1. React：通过操作数据来改变界面中的组件，不用通过改变dom来改变组件
2. 定义组件。两种方法：1工厂函数组件（简单组件），2ES6类组件（复杂组件）
3. 工厂函数组件（简单组件）：function MyComponent(){return html代码}。如function Mycomponent(){return <h2>我是简单组件</h2>}
4. ES6类组件（复杂组件）：class MyComponent2 extents React.Component{render(){return html代码}}。
5. 渲染组件标签(无论简单还是复杂组件都是用这同一种方式渲染)：ReactDOM.render(<MyComponent />,document.getelementById(“example1”));
6. < />和<></ >是一样的
7. Constructor(props){super(props)}：props（外部接口），state（内部）
8. 组件的参数props ：props 是组件对外的接口，组件内所有参数都存放在this.props。props 作为父子组件沟通的桥梁，为组件的通信和传值提供了重要手段。
9. 组件的状态state：state的固定格式为Constructor(props){super(props)；this.state={......}}
10. export可以导出多个，而export default只能导出一个
11. setState()：更新/设置状态
12. 新添加的方法里的this默认不是组件对象，而是undefined，render里的this是组件对象 。解决方法：用bind将this绑定为新添加的方法。如 this.xxx=this.xxx.bind(this)；
13. 默认属性：defaultprops
14. ref（refs）：（句柄）标识符，用于操作dom，有点类似与id和class。语法：ref=”xxx”;调用：const ab=this.refs.xxx;注意调用的时候是refs，因为ref是一个数组。Refs 是 React 提供给我们的安全访问 DOM 元素或者某个组件实例的句柄
15. propTypes:react中用来对参数进行类型检测的。要调用import proptype from ‘prop-type’
16. unshift():将新项添加到数组起始位置
17. <row>和<col>
18. export default connect：将组件component和models结合在一起。将models中的state（状态）绑定到组件的props（属性）中
19. Import和require都是导入
20. let命令，用来声明变量。它的用法类似于var，但是所声明的变量，只在let命令所在的代码块内有效
21. 自定义函数组件继承了react Component这个基类（也就继承这个react的基类），才能有render()、生命周期等方法可以使用，因此其他自定义函数组件不继承react就不能使用这些方法。
22. 类的构造方法construction()：super(props)用来调用基类（父类）的构造方法，，也将基类组件的props注入给子组件，给子组件读取（组件中props只读不可变，state可变），而constructor()用来做一些组件的初始化工作，如定义this.state的初始内容。

组件的挂载（Mounting）阶段：1componentWillMount：在组件卸载前自动调用，且只会被调用一次，在这边调用this.setState不会引起组件重新渲染，也可以把写在这边的内容提前到constructor()中，所以项目中很少用。2render：根据组件的props和state（无两者的重传递和重赋值，论值是否有变化，都可以引起组件重新render） ，return 一个React元素（描述组件，即UI），不负责组件实际渲染工作，之后由React自身根据此元素去渲染出页面DOM。render是纯函数（Pure function：函数的返回结果只依赖于它的参数；函数执行过程里面没有副作用），不能在里面执行this.setState，会有改变组件状态的副作用。3componentDidMount：组件挂载到DOM后调用，且只会被调用一次

1. 组件的更新（update）阶段：componentDidUpdate()，会在 UI 每次更新后调用（即组件挂载成功以后，每次调用 render 方法，都会触发这个方法）。
2. getAttribute：获取某个属性的值
3. setAttribute：建立一个属性，并同时给属性捆绑一个值
4. localStorage：用来作为本地存储来使用的，解决了cookie存储空间不足的问题(cookie中每条cookie的存储空间为4k)，localStorage中一般浏览器支持的是5M大小。1localStorage.setItem("key", "value")：保存数据。2var lastname = localStorage.getItem("key")：读取数据。3localStorage.removeItem("key")：删除数据
5. prop.children是一个数组
6. 路由配置：在点击导航选项的时候，让对应内容填充的到页面，实现这种效果的方式就是路由。简单来说，路由就是用来跟后端服务器进行交互的一种方式，通过不同的路径来请求不同的资源。
7. 页面地址的跳转都是在浏览器端实现的，不会去重新请求服务端获取 html。html 只是在应用初始化的时候加载一次。所有的页面都是由不同的组件构成，页面的切换其实就是不同组件的切换，你只需要在配置中把不同的路由路径和对应的组件关联上即可。
