如果一致，继续校验 Last-Modified，没有设置 ETag 则直接验证 Last-Modified，再决定是否返回 304。

- cookie 和 session 的区别，localstorage 和 sessionstorage 的区别

参考回答：
Cookie 和 session 都可用来存储用户信息，cookie 存放于客户端，session 存放于服务器端，因为 cookie 存放于客户端有可能被窃取，所以 cookie 一般用来存放不敏感的信息，比如用户设置的网站主题，敏感的信息用 session 存储，比如用户的登陆信息，session 可以存放于文件，数据库，内存中都可以，cookie 可以服务器端响应的时候设置，也可以客户端通过 JS 设置 cookie 会在请求时在 http 首部发送给客户端，cookie 一般在客户端有大小限制，一般为 4K，
下面从几个方向区分一下 cookie，localstorage，sessionstorage 的区别
1、生命周期：
Cookie：可设置失效时间，否则默认为关闭浏览器后失效
Localstorage:除非被手动清除，否则永久保存
Sessionstorage：仅在当前网页会话下有效，关闭页面或浏览器后就会被清除
2、存放数据：
Cookie：4k 左右
Localstorage 和 sessionstorage：可以保存 5M 的信息
3、http 请求：
Cookie：每次都会携带在 http 头中，如果使用 cookie 保存过多数据会带来性能问题
其他两个：仅在客户端即浏览器中保存，不参与和服务器的通信
4、易用性：
Cookie：需要程序员自己封装，原生的 cookie 接口不友好
其他两个：即可采用原生接口，亦可再次封装
5、应用场景：
从安全性来说，因为每次 http 请求都回携带 cookie 信息，这样子浪费了带宽，所以 cookie 应该尽可能的少用，此外 cookie 还需要指定作用域，不可以跨域调用，限制很多，但是用户识别用户登陆来说，cookie 还是比 storage 好用，其他情况下可以用 storage，localstorage 可以用来在页面传递参数，sessionstorage 可以用来保存一些临时的数据，防止用户刷新页面后丢失了一些参数。

- 常见的 HTTP 的头部

参考回答：
可以将 http 首部分为通用首部，请求首部，响应首部，实体首部
通用首部表示一些通用信息，比如 date 表示报文创建时间，
请求首部就是请求报文中独有的，如 cookie，和缓存相关的如 if-Modified-Since
响应首部就是响应报文中独有的，如 set-cookie，和重定向相关的 location，
实体首部用来描述实体部分，如 allow 用来描述可执行的请求方法，content-type 描述主题类型，content-Encoding 描述主体的编码方式。

- HTTP2.0 的特性

参考回答：
http2.0 的特性如下：
1、内容安全，应为 http2.0 是基于 https 的，天然具有安全特性，通过 http2.0 的特性可以避免单纯使用 https 的性能下降
2、二进制格式，http1.X 的解析是基于文本的，http2.0 将所有的传输信息分割为更小的消息和帧，并对他们采用二进制格式编码，基于二进制可以让协议有更多的扩展性，比如引入了帧来传输数据和指令
3、多路复用，这个功能相当于是长连接的增强，每个 request 请求可以随机的混杂在一起，接收方可以根据 request 的 id 将 request 再归属到各自不同的服务端请求里面，另外多路复用中也支持了流的优先级，允许客户端告诉服务器那些内容是更优先级的资源，可以优先传输。

- cache-control 的值有哪些

参考回答：
cache-control 是一个通用消息头字段被用于 HTTP 请求和响应中，通过指定指令来实现缓存机制，这个缓存指令是单向的，常见的取值有 private、no-cache、max-age、must-revalidate 等，默认为 private。

- 浏览器在生成页面的时候，会生成那两颗树？

参考回答：
构造两棵树，DOM 树和 CSSOM 规则树，
当浏览器接收到服务器相应来的 HTML 文档后，会遍历文档节点，生成 DOM 树，
CSSOM 规则树由浏览器解析 CSS 文件生成。

- csrf 和 xss 的网络攻击及防范

参考回答：
CSRF：跨站请求伪造，可以理解为攻击者盗用了用户的身份，以用户的名义发送了恶意请求，比如用户登录了一个网站后，立刻在另一个 t a b 页面访问量攻击者用来制

造攻击的网站，这个网站要求访问刚刚登陆的网站，并发送了一个恶意请求，这时候 CSRF 就产生了，比如这个制造攻击的网站使用一张图片，但是这种图片的链接却是可以修改数据库的，这时候攻击者就可以以用户的名义操作这个数据库，防御方式的话：使用验证码，检查 https 头部的 refer，使用 token
XSS：跨站脚本攻击，是说攻击者通过注入恶意的脚本，在用户浏览网页的时候进行攻击，比如获取 cookie，或者其他用户身份信息，可以分为存储型和反射型，存储型是攻击者输入一些数据并且存储到了数据库中，其他浏览者看到的时候进行攻击，反射型的话不存储在数据库中，往往表现为将攻击代码放在 url 地址的请求参数中，防御的话为 cookie 设置 httpOnly 属性，对用户的输入进行检查，进行特殊字符过滤。

- **怎么看网站的性能如何**

参考回答：
检测页面加载时间一般有两种方式，一种是被动去测：就是在被检测的页面置入脚本或探针，当用户访问网页时，探针自动采集数据并传回数据库进行分析，另一种主动监测的方式，即主动的搭建分布式受控环境，模拟用户发起页面访问请求，主动采集性能数据并分析，在检测的精准度上，专业的第三方工具效果更佳，比如说性能极客。

- **介绍 HTTP 协议(特征)**

参考回答：
HTTP 是一个基于 TCP/IP 通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于 1990 年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在 WWW 中使用的是 HTTP/1.0 的第六版，HTTP/1.1 的规范化工作正在进行之中，而且 HTTP-NG(Next Generation of HTTP)的建议已经提出。HTTP 协议工作于客户端-服务端架构为上。浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端即 WEB 服务器发送所有请求。Web 服务器根据接收到的请求后，向客户端发送响应信息。

- **输入 URL 到页面加载显示完成发生了什么？**

参考回答：
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

- **说一下对 Cookie 和 Session 的认知，Cookie 有哪些限制？**

参考回答：

1. cookie 数据存放在客户的浏览器上，session 数据放在服务器上。
2. cookie 不是很安全，别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗

考虑到安全应当使用 session。

3. session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能

考虑到减轻服务器性能方面，应当使用 COOKIE。

4. 单个 cookie 保存的数据不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie。

- **描述一下 XSS 和 CRSF 攻击？防御方法？**

参考回答：

XSS, 即为（Cross Site Scripting）, 中文名为跨站脚本, 是发生在目标用户的浏览器层面上的，当渲染 DOM 树的过程成发生了不在预期内执行的 JS 代码时，就发生了 XSS 攻击。大多数 XSS 攻击的主要方式是嵌入一段远程或者第三方域上的 JS 代码。实际上是在目标网站的作用域下执行了这段 JS 代码。

CSRF（Cross Site Request Forgery，跨站请求伪造），字面理解意思就是在别的站点伪造了一个请求。专业术语来说就是在受害者访问一个网站时，其 Cookie 还没有过期的情况下，攻击者伪造一个链接地址发送受害者并欺骗让其点击，从而形成 CSRF 攻击。

XSS 防御的总体思路是：对输入(和 URL 参数)进行过滤，对输出进行编码。也就是对提交的所有内容进行过滤，对 url 中的参数进行过滤，过滤掉会导致脚本执行的相关内容；然后对动态输出到页面的内容进行 html 编码，使脚本无法在浏览器中执行。虽然对输入过滤可以被绕过，但是也还是会拦截很大一部分的 XSS 攻击。

防御 CSRF 攻击主要有三种策略：验证 HTTP Referer 字段；在请求地址中添加 token 并验证；在 HTTP 头中自定义属性并验证。

- **知道 304 吗，什么时候用 304？**

参考回答：

304：如果客户端发送了一个带条件的 GET 请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个 304 状态码。

- **具体有哪些请求头是跟缓存相关的**

参考回答：

缓存分为两种：强缓存和协商缓存，根据响应的 header 内容来决定。

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

- cookie 和 session 的区别

参考回答：

1. cookie 数据存放在客户的浏览器上，session 数据放在服务器上。
2. cookie 不是很安全，别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗
考虑到安全应当使用 session。
3. session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能
考虑到减轻服务器性能方面，应当使用 COOKIE。
4. 单个 cookie 保存的数据不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie。

- cookie 有哪些字段可以设置

参考回答：

name 字段为一个 cookie 的名称。
value 字段为一个 cookie 的值。
domain 字段为可以访问此 cookie 的域名。
非顶级域名，如二级域名或者三级域名，设置的 cookie 的 domain 只能为顶级域名或者二级域名或者三级域名本身，不能设置其他二级域名的 cookie，否则 cookie 无法生成。
顶级域名只能设置 domain 为顶级域名，不能设置为二级域名或者三级域名，否则 cookie 无法生成。
二级域名能读取设置了 domain 为顶级域名或者自身的 cookie，不能读取其他二级域名 domain 的 cookie。所以要想 cookie 在多个二级域名中共享，需要设置 domain 为顶级域名，这样就可以在所有二级域名里面或者到这个 cookie 的值了。
顶级域名只能获取到 domain 设置为顶级域名的 cookie，其他 domain 设置为二级域名的无法获取。
path 字段为可以访问此 cookie 的页面路径。 比如 domain 是 abc.com,path 是/test，那么只有/test 路径下的页面可以读取此 cookie。
expires/Max-Age 字段为此 cookie 超时时间。若设置其值为一个时间，那么当到达此时间后，此 cookie 失效。不设置的话默认值是 Session，意思是 cookie 会和 session 一起失效。当浏览器关闭(不是浏览器标签页，而是整个浏览器) 后，此 cookie 失效。
Size 字段 此 cookie 大小。
http 字段 cookie 的 httponly 属性。若此属性为 true，则只有在 http 请求头中会带有此 cookie 的信息，而不能通过 document.cookie 来访问此 cookie。
secure 字段 设置是否只能通过 https 来传递此条 cookie

- cookie 有哪些编码方式？

参考回答：
encodeURI（）

- 除了 cookie，还有什么存储方式。说说 cookie 和 localStorage 的区别

参考回答：
还有 localStorage，sessionStorage，indexdDB 等
cookie 和 localStorage 的区别：
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
cookie 数据还有路径（path）的概念，可以限制。cookie 只属于某个路径下
存储大小限制也不同，cookie 数据不能超过 4K，同时因为每次 http 请求都会携带 cookie，所以 cookie 只适合保存很小的数据，如回话标识。
localStorage 虽然也有存储大小的限制，但是比 cookie 大得多，可以达到 5M 或更大
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，即使窗口和浏览器关闭。

- 浏览器输入网址到页面渲染全过程

参考回答：
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

- HTML5 和 CSS3 用的多吗？你了解它们的新属性吗？有在项目中用过吗？

参考回答：
html5：
1）标签增删
8 个语义元素 header section footer aside nav main article figure
内容元素 mark 高亮 progress 进度
新的表单控件 calander date time email url search
新的 input 类型 color date datetime datetime-local email
移除过时标签 big font frame frameset
2）canvas 绘图，支持内联 SVG。支持 MathML
3）多媒体 audio video source embed track
4）本地离线存储，把需要离线存储在本地的文件列在一个 manifest 配置文件

5）web 存储。localStorage、SessionStorage

css3：
CSS3 边框如 border-radius，box-shadow 等；CSS3 背景如 background-size，background-origin 等；CSS3 2D，3D 转换如 transform 等；CSS3 动画如 animation 等。 参考 https://www.cnblogs.com/xkweb/p/5862612.html

- **http 常见的请求方法**

参考回答：
get、post，这两个用的是最多的，还有很多比如 patch、delete、put、options 等等

- **get 和 post 的区别**

参考回答：
GET - 从指定的资源请求数据。
POST - 向指定的资源提交要被处理的数据。
GET：不同的浏览器和服务器不同，一般限制在 2~8K 之间，更加常见的是 1k 以内。
GET 和 POST 的底层也是 TCP/IP，GET/POST 都是 TCP 链接。
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）。

- **说说 302，301，304 的状态码**

参考回答：
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
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

- **web 性能优化**

参考回答：
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。

- 浏览器缓存机制

参考回答：
缓存分为两种：强缓存和协商缓存，根据响应的 header 内容来决定。
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

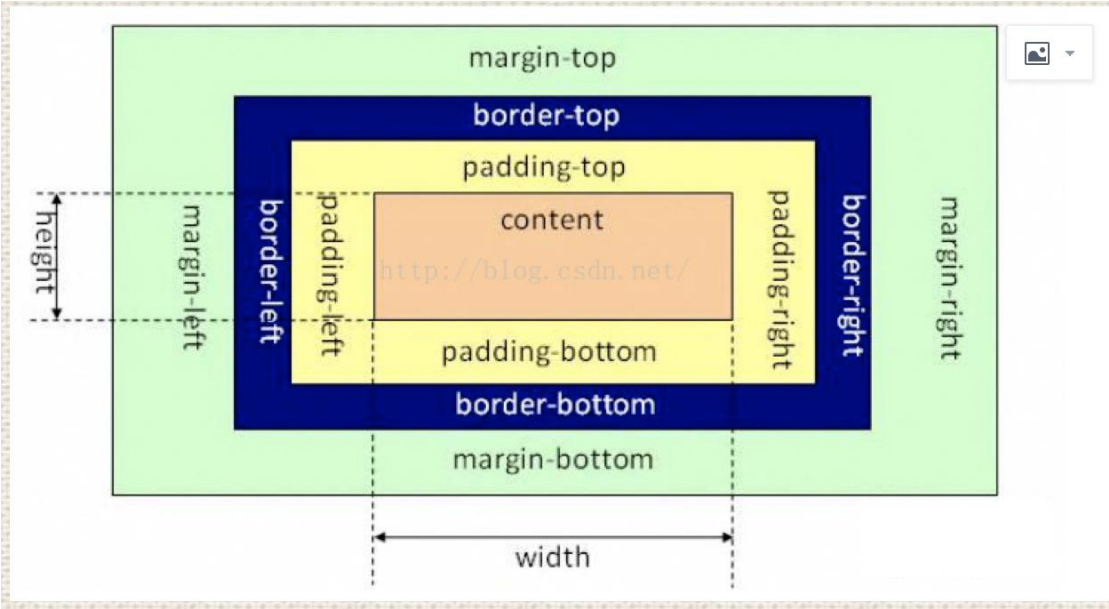
- post 和 get 区别

参考回答：
GET - 从指定的资源请求数据。
POST - 向指定的资源提交要被处理的数据。
GET：不同的浏览器和服务器不同，一般限制在 2~8K 之间，更加常见的是 1k 以内。
GET 和 POST 的底层也是 TCP/IP，GET/POST 都是 TCP 链接。
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）。

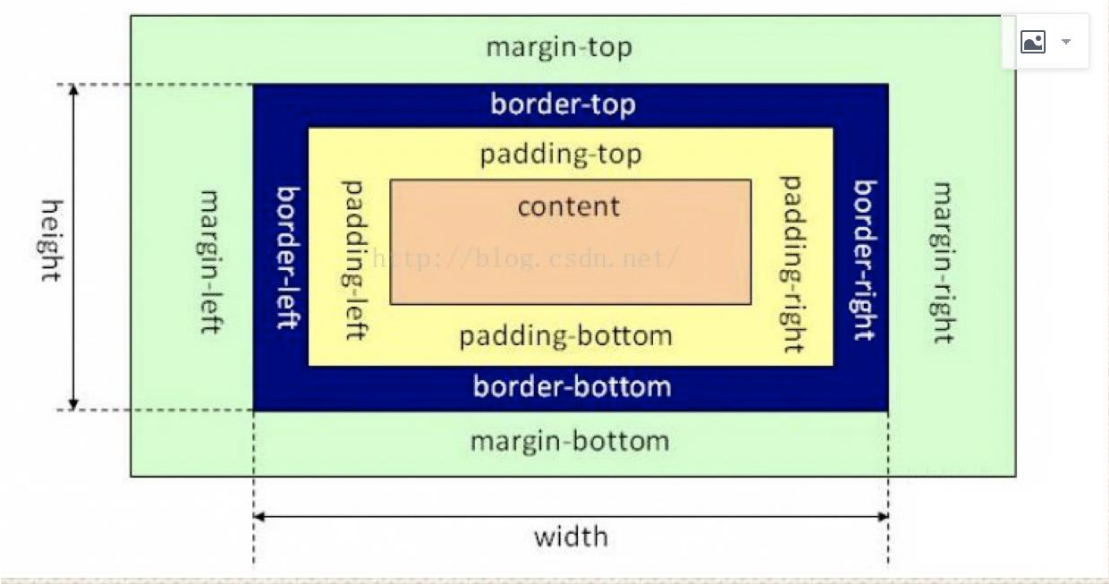
1.2 | CSS

- 说一下 css 盒模型

参考回答：
简介：就是用来装页面上的元素的矩形区域。CSS 中的盒子模型包括 IE 盒子模型和标准的 W3C 盒子模型。
box-sizing(有 3 个值哦)：border-box,padding-box,content-box.
标准盒子模型：



IE 盒子模型：



区别：从图中我们可以看出，这两种盒子模型最主要的区别就是 width 的包含范围，在标准的盒子模型中，width 指 content 部分的宽度，在 IE 盒子模型中，width 表示 content+padding+border 这三个部分的宽度，故这使得在计算整个盒子的宽度时存在着差异：

标准盒子模型的盒子宽度：左右 border+左右 padding+width

IE 盒子模型的盒子宽度：width

在 CSS3 中引入了 box-sizing 属性，box-sizing:content-box;表示标准的盒子模型，box-sizing:border-box 表示的是 IE 盒子模型

最后，前面我们还提到了，box-sizing:padding-box,这个属性值的宽度包含了左右 padding+width

也很好理解性记忆，包含什么，width 就从什么开始算起。

- 画一条 0.5px 的线

参考回答：
采用 meta viewport 的方式

<meta name="viewport" content="initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />

采用 border-image 的方式

采用 transform: scale()的方式

- link 标签和 import 标签的区别

参考回答：
link 属于 html 标签，而@import 是 css 提供的
页面被加载时，link 会同时被加载，而@import 引用的 css 会等到页面加载结束后加载。
link 是 html 标签，因此没有兼容性，而@import 只有 IE5 以上才能识别。
link 方式样式的权重高于@import 的。

- transition 和 animation 的区别

参考回答：
Animation 和 transition 大部分属性是相同的，他们都是随时间改变元素的属性值，他们的主要区别是 transition 需要触发一个事件才能改变属性，而 animation 不需要触发任何事件的情况下才会随时间改变属性值，并且 transition 为 2 帧，从 from .... to，而 animation 可以一帧一帧的。

- Flex 布局

参考回答：
文章链接：
http://www.ruanyifeng.com/blog/2015/07/flex-grammar.html?utm_source=tuicool（语法篇）
http://www.ruanyifeng.com/blog/2015/07/flex-examples.html（实例篇）Flex 是 Flexible Box 的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。

布局的传统解决方案，基于盒状模型，依赖 display 属性 + position 属性 + float 属性。它对于那些特殊布局非常不方便，比如，垂直居中就不容易实现。
简单的分为容器属性和元素属性
容器的属性：
flex-direction：决定主轴的方向（即子 item 的排列方法）

```
.box {
flex-direction: row | row-reverse | column | column-reverse;
}
```

flex-wrap：决定换行规则

```
.box{
flex-wrap: nowrap | wrap | wrap-reverse;
}
```

flex-flow：

```
.box {
flex-flow: <flex-direction> || <flex-wrap>;
}
```

justify-content：对其方式，水平主轴对齐方式
align-items：对齐方式，竖直轴线方向
项目的属性（元素的属性）：
order 属性：定义项目的排列顺序，顺序越小，排列越靠前，默认为 0
flex-grow 属性：定义项目的放大比例，即使存在空间，也不会放大
flex-shrink 属性：定义了项目的缩小比例，当空间不足的情况下会等比例的缩小，如果定义个 item 的 flow-shrink 为 0，则为不缩小
flex-basis 属性：定义了在分配多余的空间，项目占据的空间。
flex：是 flex-grow 和 flex-shrink、flex-basis 的简写，默认值为 0 1 auto。
align-self：允许单个项目与其他项目不一样的对齐方式，可以覆盖 align-items，默认属性为 auto，表示继承父元素的 align-items
比如说，用 flex 实现圣杯布局

- **BFC（块级格式化上下文，用于清楚浮动，防止 margin 重叠等）**

参考回答：
直译成：块级格式化上下文，是一个独立的渲染区域，并且有一定的布局规则。
BFC 区域不会与 float box 重叠
BFC 是页面上的一个独立容器，子元素不会影响到外面
计算 BFC 的高度时，浮动元素也会参与计算
那些元素会生成 BFC：
根元素
float 不为 none 的元素
position 为 fixed 和 absolute 的元素
display 为 inline-block、table-cell、table-caption，flex，inline-flex 的元素
overflow 不为 visible 的元素

- 垂直居中的方法

参考回答：

(1)margin:auto 法

```
css:
div{
width: 400px;
height: 400px;
position: relative;
border: 1px solid #465468;
}
img{
position: absolute;
margin: auto;
top: 0;
left: 0;
right: 0;
bottom: 0;
}
html:
<div>
<img src="mm.jpg">
</div>
```

定位为上下左右为 0，margin：0 可以实现脱离文档流的居中.

(2)margin 负值法

```
.container{
width: 500px;
height: 400px;
border: 2px solid #379;
position: relative;
}
.inner{
width: 480px;
height: 380px;
background-color: #746;
position: absolute;
top: 50%;
left: 50%;
margin-top: -190px; /*height 的一半*/
margin-left: -240px; /*width 的一半*/
}
```

补充：其实这里也可以将 marin-top 和 margin-left 负值替换成，transform：translateX(-50%)和 transform：translateY(-50%)

(3)table-cell（未脱离文档流的）

设置父元素的 display:table-cell,并且 vertical-align:middle，这样子元素可以实现垂直居中。

```
css:
div{
width: 300px;
height: 300px;
border: 3px solid #555;
display: table-cell;
vertical-align: middle;
text-align: center;
}
img{
vertical-align: middle;
}
```

(4)利用 flex

将父元素设置为 display:flex，并且设置 align-items:center;justify-content:center;

```
css:
.container{
width: 300px;
height: 200px;
border: 3px solid #546461;
display: -webkit-flex;
display: flex;
-webkit-align-items: center;
align-items: center;
-webkit-justify-content: center;
justify-content: center;
}
.inner{
border: 3px solid #458761;
padding: 20px;
}
```

- 关于 JS 动画和 css3 动画的差异性

参考回答：

渲染线程分为 main thread 和 compositor thread，如果 css 动画只改变 transform 和 opacity，这时整个 CSS 动画得以在 compositor trhead 完成（而 JS 动画则会在 main thread 执行，然后出发 compositor thread 进行下一步操作），特别注意的是如果改变 transform 和 opacity 是不会 layout 或者 paint 的。

区别：
功能涵盖面，JS 比 CSS 大
实现/重构难度不一，CSS3 比 JS 更加简单，性能跳优方向固定
对帧速表现不好的低版本浏览器，css3 可以做到自然降级
css 动画有天然事件支持
css3 有兼容性问题

- 说一下块元素和行元素

参考回答：
块元素：独占一行，并且有自动填满父元素，可以设置 margin 和 pading 以及高度和宽度
行元素：不会独占一行，width 和 height 会失效，并且在垂直方向的 padding 和 margin 会失
效。

- 多行元素的文本省略号

参考回答：

```
display: -webkit-box
-webkit-box-orient:vertical
-webkit-line-clamp:3
overflow:hidden
```

- visibility=hidden, opacity=0，display:none

参考回答：
opacity=0，该元素隐藏起来了，但不会改变页面布局，并且，如果该元素已经绑定一些事件，如 click 事件，那么点击该区域，也能触发点击事件的 visibility=hidden，该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已经绑定的事件
display=none，把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素删除掉一样。

- 双边距重叠问题（外边距折叠）

参考回答：
多个相邻（兄弟或者父子关系）普通流的块元素垂直方向 marigin 会重叠
折叠的结果为：

两个相邻的外边距都是正数时，折叠结果是它们两者之间较大的值。
两个相邻的外边距都是负数时，折叠结果是两者绝对值的较大值。
两个外边距一正一负时，折叠结果是两者的相加的和。

- position 属性　比较

参考回答：
固定定位 fixed：
元素的位置相对于浏览器窗口是固定位置，即使窗口是滚动的它也不会移动。Fixed 定位使元素的位置与文档流无关，因此不占据空间。 Fixed 定位的元素和其他元素重叠。
相对定位 relative：
如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。 在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。
绝对定位 absolute：
绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>。 absolute 定位使元素的位置与文档流无关，因此不占据空间。 absolute 定位的元素和其他元素重叠。
粘性定位 sticky：
元素先按照普通文档流定位，然后相对于该元素在流中的 flow root（BFC）和 containing block（最近的块级祖先元素）定位。而后，元素定位表现为在跨越特定阈值前为相对定位，之后为固定定位。
默认定位 Static：
默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。
inherit:
规定应该从父元素继承 position 属性的值。

- 浮动清除

参考回答：
方法一：使用带 clear 属性的空元素
在浮动元素后使用一个空元素如 ，并在 CSS 中赋予.clear{clear:both;}属性即可清理浮动。亦可使用<br class="clear" />或<hr class="clear" />来进行清理。
方法二：使用 CSS 的 overflow 属性
给浮动元素的容器添加 overflow:hidden;或 overflow:auto;可以清除浮动，另外在 IE6 中还需要触发 hasLayout ，例如为父元素设置容器宽高或设置 zoom:1。
在添加 overflow 属性后，浮动元素又回到了容器层，把容器高度撑起，达到了清理浮动的效果。
方法三：给浮动的元素的容器添加浮动

给浮动元素的容器也添加上浮动属性即可清除内部浮动，但是这样会使其整体浮动，影响布局，不推荐使用。
方法四：使用邻接元素处理
什么都不做，给浮动元素后面的元素添加 clear 属性。
方法五：使用 CSS 的:after 伪元素
结合:after 伪元素（注意这不是伪类，而是伪元素，代表一个元素之后最近的元素）和 IEhack ，可以完美兼容当前主流的各大浏览器，这里的 IEhack 指的是触发 hasLayout。
给浮动元素的容器添加一个 clearfix 的 class，然后给这个 class 添加一个:after 伪元素实现元素末尾添加一个看不见的块元素（Block element）清理浮动。
参考 https://www.cnblogs.com/ForEvErNoME/p/3383539.html

- **css3 新特性**

参考回答：
开放题。CSS3 边框如 border-radius，box-shadow 等；CSS3 背景如 background-size，background-origin 等；CSS3 2D，3D 转换如 transform 等；CSS3 动画如 animation 等。
参考 https://www.cnblogs.com/xkweb/p/5862612.html

- **CSS 选择器有哪些，优先级呢**

参考回答：
id 选择器，class 选择器，标签选择器，伪元素选择器，伪类选择器等
同一元素引用了多个样式时，排在后面的样式属性的优先级高；
样式选择器的类型不同时，优先级顺序为：id 选择器 > class 选择器 > 标签选择器；
标签之间存在层级包含关系时，后代元素会继承祖先元素的样式。如果后代元素定义了与祖先元素相同的样式，则祖先元素的相同的样式属性会被覆盖。继承的样式的优先级比较低，至少比标签选择器的优先级低；
带有!important 标记的样式属性的优先级最高；
样式表的来源不同时，优先级顺序为：内联样式> 内部样式 > 外部样式 > 浏览器用户自定义样式 > 浏览器默认样式

- **清除浮动的方法，能讲讲吗**

参考回答：
方法一：使用带 clear 属性的空元素
在浮动元素后使用一个空元素如 ，并在 CSS 中赋予.clear{clear:both;}属性即可清理浮动。亦可使用<br class="clear" />或<hr class="clear" />来进行清理。
方法二：使用 CSS 的 overflow 属性

给浮动元素的容器添加 overflow:hidden;或 overflow:auto;可以清除浮动，另外在 IE6 中还需要触发 hasLayout ，例如为父元素设置容器宽高或设置 zoom:1。
在添加 overflow 属性后，浮动元素又回到了容器层，把容器高度撑起，达到了清理浮动的效果。
方法三：给浮动的元素的容器添加浮动
给浮动元素的容器也添加上浮动属性即可清除内部浮动，但是这样会使其整体浮动，影响布局，不推荐使用。
方法四：使用邻接元素处理
什么都不做，给浮动元素后面的元素添加 clear 属性。
方法五：使用 CSS 的:after 伪元素
结合:after 伪元素（注意这不是伪类，而是伪元素，代表一个元素之后最近的元素）和 IEhack ，可以完美兼容当前主流的各大浏览器，这里的 IEhack 指的是触发 hasLayout。
给浮动元素的容器添加一个 clearfix 的 class，然后给这个 class 添加一个:after 伪元素实现元素末尾添加一个看不见的块元素（Block element）清理浮动。
参考 https://www.cnblogs.com/ForEvErNoME/p/3383539.html

- 怎么样让一个元素消失，讲讲

参考回答：
display:none; visibility:hidden; opacity: 0; 等等

- 介绍一下盒模型

参考回答：
CSS 盒模型本质上是一个盒子，封装周围的 HTML 元素，它包括：边距，边框，填充，和实际内容。
标准盒模型：一个块的总宽度=width+margin(左右)+padding(左右)+border(左右)
怪异盒模型：一个块的总宽度=width+margin（左右）（既 width 已经包含了 padding 和 border 值）
设置盒模型：box-sizing:border-box

- position 相关属性

参考回答：
固定定位 fixed：
元素的位置相对于浏览器窗口是固定位置，即使窗口是滚动的它也不会移动。Fixed 定位使元素的位置与文档流无关，因此不占据空间。 Fixed 定位的元素和其他元素重叠。
相对定位 relative：

如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。 在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。
绝对定位 absolute：
绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>。 absolute 定位使元素的位置与文档流无关，因此不占据空间。 absolute 定位的元素和其他元素重叠。
粘性定位 sticky：
元素先按照普通文档流定位，然后相对于该元素在流中的 flow root（BFC）和 containing block（最近的块级祖先元素）定位。而后，元素定位表现为在跨越特定阈值前为相对定位，之后为固定定位。
默认定位 Static：
默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。
inherit:
规定应该从父元素继承 position 属性的值。

- css 动画如何实现

参考回答：
创建动画序列，需要使用 animation 属性或其子属性，该属性允许配置动画时间、时长以及其他动画细节，但该属性不能配置动画的实际表现，动画的实际表现是由 @keyframes 规则实现，具体情况参见使用 keyframes 定义动画序列小节部分。transition 也可实现动画。transition 强调过渡，是元素的一个或多个属性发生变化时产生的过渡效果，同一个元素通过两个不同的途径获取样式，而第二个途径当某种改变发生（例如 hover）时才能获取样式，这样就会产生过渡动画。

- 如何实现图片在某个容器中居中的？

参考回答：
父元素固定宽高，利用定位及设置子元素 margin 值为自身的一半。
父元素固定宽高，子元素设置 position: absolute，margin：auto 平均分配 margin
css3 属性 transform。子元素设置 position: absolute; left: 50%; top: 50%;transform: translate(-50%,-50%);即可。
将父元素设置成 display: table, 子元素设置为单元格 display: table-cell。
弹性布局 display: flex。设置 align-items: center; justify-content: center

- 如何实现元素的垂直居中

参考回答：
法一：父元素 display:flex,align-items:center;
法二：元素绝对定位，top:50%，margin-top：-（高度/2）

法三：高度不确定用 transform：translateY（-50%）
法四：父元素 table 布局，子元素设置 vertical-align:center;

- CSS3 中对溢出的处理

参考回答：
cnk0hu
text-overflow 属性，值为 clip 是修剪文本；ellipsis 为显示省略符号来表被修剪的文本；string 为使用给定的字符串来代表被修剪的文本。

- float 的元素，display 是什么

参考回答：
display 为 block

- 隐藏页面中某个元素的方法

参考回答：
display:none; visibility:hidden; opacity: 0; position 移到外部，z-index 涂层遮盖等等

- 三栏布局的实现方式，尽可能多写，浮动布局时，三个 div 的生成顺序有没有影响

参考回答：
三列布局又分为两种，两列定宽一列自适应，以及两侧定宽中间自适应
两列定宽一列自适应：
1、使用 float+margin：
给 div 设置 float：left，left 的 div 添加属性 margin-right：left 和 center 的间隔 px,right 的 div 添加属性 margin-left：left 和 center 的宽度之和加上间隔
2、使用 float+overflow：
给 div 设置 float：left，再给 right 的 div 设置 overflow:hidden。这样子两个盒子浮动，另一个盒子触发 bfc 达到自适应
3、使用 position：
父级 div 设置 position：relative，三个子级 div 设置 position：absolute，这个要计算好盒子的宽度和间隔去设置位置，兼容性比较好，
4、使用 table 实现：
父级 div 设置 display：table，设置 border-spacing：10px//设置间距，取值随意，子级 div 设置 display:table-cell，这种方法兼容性好，适用于高度宽度未知的情况，但是 margin 失效，设计间隔比较麻烦，
5、flex 实现：

parent 的 div 设置 display：flex；left 和 center 的 div 设置 margin-right；然后 right 的 div 设置 flex：1；这样子 right 自适应，但是 flex 的兼容性不好
6、grid 实现：
parent 的 div 设置 display：grid，设置 grid-template-columns 属性，固定第一列第二列宽度，第三列 auto，
对于两侧定宽中间自适应的布局，对于这种布局需要把 center 放在前面，可以采用双飞翼布局：圣杯布局，来实现，也可以使用上述方法中的 grid，table，flex，position 实现

