1.React 是什么

React 是一个简单的 javascript UI 库，用于构建高效、快速的用户界面。它是一个轻量级库，因此很受欢迎。它遵循组件设计模式、声明式编程范式和函数式编程概念，以使前端应用程序更高效。它使用虚拟 DOM 来有效地操作 DOM。它遵循从高阶组件到低阶组件的单向数据流。

2.REACT 全家桶

react 全家桶都有什么？

react 全家桶有：react + redux(状态管理) +react-router(路由) + axios + antd。

1、react

react 的核心。

2、redux

redux 相当于一个数据库，可以当成一个本地的数据库使用，react-redux 可以完成数据订阅，redux-thunk 可以实现异步的 action，redux-logger 是 redux 的日志中间件。

3、react-router

React Router 是专为 React 设计的路由解决方案。它利用 HTML5 的 history API，来操作浏览器的 session history (会话历史)。

React Router 被拆分成四个包：react-router，react-router-dom，react-router-native 和 react-router-config。react-router 提供核心的路由组件与函数。react-router-config 用来配置静态路由（还在开发中），其余两个则提供了运行环境（浏览器与 react-native）所需的特定组件。

进行网站（将会运行在浏览器环境中）构建，我们应当安装 react-router-dom。因为 react-router-dom 已经暴露出 react-router 中暴露的对象与方法，因此你只需要安装并引用 react-router-dom 即可。

4、axios

axios 是基于 Promise 的用于浏览器和 Node.js 的 http 客户端。可以发送 get、post 等 http 请求，用来和服务器进行交互的。

5、antd

Ant Degisn 是个很好的 React UI 库，看起来跟我们熟知的 bootstrap 有点类似，从页面布局到按钮到文字提示泡应有尽有。

3.介绍 Redux，主要解决什么问题？数据流程是怎么样的？多个组件使用相同状态如何进行管理？

1.Redux 是一个状态容器，解决了组件间的状态共享问题，实现了多组件通信，也是一种 MVC 机制

2.Redux4 个模块

1. view - 组件状态改变时，dispatch 一个 action
2. action：生成一个描述状态变化的 action 对象，并传递给 store
3. store：接收 action 对象并传递给 reducer，驱动 reducer 执行状态更新任务，更新后刷新 dom 节点(即 view 中执行 render)
4. reducer：对 action 对象处理，更新组件状态，并将新的状态值返回 store

3.Redux 数据流程

1. 创建一个 store 存储数据，定义改变 store 的 action，里面是各种 reducer 的映射

2. view 层 dispatch 一个 action，store 接收到 action 修改状态，执行 update 方法，调用 subscribe 传进来的回调

3. view 层在 componentDidMount 中订阅状态更新，拿到更新后的 store 后 setState、forceUpdate 更新 view

4 多组件使用相同状态。都放到同一个 reducer 里管理即可。

4. React-Redux 到 react 组件的连接过

1.Provider 为后代提供 store：使用时在应用的最外层包一层，本质是上下文跨层级传递 store

2.connect 为组件提供数据和变更方法：本质是高阶组件，连接 React 组件与 Redux store，返回一个新的已经与 Redux Store 连接的组件类，实现多个组件使用相同状态。通过 mapStateToProps 和 mapDispatchToProps 给组件添加使用 state 和 dispatch 的方法。connect 中通过 setState 触发组件 render。

1.mapStateToProps(Function)：当 Redux store 发生变化时，调用该回调，此方法必须返回一个 Object，这个 Object 与组件的 props 融合，不传 mapStateToProps 时组件不监听 Redux store。谨慎使用第二个参数 ownProps，父组件的 prop 改变也会引起 mapStateToProps 执行

2.mapDispatchToProps：返回一个对象(action creator 映射)或 function 接收一个 dispatch，可能会使用到 bindActionCreator

5. Redux 中间件是什么东西，接受几个参数

redux 中间件是一个函数，对 dispatch 方法增强，使 dispatch 不再接收纯对象的 action，而是接受一个函数，把原来的 dispatch 当参数传进去。

接收参数是一系列的中间件函数，使用 reduce 实现 compose，在 action 发出和执行 reducer 之间做这些操作。

手写 compose:

```
function compose (...funcs) {
  funcs.reduce((a, b) => {
      return (...args) => a(b(...args)) // ...args 是第一次传入的参数
  })
}
```

compose 返回增强型的 dispatch

6. redux 请求中间件如何处理并发

1.redux-trunk 中间件 - 传入 dispatch 的 action 可以定义成方法 fun，fun 可以拿到 dispatch 控制权，这样就可以自己控制何时出发 reducer 改变 state

