【主站前端】2023 届校招题库

（原题库来自原平台前端开发中心）

一、面试构成

校招面试考核重点主要包括两部分【基础知识】+【软性素质】

1、基础知识

旨在通过一系列前端基础知识+计算机基础知识题目识别出基本功过关的同学，同时可通过深挖或引申问题识别出较为优秀的同学；

2、软性素质

旨在通过一系列可以挖掘出个人特质的问题来感知学生软性素质是否匹配我们的需求。

二、如何进行一场面试

1、第一步：确定人才画像

我们鼓励团队构成多元化，理想状态是**既要有人踏实干活，又要有人能钻研和解决比较新比较难的问题，还要有人非常擅长上下游协作和沟通。**因此在开始招聘前，请先明晰自己团队所需要的人才画像，可按照以下思路进行梳理：

团队当前人才构成是什么样？团队成员的长板和短板是什么？还需要补充哪些能力？

例 1：团队成员都偏技术型，不太擅长沟通对接，因此需要沟通方面比较灵活比较积极的同学；

例 2：团队成员平时比较踏实完成业务需求，没有思维比较活泛的人来调动技术气氛，因此需要对新技术比较敏感且喜欢钻研的同学。

2、第二步：根据人才画像制定面试侧重点

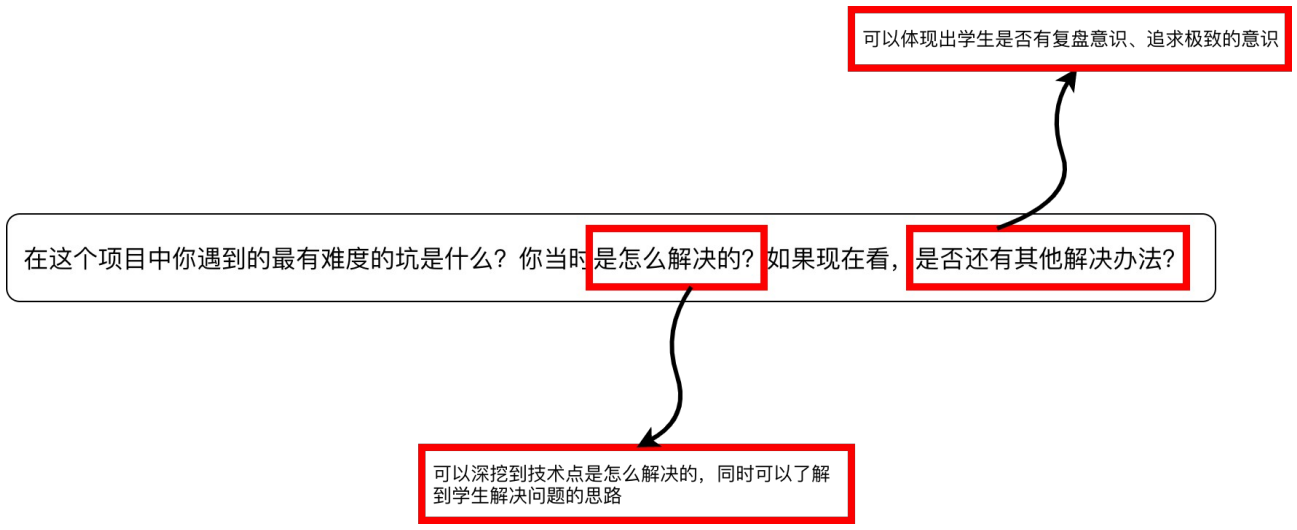
例 1：需要沟通型：除了基础知识外，面试过程中重点考察过往经历中关于【上下游协作】相关的内容；

例 2：需要技术型：将技术问题不断深挖，直到符合本部门的需求为止。注意在对校招生的考核过程中，我们**不追求答案的高准确率**，而是希望能识别出**解题思路较好、培养潜力较好**的同学。

3、第三步：根据面试侧重点设计题目

- 技术考察结构：需要依据该题库的结构进行考核，即前端基础知识+算法/计算机基础知识+编程能力+框架能力；每一项都是必考项目；
- 题目占比：70%的题目需从题库中选择，剩余 30%可根据校招同学的过往经历进行灵活提问；
- 软性素质考察：软性素质的问题可以在 30%的问题穿插进行，用软性素质问题起个头，然后在回答的问题中深挖技术点，既能深挖技术点也能考察软素质。

  例：在结合校招生简历中的某个项目来提问的时候，可以如图所示提问：



三、面试标准

考核标准	定义	B	A	S
专业技能	专业成绩	在校期间成绩优良，有奖学金或专业论文发表者尤佳	✓	✓	✓
	技术基础	考察其所具备的专业技能水平是否适用于现有需求	✓	✓	✓
	项目经验	有参与项目的经验，通过项目参与对技术在实际场景中的运用有一定了解	✓	✓	✓
	项目主导	从项目的决策开始到结束的全过程计划、组织、指挥、协调、督促和评价，以确保项目在特定的时间、预算、资源限定内，依据规范完成		✓	✓
	技术匹配	掌握的技术与现有团队技术需求吻合，能很好地独立完成现有工作任务		✓	✓
	技术互补	目前掌握的技术与现有团队的需求有效互补，有建设性意见及经验能够帮助团队获得更好的提升或飞跃			✓
	领域研究	对某一领域有强烈的好奇心，愿意为此投入更多额外精力，在自己对该领域深入了解的基础上，提出很多建设性建议和看法			✓
考核标准		**定义**	**B**	**A**	**S**
个人特质	沟通协作	能够清晰表达自己的想法，逻辑清晰。能通过正确的方式，将自己观点有效传递，并能获得理解及认同 愿意了解他人，并能准确掌握他人特点，正确理解他人想法	✓	✓	✓
	学习能力	考察其在工作过程中是否能够积极获取与工作有关的信息和知识，并对获取的信息进行加工和理解，从而不断地更新自己的知识结构、提高自己的工作技能	✓	✓	✓
	积极主动	指一种积极、乐观、提前行动的思维方式，碰到困难时不抱怨，而是积极的寻找更为有效的行动来规避或解决。有较好的进步意识和技术追求，会主动寻求成长	✓	✓	✓
	抗压能力	面对压力、委屈、和失败，能调整心态积极去应对，直面问题不逃避。乐于接受批评，会自我审视并调整；不过分骄傲也不长时间沮丧，有正向思考的能力，向善。	✓	✓	✓
	思维能力	指个人对于问题的分析、归纳、推理和判断等一系列认知，它主要包括分析推理和概念思维		✓	✓
	复盘意识	不论结果好与坏，都能够在事后进行复盘和反思，总结经验与问题，寻求更好的解决方案，追求极致		✓	✓
	好奇心	不仅局限于现有技术，会随时了解技术新动态，热衷于研究新技术并运用在工作当中			✓
	举一反三	能尝试用多种方法解决问题，有灵活性，对一件事情有不同甚至相反香法的理解与欣赏，使自己的方法能适应环境的变化			✓

==========题库分割线==========

● 技术题目

一、前端基础知识

【浏览器基础原理】

1、DOM tree 的形成过程

B-能简单说出大致过程

A-能延展出一些概念，重排重绘，时机过程

S-能说出 tokenizer，AST，简单的编译原理，paint layer，图层，合成层等有所了解

2、浏览器 render 进程是什么，大概有哪些线程？

B-说出 JS 主线程，事件队列线程，定时器线程

A-说出 JS 主线程，事件队列线程，定时器线程，GUI 线程，异步请求现成，并能提及到 worker

S-能详尽的说出 render 进程、 browser 进程 和 GPU 进程的详细交互

3、什么情况会触发重排和重绘？

B-能说出常见出发的 api

A-能说出大量重排重绘带来性能问题的解决方法

S-能从合成层和位图来解释重排重绘的深层次原理

4、请描述以下代码运行的结果和原因

宏任务/微任务执行顺序题：

```
setTimeout(() => {

console.log(1);

}, 0);

console.log(2);

(new Promise((resolve) => {

console.log(3);

})).then(() => {

console.log(4);

});

console.log(5);
```

B-回答出正确的结果 2 3 5 1

A-从 JavaScript 异步任务的角度给出原因，以及结果中为什么没有 4

S-可以完整阐述 JavaScript 事件循环机制

5、请描述 cookie、sessionStorage 和 localStorage 的区别。

	cookie	localStorage	sessionStorage
由谁初始化	客户端或服务器，服务器可以使用 Set-Cookie 请求头。	客户端	客户端
过期时间	手动设置	永不过期	当前页面关闭时
在当前浏览器会话（browser sessions）中是否保持不变	取决于是否设置了过期时间	是	否
是否随着每个 HTTP 请求发送给服务器	是，Cookies 会通过 Cookie 请求头，自动发送给服务器	否	否
容量（每个域名）	4KB	5MB	5MB
访问权限	任意窗口	任意窗口	当前页面窗口

参考：https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

B-回答出基本区别
A-细节回答准确
S-了解不同的使用场景

6、请描述<script>、<script async> 和 <script defer> 的区别。

- <script> - HTML 解析中断，脚本被提取并立即执行。执行结束后，HTML 解析继续。
- <script async> - 脚本的提取、执行的过程与 HTML 解析过程并行，脚本执行完毕可能在 HTML 解析完毕之前。当脚本与页面上其他脚本独立时，可以使用 async，比如用作页面统计分析。
- <script defer> - 脚本仅提取过程与 HTML 解析过程并行，脚本的执行将在 HTML 解析完毕后进行。如果有多个含 defer 的脚本，脚本的执行顺序将按照在 document 中出现的位置，从上到下顺序执行。
- 注意：没有 src 属性的脚本，async 和 defer 属性会被忽略。