- 什么是 BFC

参考回答：
BFC 也就是常说的块格式化上下文，这是一个独立的渲染区域，规定了内部如何布局，并且这个区域的子元素不会影响到外面的元素，其中比较重要的布局规则有内部 box 垂直放置，计算 BFC 的高度的时候，浮动元素也参与计算，触发 BFC 的规则有根元素，浮动元素，position 为 absolute 或 fixed 的元素，display 为 inline-block，table-cell，table-caption，flex，inline-flex，overflow 不为 visible 的元素

- calc 属性

参考回答：
Calc 用户动态计算长度值，任何长度值都可以使用 calc()函数计算，需要注意的是，运算符前后都需要保留一个空格，例如：width: calc(100% - 10px)；

- 有一个 width300，height300，怎么实现在屏幕上垂直水平居中

参考回答：
对于行内块级元素，
1、父级元素设置 text-alig：center，然后设置 line-height 和 vertical-align 使其垂直居中，最后设置 font-size：0 消除近似居中的 bug
2、父级元素设置 display：table-cell，vertical-align：middle 达到水平垂直居中
3、采用绝对定位，原理是子绝父相，父元素设置 position：relative，子元素设置 position：absolute，然后通过 transform 或 margin 组合使用达到垂直居中效果，设置 top：50%，left：50%，transform：translate（-50%，-50%）
4、绝对居中，原理是当 top,bottom 为 0 时，margin-top&bottom 设置 auto 的话会无限延伸沾满空间并平分，当 left，right 为 0 时,margin-left&right 设置 auto 会无限延伸占满空间并平分，
5、采用 flex，父元素设置 display：flex，子元素设置 margin：auto
6、视窗居中，vh 为视口单位，50vh 即是视口高度的 50/100，设置 margin：50vh auto 0，transform：translate(-50%)

- display：table 和本身的 table 有什么区别

参考回答：
Display:table 和本身 table 是相对应的，区别在于，display：table 的 css 声明能够让一个 html 元素和它的子节点像 table 元素一样，使用基于表格的 css 布局，是我们能够轻松定义一个单元格的边界，背景等样式，而不会产生因为使用了 table 那样的制表标签导致的语义化问题。
之所以现在逐渐淘汰了 table 系表格元素，是因为用 div+css 编写出来的文件比用 table 边写出来的文件小，而且 table 必须在页面完全加载后才显示，div 则是逐行显示，table 的嵌套性太多，没有 div 简洁

- position 属性的值有哪些及其区别

参考回答：
Position 属性把元素放置在一个静态的，相对的，绝对的，固定的位置中，
Static：位置设置为 static 的元素，他始终处于页面流给予的位置，static 元素会忽略任何 top,buttom,left,right 声明
Relative：位置设置为 relative 的元素，可将其移至相对于其正常位置的地方，因此 left：20 会将元素移至元素正常位置左边 20 个像素的位置
Absolute：此元素可定位于相对包含他的元素的指定坐标，此元素可通过 left，top 等属性规定
Fixed：位置被设为 fiexd 的元素，可定为与相对浏览器窗口的指定坐标，可以通过 left，top，right 属性来定位

- z-index 的定位方法

参考回答：
z-index 属性设置元素的堆叠顺序，拥有更好堆叠顺序的元素会处于较低顺序元素之前，z-index 可以为负，且 z-index 只能在定位元素上奏效，该属性设置一个定位元素沿 z 轴的位置，如果为正数，离用户越近，为负数，离用户越远，它的属性值有 auto，默认，堆叠顺序与父元素相等，number，inherit，从父元素继承 z-index 属性的值

- 如果想要改变一个 DOM 元素的字体颜色，不在它本身上进行操作？

参考回答：
可以更改父元素的 color

- 对 CSS 的新属性有了解过的吗？

参考回答：

CSS3 的新特性中，在布局方面新增了 flex 布局，在选择器方面新增了例如 first-of-type,nth-child 等选择器，在盒模型方面添加了 box-sizing 来改变盒模型，在动画方面增加了 animation，2d 变换，3d 变换等，在颜色方面添加透明，rbga 等，在字体方面允许嵌入字体和设置字体阴影，最后还有媒体查讯等

- **用的最多的 css 属性是啥？**

参考回答：
用的目前来说最多的是 flex 属性，灵活但是兼容性方面不强。

- **line-height 和 height 的区别**

参考回答：
line-height 一般是指布局里面一段文字上下行之间的高度，是针对字体来设置的，height 一般是指容器的整体高度。

- **设置一个元素的背景颜色，背景颜色会填充哪些区域？**

参考回答：
background-color 设置的背景颜色会填充元素的 content、padding、border 区域。

- **知道属性选择器和伪类选择器的优先级吗**

参考回答：
属性选择器和伪类选择器优先级相同

- **inline-block、inline 和 block 的区别；为什么 img 是 inline 还可以设置宽高**

参考回答：
Block 是块级元素，其前后都会有换行符，能设置宽度，高度，margin/padding 水平垂直方向都有效。
Inline：设置 width 和 height 无效，margin 在竖直方向上无效，padding 在水平方向垂直方向都有效，前后无换行符
Inline-block：能设置宽度高度，margin/padding 水平垂直方向 都有效，前后无换行符

- **用 css 实现一个硬币旋转的效果**

参考回答：

虽然不认为很多人能在面试中写出来

```
#euro {
width: 150px;
height: 150px;
margin-left: -75px;
margin-top: -75px;
position: absolute;
top: 50%;
left: 50%;
transform-style: preserve-3d;
animation: spin 2.5s linear infinite;
}
.back {
background-image: url("/uploads/160101/backeuro.png");
width: 150px;
height: 150px;
}
.middle {
background-image: url("/uploads/160101/faceeuro.png");
width: 150px;
height: 150px;
transform: translateZ(1px);
position: absolute;
top: 0;
}
.front {
background-image: url("/uploads/160101/faceeuro.png");
height: 150px;
position: absolute;
top: 0;
transform: translateZ(10px);
width: 150px;
}
@keyframes spin {
0% {
transform: rotateY(0deg);
}
100% {
transform: rotateY(360deg);
}
}
```

- **了解重绘和重排吗，知道怎么去减少重绘和重排吗，让文档脱离文档流有哪些方法**

参考回答：
DOM 的变化影响到了预算内宿的几何属性比如宽高，浏览器重新计算元素的几何属性，其他元素的几何属性也会受到影响，浏览器需要重新构造渲染书，这个过程称之为重排，浏览器将受到影响的部分重新绘制在屏幕上 的过程称为重绘，引起重排重绘的原因有：
添加或者删除可见的 DOM 元素，
元素尺寸位置的改变
浏览器页面初始化，
浏览器窗口大小发生改变，重排一定导致重绘，重绘不一定导致重排，
减少重绘重排的方法有：
不在布局信息改变时做 DOM 查询，
使用 csstext,className 一次性改变属性
使用 fragment
对于多次重排的元素，比如说动画。使用绝对定位脱离文档流，使其不影响其他元素

- CSS 画正方体，三角形

参考回答：
画三角形
```
#triangle02{
width: 0;
height: 0;
border-top: 50px solid blue;
border-right: 50px solid red;
border-bottom: 50px solid green;
border-left: 50px solid yellow;
}
```
画正方体：
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>perspective</title>
<style>
.wrapper{
width: 50%;
float: left;
}
.cube{
font-size: 4em;
width: 2em;
```

```
margin: 1.5em auto;
transform-style:preserve-3d;
transform:rotateX(-35deg) rotateY(30deg);
}
.side{
position: absolute;
width: 2em;
height: 2em;
background: rgba(255,99,71,0.6);
border: 1px solid rgba(0,0,0,0.5);
color: white;
text-align: center;
line-height: 2em;
}
.front{
transform:translateZ(1em);
}
.bottom{
transform:rotateX(-90deg) translateZ(1em);
}
.top{
transform:rotateX(90deg) translateZ(1em);
}
.left{
transform:rotateY(-90deg) translateZ(1em);
}
.right{
transform:rotateY(90deg) translateZ(1em);
}
.back{
transform:translateZ(-1em);
}
</style>
</head>
<body>
<div class="wrapper w1">
<div class="cube">
<div class="side front">1</div>
<div class="side back">6</div>
<div class="side right">4</div>
<div class="side left">3</div>
<div class="side top">5</div>
<div class="side bottom">2</div>
</div>
```

```
</div>
<div class="wrapper w2">
<div class="cube">
<div class="side front">1</div>
<div class="side back">6</div>
<div class="side right">4</div>
<div class="side left">3</div>
<div class="side top">5</div>
<div class="side bottom">2</div>
</div>
</div>
</body>
</html>
```

- overflow 的原理

参考回答：
要讲清楚这个解决方案的原理，首先需要了解块格式化上下文，A block formatting context is a part of a visual CSS rendering of a Web page. It is the region in which the layout of block boxes occurs and in which floats interact with each other.翻译过来就是块格式化上下文是 CSS 可视化渲染的一部分，它是一块区域，规定了内部块盒 的渲染方式，以及浮动相互之间的影响关系
当元素设置了 overflow 样式且值部位 visible 时，该元素就构建了一个 BFC，BFC 在计算高度时，内部浮动元素的高度也要计算在内，也就是说技术 BFC 区域内只有一个浮动元素，BFC 的高度也不会发生塌缩，所以达到了清除浮动的目的。

- 清除浮动的方法

参考回答：
给要清除浮动的元素添加样式 clear，\
父元素结束标签钱插入清除浮动的块级元素，给该元素添加样式 clear
添加伪元素，在父级元素的最后，添加一个伪元素，通过清除伪元素的浮动，注意该伪元素的 display 为 block，
父元素添加样式 overflow 清除浮动，overflow 设置除 visible 以外的任何位置

- box-sizing 的语法和基本用处

参考回答：
box-sizing 规定两个并排的带边框的框，语法为 box-sizing：content-box/border-box/inherit
content-box：宽度和高度分别应用到元素的内容框，在宽度和高度之外绘制元素的内边距和边框

border-box：为元素设定的宽度和高度决定了元素的边框盒，
inherit：继承父元素的 box-sizing

- **使元素消失的方法有哪些？**

参考回答：
1. opacity：0，该元素隐藏起来了，但不会改变页面布局，并且，如果该元素已经绑定一些事件，如 click 事件，那么点击该区域，也能触发点击事件的
2. visibility：hidden，该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已经绑定的事件
3. display：none，把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素删除掉。

- **两个嵌套的 div，position 都是 absolute，子 div 设置 top 属性，那么这个 top 是相对于父元素的哪个位置定位的。**

参考回答：
margin 的外边缘

- **说说盒子模型**

参考回答：
CSS 盒模型本质上是一个盒子，封装周围的 HTML 元素，它包括：边距，边框，填充，和实际内容。
标准盒模型：一个块的总宽度=width+margin(左右)+padding(左右)+border(左右)
怪异盒模型：一个块的总宽度=width+margin（左右）（既 width 已经包含了 padding 和 border 值）
如何设置：box-sizing:border-box

- **display**

参考回答：
主要取值有 none,block,inline-block,inline,flex 等。具体可参考 https://developer.mozilla.org/zh-CN/docs/Web/CSS/display

- **怎么隐藏一个元素**

参考回答：
1. opacity：0，该元素隐藏起来了，但不会改变页面布局，并且，如果该元素已经绑定一些事件，如 click 事件，那么点击该区域，也能触发点击事件的

2. visibility：hidden，该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已经绑定的事件
3. display：none，把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素删除掉。

- display:none 和 visibilty:hidden 的区别

参考回答：
1. visibility：hidden，该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已经绑定的事件
2. display：none，把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素删除掉。

- 相对布局和绝对布局，position:relative 和 obsolute。

参考回答：
相对定位 relative：
如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。 在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。
绝对定位 absolute：
绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>。 absolute 定位使元素的位置与文档流无关，因此不占据空间。 absolute 定位的元素和其他元素重叠。

- flex 布局

参考回答：
flex 是 Flexible Box 的缩写，意为"弹性布局"。指定容器 display: flex 即可。
容器有以下属性：flex-direction，flex-wrap，flex-flow，justify-content，align-items，align-content。
flex-direction 属性决定主轴的方向；
flex-wrap 属性定义，如果一条轴线排不下，如何换行；
flex-flow 属性是 flex-direction 属性和 flex-wrap 属性的简写形式，默认值为 row nowrap；
justify-content 属性定义了项目在主轴上的对齐方式。
align-items 属性定义项目在交叉轴上如何对齐。
align-content 属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。
项目（子元素）也有一些属性：order，flex-grow，flex-shrink，flex-basis，flex，align-self。
order 属性定义项目的排列顺序。数值越小，排列越靠前，默认为 0。

flex-grow 属性定义项目的放大比例，默认为 0，即如果存在剩余空间，也不放大。
flex-shrink 属性定义了项目的缩小比例，默认为 1，即如果空间不足，该项目将缩小。
flex-basis 属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。
flex 属性是 flex-grow, flex-shrink 和 flex-basis 的简写，默认值为 0 1 auto。后两个属性可选。
align-self 属性允许单个项目有与其他项目不一样的对齐方式，可覆盖 align-items 属性。默认值为 auto，表示继承父元素的 align-items 属性，如果没有父元素，则等同于 stretch。
参考 http://www.ruanyifeng.com/blog/2015/07/flex-grammar.html

- block、inline、inline-block 的区别。

参考回答：
block 元素会独占一行，多个 block 元素会各自新起一行。默认情况下，block 元素宽度自动填满其父元素宽度。
block 元素可以设置 width,height 属性。块级元素即使设置了宽度,仍然是独占一行。
block 元素可以设置 margin 和 padding 属性。

inline 元素不会独占一行，多个相邻的行内元素会排列在同一行里，直到一行排列不下，才会新换一行，其宽度随元素的内容而变化。
inline 元素设置 width,height 属性无效。
inline 元素的 margin 和 padding 属性，水平方向的 padding-left, padding-right, margin-left, margin-right 都产生边距效果；但竖直方向的 padding-top, padding-bottom, margin-top, margin-bottom 不会产生边距效果。

inline-block：简单来说就是将对象呈现为 inline 对象，但是对象的内容作为 block 对象呈现。之后的内联对象会被排列在同一行内。比如我们可以给一个 link（a 元素）inline-block 属性值，使其既具有 block 的宽度高度特性又具有 inline 的同行特性。

- css 的常用选择器

参考回答：
id 选择器，类选择器，伪类选择器等

- css 布局

参考回答：
六种布局方式总结：圣杯布局、双飞翼布局、Flex 布局、绝对定位布局、表格布局、网格布局。

圣杯布局是指布局从上到下分为 header、container、footer，然后 container 部分定为三栏布局。这种布局方式同样分为 header、container、footer。圣杯布局的缺陷在于 center 是在 container 的 padding 中的，因此宽度小的时候会出现混乱。
双飞翼布局给 center 部分包裹了一个 main 通过设置 margin 主动地把页面撑开。
Flex 布局是由 CSS3 提供的一种方便的布局方式。
绝对定位布局是给 container 设置 position: relative 和 overflow: hidden，因为绝对定位的元素的参照物为第一个 postion 不为 static 的祖先元素。 left 向左浮动，right 向右浮动。center 使用绝对定位，通过设置 left 和 right 并把两边撑开。 center 设置 top: 0 和 bottom: 0 使其高度撑开。
表格布局的好处是能使三栏的高度统一。
网格布局可能是最强大的布局方式了，使用起来极其方便，但目前而言，兼容性并不好。网格布局，可以将页面分割成多个区域，或者用来定义内部元素的大小，位置，图层关系。

- css 定位

参考回答：
固定定位 fixed：
元素的位置相对于浏览器窗口是固定位置，即使窗口是滚动的它也不会移动。Fixed 定位使元素的位置与文档流无关，因此不占据空间。 Fixed 定位的元素和其他元素重叠。
相对定位 relative：
如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。 在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。
绝对定位 absolute：
绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>。 absolute 定位使元素的位置与文档流无关，因此不占据空间。 absolute 定位的元素和其他元素重叠。
粘性定位 sticky：
元素先按照普通文档流定位，然后相对于该元素在流中的 flow root（BFC）和 containing block（最近的块级祖先元素）定位。而后，元素定位表现为在跨越特定阈值前为相对定位，之后为固定定位。
默认定位 Static：
默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。
inherit:
规定应该从父元素继承 position 属性的值。

- relative 定位规则

参考回答：

如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。 在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。

- 垂直居中

参考回答：
父元素固定宽高，利用定位及设置子元素 margin 值为自身的一半。
父元素固定宽高，子元素设置 position: absolute，margin：auto 平均分配 margin
css3 属性 transform。子元素设置 position: absolute; left: 50%; top: 50%;transform: translate(-50%,-50%);即可。
将父元素设置成 display: table, 子元素设置为单元格 display: table-cell。
弹性布局 display: flex。设置 align-items: center; justify-content: center;

- css 预处理器有什么

参考回答：
less，sass 等

1.3 | JavaScript

- get 请求传参长度的误区

参考回答：
误区：我们经常说 get 请求参数的大小存在限制，而 post 请求的参数大小是无限制的。
实际上 HTTP 协议从未规定 GET/POST 的请求长度限制是多少。对 get 请求参数的限制是来源与浏览器或 web 服务器，浏览器或 web 服务器限制了 url 的长度。为了明确这个概念，我们必须再次强调下面几点:
HTTP 协议 未规定 GET 和 POST 的长度限制
GET 的最大长度显示是因为 浏览器和 web 服务器限制了 URI 的长度
不同的浏览器和 WEB 服务器，限制的最大长度不一样
要支持 IE，则最大长度为 2083byte，若只支持 Chrome，则最大长度 8182byte

- 补充 get 和 post 请求在缓存方面的区别

参考回答：
post/get 的请求区别，具体不再赘述。
补充补充一个 get 和 post 在缓存方面的区别：

get 请求类似于查找的过程，用户获取数据，可以不用每次都与数据库连接，所以可以使用缓存。
post 不同，post 做的一般是修改和删除的工作，所以必须与数据库交互，所以不能使用缓存。因此 get 请求适合于请求缓存。

- **说一下闭包**

参考回答：
一句话可以概括：闭包就是能够读取其他函数内部变量的函数，或者子函数在外调用，子函数所在的父函数的作用域不会被释放。

- **说一下类的创建和继承**

参考回答：
（1）类的创建（es5）：new 一个 function，在这个 function 的 prototype 里面增加属性和方法。
下面来创建一个 Animal 类：

```
// 定义一个动物类
function Animal (name) {
// 属性
this.name = name || 'Animal';
// 实例方法
this.sleep = function(){
console.log(this.name + '正在睡觉！');
}
}
// 原型方法
Animal.prototype.eat = function(food) {
console.log(this.name + '正在吃：' + food);
};
```

这样就生成了一个 Animal 类，实力化生成对象后，有方法和属性。
（2）类的继承——原型链继承

```
--原型链继承
function Cat(){ }
Cat.prototype = new Animal();
Cat.prototype.name = 'cat';
//  Test Code
var cat = new Cat();
console.log(cat.name);
console.log(cat.eat('fish'));
console.log(cat.sleep());
console.log(cat instanceof Animal); //true
console.log(cat instanceof Cat); //true
```

介绍：在这里我们可以看到 new 了一个空对象,这个空对象指向 Animal 并且 Cat.prototype 指向了这个空对象，这种就是基于原型链的继承。
特点：基于原型链，既是父类的实例，也是子类的实例
缺点：无法实现多继承
（3）构造继承：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）

```
function Cat(name){
Animal.call(this);
this.name = name || 'Tom';
}
// Test Code
var cat = new Cat();
console.log(cat.name);
console.log(cat.sleep());
console.log(cat instanceof Animal); // false
console.log(cat instanceof Cat); // true
```

特点：可以实现多继承
缺点：只能继承父类实例的属性和方法，不能继承原型上的属性和方法。
（4）实例继承和拷贝继承
实例继承：为父类实例添加新特性，作为子类实例返回
拷贝继承：拷贝父类元素上的属性和方法
上述两个实用性不强，不一一举例。
（5）组合继承：相当于构造继承和原型链继承的组合体。通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用

```
function Cat(name){
Animal.call(this);
this.name = name || 'Tom';
}
Cat.prototype = new Animal();
Cat.prototype.constructor = Cat;
// Test Code
var cat = new Cat();
console.log(cat.name);
console.log(cat.sleep());
console.log(cat instanceof Animal); // true
console.log(cat instanceof Cat); // true
```

特点：可以继承实例属性/方法，也可以继承原型属性/方法
缺点：调用了两次父类构造函数，生成了两份实例
（6）寄生组合继承：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性

```
function Cat(name){
Animal.call(this);
this.name = name || 'Tom';
}
```

```
(function(){
// 创建一个没有实例方法的类
var Super = function(){};
Super.prototype = Animal.prototype;
//将实例作为子类的原型
Cat.prototype = new Super();
})();
// Test Code
var cat = new Cat();
console.log(cat.name);
console.log(cat.sleep());
console.log(cat instanceof Animal); // true
console.log(cat instanceof Cat); //true
```

较为推荐

- **如何解决异步回调地狱**

参考回答：

```
1 promise、generator、async/await
```

- **说说前端中的事件流**

参考回答：

HTML 中与 javascript 交互是通过事件驱动来实现的，例如鼠标点击事件 onclick、页面的滚动事件 onscroll 等等，可以向文档或者文档中的元素添加事件侦听器来预订事件。想要知道这些事件是在什么时候进行调用的，就需要了解一下“事件流”的概念。

什么是事件流：事件流描述的是从页面中接收事件的顺序,DOM2 级事件流包括下面几个阶段。

事件捕获阶段

处于目标阶段

事件冒泡阶段

addEventListener：addEventListener 是 DOM2 级事件新增的指定事件处理程序的操作，这个方法接收 3 个参数：要处理的事件名、作为事件处理程序的函数和一个布尔值。最后这个布尔值参数如果是 true，表示在捕获阶段调用事件处理程序；如果是 false，表示在冒泡阶段调用事件处理程序。

IE 只支持事件冒泡。

- **如何让事件先冒泡后捕获**

参考回答：

在 DOM 标准事件模型中，是先捕获后冒泡。但是如果要实现先冒泡后捕获的效果，对于同一个事件，监听捕获和冒泡，分别对应相应的处理函数，监听到捕获事件，先暂缓执行，直到冒泡事件被捕获后再执行捕获之间。

- 说一下事件委托

参考回答：
简介：事件委托指的是，不在事件的发生地（直接 dom）上设置监听函数，而是在其父元素上设置监听函数，通过事件冒泡，父元素可以监听到子元素上事件的触发，通过判断事件发生元素 DOM 的类型，来做出不同的响应。
举例：最经典的就是 ul 和 li 标签的事件监听，比如我们在添加事件时候，采用事件委托机制，不会在 li 标签上直接添加，而是在 ul 父元素上添加。
好处：比较合适动态元素的绑定，新添加的子元素也会有监听函数，也可以有事件触发机制。

- 说一下图片的懒加载和预加载

参考回答：
预加载：提前加载图片，当用户需要查看时可直接从本地缓存中渲染。
懒加载：懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数。

两种技术的本质：两者的行为是相反的，一个是提前加载，一个是迟缓甚至不加载。
懒加载对服务器前端有一定的缓解压力作用，预加载则会增加服务器前端压力。

- mouseover 和 mouseenter 的区别

参考回答：
mouseover：当鼠标移入元素或其子元素都会触发事件，所以有一个重复触发，冒泡的过程。对应的移除事件是 mouseout
mouseenter：当鼠标移除元素本身（不包含元素的子元素）会触发事件，也就是不会冒泡，对应的移除事件是 mouseleave

- JS 的 new 操作符做了哪些事情

参考回答：
new 操作符新建了一个空对象，这个对象原型指向构造函数的 prototype，执行构造函数后返回这个对象。

- 改变函数内部 this 指针的指向函数（bind，apply，call 的区别）

参考回答：
通过 apply 和 call 改变函数的 this 指向，他们两个函数的第一个参数都是一样的表示要改变指向的那个对象，第二个参数，apply 是数组，而 call 则是 arg1,arg2...这种形式。通过 bind 改变 this 作用域会返回一个新的函数，这个函数不会马上执行。

- JS 的各种位置，比如 clientHeight,scrollHeight,offsetHeight ，以及 scrollTop, offsetTop,clientTop 的区别？

参考回答：
clientHeight：表示的是可视区域的高度，不包含 border 和滚动条
offsetHeight：表示可视区域的高度，包含了 border 和滚动条
scrollHeight：表示了所有区域的高度，包含了因为滚动被隐藏的部分。
clientTop：表示边框 border 的厚度，在未指定的情况下一般为 0
scrollTop：滚动后被隐藏的高度，获取对象相对于由 offsetParent 属性指定的父坐标(css 定位的元素或 body 元素)距离顶端的高度。

- JS 拖拽功能的实现

参考回答：
首先是三个事件，分别是 mousedown，mousemove，mouseup
当鼠标点击按下的时候，需要一个 tag 标识此时已经按下，可以执行 mousemove 里面的具体方法。

clientX，clientY 标识的是鼠标的坐标，分别标识横坐标和纵坐标，并且我们用 offsetX 和 offsetY 来表示元素的元素的初始坐标，移动的举例应该是：

鼠标移动时候的坐标-鼠标按下去时候的坐标。

也就是说定位信息为：

鼠标移动时候的坐标-鼠标按下去时候的坐标+元素初始情况下的 offetLeft.

还有一点也是原理性的东西，也就是拖拽的同时是绝对定位，我们改变的是绝对定位条件下的 left
以及 top 等等值。

补充：也可以通过 html5 的拖放（Drag 和 drop）来实现

- 异步加载 JS 的方法

参考回答：

defer：只支持 IE 如果您的脚本不会改变文档的内容，可将 defer 属性加入到<script>标签中，以便加快处理文档的速度。因为浏览器知道它将能够安全地读取文档的剩余部分而不用执行脚本，它将推迟对脚本的解释，直到文档已经显示给用户为止。
async，HTML5 属性仅适用于外部脚本，并且如果在 IE 中，同时存在 defer 和 async，那么 defer 的优先级比较高，脚本将在页面完成时执行。
创建 script 标签，插入到 DOM 中

- Ajax 解决浏览器缓存问题

参考回答：
在 ajax 发送请求前加上 anyAjaxObj.setRequestHeader("If-Modified-Since","0")。

在 ajax 发送请求前加上 anyAjaxObj.setRequestHeader("Cache-Control","no-cache")。

在 URL 后面加上一个随机数： "fresh=" + Math.random()。

在 URL 后面加上时间搓："nowtime=" + new Date().getTime()。

如果是使用 jQuery，直接这样就可以了 $.ajaxSetup({cache:false})。这样页面的所有 ajax 都会执行这条语句就是不需要保存缓存记录。

- JS 的节流和防抖

参考回答：
http://www.cnblogs.com/cocols/p/5499469.html

- JS 中的垃圾回收机制

参考回答：
必要性：由于字符串、对象和数组没有固定大小，所有当他们的大小已知时，才能对他们进行动态的存储分配。JavaScript 程序每次创建字符串、数组或对象时，解释器都必须分配内存来存储那个实体。只要像这样动态地分配了内存，最终都要释放这些内存以便他们能够被再用，否则，JavaScript 的解释器将会消耗完系统中所有可用的内存，造成系统崩溃。

这段话解释了为什么需要系统需要垃圾回收，JS 不像 C/C++，他有自己的一套垃圾回收机制（Garbage Collection）。JavaScript 的解释器可以检测到何时程序不再使用一个对象了，当他确定了一个对象是无用的时候，他就知道不再需要这个对象，可以把它所占用的内存释放掉了。例如：

```
var a="hello world";
var b="world";
var a=b;
//这时，会释放掉"hello world"，释放内存以便再引用
```

垃圾回收的方法：标记清除、计数引用。

标记清除

这是最常见的垃圾回收方式，当变量进入环境时，就标记这个变量为”进入环境“,从逻辑上讲，永远不能释放进入环境的变量所占的内存，永远不能释放进入环境变量所占用的内存，只要执行流程进入相应的环境，就可能用到他们。当离开环境时，就标记为离开环境。

垃圾回收器在运行的时候会给存储在内存中的变量都加上标记（所有都加），然后去掉环境变量中的变量，以及被环境变量中的变量所引用的变量（条件性去除标记），删除所有被标记的变量，删除的变量无法在环境变量中被访问所以会被删除，最后垃圾回收器，完成了内存的清除工作，并回收他们所占用的内存。

引用计数法

另一种不太常见的方法就是引用计数法，引用计数法的意思就是每个值没引用的次数，当声明了一个变量，并用一个引用类型的值赋值给改变量，则这个值的引用次数为 1,；相反的，如果包含了对这个值引用的变量又取得了另外一个值，则原先的引用值引用次数就减 1，当这个值的引用次数为 0 的时候，说明没有办法再访问这个值了，因此就把所占的内存给回收进来，这样垃圾收集器再次运行的时候，就会释放引用次数为 0 的这些值。

用引用计数法会存在内存泄露，下面来看原因：

```
function problem() {
var objA = new Object();
var objB = new Object();
objA.someOtherObject = objB;
objB.anotherObject = objA;
}
```

在这个例子里面，objA 和 objB 通过各自的属性相互引用，这样的话，两个对象的引用次数都为 2，在采用引用计数的策略中，由于函数执行之后，这两个对象都离开了作用域，函数执行完成之后，因为计数不为 0，这样的相互引用如果大量存在就会导致内存泄露。

特别是在 DOM 对象中，也容易存在这种问题：

```
var element=document.getElementById（’ ‘）；
var myObj=new Object();
myObj.element=element;
element.someObject=myObj;
```

这样就不会有垃圾回收的过程。

- eval 是做什么的

参考回答：
它的功能是将对应的字符串解析成 JS 并执行，应该避免使用 JS，因为非常消耗性能（2 次，一次解析成 JS，一次执行）

- 如何理解前端模块化

参考回答：
前端模块化就是复杂的文件编程一个一个独立的模块，比如 JS 文件等等，分成独立的模块有利于重用（复用性）和维护（版本迭代），这样会引来模块之间相互依赖的问题，所以有了 commonJS 规范，AMD，CMD 规范等等，以及用于 JS 打包（编译等处理）的工具 webpack

- 说一下 CommonJS、AMD 和 CMD

参考回答：
一个模块是能实现特定功能的文件，有了模块就可以方便的使用别人的代码，想要什么功能就能加载什么模块。
CommonJS：开始于服务器端的模块化，同步定义的模块化，每个模块都是一个单独的作用域，模块输出，modules.exports，模块加载 require()引入模块。
AMD：中文名异步模块定义的意思。
requireJS 实现了 AMD 规范，主要用于解决下述两个问题。
1.多个文件有依赖关系，被依赖的文件需要早于依赖它的文件加载到浏览器

2.加载的时候浏览器会停止页面渲染，加载文件越多，页面失去响应的时间越长。
语法：requireJS 定义了一个函数 define，它是全局变量，用来定义模块。
requireJS 的例子：

```
//定义模块
define(['dependency'], function(){
var name = 'Byron';
function printName(){
console.log(name);
}
return {
printName: printName
};
});
//加载模块
require(['myModule'], function (my){
my.printName();
}
```

RequireJS 定义了一个函数 define,它是全局变量，用来定义模块：

define(id?dependencies?,factory)
在页面上使用模块加载函数：
require([dependencies],factory)；
总结 AMD 规范：require（）函数在加载依赖函数的时候是异步加载的，这样浏览器不会失去响应，它指定的回调函数，只有前面的模块加载成功，才会去执行。
因为网页在加载 JS 的时候会停止渲染，因此我们可以通过异步的方式去加载 JS,而如果需要依赖某些，也是异步去依赖，依赖后再执行某些方法。

- 对象深度克隆的简单实现

参考回答：

```
function deepClone(obj){
var newObj= obj instanceof Array ? []:{};
for(var item in obj){
var temple= typeof obj[item] == 'object' ?
deepClone(obj[item]):obj[item];
newObj[item] = temple;
}
return newObj;
}
```

ES5 的常用的对象克隆的一种方式。注意数组是对象，但是跟对象又有一定区别，所以我们一开始判断了一些类型，决定 newObj 是对象还是数组。

- 实现一个 once 函数，传入函数参数只执行一次

参考回答：

```
function ones(func){
var tag=true;
return function(){
if(tag==true){
func.apply(null,arguments);
tag=false;
}
return undefined
}
}
```

- 将原生的 ajax 封装成 promise

参考回答：

```
var  myNewAjax=function(url){
return new Promise(function(resolve,reject){
```