1.通过 Promise.all 实现并发

2.使用 async、await 配合 Array.map() // forEach 循环是顺序执行不是并发？存疑

3.可以使用队列控制并发数量

2.redux-saga 中间件 - generator + promise

7. Redux 中异步的请求怎么处理

需要 dispatch 两次 action，用户触发第一次 dispatch 时执行异步操作，异步操作有返回时触发真正的 dispatch

使用 redux-trunk，拿到 dispatch 控制权，就可以自己决定何时出发 reducer 改变 state

8. Redux 状态管理器和变量挂载到 window 中有什么区别

数据可控

数据响应

9.如何配置 React-Router

4 浏览器应用引入 react-router-dom 包，会自动引入 react-router，RN 项目引入 react-router-native

2.基本组件

1.Router 路由器

2.Link 连接

3.Route 路由

4.Switch 独占

5.Redirect 重定向

3.Route 渲染的 3 种方式，按优先级排列

1.children 不匹配时 match 为 null

2.component

3.render

4.：使用内联函数渲染时，需要使用 render/children，而不是 component

5.动态路由

6.嵌套路由

7.路由守卫：创建高阶组件包装 Route 使其具有权限判断功能，一般结合 store

10.react-router 怎么实现路由切换？BrowserRouter as Router

1.Router 组件使用 Provider 向下提供 history 对象，以及 location 的变更监听

2.Route 接收 loaction(props || context)，Router 尝试其属性 path 与 location 进行匹配，匹配检测(children > component > render)，内容渲染

3.Link 跳转链接，生成一个 a 标签，禁用默认事件，js 处理点击事件跳转(history.push) -> Link 与直接使用 a 标签的区别是 a 页面会重新加载一下，Link 只重新渲染 Route 组件部分

4.Redirect to

5.从上下文中获取 history this.context.router.history.push...

11.路由的动态加载模块

component：import(...)，打包时按 import 进来的路由页面分割成多个 chunk，按需加载

使用 react-loadable 包实现懒加载

12.前端怎么控制管理路由

实现一个 PrivateRoute 高阶组件做路由守卫

13.使用路由时出现问题如何解决

版本升级带来的问题

14.多个组件之间如何拆分各自的 state，每块小的组件有自己的状态，它们之间还有一些公共的状态需要维护，如何思考这

每块小组件自己的状态放到私有 state 中，公共的才用 Redux 管理

15.React 生命周期及自己的理解，以及 V16 对生命周期的修改

1. 3 阶段

1.挂载时

1.constructor

2.getDerivedStateFromProps

3.render -> React 更新 DOM 和 refs

4.componentDidMount (commit 阶段)

2.更新时 -> 触发更新的条件：父 props 改变、setState、forceUpdate()

1.getDerivedStateFromProps

2.shouldComponentUpdate -> 可优化组件渲染

3.render

4.getSnapshotBeforeUpdate (在 commit 前夕执行) -> React 更新 DOM 和 refs

5.componentDidUpdate

3.卸载时

1.componentWillUnmount

2.生命周期的变革？原因？

1.v17 打算删除 3 个 will，fiber 出现后，任务切片可以被中断，这 3 个 will 方法可能会多次执行，不符合生命周期定义初衷

1.componentWillMount

2.componentWillReceiveProps

3.componentWillUpdate

2 新引入 2 个：

1.static getDerivedStateFromProps -> 不能获取 this，不在那些 will 里引起副作用

2.getSnapshotBeforeUpdate：使用场景不多，仅对老 api 的替代

16.react diff 算法

传统 diff：diff 算法即差异查找算法，对于 Html Dom 结构即为 tree 的差异查找算法，而对于计算两棵树差异时间复杂度为 O(n*3)，显然成本太高，react 不可能采用这种传统算法；

React Diff:React 采用虚拟 Dom 技术实现对真实 Dom 的映射，即 React Diff 算法的差异查找实质是对两个 JavaScript 对象的差异查找；只有在 React 更新阶段才有 Diff 算法的应用；事实上，Diff 算法只被调用于 React 更新阶段的 Dom 元素更新过程，因为如果为更新文本类型，内容不同就直接更新替换，并不会调用复杂的 diff 算法，在自定义组件中，自定义组件最后结合 React Diff 优化。策略（不同的组件具备不同的结构）