参考：https://www.growingwiththeweb.com/2014/02/async-vs-defer-attributes.html

B-听说过相关概念

A-回答问题比较准确

S-了解实际的使用场景

7、有哪些用于提高浏览器渲染性能的方法@崔怀祥

1. 减少渲染中的重排重绘。浏览器渲染一般要经过 Layout、Paint、Composite 三个阶段，在这个三个阶段中，Layout 和 Paint 比较耗时，而 Compoistion 则需要的时间较少
   a. 对 DOM 进行批量写入和读取（通过虚拟 DOM 或者 DocumentFragment 实现）。
   b. 避免对样式频繁操作，了解常用样式属性触发 Layout / Paint / Composite 的机制，合理使用样式。
   c. 合理利用特殊样式属性（如 transform: translateZ(0) 或者 will-change），将渲染层提升为合成层，开启 GPU 加速，提高页面性能。
   d. 使用变量对布局信息（如 clientTop）进行缓存，避免因频繁读取布局信息而触发重排和重绘。

2. 优化影响渲染的资源。常见的优化方法有
    a. 关键 CSS 资源放在头部加载。
    b. JS 通常放在页面底部。
    c. 为 JS 添加 async 和 defer 属性。
    d. body 中尽量不要出现 CSS 和 JS。
    e. 避免使用 table, iframe 等慢元素。原因是 table 会等到它的 dom 树全部生成后再一次性插入页面中；iframe 内资源的下载过程会阻塞父页面静态资源的下载及 css, dom 树的解析。
    f. 优化 CSS、JS、Image 等的大小，避免加载不需要的资源

B- 能说出 3 到 5 个优化方法

A- 针对两个优化方向，可以各自讲出 3 个以上优化方法

S- 熟悉浏览器渲染原理，能够全面、体系化的说出性能优化方法

8、请描述 Hash 路由和 History 路由的区别 @崔怀祥

前端路由根据实现方式不同，可以分为 Hash 路由和 History 路由，这两种方式各有优势和局限性

Hash 路由

1. 优点
    a. 兼容性最佳。
    b. 无需服务端配置。
2. 缺点
    a. 服务端无法获取 hash 部分内容。
    b. 可能和锚点功能冲突。
    c. SEO 不友好。

History 路由

1. 优点
    a. 服务端可获取完整的链接和参数。
    b. 前端监控友好。
    c. SEO 相对 Hash 路由友好。
2. 缺点
    a. 兼容性稍弱。
    b. 需要服务端额外配置（各 path 均指向同一个 HTML）

B- 能够讲清楚其中一种路由的优势和局限

A- 两种路由的优势和局限都可以讲清楚

S- 了解这两种前端路由的实现原理

9、如何定位内存泄露 @崔怀祥

内存泄漏是指不再使用的内存，没有被垃圾回收机制回收。当内存泄漏很大或足够频繁时，用户会有所感知：轻则影响应用性能，表现为迟缓卡顿；重则导致应用崩溃，表现为无法正常使用。

常见问题：

1. 是否滥用全局变量，没有手动回收。
2. 是否没有正确销毁定时器、闭包。
3. 是否没有正确监听事件和销毁事件。

B- 能够讲清楚什么是内存泄露，知道可以使用 DevTools 定位内存泄露

A- 能够讲出常见的内存泄露场景

S- 除了以上这些，还熟悉垃圾回收机制

10、什么时候会命中强缓存和协商缓存，两者之间的区别是什么？@张恒敏

B- 能够讲清楚什么是强缓存，什么是协商缓存

- 强缓存判断是否缓存的依据来自于是否超出某个时间或者某个时间段，而不关心服务器端文件是否已经更新
- 协商缓存就是强制缓存失效后，浏览器携带缓存标识向服务器发起请求，由服务器根据缓存标识决定是否使用缓存的过程

A- 了解 cache-control 的常见指令、cache-control 和 expires 的区别，了解强缓存和协商缓存的一些 HTTP Header

- 常见指令有 max-age、no-store、no-cache 这些
- expires 兼容性好，cache-conctrol 优先级高，支持的指令多
- 强缓存 —— expires、cache-control、协商缓存 —— Last-Modified、Etag 两对

S- 了解 ETag 和 Last-Modified 的区别。知道 Etag 和 Last-modified 同时存在时，都符合才返回 304

- Etag 精度高，modified 时间单位是秒
- 性能上，Etag 稍逊于 modified

11、移动端调试工具有哪些？@张恒敏

B- 能说出 chrome://inspect、 safari、vconsole、wenire、chii 等调试工具，如何使用

- 了解 chrome://tracing 加分

A- 能说出调试工具的差异，各自适用的使用场景

- inspect 和 safari 超强，但是建连麻烦点，尤其是 safari
- vconsole 通用，看 console、network、storage 这些面板好使方便
- wenire、chii 可以帮助排查 iOS 兼容性 bug

S- 能说出调试工具的运行原理

- vconsole 全局方法劫持
- inspect 和 safari 调试协议通信

12、你知道哪些在 Chrome 中调试代码的方法@姚泽源

B: 使用 alert/console.log/反复刷新页面 进行调试

A: 能够答出以下常用调试方法

- F12 模拟移动端页面
- console.log 找到代码位置, 添加断点进行调试
- 进入断点后可以进行单步调试
- 可以跳出当前函数(步出)/进入下一个函数内(步入)/停用断点/在遇到异常时暂停

S: 了解更细致的调试方法

- F12 模拟移动端页面时, 通过配置 ua 模拟微信等浏览器环境
- 在源代码面板调试时, 能够添加条件断点, 能够有意识的查看调用堆栈, 切换堆栈以补充断点

- 在控制台面板, 通过配置启用高级日志功能. 如保留日志/显示时间戳
- 通过 Network 面板配置节流模式, 模拟弱网/断网环境
- 通过性能面板配置节流模式, 模拟弱 cpu 环境
- 通过性能面板录制页面绘制过程, 查看执行火焰图
- 了解 Chrome DevTools Protocol 协议

参考:

- https://zh.javascript.info/debugging-chrome
- https://developer.chrome.com/docs/devtools/

【HTML/DOM 相关】

1、HTML 标签上的 data-* 特性有什么作用？如何访问/修改特性值？ @陈晓龙

```
<div data-userid="1" data-Name="John"></div>
```

B - 清楚 data-* 特性的作用
A - 了解获取 data-* 特性的方式（element.dataset），知道在不支持 dataset 的浏览器里使用 getAttribute/setAttribute 获取/修改特性值
S - 能列举出 1~2 个常用场景，比如通过 data-* 设置 CSS 样式

2、回答问题，说明属性（DOM Property）和特性（HTML Attribute）的区别与联系 @陈晓龙

```
<input id="input" type="sometype" value="1">
```

针对如上的 HTML，分别说出 type 属性/特性 和 value 属性/特性的值，即
问题一：value 的属性和特性分别是什么？如果此时用户清空输入框并输入 2，value 的属性/特性分别是什么？
问题二：type 的属性和特性分别是什么？

```
// 问题一，参考答案

input.getAttribute('value') // 1
input.value // 1

// 若用户清空输入框并输入 2
input.getAttribute('value') // 1
input.value // 1
```

```
// 问题二，参考答案
input.type // text
input.getAttribute('type') // sometype
```

B - 能回答对问题一，并能说明 value 的属性与特性的映射关系
A - 能回答对问题二，并能说明常见 type 的特性值
S - 说出属性和特性是如何映射的，并能说出一两个非一一映射的例子，比如 class 特性 vs className 属性，for 特性 vs htmlFor 属性

参考文档：

- What is the difference between properties and attributes in HTML?
- HTML attribute 和 DOM property

【CSS 相关】

1、隐藏页面元素的方式有哪些

B-display none visibility hidden opacity 0

A-了解三者在交互和文档流的区别

S-从浏览器渲染原理进行阐述

2、垂直居中一个元素的方式有哪些

B-至少说出两种

A-能说出绝大部分实现以及优缺点

3、CSS、JS 放置位置和原因

B-加快页面加载速度

A-加速原理，能说出为什么 CSS 要放头部，JS 放到 body 末尾

S-能从 dom tree、CSSOM、 render tree 形成的角度解释

4、介绍一下 CSS 的盒子模型有哪些种类？他们的区别是什么？

5、基本布局属性，float，flex，grid 等实现方法；

B-能说出不同布局之间的区别

A-能说出不同布局的工作原理

6、CSS 选择器的优先级是怎样的？

B-能够说出 id、class 等基本选择器以及内联样式的优先级

A-选择器类型比较全面，比如属性选择器、伪选择器等

S-了解 MDN 对于优先级的计算方式：https://developer.mozilla.org/zh-CN/docs/Web/CSS/Specificity

7、CSS 中的 BFC 是什么

B - 解释什么是 BFC

A - 创建 BFC

8、CSS 如何实现固定宽高比

B - 知道一种实现方式

A - 知道多种实现方式

9、介绍一些 CSS 中的合成层（Compositing Layers）？

B - 哪些属性会导致单独的合成层

A - 什么是合成层

S - 提升为合成层的原因

10、CSS 中的继承 @徐再贤

B - 了解 Inherited 属性

A - 了解未继承时会取属性默认值 https://developer.mozilla.org/zh-CN/docs/Web/CSS/initial_value

S - 了解通常情况下样式属性会继承，布局属性不继承

11、用 html + css 画一个漂亮的彩虹 @徐再贤

B - 实现一个基本多色彩虹

A - 增加颜色渐变（可以查 MDN）

12、只使用 CSS 实现一个 Dropdown Menu @徐再贤

A - 实现即可

13、用 css 实现一个红绿灯 @张恒敏

A - animation keyframes 实现即可

14、简述伪类和伪元素及区别 @周世超

B- 能够正确简述及区别 伪类 伪元素

A-能够说出 CSS3 新增的伪类及用法

15、提供 1px 边框解决方案 @杜俏

B- 能提供一到两种解决方案（伪类+transform、linear-gradient 渐变）

A- 了解 DPR、物理像素、CSS 像素概念

【网络相关】

1、前端怎么使用缓存

B-了解 cookie，能够使用

A-了解 HTTP 缓存机制

S-了解 Service Worker 的特点和优劣

2、常见 http 状态码

B-能回答 20x、30x、40x、50x 等基本的编码以及含义

3、DNS 是什么以及其原理，浏览器从 URL 到展示经历了哪些

B-能够回答 DNS 概念

A-能够说明从 URL 到渲染全过程，1.根据域名，进行 DNS 域名解析；2.拿到解析的 IP 地址，建立 TCP 连接；3.向 IP 地址，发送 HTTP 请求；4.服务器处理请求；5.返回响应结果；6.关闭 TCP 连接；7.浏览器解析 HTML；8.浏览器布局渲染；