```
var xhr = new XMLHttpRequest();
xhr.open('get',url);
xhr.send(data);
xhr.onreadystatechange=function(){
if(xhr.status==200&&readyState==4){
var json=JSON.parse(xhr.responseText);
resolve(json)
}else if(xhr.readyState==4&&xhr.status!=200){
reject('error');
}
}
})
}
```

- JS 监听对象属性的改变

参考回答：

我们假设这里有一个 user 对象,

(1)在 ES5 中可以通过 Object.defineProperty 来实现已有属性的监听

```
Object.defineProperty(user,'name',{
set：function(key,value){
}
})
```

缺点：如果 id 不在 user 对象中，则不能监听 id 的变化

(2)在 ES6 中可以通过 Proxy 来实现

```
var  user = new Proxy({}, {
set：function(target,key,value,receiver){
}
})
```

这样即使有属性在 user 中不存在，通过 user.id 来定义也同样可以这样监听这个属性的变化哦。

- 如何实现一个私有变量，用 getName 方法可以访问，不能直接访问

参考回答：

(1)通过 defineProperty 来实现

```
obj={
name:yuxiaoliang,
getName:function(){
return this.name
}
}
object.defineProperty(obj,"name",{
```

```
//不可枚举不可配置
});
(2)通过函数的创建形式
function product(){
var name='yuxiaoliang';
this.getName=function(){
return name;
}
}
var obj=new product();
```

- ==和===、以及 Object.is 的区别

参考回答：

(1) ==
主要存在：强制转换成 number,null==undefined
" "==0 //true
"0"==0 //true
" " !="0" //true
123=="123" //true
null==undefined //true
(2)Object.js
主要的区别就是+0! =-0 而 NaN==NaN
(相对比===和==的改进)

- setTimeout、setInterval 和 requestAnimationFrame 之间的区别

参考回答：

这里有一篇文章讲的是 requestAnimationFrame：
http://www.cnblogs.com/xiaohuochai/p/5777186.html
与 setTimeout 和 setInterval 不同，requestAnimationFrame 不需要设置时间间隔，大多数电脑显示器的刷新频率是 60Hz，大概相当于每秒钟重绘 60 次。大多数浏览器都会对重绘操作加以限制，不超过显示器的重绘频率，因为即使超过那个频率用户体验也不会有提升。因此，最平滑动画的最佳循环间隔是 1000ms/60，约等于 16.6ms。
RAF 采用的是系统时间间隔，不会因为前面的任务，不会影响 RAF，但是如果前面的任务多的话，
会响应 setTimeout 和 setInterval 真正运行时的时间间隔。
特点：
（1）requestAnimationFrame 会把每一帧中的所有 DOM 操作集中起来，在一次重绘或回流中就完成，并且重绘或回流的时间间隔紧紧跟随浏览器的刷新频率。
（2）在隐藏或不可见的元素中，requestAnimationFrame 将不会进行重绘或回流，这当然就意味着更少的 CPU、GPU 和内存使用量

（3）requestAnimationFrame 是由浏览器专门为动画提供的 API，在运行时浏览器会自动优化方法的调用，并且如果页面不是激活状态下的话，动画会自动暂停，有效节省了 CPU 开销。

- **实现一个两列等高布局，讲讲思路**

参考回答：
为了实现两列等高，可以给每列加上 padding-bottom:9999px; margin-bottom:-9999px;同时父元素设置 overflow:hidden;

- **自己实现一个 bind 函数**

参考回答：
原理：通过 apply 或者 call 方法来实现。
(1)初始版本

```
Function.prototype.bind=function(obj,arg){
var arg=Array.prototype.slice.call(arguments,1);
var context=this;
return function(newArg){
arg=arg.concat(Array.prototype.slice.call(newArg));
return context.apply(obj,arg);
}
}
```

(2) 考虑到原型链
为什么要考虑？因为在 new 一个 bind 过生成的新函数的时候，必须的条件是要继承原函数的原型

```
Function.prototype.bind=function(obj,arg){
var arg=Array.prototype.slice.call(arguments,1);
var context=this;
var bound=function(newArg){
arg=arg.concat(Array.prototype.slice.call(newArg));
return context.apply(obj,arg);
}
var F=function(){}
//这里需要一个寄生组合继承
F.prototype=context.prototype;
bound.prototype=new F();
return bound;
}
```

- **用 setTimeout 来实现 setInterval**

参考回答：

(1)用 setTimeout()方法来模拟 setInterval()与 setInterval()之间的什么区别？

首先来看 setInterval 的缺陷，使用 setInterval()创建的定时器确保了定时器代码规则地插入队列中。这个问题在于：如果定时器代码在代码再次添加到队列之前还没完成执行，结果就会导致定时器代码连续运行好几次。而之间没有间隔。不过幸运的是：javascript 引擎足够聪明，能够避免这个问题。当且仅当没有该定时器的如何代码实例时，才会将定时器代码添加到队列中。这确保了定时器代码加入队列中最小的时间间隔为指定时间。

这种重复定时器的规则有两个问题：1.某些间隔会被跳过 2.多个定时器的代码执行时间可能会比预期小。

下面举例子说明：

假设，某个 onclick 事件处理程序使用啦 setInterval()来设置了一个 200ms 的重复定时器。如果事件处理程序花了 300ms 多一点的时间完成。

这个例子中的第一个定时器是在 205ms 处添加到队列中，但是要过 300ms 才能执行。在 405ms 又添加了一个副本。在一个间隔，605ms 处，第一个定时器代码还在执行中，而且队列中已经有了一个定时器实例，结果是 605ms 的定时器代码不会添加到队列中。结果是在 5ms 处添加的定时器代码执行结束后，405 处的代码立即执行。

```
function say(){
//something
setTimeout(say,200);
}
setTimeout(say,200)
```

或者

```
setTimeout(function(){
//do something
setTimeout(arguments.callee,200);
},200);
```

- **JS 怎么控制一次加载一张图片，加载完后再加载下一张**

参考回答：

(1)方法 1

```
<script type="text/javascript">
var obj=new Image();
obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";
obj.onload=function(){
alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);
document.getElementById("mypic").innnerHTML="<img src='"+this.src+"' />";
```

```
}
</script>
<div id="mypic">onloading……</div>
```

(2)方法 2

```
<script type="text/javascript">
var obj=new Image();
obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";
obj.onreadystatechange=function(){
if(this.readyState=="complete"){
alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);
document.getElementById("mypic").innnerHTML="<img src='"+this.src+"' />";
}
}
</script>
<div id="mypic">onloading……</div>
```

- 代码的执行顺序

参考回答：

```
setTimeout(function(){console.log(1)},0);
new Promise(function(resolve,reject){
console.log(2);
resolve();
}).then(function(){console.log(3)
}).then(function(){console.log(4)});
process.nextTick(function(){console.log(5)});
console.log(6);
//输出 2,6,5,3,4,1
```

为什么呢？具体请参考这篇文章：

从 promise、process.nextTick、setTimeout 出发，谈谈 Event Loop 中的 Job queue

- 如何实现 sleep 的效果（es5 或者 es6）

参考回答：

(1)while 循环的方式

```
function sleep(ms){
var start=Date.now(),expire=start+ms;
while(Date.now()<expire);
console.log('1111');
return;
}
```

执行 sleep(1000)之后，休眠了 1000ms 之后输出了 1111。上述循环的方式缺点很明显，容易造成死循环。
(2)通过 promise 来实现

```
function sleep(ms){
var temple=new Promise(
(resolve)=>{
console.log(111);setTimeout(resolve,ms)
});
return temple
}
sleep(500).then(function(){
//console.log(222)
})
//先输出了 111，延迟 500ms 后输出 222
```

(3)通过 async 封装

```
function sleep(ms){
return new Promise((resolve)=>setTimeout(resolve,ms));
}
async function test(){
var temple=await sleep(1000);
console.log(1111)
return temple
}
test();
//延迟 1000ms 输出了 1111
```

(4).通过 generate 来实现

```
function* sleep(ms){
yield new Promise(function(resolve,reject){
console.log(111);
setTimeout(resolve,ms);
})
}
sleep(500).next().value.then(function(){console.log(2222)})
```

- **简单的实现一个 promise**

参考回答：
首先明确什么是 promiseA+规范，参考规范的地址：primise
A+规范
如何实现一个 promise，参考这篇文章：
实现一个完美符合 Promise/A+规范的 Promise
一般不会问的很详细，只要能写出上述文章中的 v1.0 版本的简单 promise 即可。

- Function._proto_(getPrototypeOf)是什么？

参考回答：
获取一个对象的原型，在 chrome 中可以通过_proto_的形式，或者在 ES6 中可以通过 Object.getPrototypeOf 的形式。
那么 Function.proto 是什么么？也就是说 Function 由什么对象继承而来，我们来做如下判别。
Function.__proto__==Object.prototype //false
Function.__proto__==Function.prototype//true
我们发现 Function 的原型也是 Function。
我们用图可以来明确这个关系：
<img width="646" alt="2018-07-10 2 38 27" src="https://user-images.githubusercontent.com/17233651/42493275-f55d0860-844e-11e8-983f-e04189a4f3d8.png">

- 实现 JS 中所有对象的深度克隆（包装对象，Date 对象，正则对象）

参考回答：
通过递归可以简单实现对象的深度克隆，但是这种方法不管是 ES6 还是 ES5 实现，都有同样的缺陷，就是只能实现特定的 object 的深度复制（比如数组和函数），不能实现包装对象 Number，String ， Boolean，以及 Date 对象，RegExp 对象的复制。
(1)前文的方法

```
function deepClone(obj){
var newObj= obj instanceof Array?[]:{};
for(var i in obj){
newObj[i]=typeof obj[i]=='object'?
deepClone(obj[i]):obj[i];
}
return newObj;
}
```

这种方法可以实现一般对象和数组对象的克隆，比如：

```
var arr=[1,2,3];
var newArr=deepClone(arr);
// newArr->[1,2,3]
var obj={
x:1,
y:2
}
var newObj=deepClone(obj);
// newObj={x:1,y:2}
```

但是不能实现例如包装对象 Number,String,Boolean,以及正则对象 RegExp 和 Date 对象的克隆，比如：

```
//Number 包装对象
var num=new Number(1);
```

```
typeof num // "object"
var newNum=deepClone(num);
//newNum ->   {} 空对象

//String 包装对象
var str=new String("hello");
typeof str //"object"
var newStr=deepClone(str);
//newStr->  {0:'h',1:'e',2:'l',3:'l',4:'o'};

//Boolean 包装对象
var bol=new Boolean(true);
typeof bol //"object"
var newBol=deepClone(bol);
// newBol ->{} 空对象


....
```

(2)valueof()函数

所有对象都有 valueOf 方法，valueOf 方法对于：如果存在任意原始值，它就默认将对象转换为表示它的原始值。对象是复合值，而且大多数对象无法真正表示为一个原始值，因此默认的 valueOf()方法简单地返回对象本身，而不是返回一个原始值。数组、函数和正则表达式简单地继承了这个默认方法，调用这些类型的实例的 valueOf()方法只是简单返回这个对象本身。

对于原始值或者包装类：

```
function baseClone(base){
return base.valueOf();
}
//Number
var num=new Number(1);
var newNum=baseClone(num);
//newNum->1
//String
var str=new String('hello');
var newStr=baseClone(str);
// newStr->"hello"
//Boolean
var bol=new Boolean(true);
var newBol=baseClone(bol);
//newBol-> true
```

其实对于包装类，完全可以用=号来进行克隆，其实没有深度克隆一说，

这里用 valueOf 实现，语法上比较符合规范。

对于 Date 类型：

因为 valueOf 方法，日期类定义的 valueOf()方法会返回它的一个内部表示：1970 年 1 月 1 日以来的毫秒数.因此我们可以在 Date 的原型上定义克隆的方法：

```
Date.prototype.clone=function(){
return new Date(this.valueOf());
}
var date=new Date('2010');
var newDate=date.clone();
// newDate->  Fri Jan 01 2010 08:00:00 GMT+0800
```
对于正则对象 RegExp：
```
RegExp.prototype.clone = function() {
var pattern = this.valueOf();
var flags = '';
flags += pattern.global ? 'g' : '';
flags += pattern.ignoreCase ? 'i' : '';
flags += pattern.multiline ? 'm' : '';
return new RegExp(pattern.source, flags);
};
var reg=new RegExp('/111/');
var newReg=reg.clone();
//newReg->  /\/111\//
```

- 简单实现 Node 的 Events 模块

参考回答：

简介：观察者模式或者说订阅模式，它定义了对象间的一种一对多的关系，让多个观察者对象同时监听某一个主题对象，当一个对象发生改变时，所有依赖于它的对象都将得到通知。

node 中的 Events 模块就是通过观察者模式来实现的：
```
var events=require('events');
var eventEmitter=new events.EventEmitter();
eventEmitter.on('say',function(name){
console.log('Hello',name);
})
eventEmitter.emit('say','Jony yu');
```
这样，eventEmitter 发出 say 事件，通过 On 接收，并且输出结果，这就是一个订阅模式的实现，下面我们来简单的实现一个 Events 模块的 EventEmitter。

(1)实现简单的 Event 模块的 emit 和 on 方法
```
function Events(){
this.on=function(eventName,callBack){
if(!this.handles){
this.handles={};
}
if(!this.handles[eventName]){
this.handles[eventName]=[];
}
```

```
this.handles[eventName].push(callBack);
}
this.emit=function(eventName,obj){
if(this.handles[eventName]){
for(var i=0;o<this.handles[eventName].length;i++){
this.handles[eventName][i](obj);
}
}
}
return this;
}
```

这样我们就定义了 Events，现在我们可以开始来调用：

```
var events=new Events();
events.on('say',function(name){
console.log('Hello',nama)
});
events.emit('say','Jony yu');
//结果就是通过 emit 调用之后，输出了 Jony yu
```

(2)每个对象是独立的

因为是通过 new 的方式，每次生成的对象都是不相同的，因此：

```
var event1=new Events();
var event2=new Events();
event1.on('say',function(){
console.log('Jony event1');
});
event2.on('say',function(){
console.log('Jony event2');
})
event1.emit('say');
event2.emit('say');
//event1、event2 之间的事件监听互相不影响
//输出结果为'Jony event1' 'Jony event2'
```

- **箭头函数中 this 指向举例**

参考回答：

```
var a=11;
function test2(){
this.a=22;
let b=()=>{console.log(this.a)}
b();
}
var x=new test2();
```

//输出 22
定义时绑定。

- JS 判断类型

参考回答：
判断方法：typeof()，instanceof，Object.prototype.toString.call()等

- 数组常用方法

参考回答：
push()，pop()，shift()，unshift()，splice()，sort()，reverse()，map()等

- 数组去重

参考回答：
法一：indexOf 循环去重
法二：ES6 Set 去重；Array.from(new Set(array))
法三：Object 键值对去重；把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的。

- 闭包　有什么用

参考回答：
（1）什么是闭包：
闭包是指有权访问另外一个函数作用域中的变量的函数。
闭包就是函数的局部变量集合，只是这些局部变量在函数返回后会继续存在。闭包就是就是函数的“堆栈”在函数返回后并不释放，我们也可以理解为这些函数堆栈并不在栈上分配而是在堆上分配。当在一个函数内定义另外一个函数就会产生闭包。
（2）为什么要用：
匿名自执行函数：我们知道所有的变量，如果不加上 var 关键字，则默认的会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，比如：别的函数可能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。除了每次使用变量都是用 var 关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，可以用闭包。
结果缓存：我们开发中会碰到很多情况，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。

封装：实现类和继承等。

- **事件代理在捕获阶段的实际应用**

参考回答：
可以在父元素层面阻止事件向子元素传播，也可代替子元素执行某些操作。

- **去除字符串首尾空格**

参考回答：
使用正则(^\s*)|(\s*$)即可

- **性能优化**

参考回答：
减少 HTTP 请求
使用内容发布网络（CDN）
添加本地缓存
压缩资源文件
将 CSS 样式表放在顶部，把 javascript 放在底部（浏览器的运行机制决定）
避免使用 CSS 表达式
减少 DNS 查询
使用外部 javascript 和 CSS
避免重定向
图片 lazyLoad

- **来讲讲 JS 的闭包吧**

参考回答：
闭包是指有权访问另外一个函数作用域中的变量的函数。
闭包就是函数的局部变量集合，只是这些局部变量在函数返回后会继续存在。闭包就是就是函数的“堆栈”在函数返回后并不释放，我们也可以理解为这些函数堆栈并不在栈上分配而是在堆上分配。当在一个函数内定义另外一个函数就会产生闭包。
（2）为什么要用：
匿名自执行函数：我们知道所有的变量，如果不加上 var 关键字，则默认的会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，比如：别的函数可能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。除了每次使用变量都是用 var 关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，可以用闭包。
结果缓存：我们开发中会碰到很多情况，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，那么我们就需要将计算出来的值存储起来，当调用

这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。

- **能来讲讲 JS 的语言特性吗**

参考回答：
运行在客户端浏览器上；
不用预编译，直接解析执行代码；
是弱类型语言，较为灵活；
与操作系统无关，跨平台的语言；
脚本语言、解释性语言

- **如何判断一个数组(讲到 typeof 差点掉坑里)**

参考回答：
Object.prototype.call.toString()
instanceof

- **你说到 typeof，能不能加一个限制条件达到判断条件**

参考回答：
typeof 只能判断是 object,可以判断一下是否拥有数组的方法

- **JS 实现跨域**

参考回答：
JSONP：通过动态创建 script，再请求一个带参网址实现跨域通信。document.domain + iframe 跨域：两个页面都通过 js 强制设置 document.domain 为基础主域，就实现了同域。
location.hash + iframe 跨域：a 欲与 b 跨域相互通信，通过中间页 c 来实现。 三个页面，不同域之间利用 iframe 的 location.hash 传值，相同域之间直接 js 访问来通信。
window.name + iframe 跨域：通过 iframe 的 src 属性由外域转向本地域，跨域数据即由 iframe 的 window.name 从外域传递到本地域。
postMessage 跨域：可以跨域操作的 window 属性之一。
CORS：服务端设置 Access-Control-Allow-Origin 即可，前端无须设置，若要带 cookie 请求，前后端都需要设置。
代理跨域：启一个代理服务器，实现数据的转发
参考 https://segmentfault.com/a/1190000011145364

- JS 基本数据类型

参考回答：
基本数据类型：undefined、null、number、boolean、string、symbol

- JS 深度拷贝一个元素的具体实现

参考回答：
```
var deepCopy = function(obj) {
if (typeof obj !== 'object') return;
var newObj = obj instanceof Array ? [] : {};
for (var key in obj) {
if (obj.hasOwnProperty(key)) {
newObj[key] = typeof obj[key] === 'object' ? deepCopy(obj[key]) :
obj[key];
}
}
return newObj;
}
```

- 之前说了 ES6set 可以数组去重，是否还有数组去重的方法

参考回答：
法一：indexOf 循环去重
法二：Object 键值对去重；把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的。

- 重排和重绘，讲讲看

参考回答：
重绘（repaint 或 redraw）：当盒子的位置、大小以及其他属性，例如颜色、字体大小等都确定下来之后，浏览器便把这些原色都按照各自的特性绘制一遍，将内容呈现在页面上。重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。
触发重绘的条件：改变元素外观属性。如：color，background-color 等。
注意：table 及其内部元素可能需要多次计算才能确定好其在渲染树中节点的属性值，比同等元素要多花两倍时间，这就是我们尽量避免使用 table 布局页面的原因之一。
重排（重构/回流/reflow）：当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建，这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。

重绘和重排的关系：在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新构造这部分渲染树，完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程称为重绘。所以，重排必定会引发重绘，但重绘不一定会引发重排。

- JS 的全排列

参考回答：

```
function permutate(str) {
var result = [];
if(str.length > 1) {
var left = str[0];
var rest = str.slice(1, str.length);
var preResult = permutate(rest);
for(var i=0; i<preResult.length; i++) {
for(var j=0; j<preResult[i].length; j++) {
var tmp = preResult[i],slice(0, j) + left + preResult[i].slice(j,
preResult[i].length);
result.push(tmp);
}
}
} else if (str.length == 1) {
return [str];
}
return result;
}
```

- 跨域的原理

参考回答：

跨域，是指浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的，是浏览器对 JavaScript 实施的安全限制，那么只要协议、域名、端口有任何一个不同，都被当作是不同的域。跨域原理，即是通过各种方式，避开浏览器的安全限制。

- 不同数据类型的值的比较，是怎么转换的，有什么规则

参考回答：

比较运算x==y, 其中x和 y是值，产生true或者false。这样的比较按如下方式进行：

1. 若Type(x)与Type(y)相同， 则
    a. 若Type(x)为Undefined， 返回true。
    b. 若Type(x)为Null， 返回true。
    c. 若Type(x)为Number， 则
        i. 若x为NaN， 返回false。
        ii. 若y为NaN， 返回false。
        iii. 若x与y为相等数值， 返回true。
        iv. 若x 为 +0 且 y为-0， 返回true。
        v. 若x 为 -0 且 y为+0， 返回true。
        vi. 返回false。
    d. 若Type(x)为String, 则当x和y为完全相同的字符序列（长度相等且相同字符在相同位置）时返回true。 否则, 返回false。
    e. 若Type(x)为Boolean, 当x和y为同为true或者同为false时返回true。 否则， 返回false。
    f. 当x和y为引用同一对象时返回true。否则，返回false。
2. 若x为null且y为undefined， 返回true。
3. 若x为undefined且y为null， 返回true。
4. 若Type(x) 为 Number 且 Type(y)为String， 返回comparison x == ToNumber(y)的结果。
5. 若Type(x) 为 String 且 Type(y)为Number,
6. 返回比较ToNumber(x) == y的结果。
7. 若Type(x)为Boolean， 返回比较ToNumber(x) == y的结果。
8. 若Type(y)为Boolean， 返回比较x == ToNumber(y)的结果。
9. 若Type(x)为String或Number，且Type(y)为Object，返回比较x == ToPrimitive(y)的结果。
10. 若Type(x)为Object且Type(y)为String或Number， 返回比较ToPrimitive(x) == y的结果。
11. 返回false。

- null == undefined 为什么

参考回答：
要比较相等性之前，不能将 null 和 undefined 转换成其他任何值，但 null == undefined 会返回 true 。ECMAScript 规范中是这样定义的。

- this 的指向 哪几种

参考回答：
默认绑定：全局环境中，this 默认绑定到 window。
隐式绑定：一般地，被直接对象所包含的函数调用时，也称为方法调用，this 隐式绑定到该直接对象。
隐式丢失：隐式丢失是指被隐式绑定的函数丢失绑定对象，从而默认绑定到 window。
显式绑定：通过 call()、apply()、bind()方法把对象绑定到 this 上，叫做显式绑定。
new 绑定：如果函数或者方法调用之前带有关键字 new，它就构成构造函数调用。对于 this 绑定来说，称为 new 绑定。
【1】构造函数通常不使用 return 关键字，它们通常初始化新对象，当构造函数的函数体执行完毕时，它会显式返回。在这种情况下，构造函数调用表达式的计算结果就是这个新对象的值。
【2】如果构造函数使用 return 语句但没有指定返回值，或者返回一个原始值，那么这时将忽略返回值，同时使用这个新对象作为调用结果。

【3】如果构造函数显式地使用 return 语句返回一个对象，那么调用表达式的值就是这个对象。

- 暂停死区

参考回答：
在代码块内，使用 let、const 命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”

- AngularJS 双向绑定原理

参考回答：
Angular 将双向绑定转换为一堆 watch 表达式，然后递归这些表达式检查是否发生过变化，如果变了则执行相应的 watcher 函数（指 view 上的指令，如 ng-bind，ng-show 等或是{{}}）。等到 model 中的值不再发生变化，也就不会再有 watcher 被触发，一个完整的 digest 循环就完成了。
Angular 中在 view 上声明的事件指令，如：ng-click、ng-change 等，会将浏览器的事件转发给$scope 上相应的 model 的响应函数。等待相应函数改变 model，紧接着触发脏检查机制刷新 view。
watch 表达式：可以是一个函数、可以是$scope 上的一个属性名，也可以是一个字符串形式的表达式。$watch 函数所监听的对象叫做 watch 表达式。watcher 函数：指在 view 上的指令（ngBind，ngShow、ngHide 等）以及{{}}表达式，他们所注册的函数。每一个 watcher 对象都包括：监听函数，上次变化的值，获取监听表达式的方法以及监听表达式，最后还包括是否需要使用深度对比（angular.equals()）

- 写一个深度拷贝

参考回答：
```
function clone( obj ) {
var copy;
switch( typeof obj ) {
case "undefined":
break;
case "number":
copy = obj - 0;
break;
case "string":
copy = obj + "";
break;
case "boolean":
copy = obj;
break;
```

```
case "object":   //object 分为两种情况 对象（Object）和数组（Array）

if(obj === null) {
copy = null;
} else {
if( Object.prototype.toString.call(obj).slice(8, -1) === "Array") {
copy = [];
for( var i = 0 ; i < obj.length ; i++ ) {
copy.push(clone(obj[i]));
}
} else {
copy = {};
for( var j in obj) {
copy[j] = clone(obj[j]);
}
}
}
break;
default:
copy = obj;
break;
}
return copy;
}
```

- **简历中提到了 requestAnimationFrame，请问是怎么使用的**

参考回答：

requestAnimationFrame() 方法告诉浏览器您希望执行动画并请求浏览器在下一次重绘之前调用指定的函数来更新动画。该方法使用一个回调函数作为参数，这个回调函数会在浏览器重绘之前调用。

- **有一个游戏叫做 Flappy Bird，就是一只小鸟在飞，前面是无尽的沙漠，上下不断有钢管生成，你要躲避钢管。然后小明在玩这个游戏时候老是卡顿甚至崩溃，说出原因（3-5 个）以及解决办法（3-5 个）**

参考回答：

原因可能是：
1.内存溢出问题。
2.资源过大问题。
3.资源加载问题。
4.canvas 绘制频率问题
解决办法：

1.针对内存溢出问题，我们应该在钢管离开可视区域后，销毁钢管，让垃圾收集器回收钢管，因为不断生成的钢管不及时清理容易导致内存溢出游戏崩溃。
2.针对资源过大问题，我们应该选择图片文件大小更小的图片格式，比如使用 webp、png 格式的图片，因为绘制图片需要较大计算量。
3.针对资源加载问题，我们应该在可视区域之前就预加载好资源，如果在可视区域生成钢管的话，用户的体验就认为钢管是卡顿后才生成的，不流畅。
4.针对 canvas 绘制频率问题，我们应该需要知道大部分显示器刷新频率为 60 次/s,因此游戏的每一帧绘制间隔时间需要小于 1000/60=16.7ms，才能让用户觉得不卡顿。
（注意因为这是单机游戏，所以回答与网络无关）

- **编写代码，满足以下条件： （1）Hero("37er");执行结果为 Hi! This is 37er （2）Hero("37er").kill(1).recover(30);执行结果为 Hi! This is 37er Kill 1 bug Recover 30 bloods （3）Hero("37er").sleep(10).kill(2)执行结果为 Hi! This is 37er //等待 10s 后 Kill 2 bugs //注意为 bugs （双斜线后的为提示信息，不需要打印）**

参考回答：

```
function Hero(name){
let o=new Object();
o.name=name;
o.time=0;
console.log("Hi! This is "+o.name);
o.kill=function(bugs) {
if(bugs==1){
console.log("Kill "+(bugs)+" bug");
}else {
setTimeout(function () {
console.log("Kill " + (bugs) + " bugs");
}, 1000 * this.time);
}
return o;
};
o.recover=function (bloods) {
console.log("Recover "+(bloods)+" bloods");
return o;
}
o.sleep=function (sleepTime) {
o.time=sleepTime;
return o;
}
return o;
}
```

- 什么是按需加载

参考回答：
当用户触发了动作时才加载对应的功能。触发的动作，是要看具体的业务场景而言，包括但不限于以下几个情况：鼠标点击、输入文字、拉动滚动条，鼠标移动、窗口大小更改等。加载的文件，可以是 JS、图片、CSS、HTML 等。

- 说一下什么是 virtual dom

参考回答：
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。

- webpack 用来干什么的

参考回答：
webpack 是一个现代 JavaScript 应用程序的静态模块打包器(module bundler)。当 webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle。

- ant-design 优点和缺点

参考回答：
优点：组件非常全面，样式效果也都比较不错。
缺点：框架自定义程度低，默认 UI 风格修改困难。

- JS 中继承实现的几种方式，

参考回答：
1、原型链继承，将父类的实例作为子类的原型，他的特点是实例是子类的实例也是父类的实例，父类新增的原型方法/属性，子类都能够访问，并且原型链继承简单易于实现，缺点是来自原型对象的所有属性被所有实例共享，无法实现多继承，无法向父类构造函数传参。
2、构造继承，使用父类的构造函数来增强子类实例，即复制父类的实例属性给子类，构造继承可以向父类传递参数，可以实现多继承，通过 call 多个父类对象。但是构造继承只能继承父类的实例属性和方法，不能继承原型属性和方法，无法实现函数服用，每个子类都有父类实例函数的副本，影响性能

3、实例继承，为父类实例添加新特性，作为子类实例返回，实例继承的特点是不限制调用方法，不管是 new 子类（）还是子类（）返回的对象具有相同的效果，缺点是实例是父类的实例，不是子类的实例，不支持多继承
4、拷贝继承：特点：支持多继承，缺点：效率较低，内存占用高（因为要拷贝父类的属性）无法获取父类不可枚举的方法（不可枚举方法，不能使用 for in 访问到）
5、组合继承：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
6、寄生组合继承：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性，避免的组合继承的缺点

- **写一个函数，第一秒打印 1，第二秒打印 2**

参考回答：
两个方法，第一个是用 let 块级作用域

```
for(let i=0;i<5;i++){
setTimeout(function(){
console.log(i)
},1000*i)
}
```

第二个方法闭包

```
for(var i=0;i<5;i++){
(function(i){
setTimeout(function(){
console.log(i)
},1000*i)
})(i)
}
```

- **Vue 的生命周期**

参考回答：
Vue 实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模板、挂载 Dom、渲染→更新→渲染、销毁等一系列过程，我们称这是 Vue 的生命周期。通俗说就是 Vue 实例从创建到销毁的过程，就是生命周期。
每一个组件或者实例都会经历一个完整的生命周期，总共分为三个阶段：初始化、运行中、销毁。
实例、组件通过 new Vue() 创建出来之后会初始化事件和生命周期，然后就会执行 beforeCreate 钩子函数，这个时候，数据还没有挂载呢，只是一个空壳，无法访问到数据和真实的 dom，一般不做操作
挂载数据，绑定事件等等，然后执行 created 函数，这个时候已经可以使用到数据，也可以更改数据,在这里更改数据不会触发 updated 函数，在这里可以在渲染前倒数第二次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取

接下来开始找实例或者组件对应的模板，编译模板为虚拟 dom 放入到 render 函数中准备渲染，然后执行 beforeMount 钩子函数，在这个函数中虚拟 dom 已经创建完成，马上就要渲染,在这里也可以更改数据，不会触发 updated，在这里可以在渲染前最后一次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取
接下来开始 render，渲染出真实 dom，然后执行 mounted 钩子函数，此时，组件已经出现在页面中，数据、真实 dom 都已经处理好了,事件都已经挂载好了，可以在这里操作真实 dom 等事情...
当组件或实例的数据更改之后，会立即执行 beforeUpdate，然后 Vue 的虚拟 dom 机制会重新构建虚拟 dom 与上一次的虚拟 dom 树利用 diff 算法进行对比之后重新渲染，一般不做什么事儿
当更新完成后，执行 updated，数据已经更改完成，dom 也重新 render 完成，可以操作更新后的虚拟 dom
当经过某种途径调用$destroy 方法后，立即执行 beforeDestroy，一般在这里做一些善后工作，例如清除计时器、清除非指令绑定的事件等等
组件的数据绑定、监听...去掉后只剩下 dom 空壳，这个时候，执行 destroyed，在这里做善后工作也可以

- 简单介绍一下 symbol

参考回答：
Symbol 是 ES6 的新增属性，代表用给定名称作为唯一标识，这种类型的值可以这样创建，let id=symbol(“id”)
Symbl 确保唯一，即使采用相同的名称，也会产生不同的值，我们创建一个字段，仅为知道对应 symbol 的人能访问，使用 symbol 很有用，symbol 并不是 100%隐藏，有内置方法 Object.getOwnPropertySymbols(obj)可以获得所有的 symbol。
也有一个方法 Reflect.ownKeys(obj)返回对象所有的键，包括 symbol。
所以并不是真正隐藏。但大多数库内置方法和语法结构遵循通用约定他们是隐藏的。

- 什么是事件监听

参考回答：
addEventListener()方法，用于向指定元素添加事件句柄，它可以更简单的控制事件，语法为
element.addEventListener(event, function, useCapture);
第一个参数是事件的类型(如 "click" 或 "mousedown").
第二个参数是事件触发后调用的函数。
第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。
事件传递有两种方式，冒泡和捕获
事件传递定义了元素事件触发的顺序，如果你将 P 元素插入到 div 元素中，用户点击 P 元素，
在冒泡中，内部元素先被触发，然后再触发外部元素，
捕获中，外部元素先被触发，在触发内部元素。

- 介绍一下 promise，及其底层如何实现

参考回答：

Promise 是一个对象，保存着未来将要结束的事件，她有两个特征:
1、对象的状态不受外部影响，Promise 对象代表一个异步操作，有三种状态，pending 进行中，fulfilled 已成功，rejected 已失败，只有异步操作的结果，才可以决定当前是哪一种状态，任何其他操作都无法改变这个状态，这也就是 promise 名字的由来
2、一旦状态改变，就不会再变，promise 对象状态改变只有两种可能，从 pending 改到 fulfilled 或者从 pending 改到 rejected，只要这两种情况发生，状态就凝固了，不会再改变，这个时候就称为定型 resolved,
Promise 的基本用法，