Diff 策略：

DOM 节点跨层级的操作特别少，可以忽略不计 拥有相同类的两个组件将会生成相似的树形结构，拥有不同类的两个组件将会生成不同的树形结构 同一层级的一组子节点，他们可以通过 uuild 进行区分

对于 Diff 的开发建议：

基于 tree diff：

开发组件时，注意保持 Dom 结构的稳定；即，尽可能少的动态操作 Dom 结构，尤其是移动操作。 当节点过大或者页面更新次数过多时，页面卡顿的现象会比较明显。这时可以通过 css 隐藏或显示节点，而不是真的移除或添加 DOM 节点。

==基于 component diff： == 基于 shouleComponentUpdate()来减少组件不必要的更新 对于类似的结构应该尽量封装成组件，减少代码量，又能减少 component Diff 的性能消耗

基于 element diff：

对于列表结构，尽量较少类似将最后一个节点移动到列表首部的操作，当节点数量过大或更 新操作过于频繁时，在一定程度上会影响 React 的渲染性能。

17.为什么使用虚拟 dom?

优点:

1. 保证性能下限: 虚拟 DOM 可以经过 diff 找出最小差异,然后批量进行 patch,这种操作虽然比不上手动优化,但是比起粗暴的 DOM 操作性能要好很多,因此虚拟 DOM 可以保证性能下限

2. 无需手动操作 DOM: 虚拟 DOM 的 diff 和 patch 都是在一次更新中自动进行的,我们无需手动操作 DOM,极大提高开发效率 跨平台: 虚拟 DOM 本质上是 JavaScript 对象,而 DOM 与平台强相关,相比之下虚拟 DOM 可以进行更方便地跨平台操作,例如服务器渲染、移动端开发等等

缺点：

无法进行极致优化: 在一些性能要求极高的应用中虚拟 DOM 无法进行针对性的极致优化,比如 VScode 采用直接手动操作 DOM 的方式进行极端的性能优化

18.虚拟 dom 实现原理：

虚拟 DOM 本质上是 JavaScript 对象,是对真实 DOM 的抽象;

状态变更时，记录新树和旧树的差异

最后把差异更新到真正的 dom 中

19.react 合成事件

采用事件冒泡的形式冒泡到 document 上面，然后 React 将事件封装给正式的函数处理运行和处理。围绕浏览器原生事件充当跨浏览器包装器的对象。它们将不同浏览器的行为合并为一个 API。这样做是为了确保事件在不同浏览器中显示一致的属性。

20.什么是高阶组件（HOC）

高阶组件是重用组件逻辑的高级方法，是一种源于 React 的组件模式。 HOC 是自定义组件，在它之内包含另一个组件。它们可以接受子组件提供的任何动态，但不会修改或复制其输入组件中的任何行为。你可以认为 HOC 是“纯（Pure）”组件。

可以用于：

代码重用，逻辑和引导抽象

渲染劫持

状态抽象和控制

Props 控制

21.react 中 key 的重要性：

key 用于识别唯一的 Virtual DOM 元素及其驱动 UI 的相应数据。它们通过回收 DOM 中当前所有的元素来帮助 React 优化渲染。在虚拟 dom 节点中赋予 key 值，会更加快速的拿到需要的目标节点，不会造成就地复用的情况，对于节点的把控更加精准。

22.react-hook

Hook 是 React 16.8 的新增特性。它可以让你在不编写 class 的情况下使用 state 以及其他的 React 特性。

useState:函数组件里面没有 state，所以我们使用 useState，只接受一个参数，就是该 state 属性的初始值，它会返回一个数组，里面包含两个值，第一个值是初始值，第二个用来更改初始值。

useEffect:函数式组件没有生命周期，使用 useEffect 来替代 componentDidMount 和 componentDidUpdate。有两个参数，第一个是一个回调函数，第二个是空数组。如果第二个数组为空数组，就会在初始化执行完成之后，执行一次，相当于 componentDidMount，数组有内容，会根据数组内容改变去执行前面的回调函数。相当于 componentDidUpdate 生命周期。

userContext：组件之间共享状
https://blog.csdn.net/weixin_43606158/article/details/100750602
useReducer：相当于简单的 redux。接收两个参数，第一个参数是一个回调函数，里面接收一个 state 数据，以及 action，通过 dispatch 来更改内容。第二个参数是一个初始值。
useCallback：传入两个参数，第一个是回调函数，第二个是依赖，这个回调函数只在某个依赖改变时才会更新。
useMemo：可以优化用以优化每次渲染的耗时工作。
useRef：它可以用来获取组件实例对象或者是 DOM 对象，来跨越渲染周期存储数据，而且对它修改也不会引起组件渲染。