S-能够说明 DNS 工作原理

4、网络协议的分层和基本原理

B-tcp 基本原理，与 udp 的不同，dns 常见使用协议和原因 http 常见状态码和意义

A-关注 http2/3，quic 等优化协议 能说出一些常见优化手段

S-极其扎实的计算机网络基础，除了传输层对于网络层和基本算法也能说出一二

5、常见的网络攻击有哪些

B: 说出一种网络攻击的原理

A: 知道多种网络攻击的原理

S: 知道多种网络攻击原理，并且知晓如何防范

6、什么是存储型、反射性攻击

B：能大概说出概念的含义

A: 清楚说出概念的含义，并且知道常见攻击的相应归类

7、什么是 xss

B：大概知道原理

A: 清楚知道原理，能说出具体 case，知道如何防范

核心原因:

- 浏览器把用户的输入当成了脚本进行了执行

解决思路:

- 避免注入(转义字符)
- 避免执行(声明为普通的 text 文本, 避免浏览器误执行)
- 用好框架(React/Vue/owasp 提供的专业 encode 工具), 自行编写总会存在未考虑周全的地方

参考资料:

- 前端安全系列（一）：如何防止 XSS 攻击？ - 美团技术团队
- OWASP: 2021 十大 web 安全风险 / OWASP:2017 十大 web 安全风险:介绍与常见防御策略-中文版

8、一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？@周纤纤

B: 大致能了解

A：理解主要过程

S：非常清楚整个过程

9、说一下 http 和 https 的区别及 https 协议的工作原理 @周纤纤

B：区别可以回答 3-4 点以上，原理基本了解；

A：区别可以回答 5 点以上；原理能了解 6 大主要过程

S：区别回答完整；原理清楚阐述；

- https 在 http 基础上, 添加 TLS1.3, 通过 TLS 建立信道, 交换会话密钥, 后续使用会话秘钥进行加密
  - 建立信道使用非对称加密(RSA)验证身份, 速度慢, 安全性高
  - 通过安全信道交换会话秘钥, 后续使用该会话秘钥进行对称加密(AES), 速度快(为什么不能一开始就用对称加密=> 无法安全传递会话秘钥)

参考资料:

彻底搞懂 HTTP 和 HTTPS 协议-探究 http 和 https 的发展史

TLS1.2-rfc5246-中英对照翻译(目前 TLS1.2 已废弃, 18 年推出 TLS1.3 规范-rfc8446, 大致思路一样)

10、说说三次握手和四次挥手过程

B：了解基本过程

A：掌握原理，区别握手和挥手

S: 能够进行扩展

11、什么 CSRF

B：了解原理

A：能阐述执行攻击、防护方法

S：能横向延伸其他攻击原理、防护方法

12、 页面劫持有哪些

B：了解劫持含义

A：掌握原理，能阐述防护方法

S： 能延伸 https、ssl 的方面知识

13、 什么是 **sql** 注入

B：了解原理

A：掌握原理，能阐述防护方法

14、 HTTP1.1 和 **HTTP2.0** 的区别 **@**杜俏

B：能说出 HTTP1.1 和 HTTP2.0 的主要区别（多路复用，头部压缩，服务器推送）

A：能够说出 HTTP 的发展历程以及 HTTP1.0 和 HTTP1.1 的区别

参考：https://juejin.cn/post/6844903489596833800

15、 HTTP 的几种请求方法与用途 **@**杜俏

B：能说出 GET、POST 的用途与区别

A：能清晰的说出请求的响应步骤

S：能说出 PUT、HEAD、DELETE、OPTIONS、TRACE、CONNECT 等其余请求方法与用途

参考：https://segmentfault.com/a/1190000015853611

【JS 基础-常规】

1、如何判断一个变量是 Array 类型

B-isArray

A-Object.prototype.toString.call()

2、常用的数组 api，字符串 api

B-能说出大部分数组的 api

A-针对数据的操作能说出哪些是修改原数组的，哪些是返回新数组

S-能说出一些关键 api 的实现，比如 indexof

3、前端可操作的本地存储有哪些，有什么区别

B-能说出 cookie，localstorage，sessionStorage 的区别

A-能说出 indexDB 的一些常见区别

S-能延展出一些变量本地存储，堆栈等

4、this 指向问题：

```
var name = '123';

var obj = {
  name: '456',
  getName: function () {
     function printName () {
        console.log(this.name);
     }
```

```
      printName();
  }
}
obj.getName();
```

B-知晓问题答案

A-进一步趋稳如何修改能回答，知道 call、apply、bind

S-知道如何实现 bind

5、原型、构造函数、实例 之间的关系？

B-能够说出 constructor、__proto__、prototype 等

A-继续追问 instanceof 实现比较明晰

S-继续追问原型链顶端相关问题明晰，原型链查找机制明晰

6、Map、WeakMap、Set、WeakSet 的区别

B-能说出四者的主要区别

A-能说出应用场景

S-能说出 iterable、垃圾回收机制、弱引用含义和作用

7、简述 JavaScript 原型，原型链原理，如何基于原型链实现继承？

B-能实现组合继承

A-能实现寄生组合式继承，并能说出为什么要寄生，以及寄生的优点

8、DOMContentLoaded，load，beforeunload，unload

B-能说出大致顺序

A-能精准说出所有的时机区别和先后顺序

9、介绍下 new 操作符作用 @黄义珊

B-能说出大概作用

A-能比较准备描述 new 操作符整个工作流程

S-能自己实现一个 new

10、Javascript 数字精度丢失 @黄义珊

B- 能够说出 0.1 + 0.2 正确结果（不等于 3 就行，不需要精确到小数点）且了解大概原因

A- 1. 能够说出 Javascript 底层计算逻辑

或者 2. 能够给出精度丢失时的解决方案

S - A 里 2 条都能答出

11、函数参数值传递问题 @黄义珊

题 1： 代码执行完后 a、b 的值是什么 ？

```
let a = [1, 2, 3]
let b = 1
```

```
const foo = (obj, num) => {
    obj = [5, 4, 3]
    obj[1] = 999
          obj = [5, 4, 3]
          num = 2
}

foo(a, b)
console.log(a, b);
```

题 2：给出 a(1)返回值

```
function a(x, f=()=>x) {
    var x;
    y = x;
    x = 2;
    return [x,y,f()];
}
a(1)
```

B - 能够正确给出题目 1 的输出

A - 解释 JS 函数传参的逻辑, 以及正确阐述代码过程中各变量的值

S - 正确给出题目 2 的返回 并阐述流程

12.跨域相关@冯盼

考察对跨域问题的原因，解决方案，优缺点的了解

B-说出至少 2 中解决方案（转发，jsonp，cors）

A-能够知道原因，实现细节

S-能够说出优缺点，使用中的注意事项，比如 cookie，前后端的配合，安全

同源策略:

(网络协议, 域名, 端口号) 构成三元组, 三元组相同才是同源, 否则则为跨域

可以通过 document.domain = "company.com"将子域 a.company.com 向上提升到父域 company.com, 若页面内 document.domain 相同, 则不视为同域, 通过 document.domain 处理时, 会抹除端口号信息

13.performance 有了解吗@冯盼

考察对首屏性能相关的关注

B-说出和性能有关，可以量化性能数据

A-了解常见的量化性能的工具，说出如何使用，用到了哪些属性，

S-引导下能够说出常用的性能优化手段

14.防抖@冯盼

考察对用户体验的关注度，闭包的应用

B-能够讲清楚原理，知道和闭包相关

A-能够实现代码

S-引导下说出其他的一些提升用户体验的手段

15.节流

考察对用户体验的关注度，闭包的应用

B-能够讲清楚原理，知道和闭包相关

A-能够实现代

S-引导下说出其他的一些提升用户体验的手段

16.请说一下对 JS 原型链的理解 @刘卓

B-能讲出原型链的概念，通过相关代码示例进行解释 讲清楚原型 构造函数 实例 原型链的关系；讲出原型链和原型对象之间到底起到的作用；

A-引导提问 JS 创建对象的几种方法，追问 instanceof 原型链判断标准，能准确回答；进而追问如何手动创建一个原型链，答出 Object.create 或者 obj.__proto__ = constructor.prototype 并说明

S-追问 new 操作符内部做了什么? 能答出新生成一个对象，链接到原型、绑定 this、返回对象；自己用 JS 实现 new 操作符？ 能实现代码

17.object.assign 和扩展运算法@高瑞

B-object.assign 和扩展运算法是深拷贝还是浅拷贝，两者区别

A-适用场景

S-手写实现深拷贝

18.请说出对 JS 事件机制的理解　@刘卓

B-对事件流解释，说出事件捕获阶段，处于目标阶段，事件冒泡阶段过程

A-事件绑定的几种方式，以及对应的事件触发阶段

S-追问是否了解事件委托，实现一个 todo list 添加和删除的功能，能通过事件委托实现

19.异步编程　@刘卓

JS 中异步编程的几种常用方式

B-答出回调、promise、generator、await 和 async 几种方式

A-能说出 几种异步方式写法优缺点发展过程

S-手动实现一个 Promise 和 generator 简单实现; 以下 async await 代码编译成 es5 后是什么样子的，大体说出 async 函数是 generator 的语法糖，其编译后也是对 generator 的的自动调用

```
async function get() {
   await fetch('/');
   await fetch('/a');
   return 1;
}
```