```
let promise1 = new Promise(function(resolve,reject){
setTimeout(function(){
resolve('ok')
},1000)
})
promise1.then(function success(val){
console.log(val)
})
```

最简单代码实现 promise

```
class PromiseM {
constructor (process) {
this.status = 'pending'
this.msg = ''
process(this.resolve.bind(this), this.reject.bind(this))
return this
}
resolve (val) {
this.status = 'fulfilled'
this.msg = val
}
reject (err) {
this.status = 'rejected'
this.msg = err
}
then (fufilled, reject) {
if(this.status === 'fulfilled') {
fufilled(this.msg)
}
if(this.status === 'rejected') {
reject(this.msg)
}
}
}
```

```
//测试代码
var mm=new PromiseM(function(resolve,reject){
resolve('123');
});
mm.then(function(success){
console.log(success);
},function(){
console.log('fail!');
});
```

- **说说 C++,Java，JavaScript 这三种语言的区别**

参考回答：

从静态类型还是动态类型来看

静态类型，编译的时候就能够知道每个变量的类型，编程的时候也需要给定类型，如 Java 中的整型 int，浮点型 float 等。C、C++、Java 都属于静态类型语言。

动态类型，运行的时候才知道每个变量的类型，编程的时候无需显示指定类型，如 JavaScript 中的 var、PHP 中的$。JavaScript、Ruby、Python 都属于动态类型语言。

静态类型还是动态类型对语言的性能有很大影响。

对于静态类型，在编译后会大量利用已知类型的优势，如 int 类型，占用 4 个字节，编译后的代码就可以用内存地址加偏移量的方法存取变量，而地址加偏移量的算法汇编很容易实现。

对于动态类型，会当做字符串通通存下来，之后存取就用字符串匹配。

从编译型还是解释型来看

编译型语言，像 C、C++，需要编译器编译成本地可执行程序后才能运行，由开发人员在编写完成后手动实施。用户只使用这些编译好的本地代码，这些本地代码由系统加载器执行，由操作系统的 CPU 直接执行，无需其他额外的虚拟机等。

源代码=》抽象语法树=》中间表示=》本地代码

解释性语言，像 JavaScript、Python，开发语言写好后直接将代码交给用户，用户使用脚本解释器将脚本文件解释执行。对于脚本语言，没有开发人员的编译过程，当然，也不绝对。

源代码=》抽象语法树=》解释器解释执行。

对于 JavaScript，随着 Java 虚拟机 JIT 技术的引入，工作方式也发生了改变。可以将抽象语法树转成中间表示（字节码），再转成本地代码，如 JavaScriptCore，这样可以大大提高执行效率。也可以从抽象语法树直接转成本地代码，如 V8

Java 语言，分为两个阶段。首先像 C++语言一样，经过编译器编译。和 C++的不同，C++编译生成本地代码，Java 编译后，生成字节码，字节码与平台无关。第二阶段，由 Java 的运行环境也就是 Java 虚拟机运行字节码，使用解释器执行这些代码。一般情况下，Java 虚拟机都引入了 JIT 技术，将字节码转换成本地代码来提高执行效率。

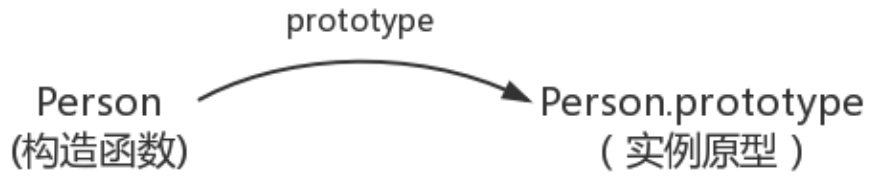
注意，在上述情况中，编译器的编译过程没有时间要求，所以编译器可以做大量的代码优化措施。

对于 JavaScript 与 Java 它们还有的不同：

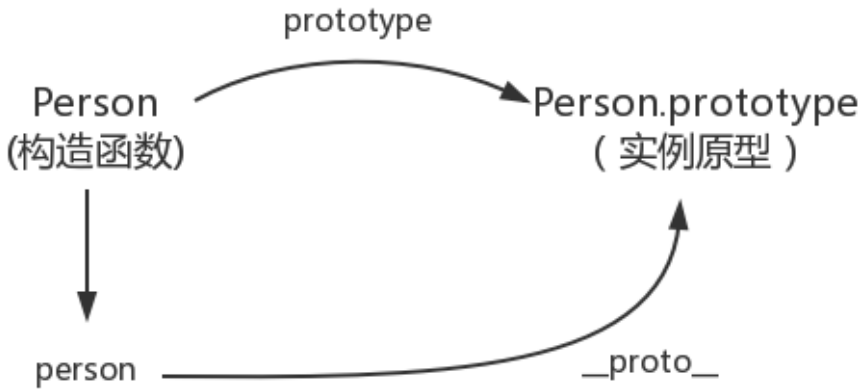
对于 Java，Java 语言将源代码编译成字节码，这个同执行阶段是分开的。也就是从源代码到抽象语法树到字节码这段时间的长短是无所谓的。
对于 JavaScript，这些都是在网页和 JavaScript 文件下载后同执行阶段一起在网页的加载和渲染过程中实施的，所以对于它们的处理时间有严格要求。

- **JS 原型链，原型链的顶端是什么？Object 的原型是什么？Object 的原型的原型是什么？在数组原型链上实现删除数组重复数据的方法**

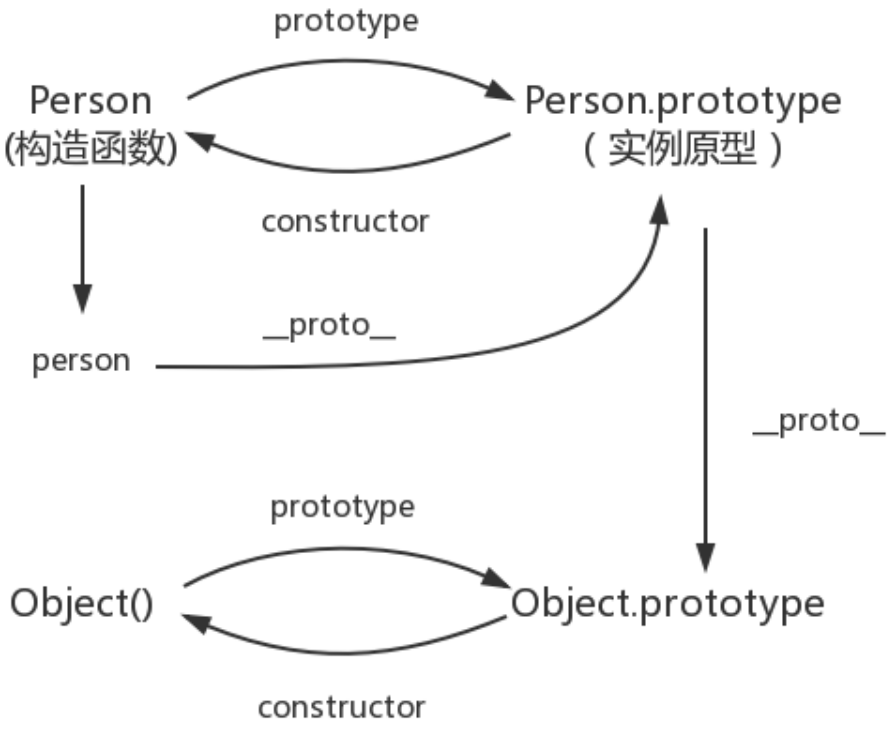
参考回答：
能够把这个讲清楚弄明白是一件很困难的事，
首先明白原型是什么，在 ES6 之前，JS 没有类和继承的概念，JS 是通过原型来实现继承的，在 JS 中一个构造函数默认带有一个 prototype 属性，这个的属性值是一个对象，同时这个 prototype 对象自带有一个 constructor 属性，这个属性指向这个构造函数，同时每一个实例都会有一个_proto_属性指向这个 prototype 对象，我们可以把这个叫做隐式原型，我们在使用一个实例的方法的时候，会先检查这个实例中是否有这个方法，没有的话就会检查这个 prototype 对象是否有这个方法，
基于这个规则，如果让原型对象指向另一个类型的实例，即 constructor1.protoytpe=instance2，这时候如果试图引用 constructor1 构造的实例 instance1 的某个属性 p1，
首先会在 instance1 内部属性中找一遍，
接着会在 instance1._proto_（constructor1.prototype）即是 instance2 中寻找 p1
搜寻轨迹：
instance1->instance2->constructor2.prototype……->Object.prototype;这即是原型链，原型链顶端是 Object.prototype
补充学习：
每个函数都有一个 prototype 属性，这个属性指向了一个对象，这个对象正是调用该函数而创建的实例的原型，那么什么是原型呢，可以这样理解，每一个 JavaScript 对象在创建的时候就会预制管理另一个对象，这个对象就是我们所说的原型，每一个对象都会从原型继承属性，如图：



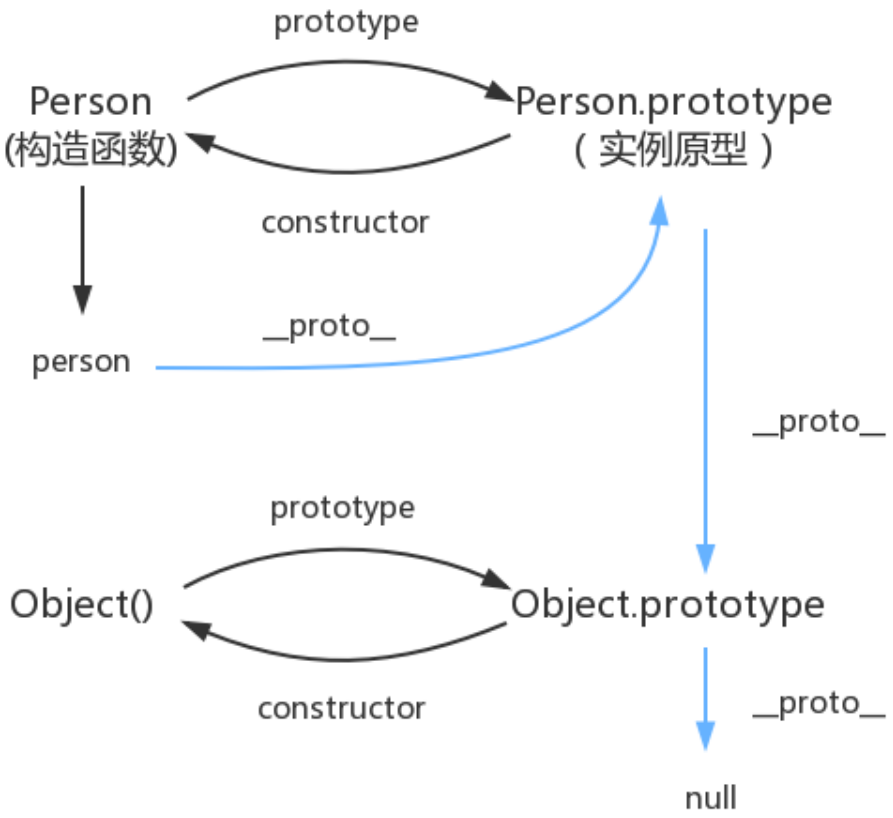
那么怎么表示实例与实例原型的关系呢，这时候就要用到第二个属性_proto_
这是每一个 JS 对象都会有的一个属性，指向这个对象的原型，如图：



既然实例对象和构造函数都可以指向原型，那么原型是否有属性指向构造函数或者实例呢，指向实例是没有的，因为一个构造函数可以生成多个实例，但是原型有属性可以直接指向构造函数，通过 constructor 即可
接下来讲解实例和原型的关系：
当读取实例的属性时，如果找不到，就会查找与对象相关的原型中的属性，如果还查不到，就去找原型的原型，一直找到最顶层，那么原型的原型是什么呢，首先，原型也是一个对象，既然是对象，我们就可以通过构造函数的方式创建它，所以原型对象就是通过 Object 构造函数生成的，如图：



那么 Object.prototype 的原型呢，我们可以打印
console.log(Object.prototype.__proto__ === null)，返回 true
null 表示没有对象，即该处不应有值，所以 Object.prototype 没有原型，如图：



图中这条蓝色的线即是原型链，
最后补充三点：
constructor：

```
function Person(){

}
var person = new Person();
console.log(Person === person.constructor);
```

原本 person 中没有 constructor 属性，当不能读取到 constructor 属性时，会从 person 的原型中读取，所以指向构造函数 Person
__proto__：
绝大部分浏览器支持这个非标准的方法访问原型，然而它并不存在与 Person.prototype 中，实际上它来自 Object.prototype，当使用 obj.__proto__时，可以理解为返回来 Object.getPrototype(obj)
继承：

前面说到，每个对象都会从原型继承属性，但是引用《你不知道的 JS》中的话，继承意味着复制操作，然而 JS 默认不会复制对象的属性，相反，JS 只是在两个对象之间创建一个关联，这样子一个对象就可以通过委托访问另一个对象的属性和函数，所以与其叫继承，叫委托更合适。

- **什么是 js 的闭包？有什么作用，用闭包写个单例模式**

参考回答：
MDN 对闭包的定义是：闭包是指那些能够访问自由变量的函数，自由变量是指在函数中使用的，但既不是函数参数又不是函数的局部变量的变量，由此可以看出，闭包=函数+函数能够访问的自由变量，所以从技术的角度讲，所有 JS 函数都是闭包，但是这是理论上的闭包，还有一个实践角度上的闭包，从实践角度上来说，只有满足 1、即使创建它的上下文已经销毁，它仍然存在，2、在代码中引入了自由变量，才称为闭包
闭包的应用：
模仿块级作用域。2、保存外部函数的变量。3、封装私有变量
单例模式：

```
var Singleton = (function(){
var instance;
var CreateSingleton = function (name) {
this.name = name;
if(instance) {
return instance;
}
// 打印实例名字
this.getName();
// instance = this;
// return instance;
return instance = this;
}
// 获取实例的名字
CreateSingleton.prototype.getName = function() {
console.log(this.name)
}
return CreateSingleton;
})();
// 创建实例对象 1
var a = new Singleton('a');
// 创建实例对象 2
var b = new Singleton('b');
console.log(a===b);
```

- promise+Generator+Async 的使用

参考回答：

Promise

解决的问题:回调地狱

Promise 规范:

promise 有三种状态，等待（pending）、已完成（fulfilled/resolved）、已拒绝（rejected）.Promise 的状态只能从“等待”转到“完成”或者“拒绝”，不能逆向转换，同时“完成”和“拒绝”也不能相互转换.

promise 必须提供一个 then 方法以访问其当前值、终值和据因。

promise.then(resolve, reject),resolve 和 reject 都是可选参数。如果 resolve 或 reject 不是函数，其必须被忽略.

then 方法必须返回一个 promise 对象.

使用:

实例化 promise 对象需要传入函数(包含两个参数),resolve 和 reject,内部确定状态.resolve 和 reject 函数可以传入参数在回调函数中使用.

resolve 和 reject 都是函数,传入的参数在 then 的回调函数中接收.

```
var promise = new Promise(function(resolve, reject) {
setTimeout(function(){
resolve('好哈哈哈哈');
});
});
promise.then(function(val){
console.log(val)
})
```

then 接收两个函数,分别对应 resolve 和 reject 状态的回调,函数中接收实例化时传入的参数.

```
promise.then(val=>{
//resolved
},reason=>{
//rejected
})
```

catch 相当于.then(null, rejection)

当 then 中没有传入 rejection 时,错误会冒泡进入 catch 函数中,若传入了 rejection,则错误会被 rejection 捕获,而且不会进入 catch.此外,then 中的回调函数中发生的错误只会在下一级的 then 中被捕获,不会影响该 promise 的状态.

```
new Promise((resolve,reject)=>{
throw new Error('错误')
}).then(null,(err)=>{
console.log(err,1);//此处捕获
}).catch((err)=>{
console.log(err,2);
});

// 对比
new Promise((resolve,reject)=>{
```

```
throw new Error('错误')
}).then(null,null).catch((err)=>{
console.log(err,2);//此处捕获
});
// 错误示例
new Promise((resolve,reject)=>{
resolve('正常');
}).then((val)=>{
throw new Error('回调函数中错误')
},(err)=>{
console.log(err,1);
}).then(null,(err)=>{
console.log(err,2);//此处捕获,也可用 catch
});
```

两者不等价的情况:
此时，catch 捕获的并不是 p1 的错误，而是 p2 的错误，

```
p1().then(res=>{
return p2()//p2 返回一个 promise 对象
}).catch(err=> console.log(err))
```

一个错误捕获的错误用例:
该函数调用中即使发生了错误依然会进入 then 中的 resolve 的回调函数,因为函数 p1 中实例化 promise 对象时已经调用了 catch,若发生错误会进入 catch 中,此时会返回一个新的 promise,因此即使发生错误依然会进入 p1 函数的 then 链中的 resolve 回调函数.

```
function p1(val){
return new Promise((resolve,reject)=>{
if(val){
var len = val.length;//传入 null 会发生错误,进入 catch 捕获错
resolve(len);
}else{
reject();
}
}).catch((err)=>{
console.log(err)
})
};
p1(null).then((len)=>{
console.log(len,'resolved');
},()=>{
console.log('rejected');
}).catch((err)=>{
```

```
console.log(err,'catch');
})
```

Promise 回调链:

promise 能够在回调函数里面使用 return 和 throw， 所以在 then 中可以 return 出一个 promise 对象或其他值，也可以 throw 出一个错误对象，但如果没有 return，将默认返回 undefined，那么后面的 then 中的回调参数接收到的将是 undefined.

```
function p1(val){
return new Promise((resolve,reject)=>{
val==1?resolve(1):reject()
})
};
function p2(val){
return new Promise((resolve,reject)=>{
val==2?resolve(2):reject();
})
};
let promimse = new Promise(function(resolve,reject){
resolve(1)
})
.then(function(data1) {
return p1(data1)//如果去掉 return,则返回 undefined 而不是 p1 的返回值,会导致报错
})
.then(function(data2){
return p2(data2+1)
})
.then(res=>console.log(res))
```

Generator 函数:

generator 函数使用:

1、分段执行，可以暂停

2、可以控制阶段和每个阶段的返回值

3、可以知道是否执行到结尾

```
function* g() {
var o = 1;
yield o++;
yield o++;
}
var gen = g();
console.log(gen.next()); //  Object {value: 1, done: false}
var xxx = g();
console.log(gen.next()); // Object {value: 2, done: false}
console.log(xxx.next()); // Object {value: 1, done: false}
console.log(gen.next()); // Object {value: undefined, done: true}
```

generator 和异步控制:
利用 Generator 函数的暂停执行的效果，可以把异步操作写在 yield 语句里面，等到调用 next 方法时再往后执行。这实际上等同于不需要写回调函数了，因为异步操作的后续操作可以放在 yield 语句下面，反正要等到调用 next 方法时再执行。所以，Generator 函数的一个重要实际意义就是用来处理异步操作，改写回调函数。
async 和异步:
用法:
async 表示这是一个 async 函数，await 只能用在这个函数里面。
await 表示在这里等待异步操作返回结果，再继续执行。
await 后一般是一个 promise 对象
示例:async 用于定义一个异步函数，该函数返回一个 Promise。
如果 async 函数返回的是一个同步的值，这个值将被包装成一个理解 resolve 的 Promise，等同于 return Promise.resolve(value)。
await 用于一个异步操作之前，表示要“等待”这个异步操作的返回值。await 也可以用于一个同步的值。

```
let timer = async function timer(){
return new Promise((resolve,reject) => {
setTimeout(() => {
resolve('500');
},500);
});
}
timer().then(result => {
console.log(result);  //500
}).catch(err => {
console.log(err.message);
});


//返回一个同步的值
let sayHi = async function sayHi(){
let hi = await 'hello world';
return hi;  //等同于 return Promise.resolve(hi);
}
sayHi().then(result => {
console.log(result);
});
```

- **事件委托以及冒泡原理。**

参考回答：
事件委托是利用冒泡阶段的运行机制来实现的，就是把一个元素响应事件的函数委托到另一个元素，一般是把一组元素的事件委托到他的父元素上，委托的优点是

减少内存消耗，节约效率
动态绑定事件
事件冒泡，就是元素自身的事件被触发后，如果父元素有相同的事件，如 onclick 事件，那么元素本身的触发状态就会传递，也就是冒到父元素，父元素的相同事件也会一级一级根据嵌套关系向外触发，直到 document/window，冒泡过程结束。

- **写个函数，可以转化下划线命名到驼峰命名**

参考回答：

```
public static String UnderlineToHump(String para){
StringBuilder result=new StringBuilder();
String a[]=para.split("_");
for(String s:a){
if(result.length()==0){
result.append(s.toLowerCase());
}else{
result.append(s.substring(0, 1).toUpperCase());
result.append(s.substring(1).toLowerCase());
}
}
return result.toString();
}
}
```

- **深浅拷贝的区别和实现**

参考回答：

数组的浅拷贝：
如果是数组，我们可以利用数组的一些方法，比如 slice，concat 方法返回一个新数组的特性来实现拷贝，但假如数组嵌套了对象或者数组的话，使用 concat 方法克隆并不完整，如果数组元素是基本类型，就会拷贝一份，互不影响，而如果是对象或数组，就会只拷贝对象和数组的引用，这样我们无论在新旧数组进行了修改，两者都会发生变化，我们把这种复制引用的拷贝方法称为浅拷贝，
深拷贝就是指完全的拷贝一个对象，即使嵌套了对象，两者也互相分离，修改一个对象的属性，不会影响另一个
如何深拷贝一个数组
1、这里介绍一个技巧，不仅适用于数组还适用于对象！那就是：

```
var arr = ['old', 1, true, ['old1', 'old2'], {old: 1}]
var new_arr = JSON.parse( JSON.stringify(arr) );
console.log(new_arr);
```

原理是 JOSN 对象中的 stringify 可以把一个 js 对象序列化为一个 JSON 字符串，parse 可以把 JSON 字符串反序列化为一个 js 对象，通过这两个方法，也可以实现对象的深复制。

但是这个方法不能够拷贝函数
浅拷贝的实现：
以上三个方法 concat,slice ,JSON.stringify 都是技巧类，根据实际项目情况选择使用，我们可以思考下如何实现一个对象或数组的浅拷贝，遍历对象，然后把属性和属性值都放在一个新的对象里即可

```
var shallowCopy = function(obj) {
// 只拷贝对象
if (typeof obj !== 'object') return;


// 根据 obj 的类型判断是新建一个数组还是对象
var newObj = obj instanceof Array ? [] : {};


// 遍历 obj，并且判断是 obj 的属性才拷贝


for (var key in obj) {
if (obj.hasOwnProperty(key)) {
newObj[key] = obj[key];
}
}
return newObj;
```

深拷贝的实现
那如何实现一个深拷贝呢？说起来也好简单，我们在拷贝的时候判断一下属性值的类型，如果是对象，我们递归调用深拷贝函数不就好了~

```
var deepCopy = function(obj) {
if (typeof obj !== 'object') return;
var newObj = obj instanceof Array ? [] : {};
for (var key in obj) {
if (obj.hasOwnProperty(key)) {
newObj[key] = typeof obj[key] === 'object' ? deepCopy(obj[key]) :
obj[key];
}
}
return newObj;
}
```

- JS 中 string 的 startwith 和 indexof 两种方法的区别

参考回答：

JS 中 startwith 函数，其参数有 3 个，stringObj,要搜索的字符串对象，str，搜索的字符串，position，可选，从哪个位置开始搜索，如果以 position 开始的字符串以搜索字符串开头，则返回 true，否则返回 false
Indexof 函数，indexof 函数可返回某个指定字符串在字符串中首次出现的位置。

- JS 字符串转数字的方法

参考回答：
通过函数 parseInt（），可解析一个字符串，并返回一个整数，语法为 parseInt（string ,radix）
string：被解析的字符串
radix：表示要解析的数字的基数，默认是十进制，如果 radix<2 或>36,则返回 NaN

- let const var 的区别 ，什么是块级作用域，如何用 ES5 的方法实现块级作用域（立即执行函数），ES6 呢

参考回答：
提起这三个最明显的区别是 var 声明的变量是全局或者整个函数块的，而 let,const 声明的变量是块级的变量，var 声明的变量存在变量提升，let,const 不存在，let 声明的变量允许重新赋值，const 不允许。

- ES6 箭头函数的特性

参考回答：
ES6 增加了箭头函数，基本语法为
```
let func = value => value;
```
相当于
```
let func = function (value) {
return value;
};
```
箭头函数与普通函数的区别在于：
1、箭头函数没有 this，所以需要通过查找作用域链来确定 this 的值，这就意味着如果箭头函数被非箭头函数包含，this 绑定的就是最近一层非箭头函数的 this，
2、箭头函数没有自己的 arguments 对象，但是可以访问外围函数的 arguments 对象
3、不能通过 new 关键字调用，同样也没有 new.target 值和原型

- setTimeout 和 Promise 的执行顺序

参考回答：
首先我们来看这样一道题：
```
setTimeout(function() {
```

```
console.log(1)
}, 0);
new Promise(function(resolve, reject) {
console.log(2)
for (var i = 0; i < 10000; i++) {
if(i === 10) {console.log(10)}
i == 9999 && resolve();
}
console.log(3)
}).then(function() {
console.log(4)
})
console.log(5);
```

输出答案为2 10 3 5 4 1

要先弄清楚 settimeout（fun,0）何时执行，promise 何时执行，then 何时执行

settimeout 这种异步操作的回调，只有主线程中没有执行任何同步代码的前提下，才会执行异步回调，而 settimeout（fun,0）表示立刻执行，也就是用来改变任务的执行顺序，要求浏览器尽可能快的进行回调

promise 何时执行，由上图可知 promise 新建后立即执行，所以 promise 构造函数里代码同步执行的，

then 方法指向的回调将在当前脚本所有同步任务执行完成后执行，

那么 then 为什么比 settimeout 执行的早呢，因为 settimeout（fun,0）不是真的立即执行，

经过测试得出结论：执行顺序为：同步执行的代码-》promise.then->settimeout

- **有了解过事件模型吗，DOM0 级和 DOM2 级有什么区别，DOM 的分级是什么**

参考回答：

JSDOM 事件流存在如下三个阶段：

事件捕获阶段

处于目标阶段

事件冒泡阶段

JSDOM 标准事件流的触发的先后顺序为：先捕获再冒泡，点击 DOM 节点时，事件传播顺序：事件捕获阶段，从上往下传播，然后到达事件目标节点，最后是冒泡阶段，从下往上传播

DOM 节点添加事件监听方法 addEventListener，中参数 capture 可以指定该监听是添加在事件捕获阶段还是事件冒泡阶段，为 false 是事件冒泡，为 true 是事件捕获，并非所有的事件都支持冒泡，比如 focus，blur 等等，我们可以通过 event.bubbles 来判断

事件模型有三个常用方法：

event.stopPropagation:阻止捕获和冒泡阶段中，当前事件的进一步传播，

event.stopImmediatePropagetion，阻止调用相同事件的其他侦听器，

event.preventDefault，取消该事件（假如事件是可取消的）而不停止事件的进一步传播，
event.target：指向触发事件的元素，在事件冒泡过程中这个值不变
event.currentTarget = this，时间帮顶的当前元素，只有被点击时目标元素的 target 才会等于 currentTarget，
最后，对于执行顺序的问题，如果 DOM 节点同时绑定了两个事件监听函数，一个用于捕获，一个用于冒泡，那么两个事件的执行顺序真的是先捕获在冒泡吗，答案是否定的，绑定在被点击元素的事件是按照代码添加顺序执行的，其他函数是先捕获再冒泡

- **平时是怎么调试 JS 的**

参考回答：
一般用 Chrome 自带的控制台

- **JS 的基本数据类型有哪些，基本数据类型和引用数据类型的区别，NaN 是什么的缩写，JS 的作用域类型，undefined==null 返回的结果是什么，undefined 与 null 的区别在哪，写一个函数判断变量类型**

参考回答：
JS 的基本数据类型有字符串，数字，布尔，数组，对象，Null，Undefined,基本数据类型是按值访问的，也就是说我们可以操作保存在变量中的实际的值，
基本数据类型和引用数据类型的区别如下：
基本数据类型的值是不可变的，任何方法都无法改变一个基本类型的值，当这个变量重新赋值后看起来变量的值是改变了，但是这里变量名只是指向变量的一个指针，所以改变的是指针的指向改变，该变量是不变的，但是引用类型可以改变
基本数据类型不可以添加属性和方法，但是引用类型可以
基本数据类型的赋值是简单赋值，如果从一个变量向另一个变量赋值基本类型的值，会在变量对象上创建一个新值，然后把该值复制到为新变量分配的位置上，引用数据类型的赋值是对象引用，
基本数据类型的比较是值的比较，引用类型的比较是引用的比较，比较对象的内存地址是否相同
基本数据类型是存放在栈区的，引用数据类型同事保存在栈区和堆区
NaN 是 JS 中的特殊值，表示非数字，NaN 不是数字，但是他的数据类型是数字，它不等于任何值，包括自身，在布尔运算时被当做 false，NaN 与任何数运算得到的结果都是 NaN，党员算失败或者运算无法返回正确的数值的就会返回 NaN，一些数学函数的运算结果也会出现 NaN ，
JS 的作用域类型：
一般认为的作用域是词法作用域，此外 JS 还提供了一些动态改变作用域的方法，常见的作用域类型有：
函数作用域，如果在函数内部我们给未定义的一个变量赋值，这个变量会转变成为一个全局变量，
块作用域：块作用域吧标识符限制在{}中，
改变函数作用域的方法：

eval（），这个方法接受一个字符串作为参数，并将其中的内容视为好像在书写时就存在于程序中这个位置的代码，
with 关键字：通常被当做重复引用同一个对象的多个属性的快捷方式

undefined 与 null：目前 null 和 undefined 基本是同义的，只有一些细微的差别，null 表示没有对象，undefined 表示缺少值，就是此处应该有一个值但是还没有定义，因此 undefined==null 返回 false
此外了解== 和===的区别：
在做==比较时。不同类型的数据会先转换成一致后在做比较，===中如果类型不一致就直接返回 false，一致的才会比较
类型判断函数，使用 typeof 即可，首先判断是否为 null，之后用 typeof 哦按段，如果是 object 的话，再用 array.isarray 判断是否为数组，如果是数字的话用 isNaN 判断是否是 NaN 即可
扩展学习：
JS 采用的是词法作用域，也就是静态作用域，所以函数的作用域在函数定义的时候就决定了，
看如下例子：

```
var value = 1;
function foo() {
console.log(value);
}
function bar() {
var value = 2;
foo();
}
bar();
```

假设 JavaScript 采用静态作用域，让我们分析下执行过程：
执行 foo 函数，先从 foo 函数内部查找是否有局部变量 value，如果没有，就根据书写的位置，查找上面一层的代码，也就是 value 等于 1，所以结果会打印 1。
假设 JavaScript 采用动态作用域，让我们分析下执行过程：
执行 foo 函数，依然是从 foo 函数内部查找是否有局部变量 value。如果没有，就从调用函数的作用域，也就是 bar 函数内部查找 value 变量，所以结果会打印 2。
前面我们已经说了，JavaScript 采用的是静态作用域，所以这个例子的结果是 1。

- setTimeout(fn,100);100 毫秒是如何权衡的

参考回答：
setTimeout()函数只是将事件插入了任务列表，必须等到当前代码执行完，主线程才会去执行它指定的回调函数，有可能要等很久，所以没有办法保证回调函数一定会在 setTimeout 指定的时间内执行，100 毫秒是插入队列的时间+等待的时间

- JS 的垃圾回收机制

参考回答：
GC（garbage collection），GC 执行时，中断代码，停止其他操作，遍历所有对象，对于不可访问的对象进行回收，在 V8 引擎中使用两种优化方法，
分代回收，2、增量 GC，目的是通过对象的使用频率，存在时长来区分新生代和老生代对象，多回收新生代区，少回收老生代区，减少每次遍历的时间，从而减少 GC 的耗时
回收方法：
引用计次，当对象被引用的次数为零时进行回收，但是循环引用时，两个对象都至少被引用了一次，因此导致内存泄漏，
标记清除

- **写一个 newBind 函数，完成 bind 的功能。**

参考回答：
bind（）方法，创建一个新函数，当这个新函数被调用时，bind（）的第一个参数将作为它运行时的 this，之后的一序列参数将会在传递的实参前传入作为它的参数