23. redux

state：组件内部状态
action：组件动作，相应的改变组件内部的状态值
dispatch：发出相应的动作
单一事件源，状态只读，使用纯函数进行更改
redux 中提供 createStore 方法用于生成一个 store 对象，这个函数接收一个初始值 state 和一个 reducer 函数，当用户发出相应的 action 时，利用传入的 reducer 函数计算一个新的 state 值，并返回。当存在多个 reducer，分别管理不同的 state，需要将其合并成一个 reducer，我们使用 combinReducers 函数。
react-redux 提供了一个 Provider 组件，以及 connect 方法，Provider 组件作为上层组件，需要将 store 作为参数注入到组件中，此后在子组件中都可以访问到 store 这个对象，connect 方法接收两个参数：mapStateToProps，actionCreators，并返回处理后的组件，其中 mapStateToProps 可以将对应的 state 作为 props 注入对应的子组件，actionCreator 可以将对应的 actioncreator 作为 prop 注入对应的子组件。

24. redux 中间件

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

25. redux 优缺点

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。

26. 简述 flux 思想

Flux 的最大特点，就是数据的"单向流动"1.用户访问 View2.View 发出用户的 Action3.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

27. redux-saga

redux-saga 的使用主要是添加监听，用于处理自定义的异步处理请求，在将结果调用 put 方法发起 action，从而交给 reducer 方法处理，使得 redux 异步处理更灵活。

saga 需要一个全局监听器（watcher saga），用于监听组件发出的 action，将监听到的 action 转发给对应的接收器（worker saga），再由接收器执行具体任务，任务执行完后，再发出另一个 action 交由 reducer 修改 state，，所以这里必须注意：watcher saga 监听的 action 和对应 worker saga 中发出的 action 不能是同一个，否则造成死循环。在 saga 中，全局监听器和接收器都使用 Generator 函数和 saga 自身的一些辅助函数实现对整个流程的管控

28. redux-thunk

thunk 采用的是扩展 action 的方式：使得 redux 的 store 能 dispatch 的内容从普通对象扩展到函数

29. 组件通信

父组件向子组件通讯: 父组件可以向子组件通过传 props 的方式，向子组件进行通讯
子组件向父组件通讯: props+回调的方式,父组件向子组件传递 props 进行通讯，此 props 为作用域为父组件自身的函数，子组件调用该函数，将子组件想要传递的信息，作为参数，传递到父组件的作用域中。
pubsub 可以采用发布订阅的方式实现组件间的传值。
import Pubsub from 'pubsub-js'
Pubsub.publish('username',val)//发布
Pubsub.subscribe('username',(msg,data)=>{}//data 是你要拿回来的数据。
兄弟组件通信: 找到这两个兄弟节点共同的父节点,结合上面两种方式由父节点转发信息进行通信
跨层级通信: Context 设计目的是为了共享那些对于一个组件树而言是“全局”的数据，例如当前认证的用户、主题或首选语言,对于跨越多层的全局数据通过 Context 通信再适合不过。 context 使用了 Provider 和 Customer 模式，和 react-redux 的模式非常像。在顶层的 Provider 中传入 value， 在子孙级的 Customer 中获取该值
发布订阅模式: 发布者发布事件,订阅者监听事件并做出反应,我们可以通过引入 event 模块进行通信
全局状态管理工具: 借助 Redux 或者 Mobx 等全局状态管理工具进行通信,这种工具会维护一个全局状态中心 Store,并根据不同的事件产生新的状态

30. React-router 里面的 hash-router 和 broswer-router 的区别

hashrouter 是以#号方式匹配路由，从 url 中可以看出来，这个地址对于后端来说，全部指向同一个地址 而 browserrouter 不存在#的，不同的路由对于后端也是不同的地址 当你需要做同步渲染的时候，肯定是要用 browserrouter 的

31. 为什么虚拟 dom 会提高性能

虚拟 dom 相当于在 js 的真实 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。

32. 虚拟 dom 和真实 dom 区别

1. 虚拟 dom 不会进行排版与重绘操作 2. 虚拟 dom 进行频繁修改,然后一次性比较并修改真实 dom 中需要改的部分,最后在真实 dom 中进行排版与重绘 3. 真实 dom 频繁修改效率极低. 4. 虚拟 dom 有效降低大面积的重绘和排版