20.setTimeout 与 promise@高瑞

B-能够讲出 setTimeout 与 Promise 的使用场景

A-能够讲清楚微任务、宏任务、js 为什么是单线程的

S-手写实现 promise

21.slice splice split 的区别 @高瑞

B-能够讲清楚 **slice splice split 的基础使用方法、区别**

A-能够讲清楚 **slice splice split 的使用场景**

22.图片的懒加载和预加载@杜元丰

B - 能够讲出懒加载和预加载的能力和使用场景

A - 能够讲出懒加载和预加载的实现原理

S - 可以使用原生 JS 实现一个懒加载方法

23.数组的遍历 @杜元丰

B - 能够讲出 3 种及以上常用的遍历方法（for of 、 forEach 、 map ...）

A - 能够讲出 some 、 every 、 filter 、 reduce 的使用方法和适用场景

24.原始数据类型和引用数据类型 @杜元丰

B - 能够说出原始数据类型的种类和引用数据类型的种类有哪些， 并能描述清楚开发中可能因引用数据类型导致的问题

A - 能够说出两种类型的核心区别在于存储位置的不同， 能简述栈（stack）和堆（heap）的概念

S - 能够说出区分存储位置的原因， 能引申 栈 和 堆 的内存释放和垃圾回收机制

【JS 基础-变量作用域】

1、let var const 区别

B-能够说出三者概念和使用上的区别；

A-明晰暂时性死区；

S-能说出 JS 代码的执行原理，从右往左，ast，内存分配等，常规类型和引用类型的存储区别等/继续追问 ES6 中的 let、const 全局对象挂在哪里？变量存储机制。

2、JS 中有几种类型的作用域？如何形成？

B-块级、函数、全局

A-如何形成不同的作用域

S-提及作用域原理、作用域链、执行环境、执行上下文

3、闭包概念以及的原理

B-能够回答概念问题，是私有变量，可能会有内存泄露

A-能够用闭包解决一些问题，比如

```
fvar data = [];

for (var i = 0; i < 3; i++) {
  data[i] = function () {
    console.log(i);
  };
}

data[0]();
data[1]();
data[2]();
```

S-执行环境栈 EC stack，作用域，作用域链，GC，V8 GC

【JS 基础-内存泄露】

1、什么是内存泄露，什么情况下会发生内存泄露，以及如何解决，可以用什么工具

B-能够回答内存泄露的原理，能够举例说明内存泄露场景

A-能够完整说出场景（全局变量、计时器、闭包、引用、回调等）的内存泄露场景，以及如何处理内存泄露

S-能够详细说明 JavaScript 内存回收机制以及如何通过工具检查内存泄露并通过代码控制内存泄露

2、垃圾回收算法有哪些，浏览器端和 node 端区别是什么？

B-能够说出常见垃圾回收算法及其原理

A-说出 node 端垃圾回收算法机制

S-追问一些不会被垃圾回收的场景，能够回答

【Web 安全基础】

1、前端会遇上哪些安全问题

B-能回答跨站攻击、xss 的概念

A-能够结合场景描述攻击及处理问题

二、算法/计算机基础

【排序】

1、基本排序考一道，如冒泡、快排

B-写出正确可运行代码

A-实际场景的应用，如 settimeout 使用什么排序实现，即考察对 js 异步的理解又考差算法能力

S-实现场景题的算法可以代码编写

【递归】

1、一个正整数，拆成各种情况，输出二位数组

比如： 1 -> [[1]] 2 -> [[2], [1, 1]]

B-能够写出递归

A-能够完整实现

S-能够处理边界情况

【数组方法】

1、多维数组如何知晓它的维度，如何实现在数组原型上？Array.prototype.getLevel

B-能够明确 this、递归

A-能够完整实现

S-能够处理边界情况

【汉明距离】

1、两个相等长度的字符串之间的汉明距离是对应符号不同的位置数。

"karolin" 和 "kathrin" = 3

1011101 和 1001001 = 2

B-能够给出基本答案

A-能够完整实现

S-能够处理边界情况

【最长公共子序列】

1、LCS：

ABCDGH 和 AEDFHR = 3

AGGTAB 和 TXTXAYB = 4

B-能够给出基本答案

A-能够完整实现

S-能够处理边界情况

【和最大的子数组】

1、在一维数组中找到连续的相加和最大的数组

B-能够给出基本答案

A-能够完整实现

S-能够处理边界情况

【爬楼梯问题】

1、需要 n 阶你才能到达楼顶。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

B-暴力递归解法

A-动态规划版本

S-能延展一些动态规划常见的使用场景

【二分查找】

1、实现一个二分查找

B-递归版本

A-while 版本并能进行位运算操作

S-能延展一些常见的使用场景

【分组】

1、一个团队外出团建，要进行一些分组活动的游戏。请设计一个算法，根据游戏分组数量的要求和团队男女生人数，对团队成员进行分组。要求每组的性别比例尽量接近，男女生平均分配。

B-不考虑性能和效率，能给出符合条件的答案

A-可以用取模等方式简化计算步骤，优化算法

【去重】

1、去掉一组整型数组中重复的值

B-能够通过伪代码实现

A-能够准确使用 js 实现

【洗牌】

1、手写洗牌算法

```
Array.prototype.shuffle = function () {
    let arr = this,
        m = arr.length,
        i, tmp

    while(m) {
        i = Math.floor(Math.random() * m--)
        tmp = arr[i],
        arr[i] = arr[m]
        arr[m] = tmp
    }
    return arr
}
```

【大数相加】

```
function sum(a, b) {
    var a = a.split(''),
        b = b.split(''),
        res = '',
        c = 0;

    while(a.length || b.length || c) {
        c += ~~a.pop() + ~~b.pop();
        res += c % 10;
        c = c > 9;
    }
    return res.replace(/^0+/,'');
}
```

【数据结构基础】

1、给定两个升序链表，设计一个方法将其合并并且保持升序，保留重复值

B-写出正确可运行代码

A-扩展问题 k 个有序链表的合并，能有非递归实现思路并且说对复杂度

S-代码正确实现 k 个链表

2、双链表的 JS 实现

B-能实现出关键 api

A-能实现出大部分 api，并能运行

S-边界场景处理非常完美

【算法】@颜适

划分数组

描述

给定一个长度为 n (n <= 10^6) 的数组，包含 n 个整数 a[i] (-10^9 <= a[i] <= 10^9)。将数组的所有元素分成三个连续的非空子数组，使得每个部分的元素之和相等，求划分的方案数。

输入

[1, 2, 3, 0, 3]

[0, 1, -1, 0]

[1, 1]

输出

2

1

0

样例解释

样例 1 可以划分为 [1, 2] [3] [0, 3] 或 [1, 2] [3, 0] [3]；

样例 2 只可能划分为 [0] [1, -1] [0]；

样例 3 无法进行合法的划分。

B - O(n^2)，暴力枚举划分点 O(n^2)，利用前缀和求区间和 O(1)，要求边界条件实现完美

A - O(nlogn)，暴力枚举一个划分点 O(n)，在前缀和序列中二分查找另一个划分点的范围

S - O(n)，线性扫描，利用前缀和快速统计，思路类似下面的代码

```
let sums = [0];
for (let i = 1; i < n; ++i) {
   sums[i] = sums[i - 1] + a[i - 1];
}
let count = 0;
let result = 0;
for (let i = 1; i < n; ++i) {
   if (sums[i] === twoThirds) {
      result += count;
   }
   if (sums[i] === oneThird) {
      ++count;
   }
}
return result;
```

前缀后缀回文串

描述

给一个字符串 s（长度 <= 10^6），仅包含小写英文字母。求一个最长的回文字符串 t，要求 t 的长度不大于 s 的长度，且满足 t = a + b，其中 a 为 s 的前缀，b 为 s 的后缀（此处 a、b 均可能为空字符串）。如果存在多种答案，返回任意一种即可。

输入

abcdfecba

acbdba

kuaishou

输出

abcdcba

abdba

k

样例解释

样例 1 由前缀 abcd 和后缀 cba 组成

样例 2 由前缀 a 和后缀 bdba 组成

样例 3 由前缀 k 和空后缀组成

B - O(n^3)，暴力枚举前缀和后缀的长度 O(n^2)，验证是否是回文字符串 O(n)，要求代码能实际运行

A - O(n^2)，移除前后本来就相等的字符后，问题转化为从剩余字符串 r 的前缀或后缀中，找到最长的回文字符串，枚举前缀或后缀的长度 O(n)，验证是否是回文字符串 O(n)

S - O(n)，问题转化同 A 档，随后构造 r + '#' + reverse(r) 与 reverse(r) + '#' + r，kmp 算法中前缀函数最后一项的值，即为 r 的前缀或后缀最长回文字符串的长度

```
const kmp = s => {
    let n = s.length;
    let p = [0];
    for (let i = 1, j = 0; i < n; ++i) {
        while (j && s[i] !== s[j]) {
            j = p[j - 1];
        }
        if (s[i] === s[j]) {
            ++j;
        }
        p[i] = j;
    }
    return p[n - 1];
};
```

归并序列

描述

merge 函数常见于归并排序，比如 a = [3, 1] b = [2, 4]，每次合并从 a b 头部选择较小的数字，则 merge(a, b) 为 [2, 3, 1, 4]。给出一个长度为 2n (n <= 2000) 的数组，内容为整数 1 ~ 2n 的一个排列，判断是否能由两个长度为 n 的数组 merge 得到。