```
Function.prototype.bind2 = function (context) {
if (typeof this !== "function") {
throw new Error("Function.prototype.bind - what is trying to be bound is not callable");
}
var self = this;
var args = Array.prototype.slice.call(arguments, 1);
var fNOP = function () {};
var fbound = function () {
self.apply(this instanceof self ? this : context,
args.concat(Array.prototype.slice.call(arguments)));
}
fNOP.prototype = this.prototype;
fbound.prototype = new fNOP();
return fbound;
}
```

- **怎么获得对象上的属性：比如说通过 Object.key（）**

参考回答：
从 ES5 开始，有三种方法可以列出对象的属性
for（let I in obj）该方法依次访问一个对象及其原型链中所有可枚举的类型
object.keys:返回一个数组，包括所有可枚举的属性名称
object.getOwnPropertyNames:返回一个数组包含不可枚举的属性

- **简单讲一讲 ES6 的一些新特性**

参考回答：
ES6 在变量的声明和定义方面增加了 let、const 声明变量，有局部变量的概念，赋值中有比较吸引人的结构赋值，同时 ES6 对字符串、 数组、正则、对象、函数等拓展了一些方法，如字符串方面的模板字符串、函数方面的默认参数、对象方面属性的简洁表达方式，ES6 也 引入了新的数据类型 symbol，新的数据结构 set 和 map,symbol 可以通过 typeof 检测出来，为解决异步回调问题，引入了 promise 和 generator，还有最为吸引人了实现 Class 和模块，通过 Class 可以更好的面向对象编程，使用模块加载方便模块化编程，当然考虑到 浏览器兼容性，我们在实际开发中需要使用 babel 进行编译
重要的特性：
块级作用域：ES5 只有全局作用域和函数作用域，块级作用域的好处是不再需要立即执行的函数表达式，循环体中的闭包不再有问题
rest 参数：用于获取函数的多余参数，这样就不需要使用 arguments 对象了，
promise:一种异步编程的解决方案，比传统的解决方案回调函数和事件更合理强大
模块化：其模块功能主要有两个命令构成，export 和 import，export 命令用于规定模块的对外接口，import 命令用于输入其他模块提供的功能

- call 和 apply 是用来做什么？

参考回答：
Call 和 apply 的作用是一模一样的，只是传参的形式有区别而已
1、改变 this 的指向
2、借用别的对象的方法，
3、调用函数，因为 apply，call 方法会使函数立即执行

- 了解事件代理吗，这样做有什么好处

参考回答：
事件代理/事件委托：利用了事件冒泡，只指定一个事件处理程序，就可以管理某一类型的事件，
简而言之：事件代理就是说我们将事件添加到本来要添加的事件的父节点，将事件委托给父节点来触发处理函数，这通常会使用在大量的同级元素需要添加同一类事件的时候，比如一个动态的非常多的列表，需要为每个列表项都添加点击事件，这时就可以使用事件代理，通过判断 e.target.nodeName 来判断发生的具体元素，这样做的好处是减少事件绑定，同事动态的 DOM 结构任然可以监听，事件代理发生在冒泡阶段

- 如何写一个继承？

参考回答：
原型链继承
核心： 将父类的实例作为子类的原型
特点：

非常纯粹的继承关系，实例是子类的实例，也是父类的实例
父类新增原型方法/原型属性，子类都能访问到
简单，易于实现
缺点：
要想为子类新增属性和方法，不能放到构造器中
无法实现多继承
来自原型对象的所有属性被所有实例共享
创建子类实例时，无法向父类构造函数传参

构造继承
核心：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）
特点：
解决了子类实例共享父类引用属性的问题
创建子类实例时，可以向父类传递参数
可以实现多继承（call 多个父类对象）
缺点：
实例并不是父类的实例，只是子类的实例
只能继承父类的实例属性和方法，不能继承原型属性/方法
无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

实例继承
核心：为父类实例添加新特性，作为子类实例返回
特点：
不限制调用方式，不管是 new 子类()还是子类(),返回的对象具有相同的效果
缺点：
实例是父类的实例，不是子类的实例
不支持多继承

拷贝继承
特点：
支持多继承
缺点：
效率较低，内存占用高（因为要拷贝父类的属性）

组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
特点：
可以继承实例属性/方法，也可以继承原型属性/方法
既是子类的实例，也是父类的实例
不存在引用属性共享问题
可传参
函数可复用

寄生组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
参考 https://www.cnblogs.com/humin/p/4556820.html

- 给出以下代码，输出的结果是什么？原因？ for(var i=0;i<5;i++){ setTimeout(function(){ console.log(i); },1000); } console.log(i)

参考回答：
在一秒后输出 5 个 5
每次 for 循环的时候 setTimeout 都会执行，但是里面的 function 则不会执行被放入任务队列，因此放了 5 次；for 循环的 5 次执行完之后不到 1000 毫秒；1000 毫秒后全部执行任务队列中的函数，所以就是输出 5 个 5。

- 给两个构造函数 A 和 B，如何实现 A 继承 B？

参考回答：
```
function A(...) {}  A.prototype...
function B(...) {}  B.prototype...
A.prototype = Object.create(B.prototype);
```

```
// 再在 A 的构造函数里 new B(props);
for(var i = 0; i < lis.length; i++) {
lis[i].addEventListener('click', function(e) {
alert(i);
}, false)
}
```

- 问能不能正常打印索引

参考回答：
在 click 的时候，已经变成 length 了

- 如果已经有三个 promise，A、B 和 C，想串行执行，该怎么写？

参考回答：
```
// promise
A.then(B).then(C).catch(...)
```

```
// async/await
(async ()=>{
await a();
await b();
await c();
})()
```

- 知道 private 和 public 吗

参考回答：
public：public 表明该数据成员、成员函数是对所有用户开放的，所有用户都可以直接进行调用
private：private 表示私有，私有的意思就是除了 class 自己之外，任何人都不可以直接使用

- 基础的 js

参考回答：

```
Function.prototype.a = 1;
Object.prototype.b = 2;
function A() {}
var a = new A();
console.log(a.a, a.b); // undefined, 2
console.log(A.a, A.b); // 1, 2
```

- async 和 await 具体该怎么用？

参考回答：

```
(async () = > {
await new promise();
})()
```

- 知道哪些 ES6，ES7 的语法

参考回答：
promise，await/async，let、const、块级作用域、箭头函数

- promise 和 await/async 的关系

参考回答：

都是异步编程的解决方案

- JS 的数据类型

参考回答：
字符串，数字，布尔，数组，null，Undefined，symbol，对象。

- JS 加载过程阻塞，解决方法。

参考回答：
指定 script 标签的 async 属性。
如果 async="async"，脚本相对于页面的其余部分异步地执行（当页面继续进行解析时，脚本将被执行）
如果不使用 async 且 defer="defer"：脚本将在页面完成解析时执行

- JS 对象类型，基本对象类型以及引用对象类型的区别

参考回答：
分为基本对象类型和引用对象类型
基本数据类型：按值访问，可操作保存在变量中的实际的值。基本类型值指的是简单的数据段。基本数据类型有这六种:undefined、null、string、number、boolean、symbol。
引用类型：当复制保存着对象的某个变量时，操作的是对象的引用，但在为对象添加属性时，操作的是实际的对象。引用类型值指那些可能为多个值构成的对象。
引用类型有这几种：Object、Array、RegExp、Date、Function、特殊的基本包装类型(String、Number、Boolean)以及单体内置对象(Global、Math)。

- JavaScript 中的轮播实现原理？假如一个页面上有两个轮播，你会怎么实现？

参考回答：
图片轮播的原理就是图片排成一行，然后准备一个只有一张图片大小的容器，对这个容器设置超出部分隐藏，在控制定时器来让这些图片整体左移或右移，这样呈现出来的效果就是图片在轮播了。
如果有两个轮播，可封装一个轮播组件，供两处调用

- 怎么实现一个计算一年中有多少周？

参考回答：
首先你得知道是不是闰年，也就是一年是 365 还是 366.

其次你得知道当年 1 月 1 号是周几。假如是周五，一年 365 天把 1 号 2 号 3 号减去，也就是把第一个不到一周的天数减去等于 362
还得知道最后一天是周几，加入是周五，需要把周一到周五减去，也就是 362-5=357.
正常情况 357 这个数计算出来是 7 的倍数。357/7=51 。即为周数。

- **面向对象的继承方式**

参考回答：
原型链继承
核心： 将父类的实例作为子类的原型
特点：
非常纯粹的继承关系，实例是子类的实例，也是父类的实例
父类新增原型方法/原型属性，子类都能访问到
简单，易于实现
缺点：
要想为子类新增属性和方法，不能放到构造器中
无法实现多继承
来自原型对象的所有属性被所有实例共享
创建子类实例时，无法向父类构造函数传参

构造继承
核心：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）
特点：
解决了子类实例共享父类引用属性的问题
创建子类实例时，可以向父类传递参数
可以实现多继承（call 多个父类对象）
缺点：
实例并不是父类的实例，只是子类的实例
只能继承父类的实例属性和方法，不能继承原型属性/方法
无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

实例继承
核心：为父类实例添加新特性，作为子类实例返回
特点：
不限制调用方式，不管是 new 子类()还是子类(),返回的对象具有相同的效果
缺点：
实例是父类的实例，不是子类的实例
不支持多继承

拷贝继承
特点：
支持多继承

缺点：
效率较低，内存占用高（因为要拷贝父类的属性）

组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
特点：
可以继承实例属性/方法，也可以继承原型属性/方法
既是子类的实例，也是父类的实例
不存在引用属性共享问题
可传参
函数可复用

寄生组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
参考 https://www.cnblogs.com/humin/p/4556820.html

- JS 的数据类型

参考回答：
字符串，数字，布尔，数组，null，Undefined，symbol，对象。

- 引用类型常见的对象

参考回答：
Object、Array、RegExp、Date、Function、特殊的基本包装类型(String、Number、Boolean)以及单体内置对象(Global、Math)等

- es6 的常用

参考回答：
promise，await/async，let、const、块级作用域、箭头函数

- class

参考回答：
ES6 提供了更接近传统语言的写法，引入了 Class（类）这个概念，作为对象的模板。通过 class 关键字，可以定义类。

- **口述数组去重**

参考回答：
法一：indexOf 循环去重
法二：ES6 Set 去重；Array.from(new Set(array))
法三：Object 键值对去重；把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的。

- **继承**

参考回答：
原型链继承
核心：　将父类的实例作为子类的原型
特点：
非常纯粹的继承关系，实例是子类的实例，也是父类的实例
父类新增原型方法/原型属性，子类都能访问到
简单，易于实现
缺点：
要想为子类新增属性和方法，不能放到构造器中
无法实现多继承
来自原型对象的所有属性被所有实例共享
创建子类实例时，无法向父类构造函数传参

构造继承
核心：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）
特点：
解决了子类实例共享父类引用属性的问题
创建子类实例时，可以向父类传递参数
可以实现多继承（call 多个父类对象）
缺点：
实例并不是父类的实例，只是子类的实例
只能继承父类的实例属性和方法，不能继承原型属性/方法
无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

实例继承
核心：为父类实例添加新特性，作为子类实例返回
特点：
不限制调用方式，不管是 new 子类()还是子类(),返回的对象具有相同的效果
缺点：
实例是父类的实例，不是子类的实例
不支持多继承

拷贝继承
特点：
支持多继承
缺点：
效率较低，内存占用高（因为要拷贝父类的属性）

组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
特点：
可以继承实例属性/方法，也可以继承原型属性/方法
既是子类的实例，也是父类的实例
不存在引用属性共享问题
可传参
函数可复用

寄生组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
参考 https://www.cnblogs.com/humin/p/4556820.html

- call 和 apply 的区别

参考回答：
apply：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.apply(A, arguments);即 A 对象应用 B 对象的方法。
call：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.call(A, args1,args2);即 A 对象调用 B 对象的方法。

- es6 的常用特性

参考回答：
promise，await/async，let、const、块级作用域、箭头函数

- 箭头函数和 function 有什么区别

参考回答：
箭头函数根本就没有绑定自己的 this，在箭头函数中调用 this 时，仅仅是简单的沿着作用域链向上寻找，找到最近的一个 this 拿来使用

- new 操作符原理

参考回答：
1. 创建一个类的实例：创建一个空对象 obj，然后把这个空对象的__proto__设置为构造函数的 prototype。
2. 初始化实例：构造函数被传入参数并调用，关键字 this 被设定指向该实例 obj。
3. 返回实例 obj。

- bind,apply,call

参考回答：
apply：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.apply(A, arguments);即 A 对象应用 B 对象的方法。
call：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.call(A, args1,args2);即 A 对象调用 B 对象的方法。
bind 除了返回是函数以外，它的参数和 call 一样。

- bind 和 apply 的区别

参考回答：
返回不同：bind 返回是函数
参数不同：apply(A, arguments)，bind(A, args1,args2)

- 数组的去重

参考回答：
法一：indexOf 循环去重
法二：ES6 Set 去重；Array.from(new Set(array))
法三：Object 键值对去重；把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果 Object[value2]存在的话，就说明该值是重复的。

- 闭包

参考回答：
（1）什么是闭包：
闭包是指有权访问另外一个函数作用域中的变量的函数。
闭包就是函数的局部变量集合，只是这些局部变量在函数返回后会继续存在。闭包就是就是函数的“堆栈”在函数返回后并不释放，我们也可以理解为这些函数堆栈并不在栈上分配而是在堆上分配。当在一个函数内定义另外一个函数就会产生闭包。
（2）为什么要用：
匿名自执行函数：我们知道所有的变量，如果不加上 var 关键字，则默认的会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，比如：别的函数可

能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。除了每次使用变量都是用 var 关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，可以用闭包。
结果缓存：我们开发中会碰到很多情况，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。

- promise 实现

参考回答：

Promise 实现如下

```
function Promise(fn) {
var state = 'pending',
value = null,
callbacks = [];
this.then = function (onFulfilled, onRejected) {
return new Promise(function (resolve, reject) {
handle({
onFulfilled: onFulfilled || null,
onRejected: onRejected || null,
resolve: resolve,
reject: reject
});
});
};
function handle(callback) {
if (state === 'pending') {
callbacks.push(callback);
return;
}
var cb = state === 'fulfilled' ? callback.onFulfilled :
callback.onRejected,
ret;
if (cb === null) {
cb = state === 'fulfilled' ? callback.resolve : callback.reject;
cb(value);
return;
}
ret = cb(value);
callback.resolve(ret);
}
```

```
function resolve(newValue) {
if (newValue && (typeof newValue === 'object' || typeof newValue ===
'function')) {
var then = newValue.then;
if (typeof then === 'function') {
then.call(newValue, resolve, reject);
return;
}
}
state = 'fulfilled';
value = newValue;
execute();
}
function reject(reason) {
state = 'rejected';
value = reason;
execute();
}
function execute() {
setTimeout(function () {
callbacks.forEach(function (callback) {
handle(callback);
});
}, 0);
}
fn(resolve, reject);
}
```

- assign 的深拷贝

参考回答：

```
function clone( obj ) {
var copy;
switch( typeof obj ) {
case "undefined":
break;
case "number":
copy = obj - 0;
break;
case "string":
copy = obj + "";
break;
case "boolean":
```

```
copy = obj;
break;
case "object":   //object 分为两种情况 对象（Object）和数组（Array）
```

```
if(obj === null) {
copy = null;
} else {
if( Object.prototype.toString.call(obj).slice(8,  -1) ===  "Array") {
copy = [];
for( var i =  0 ; i < obj.length ; i++ ) {
copy.push(clone(obj[i]));
}
} else {
copy = {};
for( var j in obj) {
copy[j] = clone(obj[j]);
}
}
}
break;
default:
copy = obj;
break;
}
return copy;
}
```

- **说 promise，没有 promise 怎么办**

参考回答：
没有 promise，可以用回调函数代替

- **事件委托**

参考回答：
把一个元素响应事件（click、keydown.....）的函数委托到另一个元素；
优点：减少内存消耗、动态绑定事件。

- **箭头函数和 function 的区别**

参考回答：
箭头函数根本就没有绑定自己的 this，在箭头函数中调用 this 时，仅仅是简单的沿着作用域链向上寻找，找到最近的一个 this 拿来使用

- arguments

参考回答：
arguments 是类数组对象，有 length 属性，不能调用数组方法
可用 Array.from()转换

- 箭头函数获取 arguments

参考回答：
可用…rest 参数获取

- Promise

参考回答：
Promise 对象是 CommonJS 工作组提出的一种规范，目的是为异步编程提供统一接口。
每一个异步任务返回一个 Promise 对象，该对象有一个 then 方法，允许指定回调函数。
f1().then(f2);
一个 promise 可能有三种状态：等待（pending）、已完成（resolved，又称 fulfilled）、已拒绝（rejected）。
promise 必须实现 then 方法（可以说，then 就是 promise 的核心），而且 then 必须返回一个 promise，同一个 promise 的 then 可以调用多次，并且回调的执行顺序跟它们被定义时的顺序一致。
then 方法接受两个参数，第一个参数是成功时的回调，在 promise 由“等待”态转换到“完成”态时调用，另一个是失败时的回调，在 promise 由“等待”态转换到“拒绝”态时调用。同时，then 可以接受另一个 promise 传入，也接受一个“类 then”的对象或方法，即 thenable 对象。

- 事件代理

参考回答：
事件代理是利用事件的冒泡原理来实现的，何为事件冒泡呢？就是事件从最深的节点开始，然后逐步向上传播事件，举个例子：页面上有这么一个节点树，div>ul>li>a;
比如给最里面的 a 加一个 click 点击事件，那么这个事件就会一层一层的往外执行，执行顺序 a>li>ul>div，有这样一个机制，那么我们给最外面的 div 加点击事件，那么里面的 ul，li，a 做点击事件的时候，都会冒泡到最外层的 div 上，所以都会触发，这就是事件代理，代理它们父级代为执行事件。

- Eventloop

参考回答：
任务队列中，在每一次事件循环中，macrotask 只会提取一个执行，而 microtask 会一直提取，直到 microsoft 队列为空为止。
也就是说如果某个 microtask 任务被推入到执行中，那么当主线程任务执行完成后，会循环调用该队列任务中的下一个任务来执行，直到该任务队列到最后一个任务为止。而事件循环每次只会入栈一个 macrotask,主线程执行完成该任务后又会检查 microtasks 队列并完成里面的所有任务后再执行 macrotask 的任务。
macrotasks: setTimeout, setInterval, setImmediate, I/O, UI rendering
microtasks: process.nextTick, Promise, MutationObserver

---

2 | 前端核心

2.1| 服务端编程

- JSONP 的缺点

参考回答：
JSON 只支持 get，因为 script 标签只能使用 get 请求；
JSONP 需要后端配合返回指定格式的数据。

- 跨域（jsonp，ajax）

参考回答：
JSONP：ajax 请求受同源策略影响，不允许进行跨域请求，而 script 标签 src 属性中的链接却可以访问跨域的 js 脚本，利用这个特性，服务端不再返回 JSON 格式的数据，而是返回一段调用某个函数的 js 代码，在 src 中进行了调用，这样实现了跨域。

- 如何实现跨域

参考回答：
JSONP：通过动态创建 script，再请求一个带参网址实现跨域通信。document.domain + iframe 跨域：两个页面都通过 js 强制设置 document.domain 为基础主域，就实现了同域。

location.hash + iframe 跨域：a 欲与 b 跨域相互通信，通过中间页 c 来实现。 三个页面，不同域之间利用 iframe 的 location.hash 传值，相同域之间直接 js 访问来通信。
window.name + iframe 跨域：通过 iframe 的 src 属性由外域转向本地域，跨域数据即由 iframe 的 window.name 从外域传递到本地域。
postMessage 跨域：可以跨域操作的 window 属性之一。
CORS：服务端设置 Access-Control-Allow-Origin 即可，前端无须设置，若要带 cookie 请求，前后端都需要设置。
代理跨域：起一个代理服务器，实现数据的转发

- dom 是什么，你的理解？

参考回答：
文档对象模型（Document Object Model，简称 DOM），是 W3C 组织推荐的处理可扩展标志语言的标准编程接口。在网页上，组织页面（或文档）的对象被组织在一个树形结构中，用来表示文档中对象的标准模型就称为 DOM。

- 关于 dom 的 api 有什么

参考回答：
节点创建型 api，页面修改型 API，节点查询型 API，节点关系型 api，元素属性型 api，元素样式型 api 等

2.2 | Ajax

- ajax 返回的状态

参考回答：
0 — （未初始化）还没有调用 send()方法
1 — （载入）已调用 send()方法，正在发送请求
2 — （载入完成）send()方法执行完成，已经接收到全部响应内容
3 — （交互）正在解析响应内容
4 — （完成）响应内容解析完成，可以在客户端调用了

- 实现一个 Ajax

参考回答：
AJAX 创建异步对象 XMLHttpRequest
操作 XMLHttpRequest 对象
（1）设置请求参数（请求方式，请求页面的相对路径，是否异步）

（2）设置回调函数，一个处理服务器响应的函数，使用 onreadystatechange ，类似函数指针
（3）获取异步对象的 readyState 属性：该属性存有服务器响应的状态信息。每当 readyState 改变时，onreadystatechange 函数就会被执行。
（4）判断响应报文的状态，若为 200 说明服务器正常运行并返回响应数据。
（5）读取响应数据，可以通过 responseText 属性来取回由服务器返回的数据。

- **如何实现 ajax 请求，假如我有多个请求，我需要让这些 ajax 请求按照某种顺序一次执行，有什么办法呢？如何处理 ajax 跨域**

参考回答：
通过实例化一个 XMLHttpRequest 对象得到一个实例，调用实例的 open 方法为这次 ajax 请求设定相应的 http 方法，相应的地址和是否异步，以异步为例，调用 send 方法，这个方法可以设定需要发送的报文主体，然后通过监听 readystatechange 事件，通过这个实例 的 readyState 属性来判断这个 ajax 请求状态，其中分为 0，1，2，3，4 这四种状态（0 未初始化，1 载入/正在发送请求 2 载入完成/数据接收，3 交互/解析数据，4 接收数据完成），当状态为 4 的时候也就是接受数据完成的时候，这时候可以通过实例的 status 属性判断这个请求是否成功

```
var xhr = new XMLHttpRequest();
xhr.open('get', 'aabb.php', true);
xhr.send(null);
xhr.onreadystatechange = function() {
if(xhr.readyState==4) {
if(xhr.status==200) {
console.log(xhr.responseText);
}
}
}
```

使 ajax 请求按照队列顺序执行，通过调用递归函数：

```
//按顺序执行多个 ajax 命令，因为数量不定，所以采用递归


function send(action, arg2) {


//将多个命令按顺序封装成数组对象，递归执行
//利用了 deferred 对象控制回调函数的特点


$.when(send_action(action[0], arg2))
.done(function () {
```

```
//前一个 ajax 回调函数完毕之后判断队列长度


if (action.length > 1) {


//队列长度大于 1，则弹出第一个，继续递归执行该队列


action.shift();
send(action, arg2);
}
}).fail(function (){


//队列中元素请求失败后的逻辑
//
//重试发送
//send(action, arg2);
//
//忽略错误进行下个
//if (action.length > 1) {
//队列长度大于 1，则弹出第一个，继续递归执行该队列
//    action.shift();
//    send(action, arg2);
//}
});
}
//处理每个命令的 ajax 请求以及回调函数
function send_action(command, arg2) {
var dtd = $.Deferred();//定义 deferred 对象
$.post(
"url",
{
command: command,
arg2: arg2
}
).done(function (json) {
json = $.parseJSON(json);
//每次请求回调函数的处理逻辑
//
//
//
//逻辑结束
```

```
dtd.resolve();
}).fail(function (){
//ajax 请求失败的逻辑
dtd.reject();
});
return dtd.promise();//返回 Deferred 对象的 promise，防止在外部
```

- 写出原生 Ajax

参考回答：

Ajax 能够在不重新加载整个页面的情况下与服务器交换数据并更新部分网页内容，实现局部刷新，大大降低了资源的浪费，是一门用于快速创建动态网页的技术，ajax 的使用分为四部分：

1、创建 XMLHttpRequest 对象 var xhr = new XMLHttpRequest();
2、向服务器发送请求，使用 xmlHttpRequest 对象的 open 和 send 方法，
3、监听状态变化，执行相应回调函数

```
var xhr = new XMLHttpRequest();
xhr.open('get', 'aabb.php', true);
xhr.send(null);
xhr.onreadystatechange = function() {
if(xhr.readyState==4) {
if(xhr.status==200) {
console.log(xhr.responseText);
}
}
}
```

- 如何实现一个 ajax 请求？如果我想发出两个有顺序的 ajax 需要怎么做？

参考回答：

AJAX 创建异步对象 XMLHttpRequest
操作 XMLHttpRequest 对象
（1）设置请求参数（请求方式，请求页面的相对路径，是否异步）
（2）设置回调函数，一个处理服务器响应的函数，使用 onreadystatechange ，类似函数指针
（3）获取异步对象的 readyState 属性：该属性存有服务器响应的状态信息。每当 readyState 改变时，onreadystatechange 函数就会被执行。
（4）判断响应报文的状态，若为 200 说明服务器正常运行并返回响应数据。
（5）读取响应数据，可以通过 responseText 属性来取回由服务器返回的数据。
发出两个有顺序的 ajax，可以用回调函数，也可以使用 Promise.then 或者 async 等。

- Fetch 和 Ajax 比有什么优缺点？

参考回答：
promise 方便异步，在不想用 jQuery 的情况下，相比原生的 ajax，也比较好写。

- 原生 JS 的 ajax

参考回答：
AJAX 创建异步对象 XMLHttpRequest
操作 XMLHttpRequest 对象
（1）设置请求参数（请求方式，请求页面的相对路径，是否异步）
（2）设置回调函数，一个处理服务器响应的函数，使用 onreadystatechange ，类似函数指针
（3）获取异步对象的 readyState 属性：该属性存有服务器响应的状态信息。每当 readyState 改变时，onreadystatechange 函数就会被执行。
（4）判断响应报文的状态，若为 200 说明服务器正常运行并返回响应数据。
（5）读取响应数据，可以通过 responseText 属性来取回由服务器返回的数据。

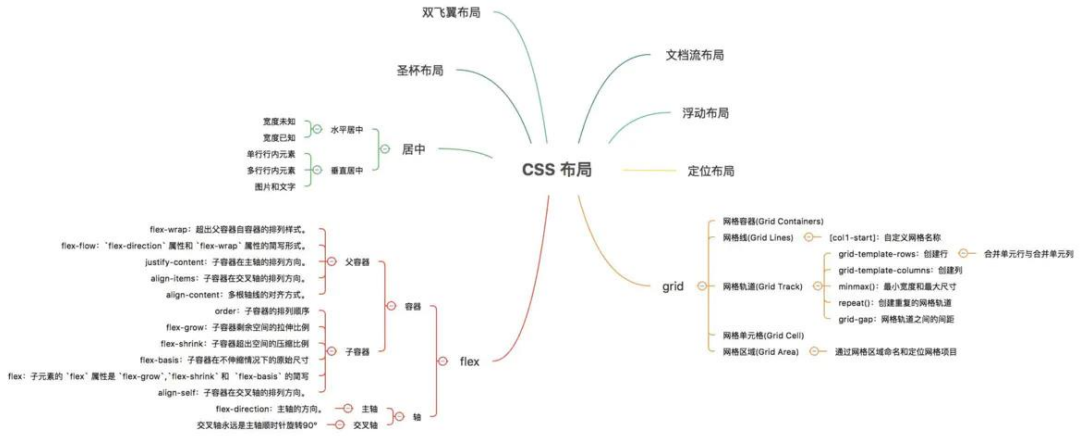
2.3 | 移动 web 开发

- 知道 PWA 吗

参考回答：
PWA 全称 Progressive Web App，即渐进式 WEB 应用。一个 PWA 应用首先是一个网页，可以通过 Web 技术编写出一个网页应用. 随后添加上 App Manifest 和 Service Worker 来实现 PWA 的安装和离线等功能

- 移动布局方案

参考回答：
https://juejin.im/post/599970f4518825243a78b9d5#heading-22



- Rem, Em

参考回答：

https://blog.csdn.net/romantic_love/article/details/80875462

一、rem 单位如何转换为像素值

1.当使用 rem 单位的时候，页面转换为像素大小取决于叶根元素的字体大小，即 HTML 元素的字体大小。根元素字体大小乘 rem 的值。例如，根元素的字体大小为 16px，那么 10rem 就等同于 10*16=160px。

二、em 是如何转换成 px 的

当使用 em 单位的时候，像素值是将 em 值乘以使用 em 单位的元素的字体大小。例如一个 div 的字体为 18px，设置它的宽高为 10em，那么此时宽高就是 18px*10em=180px。

```
.test{
    width: 10em;
    height: 10em;
    background-color: #ff7d42;
    font-size: 18px;
}
```

一定要记住的是，em 是根据使用它的元素的 font-size 的大小来变化的，而不是根据父元素字体大小。有些元素大小是父元素的多少倍那是因为继承了父元素中 font-size 的设定，所以才起到的作用。

2.em 单位的继承效果。

使用 em 单位存在继承的时候，每个元素将自动继承其父元素的字体大小，继承的效果只能被明确的字体单位覆盖，比如 px 和 vw。只要父级元素上面一直有 fontsize 为 em 单位，则会一直继承，但假如自己设置了 font-size 的单位为 px 的时候，则会直接使用自己的 px 单位的值。

三、根 html 的元素将会继承浏览器中设置的字体大小

除非显式的设置固定值去覆盖。所以 html 元素的字体大小虽然是直接确定 rem 的值，但这个字体大小首先是来源于浏览器的设置。（所以一定要设置 html 的值的大小，因为有可能用户的浏览器字体大小是不一致的。）

四、当 em 单位设置在 html 元素上时

它将转换为 em 值乘以浏览器字体大小的设置。

例如：

```
html{
    font-size: 1.5em;
}
```

可以看到，因为浏览器默认字体大小为 16px，所以当设置 HTML 的 fontsize 的值为 1.5em 的售后，其对应的 px 的值为 16*1.5=24px

所以此时，再设置其他元素的 rem 的值的时候，其对应的像素值为 n*24px。

例如，test 的 rem 的值为 10，

```
.test{
    width: 10rem;
    height: 10rem;
    background-color: #ff7d42;
}
```

可以看到 test 的 font-size 继承了 html 的值 24px，而此时宽高为 24*10=240px

总结

1. rem 单位翻译为像素值的时候是由 html 元素的字体大小决定的。此字体大小会被浏览器中字体大小的设置影响，除非显式的在 html 为 font-size 重写一个单位。

2. em 单位转换为像素值的时候，取决于使用它们的元素的 font-size 的大小，但是有因为有继承关系，所以比较复杂。

优缺点

em 可以让我们的页面更灵活，更健壮，比起到处写死的 px 值，em 似乎更有张力，改动父元素的字体大小，子元素会等比例变化，这一变化似乎预示了无限可能，

em 做弹性布局的缺点还在于牵一发而动全身，一旦某个节点的字体大小发生变化，那么其后代元素都得重新计算

- **flex 布局及优缺点**

参考回答：

https://juejin.im/post/599970f4518825243a78b9d5#heading-22

css3 引入的，flex 布局；优点在于其容易上手，根据 flex 规则很容易达到某个布局效果，然而缺点是：浏览器兼容性比较差，只能兼容到 ie9 及以上；

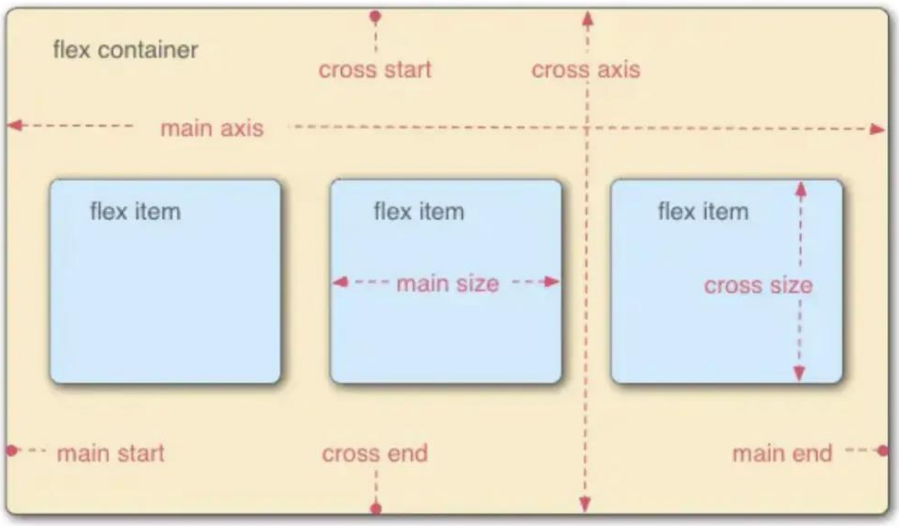
flex 的使用方法很简单，只需要将其 `display` 属性设置为 `flex` 就可以，也可以设置行内的 flex，记得 Webkit 内核的浏览器，必须加上 `-webkit` 前缀。**注意，设为 Flex 布局以后，子元素的 `float` 、 `clear` 和 `vertical-align` 属性将失效。**

复制代码