33. mobx

MobX 是响应式编程，实现状态的存储和管理。使用 MobX 将应用变成响应式可归纳为三部曲：
定义状态并使其可观察

创建视图以响应状态的变化
更改状态
observable 是将类属性等进行标记，实现对其的观察。三部曲中的第一曲，就是通过 Observable 实现的。
通过 action 改变 state。三部曲中的第一曲通过 action 创建一个动作。action 函数是对传入的 function 进行一次包装，使得 function 中的 observable 对象的变化能够被观察到，从而触发相应的衍生。

34. computed

计算值(computed values)是可以根据现有的状态或其它计算值衍生出的值。简单理解为对可观察数据做出的反应，多个可观察属性进行处理，然后返回一个可观察属性
使用方式：1、作为普通函数，2、作为 decorator

35. autorun

当我们使用 decorator 来使用 computed,我们就无法得到改变前后的值了，这样我们就要使用 autorun 方法。
从方法名可以看出是“自动运行”。 所以我们要明确两点： 自动运行什么，怎么触发自动运行
自动运行传入 autorun 的参数 ，修改传入的 autorun 的参数修改的时候会触发自动运行

36. when

when 观察并运行给定的 predicate，直到返回 true。 一旦返回 true，给定的 effect 就会被执行，然后 autorunner(自动运行程序) 会被清理。
该函数返回一个清理器以提前取消自动运行程序。

37. reaction

它接收两个函数参数，第一个(数据
函数)是用来追踪并返回数据作为第二个函数(效果 函数)的输入。 不同于 autorun 的是当创建时效果函数不会直接运行(第二个参数不会立即执行，autorun 会立即执行传入的参数方法)，只有在数据表达式首次返回一个新值后才会运行。 在执行 效果 函数时访问的任何 observable 都不会被追踪。

38. action

在 redux 中，唯一可以更改 state 的途径便是 dispatch 一个 action。这种约束性带来的一个好处是可维护性。整个 state 只要改变必定是通过 action 触发的，对此只要找到 reducer 中对应的 action 便能找到影响数据改变的原因。强约束性是好的，但是 Redux 要达到约束性的目的，似乎要写许多样板代码，虽说有许多库都在解决该问题，然而 Mobx 从根本上来说会更加优雅。 首先 Mobx 并不强制所有 state 的改变必须通过 action 来改变，这主要适用于一
些较小的项目。对于较大型的，需要多人合作的项目来说，可以使用 Mobx 提供的 api configure 来强制

39. observer

mobx-react 的 observer 就将组件的 render 方法包装为 autorun，所以当可观察属性的改变的时候，会执行 render 方法。

40. observable

observable 是一种让数据的变化可以被观察的方法

observable(value) 是一个便捷的 API ，此 API 只有在它可以被制作成可观察的数据结构(数组、映射或 observable 对象)时才会成功。对于所有其他值，不会执行转换。

41. setState 为啥是异步

保证内部的一致性

因为 在没有重新渲染 父组件 的情况下，React 不会立即更新 props，如果 将 props 也设为同步更新的话，那么就必须放弃 批处理（batching）。 state、props 变化一次，就重渲染一次，这会显著地降低 React 性能。

能够使用并发更新，从而优化性能

我们可能认为 状态更新是按照一定顺序更新的，但事实上，React 会根据不同的调用源，给不同的 setState() 分配不同的优先级。

42. jsx 的优点

1、JSX 执行更快，因为它在编译为 JavaScript 代码后进行了优化

2、JSX 是类型安全的，在编译过程中就能发现错误

3、使用 JSX 编写模板更简单快速

JSX 是 JavaScript 的一种扩展，为函数调用和对象构造提供了语法糖，特别是 React.createElement()。JSX 看起来可能更像是模板引擎或 HTML，但它不是。JSX 生成 React 元素，同时允许你充分利用 JavaScript 的全部功能。

JSX 是编写 React 组件的极好方法，有以下优点：

改进的开发人员体验（Developer Experience,DX）：代码更易读，因为它们更加形象，感谢类 XML 语法，从而可以更好地表示嵌套的声明式结构。

更具生产力的团队成员：非正式开发人员（如设计师）可以更容易地修改代码，因为 JSX 看起来像 HTML，这正是他们所熟悉的。

更少的手腕磨损和语法错误：开发人员需要敲入的代码量变得更少，这意味着他们犯错的概率降低了，能够减少重复性伤害。