输入

[2, 3, 1, 4]

[3, 1, 2, 4]

[3, 2, 6, 1, 5, 7, 8, 4]

[1, 2, 3, 4, 5, 6]

[6, 1, 3, 7, 4, 5, 8, 2]

输出

true

false

true

true

false

样例解释

样例 1 由 merge([3, 1], [2, 4]) 得到

样例 2 不能由 merge 得到

样例 3 由 merge([3, 2, 8, 4], [6, 1, 5, 7]) 得到

样例 4 可以由 merge([1, 3, 5], [2, 4, 6]) 得到，也有其他方案

样例 5 不能由 merge 得到

B - O(n!)，暴力 DFS 求全排列，每个排列都尝试 merge，判断是否相等

A - 能注意到进行区间划分，如 [3, 2, 6, 1, 5, 7, 8, 4] 划分为 [3, 2] [6, 1, 5] [7] [8, 4] 四个区间，第一个元素都是区间最大值，并且每个区间一定都在 merge 的其中一个数组内，问题转化为关于区间长度的 01 背包问题

S - O(n^2)，DP 实现 01 背包，代码能实际运行

```
f[0] = true;
```

```
for (let i = 0; i < m; ++i) {
    for (let j = n; j >= c[i]; --j) {
        f[j] = f[j] || f[j - c[i]];
    }
}
return f[n];
```

吃汉堡

描述

一个汉堡的食谱 recipe 由若干份面包、肉和蔬菜组成。已知冰箱里现存的面包、肉和蔬菜的数量 storage，以及这些食材在市场上的单价 price，还有你拥有的钱 money (1 <= money <= 10^12)，求最多能做出多少个汉堡。其中 recipe、storage 和 price 均为三元组，分别对应三种食材，并满足 1 <= recipe[i], storage[i], price[i] <= 100。假设食材在市场上无限量供应。

输入

// recipe, storage, price, money
[3, 2, 1], [6, 4, 1], [1, 2, 3], 4
[1, 1, 1], [1, 1, 1], [1, 1, 3], 1000

输出

2
201

样例解释

样例 1 的 recipe 为 [3, 2, 1]，storage 为 [6, 4, 1]，price 为 [1, 2, 3]，money 为 4，只要在市场上花 3 块买一份蔬菜后，存货就变为 [6, 4, 2]，可以做出 2 个汉堡；

样例 2 中存货正好能做一个汉堡，然后用 5 块钱可以去市场上买一套食材，1000 块正好可以买 200 套，所以一共可以做出 201 个汉堡。

B - 大模拟，边界条件非常非常多，综合考虑思路和代码实现打分

A - 二分查找，确定上下界后二分答案，验证当前答案是否满足条件即可

S - 二分查找，要求思路清晰，边界条件实现完美，代码能实际运行

```
let l = 0;
let r = money + 101;
while (l < r) {
    const m = l + ((r - l) >> 1);
    const cost = check(m);
    if (cost <= money) {
        l = m + 1;
    } else {
        r = m;
    }
}
return l - 1;
```

喝饮料

描述

有 n (n <= 10^6) 杯饮料排成一行，每杯饮料喝下后会改变体力值 a[i] (-10^9 <= a[i] <= 10^9)。现在开始按顺序喝饮料，每杯饮料可以选择喝或者倒掉。已知初始体力值为 0，且始终不能小于 0，求能喝下的最多饮料数量。

输入

[4, -4, 1, -3, 1, -3]

输出

5

样例解释

喝下第一杯饮料，体力值为 4；倒掉第二杯 -4；剩余 4 杯饮料全部喝下。一共喝下 5 杯饮料，且体力值始终非负。

B - O(2^n)，暴力 DFS，枚举所有可能性，要求代码能实际运行

A - O(n^2)

DP，定义 f[i][j] 为 前 i 杯饮料喝下 j 杯的最大体力，倒掉时 f[i][j] = f[i - 1][j]，喝下时 f[i][j] = max(f[i - 1][j - 1] + a[i], f[i - 1][j])；

贪心，从最大的饮料开始喝，每次喝下 a[i] 后更新 [i, n) 区间的体力值，确保区间的体力值非负，直到不能喝为止。

S - O(nlogn)

贪心，用优先队列维护过去喝过的饮料，不能喝的时候，就把过去喝过的最小的饮料倒掉，具体类似下面的代码；

DP，同样可以用优先队列优化；

线段树，A 档位贪心算法的区间更新可以用线段树优化。

```
let sum = 0;
let result = 0;
for (let i = 0; i < n; ++i) {
    sum += a[i];
    queue.push(a[i]);
    while (sum < 0) {
        sum -= queue.top();
        queue.pop();
    }
    result = Math.max(result, queue.size());
}
```

树上距离

描述

给定 n (n <= 50000) 个编号为 1 ~ n 的节点和 n - 1 条边，这些点和边会组成一棵树。求树上最短距离恰好为 k (1 <= k <= 500) 的点对的数量。点对 (u, v) 和 (v, u) 视为同一个。

输入

// edges, k

[[1, 2], [2, 3], [3, 4], [2, 5]], 2

[[1, 2], [2, 3], [3, 4], [4, 5]], 3

输出

4

2

样例解释

样例 1 中距离为 2 的点对包括 (1, 3) (1, 5) (3, 5) (2, 4)

样例 2 中距离为 3 的点对包括 (1, 4) (2, 5)

B - O(n^3)，暴力枚举点对 O(n^2)，遍历计算点对距离 O(n)，判断是否符合条件

A - O(n^2)，枚举一个点 O(n)，遍历求树上单源最短路径 O(n)，统计符合条件的点对数量

S - O(nk)，树上 DP，定义 f[i][j] 为以 i 为根的子树中距离为 j 的节点数量，可以在一次 DFS 中进行统计

```
const dfs = (u, p) => {
    let result = 0;
    f[u][0] = 1;
    for (let v of tree[u]) {
        if (v === p) {
            continue;
        }
        result += dfs(v, u);
        for (let i = 1; i <= k; ++i) {
            f[u][i] += f[v][i - 1];
        }
        for (let i = 1; i < k; ++i) {
            result += (f[u][i] - f[v][i - 1]) * f[v][k - i - 1];
        }
    }
    result += f[u][k];
    return result;
};
```

【计算机基础 - 线程/进程】

1、进程/线程的区别

B-内存共享，沙盒机制等等

A-能说出浏览器大致的进程/线程的进化史

S-能说出浏览器大致的进程/线程的进化史，并能理解为何要这样演进

三、编程能力

【原生 API 实现】

1、简述实现一个开根号算法的思路。

```
var mySqrt = function (x) {
  let lt = 0, rt = x


  while (rt>lt) {
    mid = lt + Math.floor((rt - lt) / 2);
    if (mid * mid <= x && x < (mid + 1) * (mid + 1)) {
      return mid;
    }
    if (mid * mid < x) {
      lt = mid;
    } else {
      rt = mid
    }

  }
};
```

2、forEach 的实现

B-能够实现 forEach 的基本用法

A-能够考虑实现的方法的优劣

S-知晓扩展问题，比如 forEach v.s. for 等

3、parseInt 的实现 @纪鹏

B- 能够实现 parseInt 的基本用法(不考虑 radix)

A- 可以基本实现 radix(2-36)的基本用法

S- 实现的基础上，考虑到大部分边界情况，并通过 test case

4、JSON.stringify 的实现 @金黎明

B- 能够实现基本类型的 stringify 能力

A- 考虑到特殊数据的处理（Date 对象 toJSON、函数等）

S- 对输出结果的格式化（换行、前导空格），处理字符串转义和循环嵌套的情况

5、JSON.parse 的实现 @金黎明

B- 能够将给出的字符串解析为 JS 对象

A- 考虑到最外层不是对象的形式（数组、纯字符串、null 等），考虑到简单的字符串转义（\"），不需要考虑复杂的转义（unicode \u1234 之类的）

S- 处理边界情况，给定的字符串不是有效的 JSON

6.call, apply, bind 的模拟实现 @周纤纤

B-关键点 this 指向改造正确，执行函数；

A-this 指向改造和执行函数实现正确，能考虑到传参的不确定性和 this 参数为 null 的情况；

S-实现过程正确，边界 case 处理考虑周全；

7、实现页面滚动至底部加载更多数据的功能，设计合理的接口请求参数和接口返回数据，可使用 React、Vue、Angular 等框架 @刘洪燕

B：滚动至底部的逻辑基本没问题，接口设计大差不差

A：调用时机正确，调用接口参数和返回数据结构合理

S：调用时机准确，接口设计合理，组件完整度高 / 各类状态考虑周全 / 代码质量高

8、编写一个顶导航的三级菜单组件，鼠标 hover 或者 click 出现子菜单浮层，数据结构自自行定义，可使用 react、vue、angular，不得使用任意 UI 框架组件 @刘洪燕

B：数据结构合理，html 结构问题不大

A：数据结构合理，功能基本实现，代码相对完整

S：A 的基础上，扩展性强，代码质量高

9、请编写一个函数将平级数组 list 转换成树形结构的 tree，parentId 为 null 是根结点 @刘洪燕

```
// 输入
const list = [
   { id: 1, parentId: null, name: 'x1' },
   { id: 3, parentId: 5, name: 'x3' },
   { id: 5, parentId: 2, name: 'x5' },
   { id: 2, parentId: 1, name: 'x2' },
   { id: 4, parentId: 1, name: 'x4' },
   { id: 6, parentId: 2, name: 'x6' },
   { id: 7, parentId: 4, name: 'x7' },
   { id: 8, parentId: 4, name: 'x8' },
   { id: 9, parentId: 8, name: 'x9' },
   { id: 10, parentId: 8, name: 'x10' },
]

// 输出
const tree = {
id: 1,
root: true,
name: 'x1',
children: [
   {
      id: 2,
      parentId: 1,
      name: 'x2',
      children: [
         {
            id: 5,
            parentId: 2,
            name: 'x5',
            children: [
               { id: 3, parentId: 5, name: 'x3' },
            ]
         },
         {
            id: 6,
```

```
          parentId: 2,
          name: 'x6'
        },
      ]
    },
    {
      id: 4,
      parentId: 1,
      name: 'x4',
      children: [
        { id: 7, parentId: 4, name: 'x7' },
        {
          id: 8,
          parentId: 4,
          name: 'x8',
          children: [
            { id: 9, parentId: 8, name: 'x9' },
            { id: 10, parentId: 8, name: 'x10' },
          ]
        },
      ]
    }
```