```
.ele{
    display: -webkit-flex;
    display: flex;
    display: inline-flex;
    display: -webkit-inline-flex;
}
```

> 在 flex 中，最核心的概念就是容器和轴，所有的属性都是围绕容器和轴设置的。其中，容器分为父容器和子容器。轴分为主轴和交叉轴（主轴默认为水平方向，方向向右，交叉轴为主轴顺时针旋转 90°）。

在使用 flex 的元素中，默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）
主轴开始的位置称为 `main start` ，主轴结束的位置称为 `main end` 。
同理，交叉轴开始的位置称为 `cross start` ，交叉轴结束的位置称为 `cross end` 。
在使用 flex 的子元素中，占据的主轴空间叫做 `main size` ，占据的交叉轴空间叫做 `cross size` 。



flex基本概念

- Rem 布局及其优缺点

参考回答：
首先 Rem 相对于根(html)的 font-size 大小来计算。简单的说它就是一个相对单例如:font-size:10px;,那么（1rem = 10px）了解计算原理后

首先解决怎么在不同设备上设置 html 的 font-size 大小。其实 rem 布局的本质是等比缩放，一般是基于宽度。

优点

可以快速适用移动端布局 字体图片 高度

缺点

①目前 ie 不支持，对 pc 页面来讲使用次数不多；

②数据量大：所有的图片，盒子都需要我们去给一个准确的值；才能保证不同机型的适配；

③在响应式布局中，必须通过 js 来动态控制根元素 font-size 的大小。

也就是说 css 样式和 js 代码有一定的耦合性。且必须将改变 font-size 的代码放在 css 样式之前。

- **百分比布局**

参考回答：

1、具体分析

除了用 px 结合媒体查询实现响应式布局外，我们也可以通过百分比单位 ″ % ″ 来实现响应式的效果。比如当浏览器的宽度或者高度发生变化时，通过百分比单位，通过百分比单位可以使得浏览器中的组件的宽和高随着浏览器的变化而变化，从而实现响应式的效果。为了了解百分比布局，首先要了解的问题是：css 中的子元素中的百分比（%）到底是谁的百分比？

直观的理解，我们可能会认为子元素的百分比完全相对于直接父元素，height 百分比相对于 height，width 百分比相对于 width。当然这种理解是正确的，但是根据 css 的盒式模型，除了 height、width 属性外，还具有 padding、border、margin 等等属性。那么这些属性设置成百分比，是根据父元素的那些属性呢？此外还有 border-radius 和 translate 等属性中的百分比，又是相对于什么呢？下面来具体分析。

padding、border、margin 等等属不论是垂直方向还是水平方向，都相对于直接父元素的 width。

除了 border-radius 外，还有比如 translate、background-size 等都是相对于自身的。

2、百分比单位布局应用

比如我们要实现一个固定长宽比的长方形，比如要实现一个长宽比为 4:3 的长方形,我们可以根据 padding 属性来实现，因为 padding 不管是垂直方向还是水平方向，百分比单位都相对于父元素的宽度，因此我们可以设置 padding-top 为百分比来实现，长宽自适应的长方形：

```
.trangle{
  height:0;
  width:100%;
  padding-top:75%;
}
```

3、百分比单位缺点

从上述对于百分比单位的介绍我们很容易看出如果全部使用百分比单位来实现响应式的布局，有明显的以下两个缺点：

（1）计算困难，如果我们要定义一个元素的宽度和高度，按照设计稿，必须换算成百分比单位。
（2）从小节 1 可以看出，各个属性中如果使用百分比，相对父元素的属性并不是唯一的。比如 width 和 height 相对于父元素的 width 和 height，而 margin、padding 不管垂直还是水平方向都相对比父元素的宽度、border-radius 则是相对于元素自身等等，造成我们使用百分比单位容易使布局问题变得复杂。

这里我们发现视窗宽高都是100vw / 100vh，那么vw或者vh，下简称vw，很类似百分比单位。vw和%的区别为：

单位	含义
%	大部分相对于祖先元素，也有相对于自身的情况比如（border-radius、translate等)
vw/vh	相对于视窗的尺寸

从对比中我们可以发现，vw单位与百分比类似，单确有区别，前面我们介绍了百分比单位的换算困难，这里的vw更像"理想的百分比单位"。任意层级元素，在使用vw单位的情况下，1vw都等于视图宽度的百分之一。

2. vw单位换算

同样的，如果要将px换算成vw单位，很简单，只要确定视图的窗口大小（布局视口），如果我们将布局视口设置成分辨率大小，比如对于iphone6/7 375*667的分辨率，那么px可以通过如下方式换算成vw：

```
1px =  (1/375) *100 vw
```

此外，也可以通过postcss的相应插件，预处理css做一个自动的转换，postcss-px-to-viewport可以自动将px转化成vw。 postcss-px-to-viewport的默认参数为：

- 移动端适配 1px 的问题

参考回答：
https://blog.csdn.net/weixin_43675871/article/details/84023447
首先，我们了解 devicePixelRatio（DPR）这个东西
在 window 对象中有一个 devicePixelRatio 属性，他可以反应 css 中的像素与设备的像素比。然而 1px 在不同的移动设备上都等于这个移动设备的 1px，这是因为不同的移动设备有不同的像素密度。有关这个属性，它的官方的定义为：设备物理像素和设备独立像素的比例，也就是 devicePixelRatio = 物理像素 / 独立像素 1px 变粗的原因：viewport 的设置和屏幕物理分辨率是按比例而不是相同的. 移动端 window 对象有个 devicePixelRatio 属性,它表示设备

物理像素和 css 像素的比例，在 retina 屏的 iphone 手机上，这个值为 2 或 3,css 里写的 1px 长度映射到物理像素上就有 2px 或 3px 那么长

1.用小数来写 px 值 （不推荐）

IOS8 下已经支持带小数的 px 值，media query 对应 devicePixelRatio 有个查询值 -webkit-min-device-pixel-ratio，css 可以写成这样

通过-webkit-min-device-pixel-ratio 设置。

```
.border { border: 1px solid #999 }
@media screen and (-webkit-min-device-pixel-ratio: 2) {
    .border { border: 0.5px solid #999 }
}
@media screen and (-webkit-min-device-pixel-ratio: 3) {
    .border { border: 0.333333px solid #999 }
}
```

如果使用 less/sass 的话只是加了 1 句 mixin

缺点：安卓与低版本 IOS 不适用，这个或许是未来的标准写法，现在不做指望

2、flexible.js

这是淘宝移动端采取的方案，github 的地址:https://github.com/amfe/lib-flexible. 前面已经说过 1px 变粗的原因就在于一刀切的设置 viewport 宽度，如果能把 viewport 宽度设置为实际的设备物理宽度，css 里的 1px 不就等于实际 1px 长了么. flexible.js 就是这样干的.

<meta name=”viewport”>里面的 scale 值指的是对 ideal viewport 的缩放，flexible.js 检测到 IOS 机型，会算出 scale = 1/devicePixelRatio，然后设置 viewport

3、伪类+transform 实现

对于解决 1px 边框问题，我个人觉得最完美的解决办法还是伪类+transform 比较好。

原理：是把原先元素的 border 去掉，然后利用 :before 或者 :after 重做 border ，并 transform 的 scale 缩小一半，原先的元素相对定位，新做的 border 绝对定位。

media query

通过媒体查询，可以通过给不同分辨率的设备编写不同的样式来实现响应式的布局，比如我们为不同分辨率的屏幕，设置不同的背景图片。比如给小屏幕手机设置@2x 图，为大屏幕手机设置@3x 图，通过媒体查询就能很方便的实现。

但是媒体查询的缺点也很明显，如果在浏览器大小改变时，需要改变的样式太多，那么多套样式代码会很繁琐。

```
@media screen and (min-width: 320px) {
    html {
        font-size: 50px;
    }
}
```

@media

方便应用广泛 适用于 pc 端 手机页面，通常做自适应布局时 我们比较常用。

缺点：相对于代码要重复很多，得知道设备的宽度，手机的分辨率很多所以麻烦了点，不过性能方面肯定最高； 可能存在闪屏的问题

@media 处理手机和 pc 端界面兼容的问题，在 IE 上的访问出现问题，百度方法，找找两种，一种是 respond.js，另一种是 css3-mediaquerieshttp://blog.csdn.net/small_tu/article/details/47317453

- **移动端性能优化相关经验**

参考回答：

https://blog.csdn.net/tangxiujiang/article/details/79791545

- **toB 和 toC 项目的区别**

参考回答：

to B（business）即面向企业，to C（ customer）即面向普通用户
简单的事情重复去做，重复的事情用心去做，长期坚持，自然功成，无论是 B 端还是 C 端都同样适用。
Tob 与 Toc 的区别
作者 猪八戒网 • 10-11 10:18:49 阅读 419
oB 产品价值何来?
最近团队在 toB 产品研究的过程中，得出结论，相对于 toC 产品与服务，toC 产品更注重产品用户的共性而淡化角色关系，而 toB 产品则更强调面向用户、客户的角色关系，而淡化共性提取。实际上，这是由服务对象所引起的，C 端产品的服务对象，由终端所限，是一个面向个体的服务。而 B 端服务使用最终是面向一个系统体系组织，在干系人间配合使用中发挥产品价值。
一个好的产品 toB 可以让组织的系统变得更好，最终反哺于系统中的各个单位。
需求动力之不同 toC 的产品方法论，用户体验是几乎最为重要的需求来源，腾讯此前，也以"以用户体验为归依"来驱动企业产品打造。
但 B 端产品则不同，B 端在一个商业的背景之下，B 端的决策思路是，"以企业获益为归依"，系统是否有利于企业的生产力，竞争力等，单纯的用户体验，仅能让员工得到片刻的享受， 但无法说服企业，企业并不会为一个不能"赚钱"的东西买单。
需求动力的不同，引发的这是购买使用决策体系的变化。
toB 产品应更考虑 获益与系统性价值，部分情况还有可能会牺牲掉局部个体的利益，对于使用者而言应该是自律或他律的，toC 产品则更考虑的是个体用户的偏好，并长时间内，基于技术效率的提升，产品的服务中心更多地围绕着更高效地帮助用户的"欲望"释放进行设计，对于使用者而言是一个释放自我的存在。

- **移动端兼容性**

参考回答：

https://zhuanlan.zhihu.com/p/28206065
1.IOS 移动端 click 事件 300ms 的延迟相应
3.h5 底部输入框被键盘遮挡问题
10.CSS 动画页面闪白,动画卡顿
解决方法:
1.尽可能地使用合成属性 transform 和 opacity 来设计 CSS3 动画，不使用 position 的 left 和 top 来定位

2.开启硬件加速
9.上下拉动滚动条时卡顿、慢
body {-webkit-overflow-scrolling:
touch;overflow-scrolling: touch;}
Android3+和 iOS5+支持 CSS3 的新属性为 overflow-scrolling

- 小程序

参考回答：
移动端手势

手势事件

手势事件	事件相详解
touchstart	当手指接触屏幕时触发
touchmove	当已经接触屏幕的手指开始移动的时候触发
touchend	当手指离开屏幕时触发

手势事件的应用

- 可以使用 `touchstart` 或者 `touchend` 事件，来代替 `click` 事件，用来触发移动端的点击事件。
- 可以使用 `touchmove` 事件代替 `scroll` 事件，来检测移动端的滑动事件。并且可以通过 `touchmove` 事件，来是实施获取滚动条滚动的高度。示例代码如下：

- 2X 图 3X 图适配

参考回答：
实际程序开发当中，我们代码中用的值是指逻辑分辨率 pt，而不是像素分辨率 px，比如我们定义一个按钮的高度为 45，这个 45 指的是 45pt 而不是 45px。在非 Retina 屏下 1pt = 1px，4 和 4.7 寸 Retina 屏下 1pt = 2px，5.5 和 x 下 1pt = 3px.我们制作不同尺寸的图片，比如@1x 为 22px，则@2x 为 44px，@3x 为 66px，命名分别为 image.png，在项目的 Assets.xcassets 中新建 New Image Set，修改名字为 image，并把相应尺寸的图片拖放至相应位置。
/* 根据 dpr 显示 2x 图/3x 图 */

```
    .bg-image(@url){
      background-image:~"url('@{url}@2x.png')";
      @media (-webkit-min-device-pixel-ratio: 3),(min-device-pixel-
ratio: 3){
        background-image:~"url('@{url}@3x.png')";
```

```
  }
}

.bg-color(@color) {
  background-color: @color;
}
```

- 图片在安卓上，有些设备模糊问题

参考回答：

4. 🙋‍♂️：如何解决 Android 浏览器查看背景图片模糊的问题？

👨‍🏫：这个问题是 devicePixelRatio 的不同导致的，因为手机分辨率太小，如果按照分辨率来显示网页，字会非常小，所以苹果系统当初就把 iPhone 4 的 960×640 像素的分辨率在网页里更改为 480×320 像素，这样 devicePixelRatio = 2。而 Android 的 devicePixelRatio 比较乱，值有 1.5、2 和 3。为了在手机里更为清晰地显示图片，必须使用 2 倍宽高的背景图来代替 img 标签（一般情况下都使用 2 倍）。

例如一个 div 的宽高是 100px×100px，背景图必须是 200px×200px，然后设置 background-size:contain 样式，显示出来的图片就比较清晰了。

- 固定定位布局键盘挡住输入框内容

参考回答：

2.通过定时器实时监听是否触发input

如果触发input框 就把固定定位，改变成静态定位。这样就会浏览器会总动把内容顶上去。

```
function fixedWatch(el) {
    //activeElement 获取焦点元素
    if(document.activeElement.nodeName == 'INPUT') {
        el.css('position', 'static');
    } else {
        el.css('position', 'fixed');
    }
}
setInterval(function() {
    fixedWatch($('.mian'));
}, 500);
```

三.移动端底部input被弹出的键盘遮挡

Element.scrollIntoView():方法让当前的元素滚动到浏览器窗口的可视区域内。

```
document.querySelector('#inputId').scrollIntoView();
//只要在input的点击事件，或者获取焦点的事件中，加入这个api就好了
```

- click 的 300ms 延迟问题和点击穿透问题

参考回答：
https://www.jianshu.com/p/6e2b68a93c88

13. ：如何解决移动端 click 事件有 300ms 延迟的问题？

：300ms 延迟导致用户体验不好。为了解决这个问题，一般在移动端用 touchstart、touchend、touchmove、tap（模拟的事件）事件来取代 click 事件。

方案二：FastClick

FastClick 是 FT Labs 专门为解决移动端浏览器 300 毫秒点击延迟问题所开发的一个轻量级的库。FastClick 的实现原理是在检测到 touchend 事件的时候，会通过 DOM 自定义事件立即出发模拟一个 click 事件，并把浏览器在 300ms 之后的 click 事件阻止掉。

二、点击穿透问题

说完移动端点击300ms延迟的问题，还不得不提一下移动端点击穿透的问题。可能有人会想，既然click点击有300ms的延迟，那对于触摸屏，我们直接监听touchstart事件不就好了吗？
使用touchstart去代替click事件有两个不好的地方。
第一：touchstart是手指触摸屏幕就触发，有时候用户只是想滑动屏幕，却触发了touchstart事件，这不是我们想要的结果；
第二：使用touchstart事件在某些场景下可能会出现点击穿透的现象。

什么是**点击穿透**？
假如页面上有两个元素A和B。B元素在A元素之上。我们在B元素的touchstart事件上注册了一个回调函数，该回调函数的作用是隐藏B元素。我们发现，当我们点击B元素，B元素被隐藏了，随后，A元素触发了click事件。

这是因为在移动端浏览器，事件执行的顺序是touchstart > touchend > click。而click事件有300ms的延迟，当touchstart事件把B元素隐藏之后，隔了300ms，浏览器触发了click事件，但是此时B元素不见了，所以该事件被派发到了A元素身上。如果A元素是一个链接，那此时页面就会意外地跳转。

- phone 及 ipad 下输入框默认内阴影

参考回答：

6. ：如何解决 iPhone 及 iPad 下输入框的默认内阴影问题？

：通过以下代码设置样式。

```
element{
  -webkit-appearance: none;
}
```

- 防止手机中页面放大和缩小

参考回答：
<meta name="viewport" content="user-scalable=no">

<meta name="viewport" content="initial-scale=1,maximum-scale=1">

- px、em、rem、%、vw、vh、vm 这些单位的区别

参考回答：
https://www.jianshu.com/p/ba26509bc5b3

- 移动端适配- dpr 浅析

参考回答：
https://www.jianshu.com/p/cf600c2930cb
dpr = 物理像素 / css 像素
在 dpr = 2； 1px 的 css 像素在设备中是 2px 的物理像素，这会导致在设备上看上去 1px 的边框是 2px
解决方法：
用 transfrom： scale（）缩小 dpr 倍数
在 meta 标签中设定 scale 缩小两倍

- 移动端扩展点击区域

参考回答：
父级代理事件
将 a 标签设置成块级元素

- 上下拉动滚动条时卡顿、慢

参考回答：

18. ：如何解决上下拖动滚动条时的卡顿问题?
：通过以下代码设置样式。

```
body {
  -webkit-overflow-scrolling: touch;
  overflow-scrolling: touch;
}
```

Android 3+和 iOS 5+支持 CSS3 的新属性 overflow-scrolling，该属性也可以解决上述问题。

- 长时间按住页面出现闪退

参考回答：

5. : 如何解决长时间按住页面出现闪退的问题?

: 通过以下代码设置样式。

```
element {
  -webkit-touch-callout: none;
}
```

- ios 和 android 下触摸元素时出现半透明灰色遮罩

参考回答:

7. : 在 iOS 和 Android 下，如何实现触摸元素时出现半透明灰色遮罩?

: 通过以下代码设置样式。

```
element {
  -webkit-tap-highlight-color:rgba(255,255,255,0)
}
```

- active 兼容处理 即 伪类：active 失效

参考回答:

https://blog.csdn.net/diaobuwei1238/article/details/101716814

将 :visited 放到最后，则会导致以下结果：若链接已经被访问过，a:visited 会覆盖:active 和:hover 的样式声明，链接将总是呈现为紫色，无论鼠标悬停还是按下激活，链接都将保持为紫色。

基于此原因，上述代码必须按照顺序定义，一般称为 LVHA-order: :link — :visited — :hover — :active，为方便记忆，可记为“LOVE HATE”

看来在iOS系统的移动设备中，需要在按钮元素或body/html上绑定一个touchstart事件才能激活:active状态。

```
[html]
1. document.body.addEventListener('touchstart', function () { //...空函数即可});
```

- webkit mask 兼容处理

参考回答:

https://segmentfault.com/a/1190000011838367

CSS中的mask属性允许用户屏蔽或剪裁特定点的图像来实现，部分或完全隐藏某个元素的可见性。
好吧，这个概念可能有点不好理解，先看图。

看了这个等式，似乎明白点什么了吧，朋友们，第一张图就是一张普通的图，第二张图，黑色部分是不透明的，白色部分是透明的，用上mask之后，两张图重叠，黑色区域中的会显示出来，白色区域不显示。

用过ps的朋友，应该很清楚，蒙版这东西，这就和蒙版很像，好吧，没用过ps的朋友，又要问蒙版是什么了，相信看完这篇文章，你应该连蒙版也知道了。

mask和background用法是相仿的，mask的值有这些

9.webkit mask兼容处理

某些低端手机不支持css3mask，可以选择性的进降级处理

比如可以使用js判断来引用不同class：

```
if('WebkitMask' in documnet.documentElement.style){ alert('支持 mask')}else{ ale
```

- transiton 闪屏

参考回答：
//设置内联的元素在 3D 空间如何呈现：保留 3D-webkit-transform-style:preserve-3D;
//设置进行转换的元素的背面在面对用户时是否可见：隐藏-webkit-backface-visibility:hidden;

- 圆角 bug

参考回答：

9. ：如何解决 Android 手机圆角失效问题?
：通过 background-clip: padding-box 为失效的元素设置样式。

作者：Aniugel
链接：https://www.jianshu.com/p/610123c6ed45
来源：简书

---

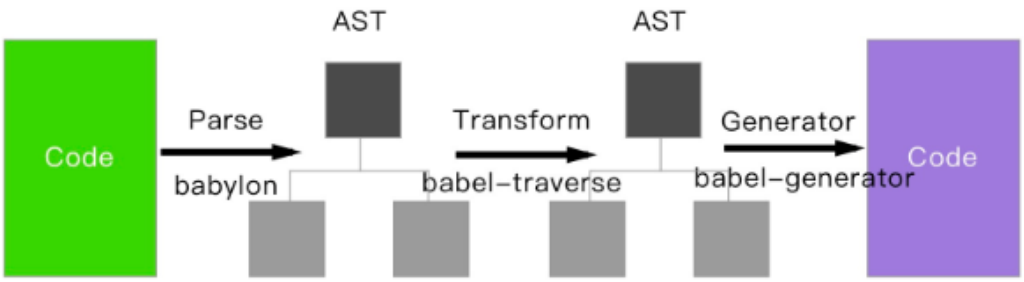
3 | 前端进阶

3.1 | 前端工程化

- Babel 的原理是什么?

参考回答：
babel 的转译过程也分为三个阶段，这三步具体是：

- 解析 Parse：将代码解析生成抽象语法树（ 即 AST ），即词法分析与语法分析的过程
- 转换 Transform：对于 AST 进行变换一系列的操作，babel 接受得到 AST 并通过 babel-traverse 对其进行遍历，在此过程中进行添加、更新及移除等操作
- 生成 Generate：将变换后的 AST 再转换为 JS 代码，使用到的模块是 babel-generator



- 如何写一个 babel 插件?

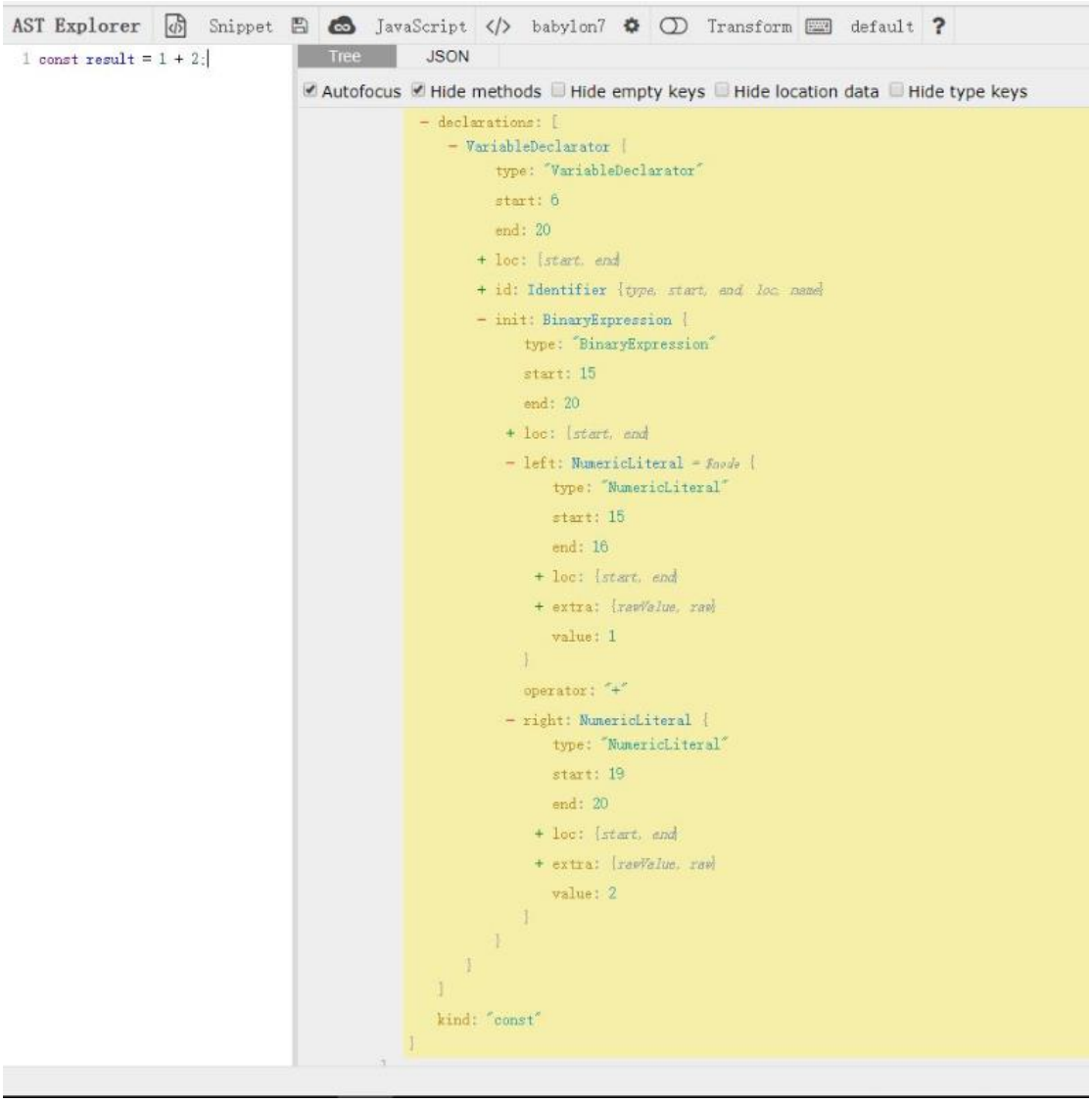
参考回答：
Babel 解析成 AST，然后插件更改 AST，最后由 Babel 输出代码
那么 Babel 的插件模块需要你暴露一个 function，function 内返回 visitor

```
module.export = function(babel){
  return {
    visitor:{
```

```
    }
  }
}
```

visitor 是对各类型的 AST 节点做处理的地方，那么我们怎么知道 Babel 生成了的 AST 有哪些节点呢？

很简单，你可以把 Babel 转换的结果打印出来，或者这里有传送门:[AST explorer]



这里我们看到 const result = 1 + 2 中的 1 + 1 是一个 BinaryExpression 节点，那么在 visitor 中，我们就处理这个节点

```
var babel = require('babel-core');
var t = require('babel-types');
const visitor = {
  BinaryExpression(path) {
```

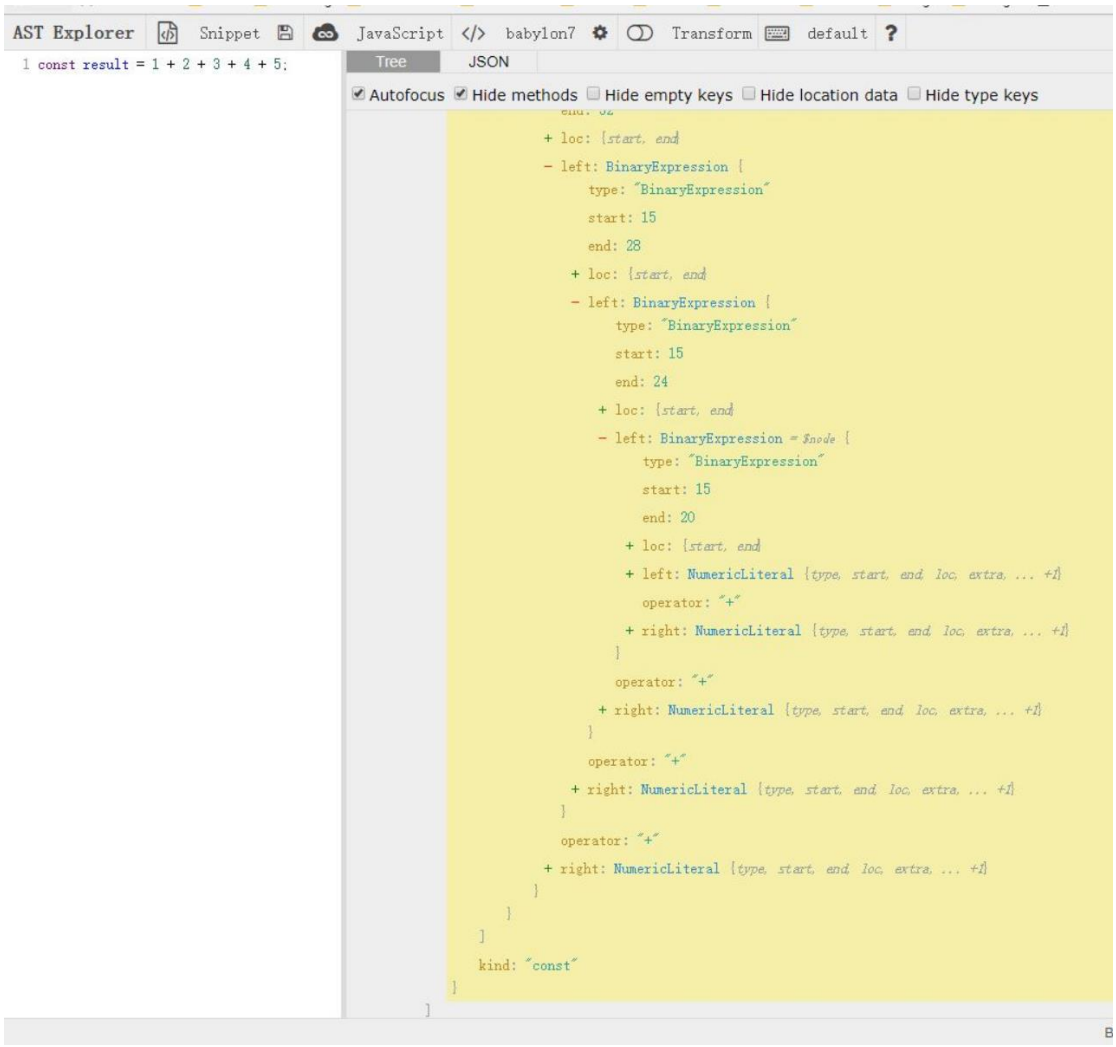
```
    const node = path.node;
    let result;
    // 判断表达式两边，是否都是数字
    if (t.isNumericLiteral(node.left) &&
t.isNumericLiteral(node.right)) {
      // 根据不同的操作符作运算
      switch (node.operator) {
        case "+":
          result = node.left.value + node.right.value;
          break
        case "-":
          result = node.left.value - node.right.value;
          break;
        case "*":
          result =  node.left.value * node.right.value;
          break;
        case "/":
          result =  node.left.value / node.right.value;
          break;
        case "**":
          let i = node.right.value;
          while (--i) {
            result = result || node.left.value;
            result =  result * node.left.value;
          }
          break;
        default:
      }
    }
    // 如果上面的运算有结果的话
    if (result !== undefined) {
      // 把表达式节点替换成number字面量
      path.replaceWith(t.numericLiteral(result));
    }
  }
};
module.exports = function (babel) {
  return {
    visitor
  };
}
```

插件写好了，我们运行下插件试试

```
const babel = require("babel-core");
const result = babel.transform("const result = 1 + 2;",{
```

```
  plugins:[
    require("./index")
  ]
});
console.log(result.code); // const result = 3;
```

与预期一致，那么转换 `const result = 1 + 2 + 3 + 4 + 5;`呢?
结果是: `const result = 3 + 3 + 4 + 5;`
这就奇怪了，为什么只计算了 1 + 2 之后，就没有继续往下运算了?
我们看一下这个表达式的 AST 树



你会发现 Babel 解析成表达式里面再嵌套表达式。
表达式(表达式(表达式(表达式(1 + 2) + 3) + 4) + 5)
而我们的判断条件并不符合所有的，只符合 1 + 2

```
// 判断表达式两边，是否都是数字
    if (t.isNumericLiteral(node.left) &&
t.isNumericLiteral(node.right)) {}
```

那么我们得改一改
第一次计算 1 + 2 之后，我们会得到这样的表达式
表达式(表达式(表达式(3 + 3) + 4) + 5)
其中 3 + 3 又符合了我们的条件， 我们通过向上递归的方式遍历父级节点
又转换成这样:
表达式(表达式(6 + 4) + 5)
表达式(10 + 5)
15

```
// 如果上面的运算有结果的话
    if (result !== undefined) {
      // 把表达式节点替换成 number 字面量
      path.replaceWith(t.numericLiteral(result));
      let parentPath = path.parentPath;
      // 向上遍历父级节点
      parentPath && visitor.BinaryExpression.call(this, parentPath);
    }
```

到这里，我们就得出了结果 `const result = 15;`
那么其他运算呢:

