React

1. 遍历子节点的时候，为什么不要用 index 作为组件的 key 进行传入？
2. 怎么去设计一个组件封装？
    - 组件封装的目的是为了重用，提高开发效率和代码质量
    - 低耦合，单一职责，可复用性，可维护性
3. react 的虚拟 dom 是怎么实现的
    - 首先说说为什么要使用 Virtual Dom，因为操作真实的 Dom 耗费的性能代价太高，所以 react 内部使用 js 实现了一套 dom 结构，在每次操作真实 dom 之前，使用实现好的 diff 算法，对虚拟 dom 进行比较，递归找出有变化 的 dom 节点，然后对其进行更新操作。
    - 为了实现虚拟 Dom,我们需要把每一种节点类型抽象成对象 ,每一种节点类型有自己的属性 ,也就是 prop, 每次进行 diff 的时候 ,react 会先比较该节点类型 ,假如节点类型不一样 ,那么 react 会直接删除该节点，然后直接创建新的节点插入到其中，假如节点类型一样，那么会比较 prop 是否有更新，假如有 prop 不一样，那么 react 判定该节点有更新，那么重渲染该节点，然后在对其节点进行比较，一层一层往下，直到没有子节点。
4. react hooks 原理是什么？
    - hooks 是用闭包实现的，因为纯函数不能记住状态，只能通过闭包来实现。
5. 数据双向绑定和单向绑定的优缺点
    - 双向绑定是自动管理状态的，对处理有用户交互的场景非常合适，代码量少，当项目越来越大的时候，调试也变的越来越复杂，难以跟踪问题。
    - 单向绑定是无状态的，程序调试相对容易，可以避免程序复杂度上升时产生的各种问题，当然写代码时就没有双向绑定那么简单方便了。
6. react-router 里的 Link 标签和 a 标签有什么区别？
    - 从 DOM 渲染来看，这两者都是链接，都是 a 标签。区别是：
    - Link 是 react-router 里实现路由跳转的链接，配合 Route 使用，react-router 拦截了其默认的链接跳转行为，区别于传统的页面跳转。
    - Link 的”跳转”行为只会触发相匹配的 Route 对应的页面内容更新 ,而不会刷新整个页面。a 标签是 HTML 原生的超链接，用于跳转的 href 指向的另一个页面或锚点元素，跳转新页面会刷新页面。

7. react-hooks 的优劣如何？

    优点：

    简洁：React Hooks 解决了 HOC 和 render props 的嵌套问题，更加简介。

    解耦：React Hooks 可以更方便地把 UI 和状态分离，做到更彻底的解耦。

    组合：Hooks 中可以引用另外的 hooks 形成新的 Hooks,组合变化万千。

    函数友好：React Hooks 为函数组件而生，从而解决了类组件的几大问题：

    this 指向容易错误

    分割在不同声明周期中的逻辑使得代码难以理解和维护

    代码复用成本高(高阶组件容易使代码量剧增)

    缺点:

    额外的学习成本（Function Component 与 Class Component 之间的困惑）。

    写法上的限制（不能出在条件、循环中），并且写法限制增加了重构成本。

    破坏了 PureComponent、React.memo 浅比较的性能性能优化效果(为了取最新的 props 和 state,每次 render() 都要重新创建事件处函数）

    在闭包场景可能会引用到旧的 state、props 值。

内部实现上不直观（依赖一份可变的全局状态，不再那么”纯“）

React.memo 并不能完全替代 shouldComponentUpdate(因为拿不到 state Change,只针对 props Change)

8. diff 算法大致的机制?

   为了实现虚拟 Dom,我们需要把每一种节点类型抽象成对象，每一种节点类型有自己的属性，也就是 prop,每次进行 diff 的时候，react 会先比较该节点类型，假如节点类型不一样，那么 react 会直接删除该节点，然后直接创建新的节点插入到其中，假如节点类型一样，那么会比较 prop 是否有更新，假如有 prop 不一样，那么 react 判定该节点有更新，那么重渲染该节点，然后在对其节点进行比较，一层一层往下，直到没有子节点。

9. 为什么选择使用框架而不是原生？

   组件化 ： 其中以 React 的组件化最为彻底，甚至可以到函数级别的原子组件，高度的组件化可以使我们的工程易于维护、易于组合拓展。

   天然分层：JQuery 时代的代码大部分情况下是面条代码，耦合严重，现代框架不管是 MVC、MVP 还是 MVVM 模式都能帮助我们进行分层，代码解耦更易于读写。

   生态：现在主流前端框架都自带生态，不管是数据流管理架构还是 UI 库都有成熟的解决方案。

   开发效率：现代前端框架都默认自动更新 DOM,而非我们手动操作，解放了开发者的手动 DOM 成本，提高了开发效率，从根本上解决了 UI 与状态同步问题。

10. React vue ，在开发过程中前端处理跨域
    - 代理

JS

11. Promise
12. Ajax 封装，请求拦截

移动端

13. 移动端 HTML5 audio autoplay 失效问题

这个不是 BUG，由于自动播放网页中的音频或视频，会给用户带来一些困扰或者不必要的流量消耗，所以苹果系统和安卓系统通常都会禁止自动播放和使用 JS 的触发播放，必须由用户来触发才可以播放

解决方法思路：先通过用户 touchstart 触碰，触发播放并暂停（音频开始加载，后面用 JS 再操作就没问题了）。

解决代码：

```
document.addEventListener('touchstart',function() {
  document.getElementsByTagName('audio')[0].play();
  document.getElementsByTagName('audio')[0].pause();
});
```

14. 移动端自适应方法
    - PxtoRem
    - 媒体查询
15. React-native 有无独立开发经验
16. Sdk 封装
17. 消息推送
18. WebView，如果接入 H5 页面，数据是如何传递的？
19. UI 库的是否使用

20. 调试工具
21. 地图的使用，是自己封装的吗？ 使用的是什么地图？

Webpack

1. 有哪些常见的 Loader？你用过哪些 Loader？
2. 有哪些常见的 Plugin？你用过哪些 Plugin？
3. Webpack 的运行流程
4. 说一下 Webpack 的热更新原理

   Webpack 的热更新又称热替换（Hot Module Replacement），缩写为 HMR。 这个机制可以做到不用刷新浏览器而将新变更的模块替换掉旧的模块。

   HMR 的核心就是客户端从服务端拉去更新后的文件，准确的说是 chunk diff (chunk 需要更新的部分)，实际上 WDS 与浏览器之间维护了一个 Websocket，当本地资源发生变化时，WDS 会向浏览器推送更新，并带上构建时的 hash，让客户端与上一次资源进行对比。客户端对比出差异后会向 WDS 发起 Ajax 请求来获取更改内容(文件列表、hash)，这样客户端就可以再借助这些信息继续向 WDS 发起 jsonp 请求获取该 chunk 的增量更新。

   后续的部分(拿到增量更新之后如何处理？哪些状态该保留？哪些又需要更新？)由 HotModulePlugin 来完成，提供了相关 API 以供开发者针对自身场景进行处理，像 react-hot-loader 和 vue-loader 都是借助这些 API 实现 HMR。