B：逻辑基本正确，有少量 bug

A：验证无误，复杂度大于 O(n)

S：完美实现，复杂度为 O(n)，代码逻辑清晰

【Request Pool】

1、请求池实现，要求：处理异步，考虑并发；处理回调；处理重试；处理边界

B-1 2 是基本实现

A-3 是扩展

S-4 是完善

【RPC 通讯】

1、实现 worker 和 浏览器线程的协同机制

B-知晓浏览器和 worker 通信机制和限制

A-基本实现协同通信

S-完善的实现

【事件机制】

1、实现一个事件 event，具有发布订阅功能

B-能简单说出原理，写出伪代码

A-能实现关键 api 包括 on off once 等

S-能处理各种关键 case，并能运行，并说出实际的应用场景，和 vuex 的区别

2、on、off、emit、once 等 event emitter 方法

B-能够实现基础方法，对数据结构的规划完整

A-代码优雅

S-延展考察浏览器事件机制异同

【缓存系统】

1、LRU 缓存替换机制的实现（hashMap + 双链）

B-能说出大概原理是 hashMap + 双链

A-能说出设计思想，写出伪代码，get 和 put 都是 O(1)

S-手写并且可运行

2. LFU 缓存替换机制的实现 实现 @纪鹏

B-能说出大概原理是 hashMap + 双链

A-能说出设计思想，写出伪代码，get 和 put 都是 O(1)

S-手写并且可运行

【变更检测】

1、实现一个数据变更检测的方法

B-不限方式，给出至少一个可用的实现方式（脏检测、defineProperty、Proxy 等）

A-给出不止一种实现方式，并能阐述各种方案的优势和劣势

S-了解推拉模式的特点。对当前主流框架的方案进行比较，结合自己的理解给出可能的原因

【DOM API】

1、计算页面中一共出现了多少种 html tag

B-能用 querySelector/getElementsByTagName 等 DOM API 给出结果

A-了解 Set 等 ES2015+ API 并能运用

S-（如果提到正则或其他基于字符串匹配的思路）给出一种 AST 的实现思路

2. 判断一个元素是否在可视化区域中 @纪鹏

B-实现通过文档距定位的距离与可视区域距离对比进行判断(类似矩形相交)

A-了解 getBoundingClientRect or Intersection Observer 的 API，并伪代码实现

S-实现功能并通过测试，并能对其他 Web Observer API 有一定了解例如:MutationObserver、PerformanceObserver 等等

【Promise】

1、实现基础 Promise 方法

B-基本实现

A-标准熟悉并能完整考虑

S-完整实现和边界处理

2、基本实现 Promise.all / Promise.race

B-基本实现

A-标准熟悉并能完整考虑

S-完整实现和边界处理

3、使用 Promise 封装异步图片加载方法 @李阳

参考答案：

```
function loadImageAsync(url) {
    return new Promise(function(resolve,reject) {
        var image = new Image();
        image.onload = function() {
            resolve(image)
        };
        image.onerror = function() {
            reject(new Error('Could not load image at' + url));
        };
        image.src = url;
     });
}
```

评分标准：

B-提醒图片加载的方式后能实现

A-能自主实现

4、使用 Promise 实现图片的条件加载 @李阳

现有 8 个图片资源的 url，已经存储在数组 urls 中，且已有一个加载图片函数，输入一个 url 链接，返回一个 Promise，该 Promise 在图片下载完成的时候 resolve，下载失败则 reject。要求：任何时刻同时下载的链接数量不可以超过 3 个。

请写一段代码实现这个需求，要求尽可能快速地将所有图片下载完成。

```
var urls = ['**1.jpg', '**2', ...];

function loadImg(url) {
    return new Promise((resolve, reject) => {
        const img = new Image()
        img.onload = () => {
            console.log('一张图片加载完成');
            resolve(url);
        }
        img.onerror = reject;
        img.src = url;
    })
};
```

参考答案：

```
function limitLoad(urls, handler, limit) {
  // 对数组做一个拷贝
  const sequence = [...urls];

  let promises = [];

  //并发请求到最大数
  promises = sequence.splice(0, limit).map((url, index) => {
    // 这里返回的 index 是任务在 promises 的脚标，用于在 Promise.race 之后找到完成的任务脚标
    return handler(url).then(() => {
```

```
    return index;
  });
});

// 利用数组的 reduce 方法来以队列的形式执行
return sequence.reduce((last, url, currentIndex) => {
  return last.then(() => {
    // 返回最快改变状态的 Promise
    return Promise.race(promises)
  }).catch(err => {
    // 这里的 catch 不仅用来捕获前面 then 方法抛出的错误
    // 更重要的是防止中断整个链式调用
    console.error(err)
  }).then((res) => {
    // 用新的 Promise 替换掉最快改变状态的 Promise
    promises[res] = handler(sequence[currentIndex]).then(() => {
      return res
    });
  })
}, Promise.resolve()).then(() => {
  return Promise.all(promises)
})

}

limitLoad(urls, loadImg, 3);

/*
因为 limitLoad 函数也返回一个 Promise，所以当 所有图片加载完成后，可以继续链式调用

limitLoad(urls, loadImg, 3).then(() => {
   console.log('所有图片加载完成');
}).catch(err => {
   console.error(err);
})
*/
```

评分标准：

B-基本实现

A-能熟练应用 Promise.reduce、Promise.race、Promise.all

S-能够包装成可复用的方法

5、使用 Promise 实现红绿灯 @李阳

红灯 3 秒亮一次，绿灯 1 秒亮一次，黄灯 2 秒亮一次；如何使用 Promise 让三个灯不断交替重复亮灯？

参考答案：

```
function red() {
  console.log('red');
}

function green() {
  console.log('green');
}

function yellow() {
  console.log('yellow');
}
```

```
let myLight = (timer, cb) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      cb();
      resolve();
    }, timer);
  });
};


let myStep = () => {
  Promise.resolve().then(() => {
    return myLight(3000, red);
  }).then(() => {
    return myLight(2000, green);
  }).then(()=>{
    return myLight(1000, yellow);
  }).then(()=>{
    myStep();
  })
};
myStep();

// output:
// => red
// => green
// => yellow
// => red
// => green
// => yellow
// => red
```

评分标准：

B-提醒如何用代码表达灯亮起后，基本实现

A-能自主实现，能将自然语言抽象成为代码

S-实现的基础上对工具方法能有合理的抽象

【深拷贝】

1、手写深拷贝方法，要考虑到非基本类型情况，比如正则、方法等？

B-能写出基本类型深拷贝 + 对象递归

A-能解决对象引用问题

S-解决对象引用问题并考虑各种类型的拷贝方法，比如正则拷贝方法，可以追问细节比

【其他】

1、多项式字符串运算 @金黎明

B- 给出一个包含加法和减法的多项式字符串，能够算出结果

A- 能够处理加减乘除混合运算，考虑运算的优先级

S- 引入带括号的情况和带小数点的情况，能够说出小数运算得到不准确结果的原因（精度问题）

2、实现一个贪吃蛇小游戏 @金黎明

B- 方案选择（DOM+CSS 或 Canvas）不限制，实现基本逻辑，包含运动方向、吃到食物变长、碰到边界游戏结束

A- 可以清晰描述代码各个部分的用途和设计思路

S- 代码可读性、函数拆分合理、命名合理，复杂逻辑部分提供了注释。若使用 TS 实现，则要求代码中的类型标注准确

3、实现一个短网址映射方案 @姚泽源

互联网上经常有把长网址缩短的需求, 请设计一个方案, 实现这两个功能: 1. 将长网址映射成 8 位短网址, 2. 将短网址还原为长网址

Tips:

- 可以假设待压缩的长网址只有 url 中的 pathname 部分, 以`/`开头
  - 示例输入(长度在 5000 以内的任意字符串): '/kuai/shou/xiao/zhao', '/kuai/le/xing/qiu/huan/ying/ni' , '/tdrfyguhij?wag=21asxyw'
  - 示例输出: '/1234abcd', '/efgh5678'
- 返回的短网址只能使用 url 允许的字符
- `shortToLong`的输入参数均来自于`longToShort`的返回值
- 可以使用伪代码导入算法包, 例如 `import quicksort from 'quicksort'`

```
class UrlService {
    shortToLong(shortPathname: string): string|undefined {}
    longToShort(longPathname: string): string {}
}
```