```
const result = 100 + 10 - 50>>>const result = 60;
const result = (100 / 2) + 50>>>const result = 100;
const result = (((100 / 2) + 50 * 2) / 50) ** 2>>>const result = 9;
```

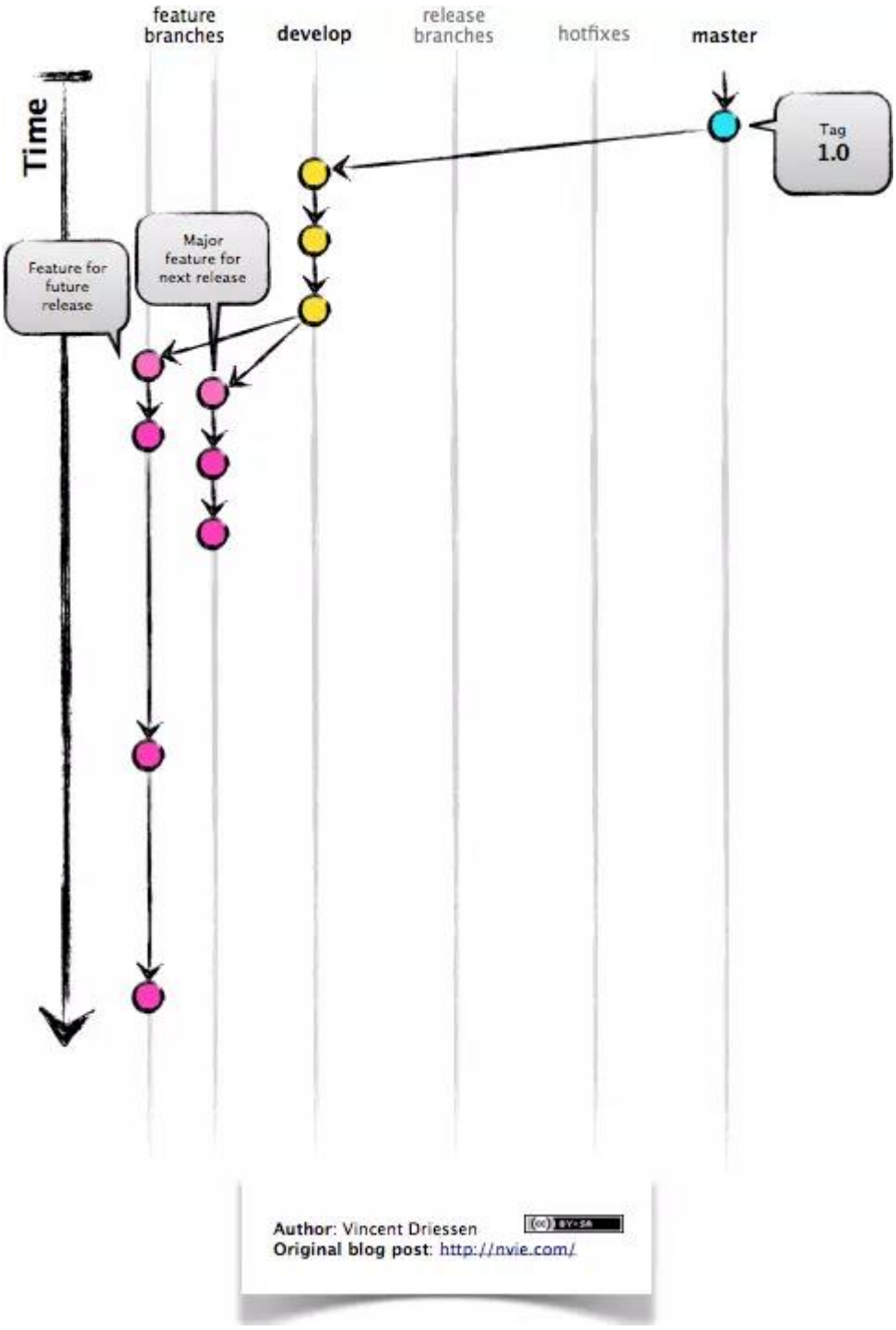
- **你的 git 工作流是怎样的?**

参考回答：
GitFlow 是由 Vincent Driessen 提出的一个 git 操作流程标准。包含如下几个关键分支：
master 主分支 develop 主开发分支，包含确定即将发布的代码
feature 新功能分支，一般一个新功能对应一个分支，对于功能的拆分需要比较合理，以避免一些后面不必要的代码冲突
release 发布分支，发布时候用的分支，一般测试时候发现的
bug 在这个分支进行修复 hotfixhotfix 分支，紧急修 bug 的时候用
GitFlow 的优势有如下几点：

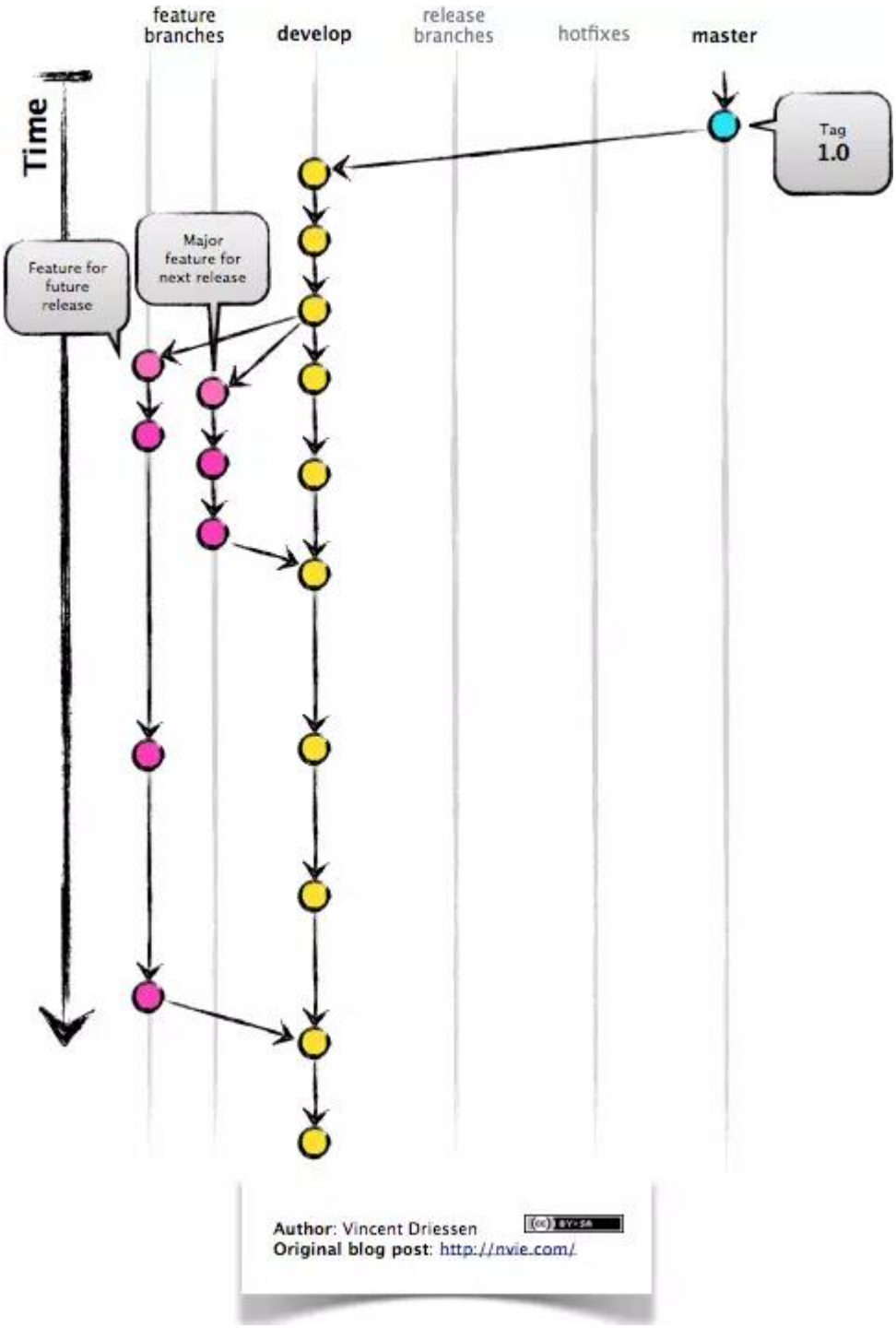
- 并行开发：GitFlow 可以很方便的实现并行开发：每个新功能都会建立一个新的 feature 分支，从而和已经完成的功能隔离开来，而且只有在新功能完成开发的情况下，其对应的 feature 分支才会合并到主开发分支上（也就是我们经常说的 develop 分支）。另外，如果你正在开发某个功能，同时又有一个新的功能需要开发，你只需要提交当前 feature 的代码，然后创建另外一个 feature 分支并完成新功能开发。然后再切回之前的 feature 分支即可继续完成之前功能的开发。
- 协作开发：GitFlow 还支持多人协同开发，因为每个 feature 分支上改动的代码都只是为了让某个新的 feature 可以独立运行。同时我们也很容易知道每个人都在干啥。

- 发布阶段：当一个新 feature 开发完成的时候，它会被合并到 develop 分支，这个分支主要用来暂时保存那些还没有发布的内容，所以如果需要再开发新的 feature，我们只需要从 develop 分支创建新分支，即可包含所有已经完成的 feature 。
- 支持紧急修复：GitFlow 还包含了 hotfix 分支。这种类型的分支是从某个已经发布的 tag 上创建出来并做一个紧急的修复，而且这个紧急修复只影响这个已经发布的 tag，而不会影响到你正在开发的新 feature。

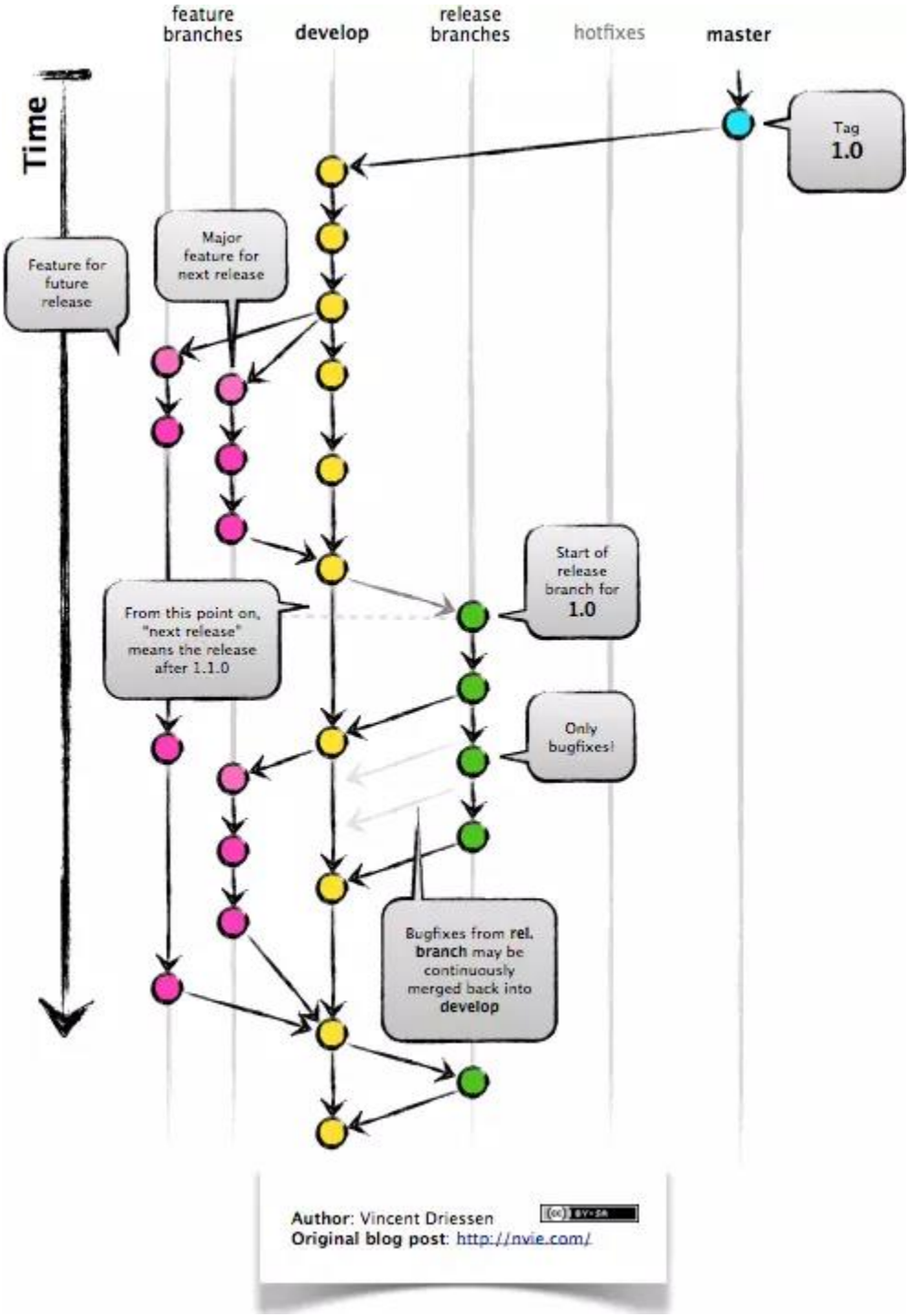
然后就是 GitFlow 最经典的几张流程图，一定要理解：



feature 分支都是从 develop 分支创建，完成后再合并到 develop 分支上，等待发布。

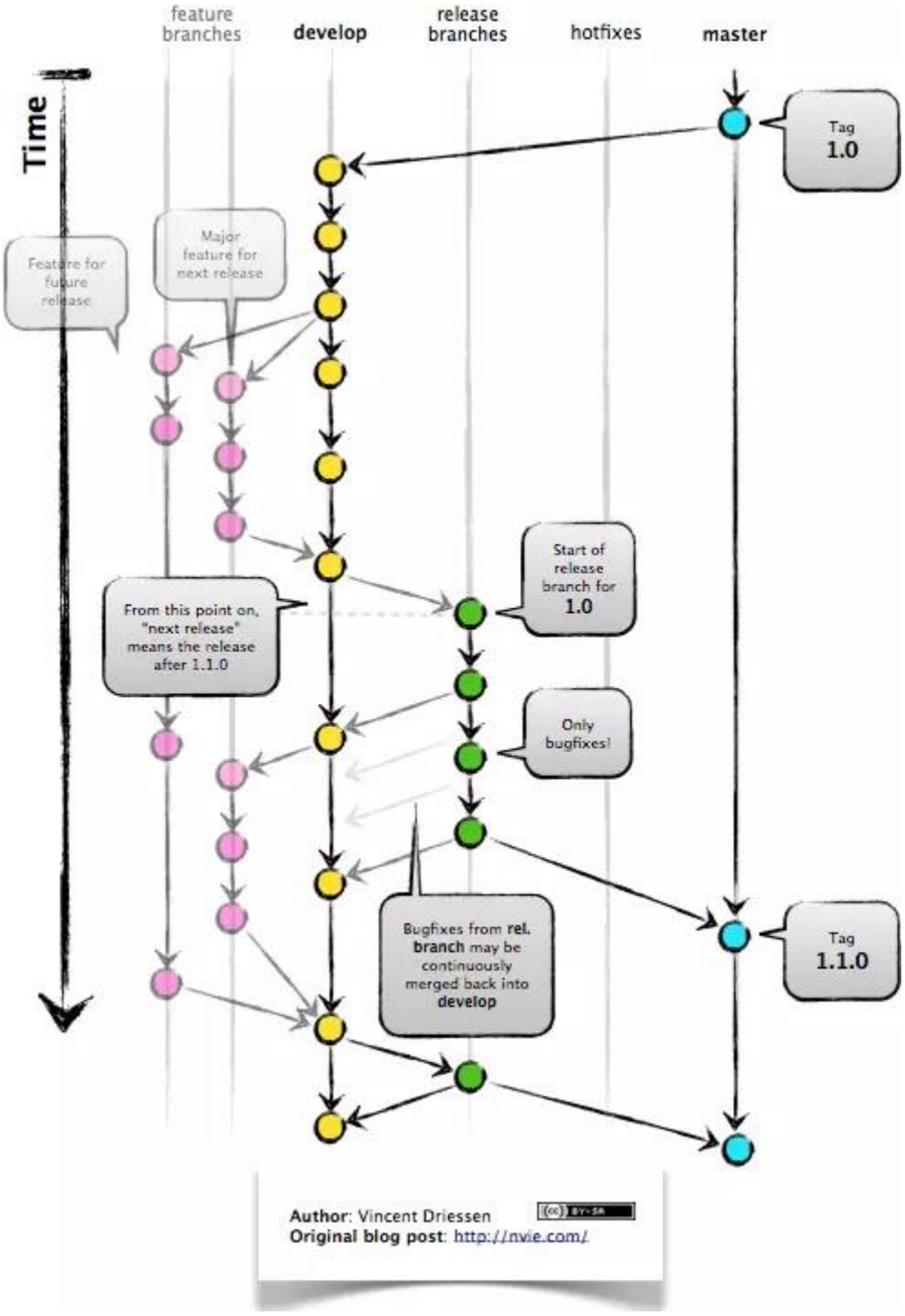


当需要发布时，我们从 develop 分支创建一个 release 分支



然后这个 release 分支会发布到测试环境进行测试，如果发现问题就在这个分支直接进行修复。在所有问题修复之前，我们会不停的重复**发布->测试->修复->重新发布->重新测试**这个流程。

发布结束后，这个 release 分支会合并到 develop 和 master 分支，从而保证不会有代码丢失。

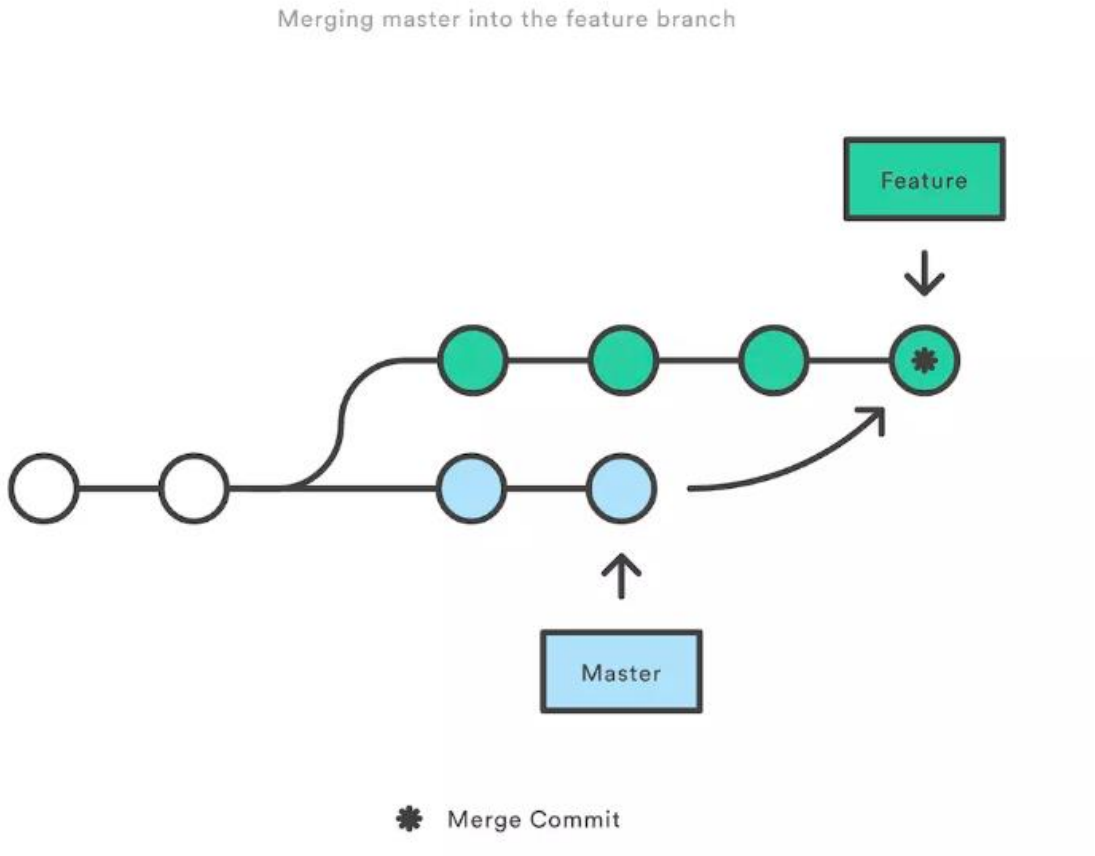


master 分支只跟踪已经发布的代码，合并到 master 上的 commit 只能来自 release 分支和 hotfix 分支。
hotfix 分支的作用是紧急修复一些 Bug。
它们都是从 master 分支上的某个 tag 建立，修复结束后再合并到 develop 和 master 分支上。

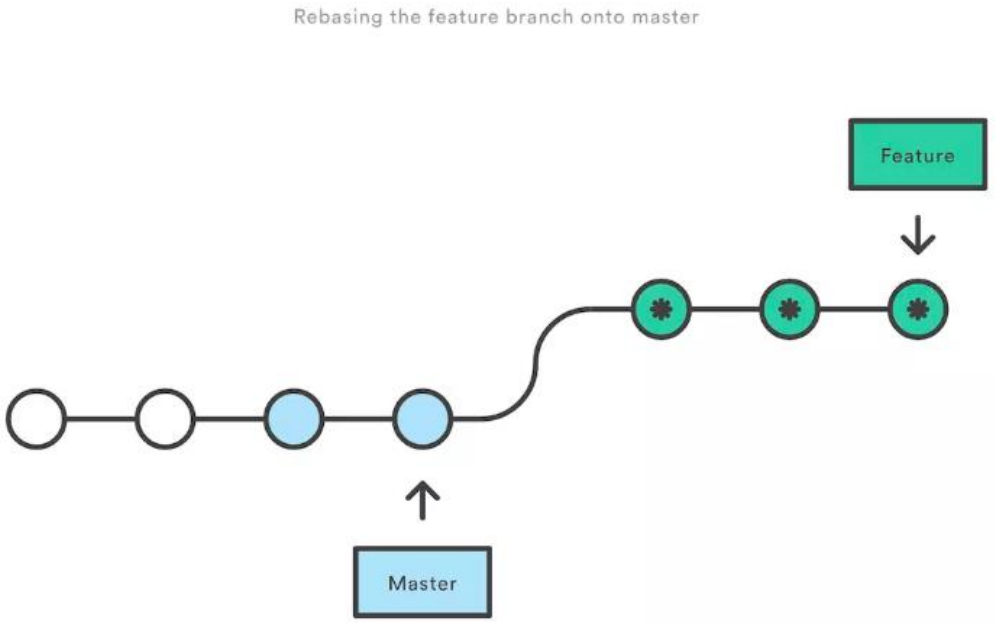
- rebase 与 merge 的区别?

参考回答：

git rebase 和 git merge 一样都是用于从一个分支获取并且合并到当前分支.
假设一个场景,就是我们开发的[feature/todo]分支要合并到 master 主分支,那么用 rebase 或者 merge 有什么不同呢?



  - marge 特点：自动创建一个新的 commit 如果合并的时候遇到冲突，仅需要修改后重新 commit
  - 优点：记录了真实的 commit 情况，包括每个分支的详情
  - 缺点：因为每次 merge 会自动产生一个 merge commit，所以在使用一些 git 的 GUI tools，特别是 commit 比较频繁时，看到分支很杂乱。



- rebase 特点：会合并之前的 commit 历史
- 优点：得到更简洁的项目历史，去掉了 merge commit
- 缺点：如果合并出现代码问题不容易定位，因为 re-write 了 history

因此,当需要保留详细的合并信息的时候建议使用 git merge，特别是需要将分支合并进入 master 分支时；当发现自己修改某个功能时，频繁进行了 git commit 提交时，发现其实过多的提交信息没有必要时，可以尝试 git rebase.

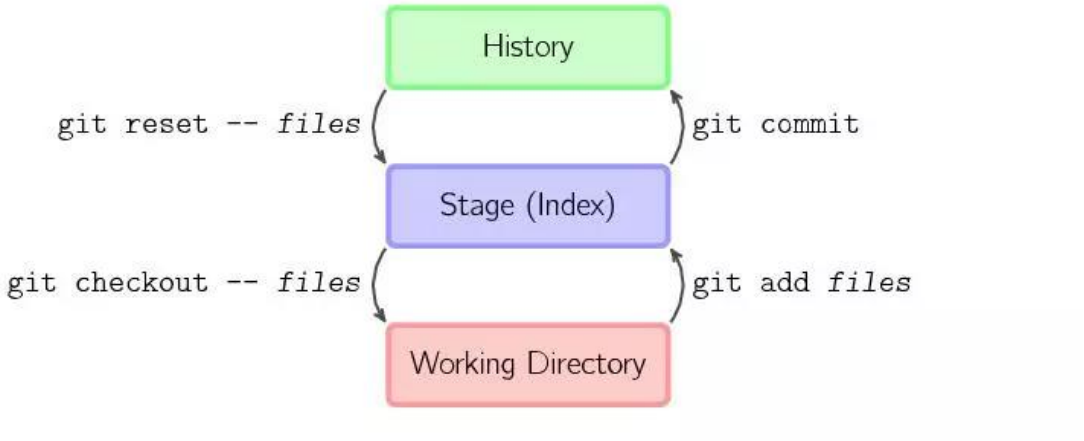
- **git reset、git revert 和 git checkout 有什么区别**

参考回答：

这个问题同样也需要先了解 git 仓库的三个组成部分：工作区（Working Directory）、暂存区（Stage）和历史记录区（History）。

- 工作区：在 git 管理下的正常目录都算是工作区，我们平时的编辑工作都是在工作区完成
- 暂存区：临时区域。里面存放将要提交文件的快照
- 历史记录区：git commit 后的记录区

三个区的转换关系以及转换所使用的命令：



git reset、git revert 和 git checkout 的共同点：用来撤销代码仓库中的某些更改。

然后是不同点：

首先，从 commit 层面来说：

- git reset 可以将一个分支的末端指向之前的一个 commit。然后再下次 git 执行垃圾回收的时候，会把这个 commit 之后的 commit 都扔掉。git reset 还支持三种标记，用来标记 reset 指令影响的范围：
  - --mixed：会影响到暂存区和历史记录区。也是默认选项
  - --soft：只影响历史记录区
  - --hard：影响工作区、暂存区和历史记录区

注意：因为 git reset 是直接删除 commit 记录，从而会影响到其他开发人员的分支，所以不要在公共分支（比如 develop）做这个操作。

  - git checkout 可以将 HEAD 移到一个新的分支，并更新工作目录。因为可能会覆盖本地的修改，所以执行这个指令之前，你需要 stash 或者 commit 暂存区和工作区的更改。
- git revert 和 git reset 的目的是一样的，但是做法不同，它会以创建新的 commit 的方式来撤销 commit，这样能保留之前的 commit 历史，比较安全。另外，同样因为可能会覆盖本地的修改，所以执行这个指令之前，你需要 stash 或者 commit 暂存区和工作区的更改。

然后，从文件层面来说：

- git reset 只是把文件从历史记录区拿到暂存区，不影响工作区的内容，而且不支持 --mixed、--soft 和 --hard。
- git checkout 则是把文件从历史记录拿到工作区，不影响暂存区的内容。
- git revert 不支持文件层面的操作。

- webpack 和 gulp 区别（模块化与流的区别）

参考回答：
gulp 强调的是前端开发的工作流程，我们可以通过配置一系列的 task，定义 task 处理的事务（例如文件压缩合并、雪碧图、启动 server、版本控制等），然后定义执行顺序，来让 gulp 执行这些 task，从而构建项目的整个前端开发流程。
webpack 是一个前端模块化方案，更侧重模块打包，我们可以把开发中的所有资源（图片、js 文件、css 文件等）都看成模块，通过 loader（加载器）和 plugins（插件）对资源进行处理，打包成符合生产环境部署的前端资源。

3.2 | Vue 框架

- 有使用过 Vue 吗？说说你对 Vue 的理解

参考回答：
Vue 是一个构建数据驱动的渐进性框架，它的目标是通过 API 实现响应数据绑定和视图更新。

- 说说 Vue 的优缺点

参考回答：
优点：
1、数据驱动视图，对真实 dom 进行抽象出 virtual dom（本质就是一个 js 对象），并配合 diff 算法、响应式和观察者、异步队列等手段以最小代价更新 dom，渲染页面
2、组件化，组件用单文件的形式进行代码的组织编写，使得我们可以在一个文件里编写 html\css（scoped 属性配置 css 隔离）\js 并且配合 Vue-loader 之后，支持更强大的预处理器等功能
3、强大且丰富的 API 提供一系列的 api 能满足业务开发中各类需求
4、由于采用虚拟 dom，让 Vue ssr 先天就足
5、生命周期钩子函数，选项式的代码组织方式，写熟了还是蛮顺畅的，但仍然有优化空间（Vue3 composition-api）
6、生态好，社区活跃
缺点：
1、由于底层基于 Object.defineProperty 实现响应式，而这个 api 本身不支持 IE8 及以下浏览器
2、csr 的先天不足，首屏性能问题（白屏）
3、由于百度等搜索引擎爬虫无法爬取 js 中的内容，故 spa 先天就对 seo 优化心有余力不足（谷歌的 puppeteer 就挺牛逼的，实现预渲染底层也是用到了这个工具）

- Vue 和 React 有什么不同？使用场景分别是什么？

参考回答：
1、Vue 是完整一套由官方维护的框架，核心库主要有由尤雨溪大神独自维护，而 React 是不要脸的书维护（很多库由社区维护），曾经一段时间很多人质疑 Vue 的后续维护性，似乎这并不是问题。
2、Vue 上手简单，进阶式框架，白话说你可以学一点，就可以在你项目中去用一点，你不一定需要一次性学习整个 Vue 才能去使用它，而 React，恐怕如果你这样会面对项目束手无策。
3、语法上 Vue 并不限制你必须 es6+完全 js 形式编写页面，可以视图和 js 逻辑尽可能分离，减少很多人看不惯 React-jsx 的恶心嵌套，毕竟都是作为前端开发者，还是更习惯于 html 干净。
4、很多人说 React 适合大型项目，适合什么什么，Vue 轻量级，适合移动端中小型项目，其实我想说，说这话的人是心里根本没点逼数，Vue 完全可以应对复杂的大型应用，甚至于说如果你 React 学的不是很好，写出来的东西或根本不如 Vue 写的，毕竟 Vue 跟着官方文档撸就行，自有人帮你规范，而 React 比较懒散自由，可以自由发挥
5、Vue 在国内人气明显胜过 React，这很大程度上得益于它的很多语法包括编程思维更符合国人思想。

- 什么是虚拟 DOM?

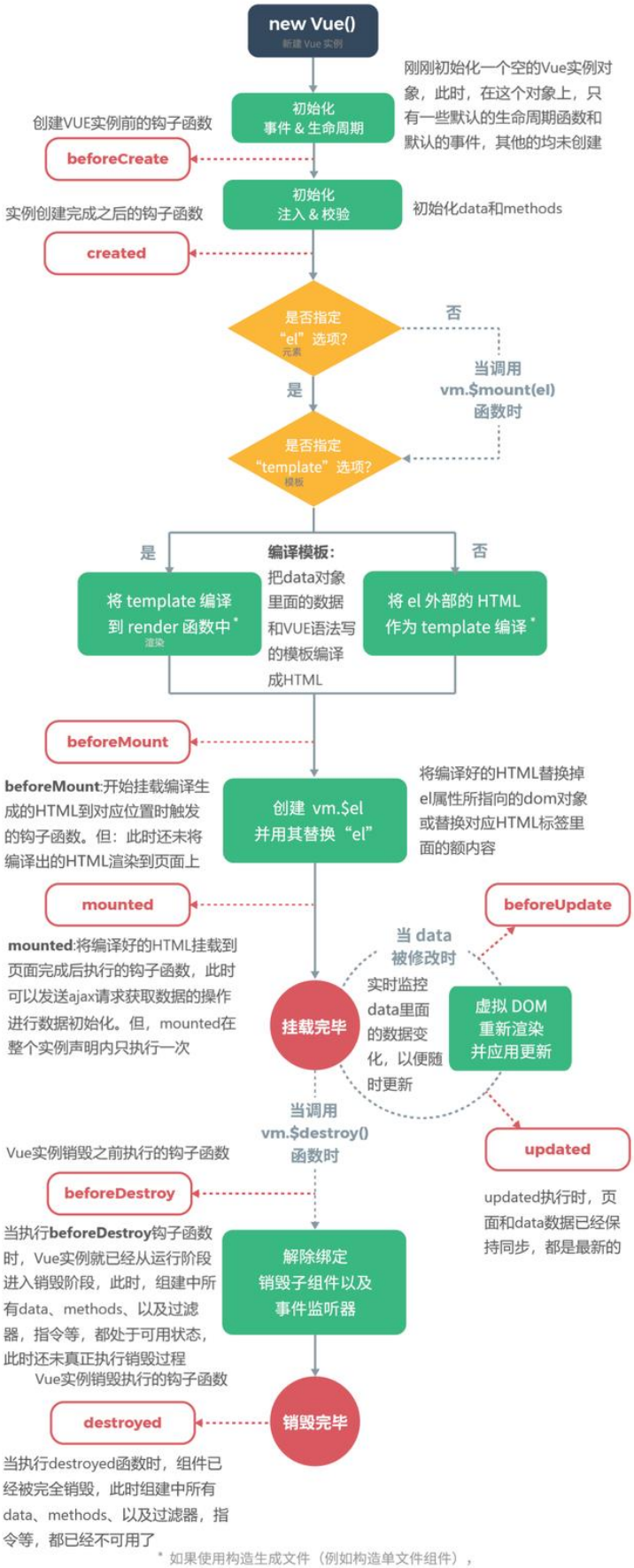
参考回答：
虚拟 dom 是相对于浏览器所渲染出来的真实 dom 的，在 react，vue 等技术出现之前，我们要改变页面展示的内容只能通过遍历查询 dom 树的方式找到需要修改的 dom 然后修改样式行为或者结构，来达到更新 ui 的目的。
这种方式相当消耗计算资源，因为每次查询 dom 几乎都需要遍历整颗 dom 树，如果建立一个与 dom 树对应的虚拟 dom 对象（ js 对象），以对象嵌套的方式来表示 dom 树，那么每次 dom 的更改就变成了 js 对象的属性的更改，这样一来就能查找 js 对象的属性变化要比查询 dom 树的性能开销小。

- 请描述下 vue 的生命周期是什么？

参考回答：
    - 生命周期就是 vue 从开始创建到销毁的过程，分为四大步（创建，挂载，更新，销毁），每一步又分为两小步，如 beforeCreate，created。beforeCreate 前，也就是 new Vue 的时候会初始化事件和生命周期；beforeCreate 和 created 之间会挂载 Data，绑定事件；接下来会根据 el 挂载页面元素，如果没有设置 el 则生命周期结束，直到手动挂载；el 挂载结束后，根据 templete/outerHTML(el)渲染页面；在 beforeMount 前虚拟 DOM 已经创建完成；之后在 mounted 前，将 vm.$el 替换掉页面元素 el;mounted 将虚拟 dom 挂载到真实页面（此时页面已经全部渲染完成）；之后发生数据变化时触发 beforeUpdate 和 updated 进行一些操作；最后主动调用销毁函数或者组件自动销毁时

beforeDestroy，手动撤销监听事件，计时器等；destroyed 时仅存在 Dom 节点，其他所有东西已自动销毁。这就是我所理解的 vue 的一个完整的生命周期。

-



* 如果使用构造生成文件（例如构造单文件组件），模板编译将提前执行

- vue 如何监听键盘事件？

参考回答：

1. @keyup. 方法