- B: 能够实现两个接口, 方案不限制
  - 可能的方案
    - 递增 id
      - 简单递增
      - id 递增后添加混淆前缀
      - id 递增后执行 hash
    - 随机 id
      - hash
      - 随机数
      - 随机数+进制转换
- A: 能够对比不同网址缩短方案, 考虑其中的安全隐患
  - 使用递增数字类方案性能最好, 但容易被找到规律进行攻击
  - 使用随机 id 类方案需要考虑 hash 碰撞概率
- S: 能够正确权衡实现成本, 考虑边界条件
  - 能够处理 hash 碰撞的情况(碰撞后添加随机数重新 hash, 尝试换一个 key)
    - hash 碰撞处理属于数据结构课程中的一部分, 可以借此考察候选人计算机知识的掌握程度, 如能答出常见碰撞解决方法为佳
      - 再 hash 法(二次 hash)
      - 链接法(将结果转换为链表挂载到 hash 值对应的桶下, 最差情况下 hash 的查找时间会劣化到和链表查找时间相同)
      - 参考: https://blog.csdn.net/kodoshinichi/article/details/109628529
  - 能够考虑到短网址若使用 base64 编码, +(空格) 和 /(分界) 有特殊含义, 不建议使用
    - 推荐使用 base62(只有 a-Z0-9)
  - 能够想到使用 hash 后只截取前 8 位, 会增大碰撞概率.
    - 常用 hash 算法(md5/sha256)的运算结果通常用 16 进制表示(`0123456789abcdef`), 但在 url 规范下, 每一位有 62 种可能, 也就是浪费了(62-16)/62 = 74%的状态空间
  - 能够考虑到短网址系统的容量和 hash 碰撞概率. 例如使用 16 进制的情况下, 其最大容量为 16^8 = 42 亿. 在容量使用一半(21 亿)后, 连续重新 hash 10 次仍没有可用 id 的可能性只有(1-1/2)^5=3.125%, 连续 10 次没有可用 id 的概率只有(1-1/2)^10 = 0.09%, 仍处于可接受的范围
  - 如果使用进制转换方案, 能够想到使用 js 自带的进制转换函数`Number.prototype.toString`和`parseInt`. 如果指出`parseInt`最大可以支持 36 进制(0-9a-z)可以加分
- 参考答案

```
// js 最大内置支持的基数
const Const_Radix = 36
// 短网址参数位数
const Const_Short_Url_Length = 8
```

```
// 在内置基数下, 所能支持的最大表达值, 当前状态下最多记录 2821109907455 条数据
// 若数据超过 Number.MAX_SAFE_INTEGER, 需考虑数字准确性问题.
// 目前离阈值仍有 3 个数量级, 因此可以先不考虑
const Const_Max_Size = parseInt(
    'z'.padEnd(Const_Short_Url_Length, 'z'),
    Const_Radix
)

const Const_Base = Math.pow(10, `${Const_Max_Size}`.length + 1)

class UrlService {
    // 将 url 按 shortUrlKey => url 的方式存储
    private db: { [key: string]: string } = {}
    // 按 url => shortUrlKey 的方式存储, 便于判断 url 是否已经生成过短网址
    private reverseDb: { [key: string]: string } = {}

    /**
     * 生成一个随机且不在 db 中存在的 key
     *
     * 也可以用随机排序的字符串作为映射字符, 但只要最终 id 是递增的, 就总会被人找到规律, 反而没有安全性. 不如随机数法, 在样本量小于 1000 亿时有良好的性能
     */
    private generateKey() {
        // 碰撞后需要重新生成一个 key, 当数据量达到 1000 亿时, 连续碰撞 5 次的概率也只有(1/10)^5, 不足万分之一.
        let isExist = true
        let randomKeyStr = ''
        while (isExist) {
            // 生成随机数
            let randomKey =
                Math.trunc(Math.random() * Const_Base) % Const_Max_Size
            // 通过 36 进制转换为字符串
            randomKeyStr = randomKey.toString(Const_Radix)
            isExist = this.db[randomKey] !== undefined
        }
        // 空缺位数补 0
        randomKeyStr = randomKeyStr.padStart(Const_Short_Url_Length, '0')
        return randomKeyStr
    }

    /**
     * 根据 key, 从数据库中获取对应 url, 没有则返回 undefined
     * @param shortPathname
     * @returns
     */
    shortToLong(shortPathname: string): string | undefined {
        return this.db[shortPathname]
    }

    /**
     * 添加 url 记录
     * @param longPathname
     * @returns
     */
    longToShort(longPathname: string): string {
        if (this.reverseDb[longPathname] !== undefined) {
            // 已经存储过, 直接返回原记录即可
            return this.reverseDb[longPathname]
        }

        let key = this.generateKey()
```

```
      this.db[key] = longPathname
      this.reverseDb[longPathname] = key // 额外存一份反向链接, 方便检测
      return key
    }
}
```

-

4、数组拍平&去重&排序 @姚泽源

已知以下纯整数构成的数组 inputNumberList, 请实现 listFormater 函数, 将数组扁平化并去除重复元素, 最终按从小到大的顺序返回结果(可以使用 ES6 语法)

```
let inputNumberList = [
   [17, 14, 0, 6, 19, 4, 19, 11, 3],
   [10, [7, 5, 5, 1], 19, 0],
   [18, 19, 16, [7, 9, 5, [14, 7, [14]]]],
   19,
   [4, 19, 16, 6],
   12,
   10,
   0,
]

const listFormater = (numList = []) => {
   return []
}

console.log(listFormater(inputNumberList))
```

- B: 能够实现 listFormater 方法, 方案不限制
- A: 能够将问题进行拆分, 按扁平化/排序/去重的顺序解决问题
- S:
  - 能够使用有意义的变量名, 能够主动编写测试方法, 自测通过后再进行提交.
  - debug 过程思路清晰
  - 编写代码时能够给出注释
- 补充: 若通过 js 内置 api 快速解答成功的话, 可以要求手工实现一份代码

参考答案

```
let inputNumberList = [
   [17, 14, 0, 6, 19, 4, 19, 11, 3],
   [10, [7, 5, 5, 1], 19, 0],
   [18, 19, 16, [7, 9, 5, [14, 7, [14]]]],
   19,
   [4, 19, 16, 6],
   12,
   10,
   0,
]

type Type_DeepArray<T> = Array<T | Type_DeepArray<T>>
```

```
const listFormater = (numList = []) => {
    function flattenList(inputList: Type_DeepArray<number>) {
        // 利用 array 内置方法直接打平数组
        // const resultList = inputList.flat(999);

        // 暴力打平数组
        // const resultList = inputList
        //   .join(",")
        //   .split(",")
        //   .map((item) => parseInt(item));

        // 递归打平数组
        let resultList: number[] = []
        for (let item of inputList) {
            if (Array.isArray(item)) {
                let flattenResult = flattenList(item)
                for (let resultItem of flattenResult) {
                    resultList.push(resultItem)
                }
            } else {
                resultList.push(item)
            }
        }

        return resultList
    }
    function uniqueEleInList(inputList: number[]) {
        // 利用对象 key 去重
        // let uniqObj = {};
        // for (let item of inputList) {
        //   uniqObj[item] = true;
        // }
        // 需要注意对象 key 只支持文本, 因此需要转换回来
        // const uniqueList = Object.keys(uniqObj).map((item) => parseInt(item));

        //  利用 set 去重
        const uniqueList = [...new Set(inputList).values()]
        return uniqueList
    }
    function sortList(inputList) {
        // 直接调用默认排序功能即可

        // 先解构, 避免对输入参数产生影响
        let resultList = [...inputList]
        // 如能指出默认排序为稳定排序可加分
        resultList.sort((a, b) => {
            return a - b
        })
        return resultList
    }

    let step1List = flattenList(numList)
    let step2List = uniqueEleInList(step1List)
    let step3List = sortList(step2List)

    return step3List
}

console.log(listFormater(inputNumberList))
```

四、框架能力

【主流框架熟悉程度- Vue/React/Angular】

1、Vue、react 、angular 区别和场景

B-能简单说出三大框架的使用场景

A-能详细说出框架间对于项目类型不同的取舍，并且能说出框架中关键思想代码的实现的区别，比如三个框架对于依赖搜集的处理等

S-能说出三个框架在细节、生命周期、模板编译并执行的大部分技术实现区别，和周边配套的简单实现与相应区别

2、Vue 中在写了一个 .vue 文件后，它最终是如何能够在浏览器运行的，大致的流程是什么样的？

B-能简单说出经历过通过 webpack loader 处理成 es5 被浏览器执行

A-能说出 webpack 中一些大致的 loader ，如 vue loader，template-loader，css-loader，babel-loader 等，并能说出大致的作用

S-能说出绝大部分 loader 的作用，以及 vue 的模板编译过程，响应式建立机制等，对首次编译并执行过程中设计的知识点有深刻的理解

3、虚拟 dom 是什么，为什么会提高性能

B-能说出减少，合并对 dom 的操作

A-能说出大致实现原理

S-能表述清楚不同框架之间虚拟 dom 库的实现

4、vue react 主要区别和相关组件的基础使用，如 router 原理使用，vuex 等

B-基本使用方式的不同；双向绑定实现原理

A-引导询问 svelte 的设计理念，继续询问其中的差异和适用场景并能说出一些

vue3 的设计理念和源码模块划分

S-了解或者研究过不同框架的核心不同，包含 vue react svelte。对于 vue 核心 diff 算法的优化和实现了解 或者 react fiber 等架构的实现原理和设计思路

5、如何实现一个 Vue 的组件/插件

B-了解基本的组件实现方式

A-自定义指令和插件的实现方式

S-了解基于 npm 生态的研发体系管理

6、Vue $nextTick 是做什么的 & 怎么实现的 @吕宪勇

[你真的理解$nextTick 么]

B-了解基本的 nextTick 用法

A-了解实现原理

S-结合 js 引擎线程和事件触发线程，清楚的认知到 nextTick 是在哪个环节发挥作用的

7、Vue diff 算法 @吕宪勇

[详解 vue 的 diff 算法]

B -了解数据发生变化后，vue 是怎么更新节点的

A - 了解 diff 算法的比较方式，diff 流程

S - 对比 react diff 算法，了解差异，或者不同 vue 版本，如 2.x 3.x 的 diff 算法差异

[React、Vue2、Vue3 的三种 Diff 算法]

[深入浅出虚拟 DOM 和 Diff 算法，及 Vue2 与 Vue3 中的区别]

8、Vue watch & computed 区别　@吕宪勇

[Vue.js 的 computed 和 watch 是如何工作的？]

B -了解 computed 和 watch 的基础定义，用法差异

A - 了解 watch 的高级用法，了解 computed 的本质：computed watch

S - 了解 watch 底层是如何工作的

9、JSX 和 HTML 模板的优劣 @陈勇

B - 知道三大框架哪些倾向于使用哪种方案

A - 知道 JSX 灵活，贴近 JS 生态，HTML 模板易于优化

S - 了解 JSX 和 HTML 模板在不同框架下的转译目标

10、数据可变、不可变的优劣 @陈勇

B - 知道三大框架哪些倾向于使用哪种方案

A - 知道数据可变方便逻辑撰写，数据不可变方便理清数据流

S - 知道在中小业务模型中适合使用数据可变模型，在大型复杂逻辑中使用数据不可变模型

11、函数式、过程式、面向对象的优劣 @陈勇

我对函数式编程、面向对象和面向过程三者的理解

B - 知道三大框架哪些 API 设计倾向于使用哪种方案

A - 知道函数式的优点，面向对象的优点

S - 知道在什么样的场合使用何种编程方式实现逻辑

12、基于 React，有哪些性能优化手段？ @任跃华

B. class 组件 shouldComponentUpdate，PureComponent，列表 key 属性等

A. 函数组件 React.memo()， useCallback，useMemo

S. 能从框架原理分析，能给出示例代码(伪代码也可)

13、使用 React 或 Vue，实现一个电话号校验组件，用户输入手机号为 11 位时提示通过？@任跃华

B. 使用任意框架，给出伪代码即可。

A. 有判空容错逻辑，体验优化逻辑。

S. 能给出 React 和 Vue 的不同实现，逻辑完整。

14、React useCallback、useDemo 区别？ @任跃华

B. 给出使用场景的区别。

A. 详细阐述参数和回调的不同，使用场景不同，并有具体例子。

15 Vue3 响应式数据 @彭彦鑫

B. 能说明白 Vue3 响应式数据的简单实现，能解释清楚 ref reactive 的区别和使用场景；

A. 能够说明白响应式数据的进阶用法，如使用 customRef 实现一个防抖 ref，能说明白那几个 shallow 的使用场景；

【 Node 相关】

1、什么样的业务场景不适合用 node？为什么？

B-了解 node 实现的基本原理，基于单线程模型，基于 js 语言特性，不适合计算密集型

A-能够理解 js java c++不同语言的实现原理和性能差异

S-除了语言差异，知道虚拟机语言的本地调用能力如何实现。也能够从业务场景和社区方面进行比较和思考

2、koa express egg midway 等不同框架的不同设计理念和关注点

B-了解 express koa egg midway 的基本差异和使用方法，了解相应框架常见的配套中间件

A-了解过 1～2 个框架的部分源码，能说出一些核心模块的挑战点，如进程管理模型、日志模块

S-能较好理解如 eggjs 框架的下沉式框架沉淀，以及动态插件机制的优势，有实践经验

3、简单说说实现下载断点续传的原理？

前端：

核心是利用 Blob.prototype.slice 方法，和数组的 slice 方法相似，调用的 slice 方法可以返回原文件的某个切片

这样我们就可以根据预先设置好的切片最大数量将文件切分为一个个切片，然后借助 http 的可并发性，同时上传多个切片，这样从原本传一个大文件，变成了同时传多个小的文件切片，可以大大减少上传时间

另外由于是并发，传输到服务端的顺序可能会发生变化，所以我们还需要给每个切片记录顺序

服务端：

服务端需要负责接受这些切片，并在接收到所有切片后合并切片

这里又引伸出两个问题

1. 何时合并切片，即切片什么时候传输完成
2. 如何合并切片

第一个问题需要前端进行配合，前端在每个切片中都携带切片最大数量的信息，当服务端接受到这个数量的切片时自动合并，也可以额外发一个请求主动通知服务端进行切片的合并

第二个问题，具体如何合并切片呢？这里可以使用 nodejs 的 读写流（readStream/writeStream），将所有切片的流传输到最终文件的流里

【常见库熟悉程度】

1、lodash axios element-ui ant-design 等不同库的使用情况，实际经验中的踩坑程度和解决办法

B-使用过，能说出关键 api 和组件使用

A-对部分核心源码有过了解

【打包工具熟悉程度】

1、babel，ts，webpack，vite 等使用情况和一些常用配置

2、webpack loader 和插件的区别，以及常用的 loader 和插件分别有哪些？

B-能简单说出 loader 和插件的试用场景

A-能说出 loader 和插件实现函数的参数区别已经可以延展出能做些什么事儿

S-能针对某一细分 loader 集合进行细致分析，比如 sass-loader；style-loader；css-loader，插件方面能举出常见热更新的实现，延展出 tree-shaking 等。

3、tree-shaking 实现原理

B-能说出是通过插件来实现

A-能说出是 unuseless 注释 + 插件

S-能说出如何打无用代码 tag，插件中如何用参数干掉，并且能说出 rollup 和 webpack 对于 tree-shaking 处理的不同

Tree-Shaking 进阶参考: tree shaking 问题排查指南
TreeShaking 需要注意意外优化的情况
- 对于以下代码

```
const obj = {};
obj.name = "obj";
export const answer = 42;
```

- rollup 编译结果
- `const answer = 42;export {answer};`
- esbuild 编译结果
- `const obj = {};obj.name = "obj";const answer = 42;export {answer};`
- 看起来 rollup 正确的清除了冗余代码, 实际上在特殊情况下, rollup 的优化会导致出错

```
// 本来代码的意思是每次设置一个变量属性的时候，都要触发一次 render，结果由于 obj.name 代码被删除，导致 render 没被触发，这明显改变了语义。
function render(val) {
  console.log("render", val);
}
Object.defineProperty(Object.prototype, "name", {
  set(val) {
    render(val);
  },
});
```

- 该 bug 在 esbuild 上的对应 issue

● 软性素质题目（需举出实例）

一、沟通协作

解读：

1、语言表达条理清晰，用语准确，重点突出；

2、积极倾听，尊重他人，正确理解他人的感受、观点和看法能力；

3、能够有效运用各种沟通技巧说服他人，影响他人观点或行为能力；

4、能够理解和容忍他人的行为，协调解决矛盾冲突的能力；

5、建立并维持富有成效的工作关系的能力。

1、合作模式：你跟你的业务搭档会在什么场景下沟通，主要沟通哪些内容？

2、处理分歧：分享一个你和你的老板/同事有分歧的事例，你是怎样处理这些分歧的？

3、寻求配合：当你的工作需要其他部门协助时，你是如何取得其他部门的配合的？请举例说明。

4、处理情绪：你曾经和一个生气的合作方交谈过吗？如果有，你是怎么处理这个情况的？

5、输出观点：你是如何让别人接受你的观点或看法的？请举例说明。

二、学习能力

解读：

1、能紧跟行业内新知识、新技术动态，并运用到实际问题解决；

2、会思考、会探索问题，对行业知识有极大的好奇心，能高效有质量的完成工作；

3、乐于分享知识经验。

1、解决挑战：请分享一个你遇到的最困难的项目/最大的挑战，你在这个项目中是什么角色？在这个项目的完成过程中起到了哪些作用？

2、新事物学习：你在 XX 公司实习的时候应该是第一次接触这类产品/项目吧，你当时是怎么快速去熟悉产品的？

3、知识获取：你是如何快速获取需要的知识的？近期看的一本技术书是哪一本？给你印象最深刻的是什么？

4、新技术学习和应用：你有运用新技术去解决过项目中遇到的问题吗？具体是什么问题？你是如何学会并运用的？

5、开源项目：你是否研究过开源项目？有什么收获？

6、技术博客：是否有技术写作的习惯？写过技术博客吗？

7、学习计划：今年有什么学习计划？今年有什么目标？

三、积极主动&思维能力

解读：指一种积极、乐观、提前行动的思维方式，碰到困难时不抱怨，而是积极地寻找更为有效的行动来规避或解决。

1、在这个项目中你遇到的最有难度的坑是什么？你当时是怎么解决的？

2、是否在工作中遇到过一些流程、协作上的问题，最后是怎么解决的？

四、抗压能力

解读：面对压力、委屈、和失败，能调整心态积极去应对，直面问题不逃避。乐于接受批评，会自我审视并调整；不过分骄傲也不长时间沮丧，有正向思考的能力，向善。

1、你的上司和同事如何评价你（含优缺点）？你怎么看他们的评价？对于缺点有哪些改进动作？

2、你有没有遇到过延迟上线的情况？如果有，当时情况是怎样应对的？如果没有，你是怎么确保自己每次都能准时完成的？

3、工作中有没有情绪特别低落的时候？当时是什么原因？对工作有影响吗？如果没有影响，是怎样调节自己的情绪的？

五、复盘意识

解读：不论结果好与坏，都能够在事后进行复盘和反思，总结经验与问题，寻求更好的解决方案，追求极致。

1、遇到过最有难度的坑是什么？你当时是怎么解决的？如果现在看，是否还有其他解决办法？

2、是否有自己产出的技术方案或思路推广给其他人使用的？使用后有什么效果？是否对你反哺新的帮助？

六、好奇心&举一反三

解读：不仅局限于现有技术，会随时了解技术新动态，热衷于研究新技术并运用在工作当中；同时，能尝试用多种方法解决问题，有灵活性，对一件事情有不同甚至相反想法的理解

与欣赏，使自己的方法能适应环境的变化。

1、工作之余是否有研究过新的技术领域？是如何去了解的？目前获得了哪些成果？

2、你觉得过往在技术上的学习/实践中，谁给你的帮助是最大的？从 ta 身上除了技术上的成长以外你还有哪些别的收获？

3、是否有在新工作中应用过往经验的情况？新的环境下如何让旧方案适应新的需求？