```
<template>
  <input ref="myInput" type="text" value="hello world" autofocus
@keyup.enter="handleKey">
</template>
<script>
  export default {
    methods: {
      handleKey(e) {
        console.log(e)
      }
    }
  }
</script>
```

2. addEventListener

```
<script>
  export default {
    mounted() {
      document.addEventListener('keyup', this.handleKey)
    },
    beforeDestroy() {
      document.removeEventListener('keyup', this.handleKey)
    },
    methods: {
      handleKey(e) {
        console.log(e)
      }
    }
  }
</script><script>
  export default {
    mounted() {
      document.addEventListener('keyup', this.handleKey)
    },
    beforeDestroy() {
      document.removeEventListener('keyup', this.handleKey)
    },
    methods: {
      handleKey(e) {
```

```
        console.log(e)
      }
    }
  }
</script>
```

- watch 怎么深度监听对象变化

参考回答：

deep 设置为 true 就可以监听到对象的变化

```
let vm=new Vue({
        el:"#first",
        data:{msg:{name:'北京'}},
        watch:{
            msg:{
                handler (newMsg,oldMsg){
                    console.log(newMsg);
                },
                immediate:true,
                deep:true
            }
        }
    })
```

- 删除数组用 delete 和 Vue.delete 有什么区别？

参考回答：

- delete：只是被删除数组成员变为 empty / undefined，其他元素键值不变
- Vue.delete：直接删了数组成员，并改变了数组的键值（对象是响应式的，确保删除能触发更新视图，这个方法主要用于避开 Vue 不能检测到属性被删除的限制）

- watch 和计算属性有什么区别？

参考回答：

通俗来讲，既能用 computed 实现又可以用 watch 监听来实现的功能，推荐用 computed，重点在于 **computed 的缓存功能**

computed 计算属性是用来声明式的描述一个值依赖了其它的值，当所依赖的值或者变量改变时，计算属性也会跟着改变；

watch 监听的是已经在 data 中定义的变量，当该变量变化时，会触发 watch 中的方法。

- Vue 双向绑定原理

参考回答：
Vue 数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的。利用了 Object.defineProperty() 这个方法重新定义了对象获取属性值(get)和设置属性值(set)。

- v-model 是什么？有什么用呢？

参考回答：
一则语法糖，相当于 v-bind:value="xxx" 和 @input，意思是绑定了一个 value 属性的值，子组件可对 value 属性监听，通过$emit('input', xxx)的方式给父组件通讯。自己实现 v-model 方式的组件也是这样的思路。

- axios 是什么？怎样使用它？怎么解决跨域的问题？

参考回答：
axios 的是一种异步请求，用法和 ajax 类似，安装 npm install axios --save 即可使用，请求中包括 get,post,put, patch ,delete 等五种请求方式，解决跨域可以在请求头中添加 Access-Control-Allow-Origin，也可以在 index.js 文件中更改 proxyTable 配置等解决跨域问题。

- 在 vue 项目中如何引入第三方库（比如 jQuery）？有哪些方法可以做到？

参考回答：
1、绝对路径直接引入
在 index.html 中用 script 引入
```
<script src="./static/jquery-1.12.4.js"></script>
```
然后在 webpack 中配置 external
```
externals: { 'jquery': 'jQuery' }
```
在组件中使用时 import
```
import $ from 'jquery'
```
2 、在 webpack 中配置 alias
```
resolve: { extensions: ['.js', '.vue', '.json'], alias: { '@': resolve('src'), 'jquery': resolve('static/jquery-1.12.4.js') } }
```
然后在组件中 import
3、在 webpack 中配置 plugins
```
plugins: [ new webpack.ProvidePlugin({ $: 'jquery' }) ]
```
全局使用，但在使用 eslint 情况下会报错，需要在使用了 $ 的代码前添加 /* eslint-disable*/ 来去掉 ESLint 的检查。

- 说说 Vue React angularjs jquery 的区别

参考回答：
JQuery 与另外几者最大的区别是，JQuery 是事件驱动，其他两者是数据驱动。
JQuery 业务逻辑和 UI 更改该混在一起， UI 里面还参杂这交互逻辑，让本来混乱的逻辑更加混乱。
Angular，Vue 是双向绑定，而 React 不是
其他还有设计理念上的区别等

- Vue3.0 里为什么要用 Proxy API 替代 defineProperty API？

参考回答：
响应式优化。
a. **defineProperty API 的局限性最大原因是它只能针对单例属性做监听。**
Vue2.x 中的响应式实现正是基于 defineProperty 中的 descriptor，对 data 中的属性做了遍历 + 递归，为每个属性设置了 getter、setter。
这也就是为什么 Vue 只能对 data 中预定义过的属性做出响应的原因，在 Vue 中使用下标的方式直接修改属性的值或者添加一个预先不存在的对象属性是无法做到 setter 监听的，这是 defineProperty 的局限性。
b. **Proxy API 的监听是针对一个对象的，那么对这个对象的所有操作会进入监听操作，这就完全可以代理所有属性，将会带来很大的性能提升和更优的代码。**
Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。
c. **响应式是惰性的**
在 Vue.js 2.x 中，对于一个深层属性嵌套的对象，要劫持它内部深层次的变化，就需要递归遍历这个对象，**执行 Object.defineProperty 把每一层对象数据都变成响应式的，这无疑会有很大的性能消耗**。
在 Vue.js 3.0 中，使用 Proxy API 并不能监听到对象内部深层次的属性变化，因此它的处理方式是在 getter 中去递归响应式，这样的好处是**真正访问到的内部属性才会变成响应式，简单的可以说是按需实现响应式，减少性能消耗。**
基础用法：

- Vue3.0 编译做了哪些优化？

参考回答：

a. 生成 Block tree

Vue.js 2.x 的数据更新并触发重新渲染的粒度是组件级的，单个组件内部 需要遍历该组件的整个 vnode 树。**在 2.0 里，渲染效率的快慢与组件大小成正相关：组件越大，渲染效率越慢。并且，对于一些静态节点，又无数据更新，这些遍历都是性能浪费。**

Vue.js 3.0 做到了通过编译阶段对静态模板的分析，编译生成了 Block tree。 **Block tree 是一个将模版基于动态节点指令切割的嵌套区块**，每个 区块内部的节点结构是固定的，每个区块只需要追踪自身包含的动态节点。所以，**在 3.0 里，渲染效率不再与模板大小成正相关，而是与模板中动态节点的数量成正相关。**

b. slot 编译优化

Vue.js 2.x 中，如果有一个组件传入了 slot，那么每次父组件更新的时候，会强制使子组件 update，造成性能的浪费。

Vue.js 3.0 优化了 slot 的生成，使得非动态 slot 中属性的更新只会触发子组件的更新。**动态 slot 指的是在 slot 上面使用 v-if，v-for，动态 slot 名字等会导致 slot 产生运行时动态变化但是又无法被子组件 track 的操作。**

c. diff 算法优化

- **Vue3.0 新特性 —— Composition API 与 React.js 中 Hooks 的异同点**

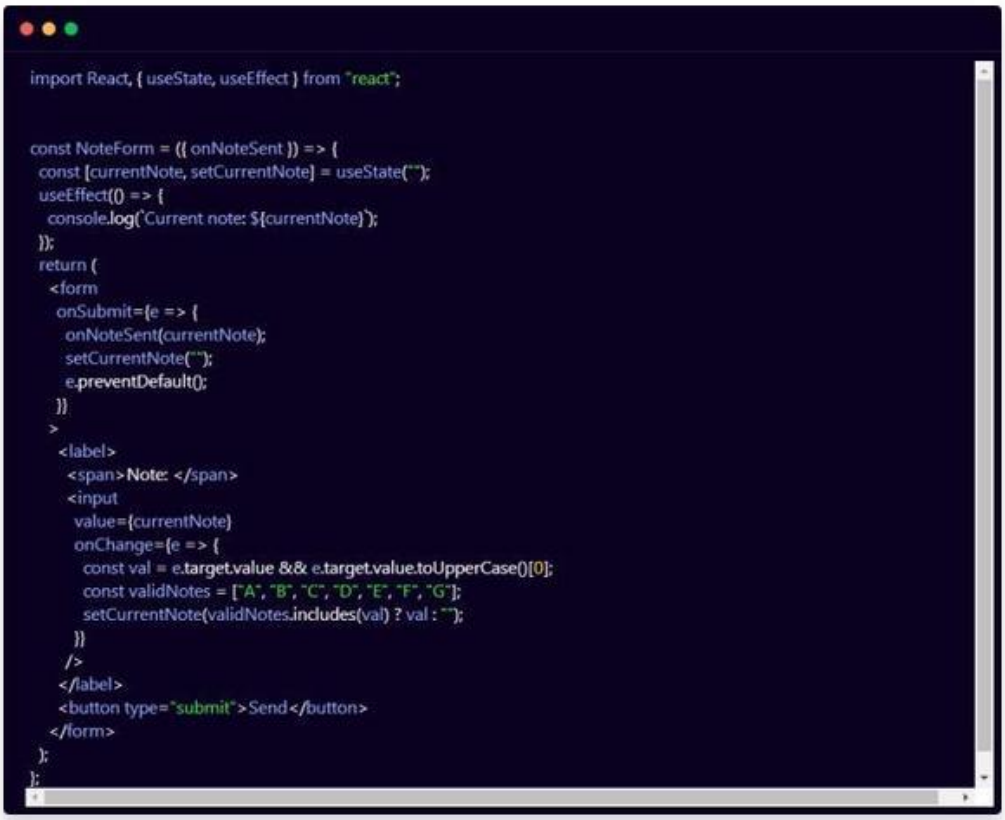
参考回答：

a. React.js 中的 Hooks 基本使用

React Hooks 允许你 "勾入" 诸如组件状态和副作用处理等 React 功能中。Hooks 只能用在函数组件中，并允许我们在不需要创建类的情况下将状态、副作用处理和更多东西带入组件中。

React 核心团队奉上的采纳策略是不反对类组件，所以你可以升级 React 版本、在新组件中开始尝试 Hooks，并保持既有组件不做任何更改。

案例：



```
import React, { useState, useEffect } from "react";


const NoteForm = ({ onNoteSent }) => {
 const [currentNote, setCurrentNote] = useState("");
 useEffect(() => {
  console.log(`Current note: ${currentNote}`);
 });
 return (
  <form
   onSubmit={e => {
    onNoteSent(currentNote);
    setCurrentNote("");
    e.preventDefault();
   }}
  >
   <label>
    <span>Note: </span>
    <input
     value={currentNote}
     onChange={e => {
      const val = e.target.value && e.target.value.toUpperCase()[0];
      const validNotes = ["A", "B", "C", "D", "E", "F", "G"];
      setCurrentNote(validNotes.includes(val) ? val : "");
     }}
    />
   </label>
   <button type="submit">Send</button>
  </form>
 );
};
```

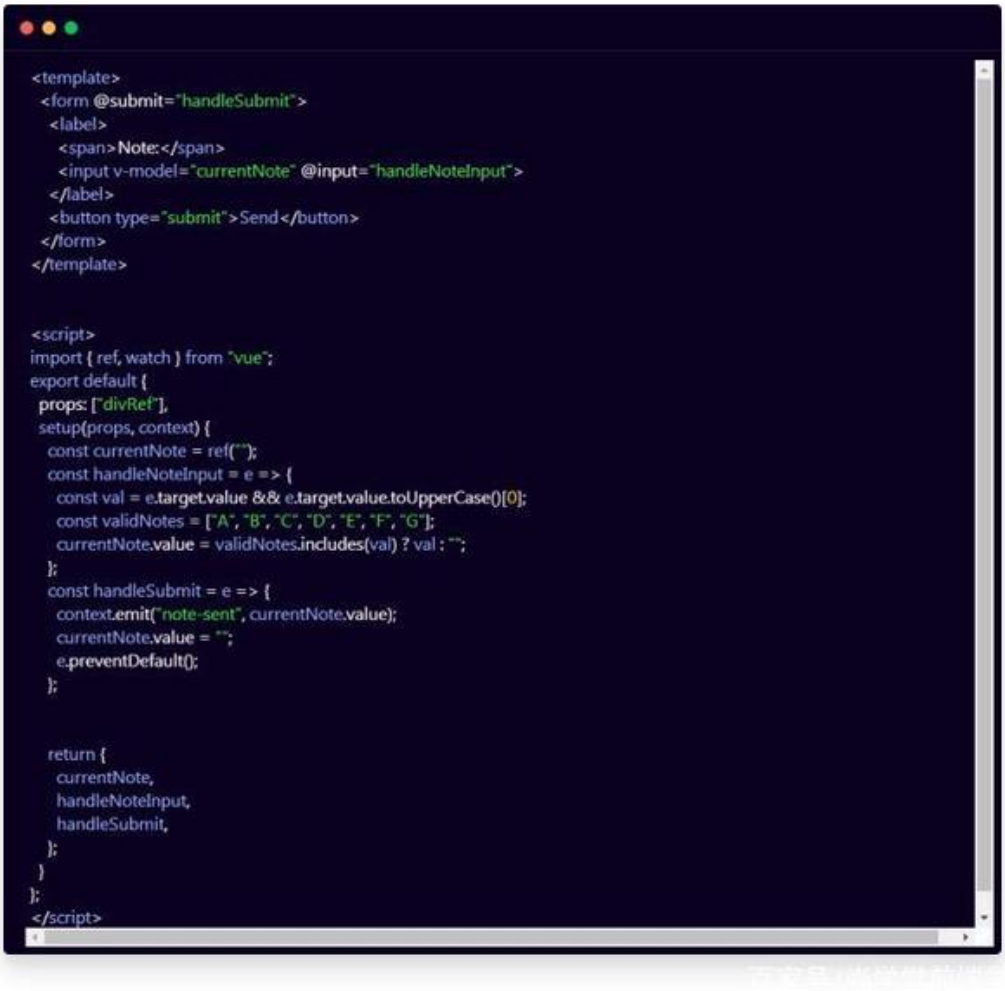
useState 和 useEffect 是 React Hooks 中的一些例子，使得函数组件中也能增加状态和运行副作用。

我们也可以自定义一个 Hooks，它打开了代码复用性和扩展性的新大门。

b. Vue Composition API 基本使用

Vue Composition API 围绕一个新的组件选项 setup 而创建。setup() 为 Vue 组件提供了状态、计算值、watcher 和生命周期钩子。

并没有让原来的 API（Options-based API）消失。允许开发者 结合使用新旧两种 API（向下兼容）。



```
<template>
 <form @submit="handleSubmit">
  <label>
   <span>Note:</span>
   <input v-model="currentNote" @input="handleNoteInput">
  </label>
  <button type="submit">Send</button>
 </form>
</template>


<script>
import { ref, watch } from "vue";
export default {
 props: ["divRef"],
 setup(props, context) {
  const currentNote = ref("");
  const handleNoteInput = e => {
   const val = e.target.value && e.target.value.toUpperCase()[0];
   const validNotes = ["A", "B", "C", "D", "E", "F", "G"];
   currentNote.value = validNotes.includes(val) ? val : "";
  };
  const handleSubmit = e => {
   context.emit("note-sent", currentNote.value);
   currentNote.value = "";
   e.preventDefault();
  };


  return {
   currentNote,
   handleNoteInput,
   handleSubmit,
  };
 }
};
</script>
```

c. 原理

React hook 底层是基于链表实现，调用的条件是每次组件被 render 的时候都会顺序执行所有的 hooks。

Vue hook 只会被注册调用一次，Vue 能避开这些麻烦的问题，原因在于它对数据的响应是基于 proxy 的，对数据直接代理观察。（这种场景下，只要任何一个更改 data 的地方，相关的 function 或者 template 都会被重新计算，因此避开了 React 可能遇到的性能上的问题）。

React 中，数据更改的时候，会导致重新 render，重新 render 又会重新把 hooks 重新注册一次，所以 React 复杂程度会高一些。

- **Vue3.0 是如何变得更快的？（底层，源码）**

参考回答：

a. diff 方法优化

Vue2.x 中的虚拟 dom 是进行全量的对比。

Vue3.0 中新增了静态标记（PatchFlag）：在与上次虚拟结点进行对比的时候，值对比带有 patch flag 的节点，并且可以通过 flag 的信息得知当前节点要对比的具体内容化。

b. hoistStatic 静态提升

Vue2.x ： 无论元素是否参与更新，每次都会重新创建。

Vue3.0 ： 对不参与更新的元素，只会被创建一次，之后会在每次渲染时候被不停的复用。

c. cacheHandlers 事件侦听器缓存

默认情况下 onClick 会被视为动态绑定，所以每次都会去追踪它的变化但是因为是同一个函数，所以没有追踪变化，直接缓存起来复用即可。

原作者姓名： 欧阳呀

- **vue 要做权限管理该怎么做？如果控制到按钮级别的权限怎么做？**

参考回答：

按钮级别的权限：

https://panjiachen.github.io/vue-element-admin-site/zh/guide/essentials/permission.html#%E6%8C%87%E4%BB%A4%E6%9D%83%E9%99%90

- **vue 在 created 和 mounted 这两个生命周期中请求数据有什么区别呢？**

参考回答：

看实际情况，一般在 created（或 beforeRouter） 里面就可以，如果涉及到需要页面加载完成之后的话就用 mounted。

在 created 的时候，视图中的 html 并没有渲染出来，所以此时如果直接去操作 html 的 dom 节点，一定找不到相关的元素

而在 mounted 中，由于此时 html 已经渲染出来了，所以可以直接操作 dom 节点，（此时 document.getelementById 即可生效了）。

- **说说你对 proxy 的理解**

参考回答：

vue 的数据劫持有两个缺点:

1、无法监听通过索引修改数组的值的变化

2、无法监听 object 也就是对象的值的变化

所以 vue2.x 中才会有$set 属性的存在

proxy 是 es6 中推出的新 api，可以弥补以上两个缺点，所以 vue3.x 版本用 proxy 替换 object.defineproperty。

3.3 | React 框架

- angularJs 和 React 区别

参考回答：
React 对比 Angular 是思想上的转变，它也并不是一个库，是一种开发理念，组件化，分治的管理，数据与 view 的一体化。它只有一个中心,发出状态，渲染 view，对于虚拟 dom 它并没有提高渲染页面的性能，它提供更多的是利用 jsx 便捷生成 dom 元素，利用组件概念进行分治管理页面每个部分(例如 header section footer slider)

- redux 中间件

参考回答：
中间件提供第三方插件的模式，自定义拦截 action -> reducer 的过程。变为 action -> middlewares -> reducer 。这种机制可以让我们改变数据流，实现如异步 action ，action 过滤，日志输出，异常报告等功能。

常见的中间件： redux-logger：提供日志输出；redux-thunk：处理异步操作；redux-promise：处理异步操作；actionCreator 的返回值是 promise

- redux 有什么缺点

参考回答：
1.一个组件所需要的数据，必须由父组件传过来，而不能像 flux 中直接从 store 取。

2.当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新 render，可能会有效率影响，或者需要写复杂的 shouldComponentUpdate 进行判断。

- React 组件的划分业务组件技术组件？

参考回答：
根据组件的职责通常把组件分为 UI 组件和容器组件。UI 组件负责 UI 的呈现，容器组件负责管理数据和逻辑。两者通过 React-Redux 提供 connect 方法联系起来。

- React 生命周期函数

参考回答：
一、初始化阶段：

getDefaultProps:获取实例的默认属性

getInitialState:获取每个实例的初始化状态

componentWillMount：组件即将被装载、渲染到页面上

render:组件在这里生成虚拟的 DOM 节点

componentDidMount:组件真正在被装载之后

二、运行中状态：

componentWillReceiveProps:组件将要接收到属性的时候调用

shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以返回 false，接收数据后不更新，阻止 render 调用，后面的函数不会被继续执行了）

componentWillUpdate:组件即将更新不能修改属性和状态

render:组件重新描绘

componentDidUpdate:组件已经更新

三、销毁阶段：

componentWillUnmount:组件即将销毁

- **React 性能优化是哪个周期函数？**

参考回答：
shouldComponentUpdate 这个方法用来判断是否需要调用 render 方法重新描绘 dom。因为 dom 的描绘非常消耗性能，如果我们能在 shouldComponentUpdate 方法中能够写出更优化的 dom diff 算法，可以极大的提高性能。

- **为什么虚拟 dom 会提高性能?**

参考回答：
虚拟 dom 相当于在 js 和真实 dom 中间加了一个缓存，利用 dom diff 算法避免了没有必要的 dom 操作，从而提高性能。

具体实现步骤如下：

1.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；

2.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 2 所记录的差异应用到步骤 1 所构建的真正的 DOM 树上，视图就更新了。

- diff 算法?

参考回答：
1.把树形结构按照层级分解，只比较同级元素。

2.给列表结构的每个单元添加唯一的 key 属性，方便比较。

3.React 只会匹配相同 class 的 component（这里面的 class 指的是组件的名字）

4.合并操作，调用 component 的 setState 方法的时候，React 将其标记为 dirty.到每一个事件循环结束，React 检查所有标记 dirty 的 component 重新绘制.

6. 选择性子树渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能。

- React 性能优化方案

参考回答：
1）重写 shouldComponentUpdate 来避免不必要的 dom 操作。

2）使用 production 版本的 React.js。

3）使用 key 来帮助 React 识别列表中所有子组件的最小变化

- 简述 flux 思想

参考回答：
Flux 的最大特点，就是数据的"单向流动"。

1.用户访问 View

2.View 发出用户的 Action

3.Dispatcher 收到 Action，要求 Store 进行相应的更新

4.Store 更新后，发出一个"change"事件

5.View 收到"change"事件后，更新页面

- React 项目用过什么脚手架？Mern? Yeoman?

参考回答：
Mern：MERN 是脚手架的工具，它可以很容易地使用 Mongo, Express, React and NodeJS 生成同构 JS 应用。它最大限度地减少安装时间，并得到您使用的成熟技术来加速开发。

- 你了解 React 吗？

参考回答：
了解，React 是 facebook 搞出来的一个轻量级的组件库，用于解决前端视图层的一些问题，就是 MVC 中 V 层的问题，它内部的 Instagram 网站就是用 React 搭建的。

- React 解决了什么问题？

参考回答：
解决了三个问题： 1.组件复用问题， 2.性能问题，3.兼容性问题：

- React 的协议？

参考回答：
React 遵循的协议是“BSD 许可证 + 专利开源协议”，这个协议比较奇葩，如果你的产品跟 facebook 没有竞争关系，你可以自由的使用 React，但是如果有竞争关系，你的 React 的使用许可将会被取消

- 了解 shouldComponentUpdate 吗？

参考回答：
React 虚拟 dom 技术要求不断的将 dom 和虚拟 dom 进行 diff 比较，如果 dom 树比价大，这种比较操作会比较耗时，因此 React 提供了 shouldComponentUpdate 这种补丁函数，如果对于一些变化，如果我们不希望某个组件刷新，或者刷新后跟原来其实一样，就可以使用这个函数直接告诉 React，省去 diff 操作，进一步的提高了效率。

- React 的工作原理？

参考回答：

React 会创建一个虚拟 DOM(virtual DOM)。当一个组件中的状态改变时，React 首先会通过 "diffing" 算法来标记虚拟 DOM 中的改变，第二步是调节(reconciliation)，会用 diff 的结果来更新 DOM。

- 使用 React 有何优点？

参考回答：
1.只需查看 render 函数就会很容易知道一个组件是如何被渲染的

2.JSX 的引入，使得组件的代码更加可读，也更容易看懂组件的布局，或者组件之间是如何互相引用的

3.支持服务端渲染，这可以改进 SEO 和性能

4.易于测试

5.React 只关注 View 层，所以可以和其它任何框架(如 Backbone.js, Angular.js)一起使用

- 展示组件(Presentational component)和容器组件(Container component)之间有何不同？

参考回答：
1.展示组件关心组件看起来是什么。展示专门通过 props 接受数据和回调，并且几乎不会有自身的状态，但当展示组件拥有自身的状态时，通常也只关心 UI 状态而不是数据的状态。

2.容器组件则更关心组件是如何运作的。容器组件会为展示组件或者其它容器组件提供数据和行为(behavior)，它们会调用 Flux actions，并将其作为回调提供给展示组件。容器组件经常是有状态的，因为它们是(其它组件的)数据源

- 类组件(Class component)和函数式组件(Functional component)之间有何不同？

参考回答：
1.类组件不仅允许你使用更多额外的功能，如组件自身的状态和生命周期钩子，也能使组件直接访问 store 并维持状态

2.当组件仅是接收 props，并将组件自身渲染到页面时，该组件就是一个 ’无状态组件(stateless component)’，可以使用一个纯函数来创建这样的组件。这种组件也被称为哑组件(dumb components)或展示组件

- (组件的)状态(state)和属性(props)之间有何不同？

参考回答：
1. State 是一种数据结构，用于组件挂载时所需数据的默认值。State 可能会随着时间的推移而发生突变，但多数时候是作为用户事件行为的结果。

2. Props(properties 的简写)则是组件的配置。props 由父组件传递给子组件，并且就子组件而言，props 是不可变的(immutable)。组件不能改变自身的 props，但是可以把其子组件的 props 放在一起(统一管理)。Props 也不仅仅是数据--回调函数也可以通过 props 传递。

- 应该在 React 组件的何处发起 Ajax 请求？

参考回答：
在 React 组件中，应该在 componentDidMount 中发起网络请求。这个方法会在组件第一次“挂载”(被添加到 DOM)时执行，在组件的生命周期中仅会执行一次。更重要的是，你不能保证在组件挂载之前 Ajax 请求已经完成，如果是这样，也就意味着你将尝试在一个未挂载的组件上调用 setState，这将不起作用。在 componentDidMount 中发起网络请求将保证这有一个组件可以更新了。

- 在 React 中，refs 的作用是什么？

参考回答：
Refs 可以用于获取一个 DOM 节点或者 React 组件的引用。何时使用 refs 的好的示例有管理焦点/文本选择，触发命令动画，或者和第三方 DOM 库集成。你应该避免使用 String 类型的 Refs 和内联的 ref 回调。Refs 回调是 React 所推荐的。

- 何为高阶组件(higher order component)？

参考回答：
高阶组件是一个以组件为参数并返回一个新组件的函数。HOC 运行你重用代码、逻辑和引导抽象。最常见的可能是 Redux 的 connect 函数。除了简单分享工具库和简单的组合，HOC 最好的方式是共享 React 组件之间的行为。如果你发现你在不同的地方写了大量代码来做同一件事时，就应该考虑将代码重构为可重用的 HOC。

- 使用箭头函数(arrow functions)的优点是什么？

参考回答：
1. 作用域安全：在箭头函数之前，每一个新创建的函数都有定义自身的 this 值(在构造函数中是新对象；在严格模式下，函数调用中的 this 是未定义的；如果函数被

称为“对象方法”，则为基础对象等)，但箭头函数不会，它会使用封闭执行上下文的 this 值。

2. 简单：箭头函数易于阅读和书写

3. 清晰：当一切都是一个箭头函数，任何常规函数都可以立即用于定义作用域。开发者总是可以查找 next-higher 函数语句，以查看 this 的值

- **为什么建议传递给 setState 的参数是一个 callback 而不是一个对象？**

参考回答：
因为 this.props 和 this.state 的更新可能是异步的，不能依赖它们的值去计算下一个 state。

- **除了在构造函数中绑定 this，还有其它方式吗？**

参考回答：
可以使用属性初始值设定项(property initializers)来正确绑定回调，create-React-app 也是默认支持的。在回调中你可以使用箭头函数，但问题是每次组件渲染时都会创建一个新的回调。

- **怎么阻止组件的渲染？**

参考回答：
在组件的 render 方法中返回 null 并不会影响触发组件的生命周期方法

- **当渲染一个列表时，何为 key？设置 key 的目的是什么？**

参考回答：
Keys 会有助于 React 识别哪些 items 改变了，被添加了或者被移除了。Keys 应该被赋予数组内的元素以赋予(DOM)元素一个稳定的标识，选择一个 key 的最佳方法是使用一个字符串，该字符串能惟一地标识一个列表项。很多时候你会使用数据中的 IDs 作为 keys，当你没有稳定的 IDs 用于被渲染的 items 时，可以使用项目索引作为渲染项的 key，但这种方式并不推荐，如果 items 可以重新排序，就会导致 re-render 变慢

- (在构造函数中)调用 super(props) 的目的是什么？

参考回答：

在 super() 被调用之前，子类是不能使用 this 的，在 ES2015 中，子类必须在 constructor 中调用 super()。传递 props 给 super() 的原因则是便于(在子类中)能在 constructor 访问 this.props。

- 何为 JSX ？

参考回答：
JSX 是 JavaScript 语法的一种语法扩展，并拥有 JavaScript 的全部功能。JSX 生产 React "元素"，你可以将任何的 JavaScript 表达式封装在花括号里，然后将其嵌入到 JSX 中。在编译完成之后，JSX 表达式就变成了常规的 JavaScript 对象，这意味着你可以在 if 语句和 for 循环内部使用 JSX，将它赋值给变量，接受它作为参数，并从函数中返回它。

3.4 | Angular 框架

- Angular 中组件之间通信的方式

参考回答：
Props down
1.调用子组件,通过自定义属性传值
2.子组件内部通过 Input 来接收属性的值

Events up
1.在父组件中定义一个有参数的方法
2.调用子组件时,绑定自定义事件和上一步方法
3.子组件内部通过 Output 和 EventEmitter 来触发事件并传值.

- Angualr 的八大组成部分并简单描述

参考回答：
model 是 Angular 开发中的基本单位,是一个容器,可以包含组件、指令、管道等
Components 是可被反复使用的带有特定功能的视图
Templates 是经过指令和管道、组件等增强过的 html
Bindings 结合着事件绑定 属性绑定 双向数据绑定等扩展 html 的功能
Directives 分为结构性和属性型指令还有其他模块中比如路由模块中的指令等,
主要是增强 html.
Pipes 可以对数据做一个筛选、过滤、格式化从而得到目的数据
Service 将组件、应用中的可共用的部分,比如数据或者方法等 封装成服务以方便服用
DependencyInjection 依赖注入

- Angular 中常见的生命周期的钩子函数?

参考回答：
ngOnChanges:当 Angular 设置其接收当前和上一个对象值的数据绑定属性时响应。
ngOnInit:在第一个 ngOnChange 触发器之后,初始化组件/指令。这是最常用的方法,用于从后端服务检索模板的数据。
ngDoCheck：检测并在 Angular 上下文发生变化时执行。
每次更改检测运行时,会被调用。
ngOnDestroy:在 Angular 销毁指令/组件之前消除。取消订阅可观察的对象并脱离事件处理程序,以避免内存泄漏。
组件特定的 hooks:
ngAfterContentInit:组件内容已初始化完成
ngAfterContentChecked:在 Angular 检查投影到其视图中的绑定的外部内容之后。
ngAfterViewInit:Angular 创建组件的视图后。
ngAfterViewChecked：在 Angular 检查组件视图的绑定之后

- Angular 中路由的工作原理

参考回答：
Angular 应用程序具有路由器服务的单个实例,并且每当 URL 改变时,相应的路由就与路由配置数组
进行匹配。在成功匹配时,它会应用重定向,此时路由器会构建 ActivatedRoute 对象的树,
同时包含路由器的当前状态。在重定向之前,路由器将通过运行保护(CanActivate)来检查是否允许新的状态。Route Guard 只是路由器运行来检查路由授权的接口方法。
保护运行后,它将解析路由数据并通过将所需的组件实例化到<router-outlet></router-outlet>
来激活路由器状态。

- 解释 rjx 在 Angular 中的使用场景

参考回答：
Rxjs 是在微软所提供的一种的异步处理数据的方式,在 Angular 中处理网络通信时用到了。
创建一个 Observable 并 subsribe
比如:this.http.get('').subscribe((data)=>{ })