8. Dva：通过把状态上提到 dva model 中，我们把数据逻辑从页面中抽离出来。dva model 中的数据可以注入给任意组件。另外，dva 允许把数据逻辑再拆分（「页面」常常就是分隔的标志），以 namespace 区分。当你觉得有必要时，不同的 namespace 之间的 state 是可以互相访问的。一个基本的 dva model 最少具备两个成员：namespace 和 state。namespace 来作为一个 model 的唯一标识，state 中就是该 model 管理的数据
9. @connect(mapStateToProps)。connect 是连接 dva 和 React 两个平行世界的关键，一定要理解。connect 让组件获取到两样东西：1. model 中的数据；2. 驱动 model 改变的方法。connect 本质上只是一个 javascript 函数，通过 @ 装饰器语法使用，放置在组件定义的上方；connect 既然是函数，就可以接受入参，第一个入参是最常用的，它需要是一个函数，我们习惯给它命名叫做 mapStateToProps，顾名思义就是把 dva model 中的 state 通过组件的 props 注入给组件。通过实现这个函数，我们就能实现把 dva model 的 state 注入给组件。
10. dva model 中可以定义一个叫做 reducers 的成员用来响应 action 并修改 state。每一个 reducer 都是一个 function，action 派发后，通过 action.type 被唯一地匹配到，随后执行函数体逻辑，返回值被 dva 使用作为新的 state。state 的改变随后会被 connect 注入到组件中，触发视图改变。reducer 是一个 "纯函数"，它的返回值作为新的 state。dva 会注入旧的 state 和 action 中的 payload，是否使用完全根据需要决定；返回值必须是一个新构造对象，绝不能把旧 state 的引用返回
11. action 被 dispatch 之后就能够 直接 到达 reducer
12. effect 充当了这么一个中间层，当 action 被 dispatch 之后，会先到达 effect 处理副作用，然后该 effect 最终会促使新的 action 发送出去，这个新的 action 可能被其他的 effect 再捕获继续处理，也可能被 reducer 捕获并结束，无论怎样，最终处理逻辑的终点都将是 reducer。
13. 第三章的搭建基于model的卡片列表页面、在抹model中请求服务端数据重新看过！
14. Less的global：允许我们声明一个 class 名称不可被改写。如:global(.ant-btn) { // ...}
15. React声明class类名：className={xxx.xxx}。如果要覆盖原本的样式则要用中括号：className={xxx[‘xxx’]};
16. umi 的话，有一个[专门的文件](https://umijs.org/guide/app-structure.html" \l "%E5%A4%8D%E6%9D%82%E5%BA%94%E7%94%A8" \t "_blank) global.less 来让我们书写全局样式。直接在 global.less内写样式，外部不用调用这个样式，可直接工作
17. 上传与下载
18. 上传：1使用fetch上传，2使用antd的Upload组件上传
19. 下载：1使用a标签下载，2使用fetch实现文件下载
20. 生命周期：构造器（constructor）、装载期间（Mounting）、渲染（render）、更新期间（Updating）、卸载期间（Unmounting）
21. 组件初始化期间：constructor构造器
22. Mounting期间：1、constructor(props)：构造函数，用于初始化这个组件的一些状态和操作，如果你是通过继承React.Component子类来创建React的组件的，那么你应当首先调用super(props) 初始化父类。在contructor函数中，你可以初始化state，比如this.state = {xxx};，不要在构造函数中使用setState()函数，强行使用的话React会报错。其次你可以在构造函数中进行函数bind。2、componentWillMount()：将要渲染到虚拟dom。3、render()：渲染到虚拟dom，用于根据状态state和属性props渲染一个React组件。我们应该保持该方法的纯洁性，这会让我们的组件更易于理解，只要state和props不变，每次调用render返回的结果应当相同，所以请不要在render方法中改变组件状态，也不要在在这个方法中和浏览器直接交互。4、componentDidMount()（只执行一次）：componentDidMount方法会在render方法之后立即被调用，该方法在整个React生命周期中只会被调用一次。React的组件树是一个树形结构，此时你可以认为这个组件以及他下面的所有子组件都已经渲染完了，所以在这个方法中你可以调用和真实DOM相关的操作了。
23. Updating期间（可执行n次）：1、omponentWillUpdate()（可执行n次）：将要更新渲染。2、render()（可执行n次）：重新更新渲染。3、componentDidUpdate()（可执行n次）：该方法会在更新完成后被立即调用，你可以在这个方法中进行DOM操作，或者做一些异步调用。这个和首次装载过程后调用componentDidMount是类似的，不一样的是你可能需要判断下属性是否变化了再发起网络请求。
24. Unmounting期间：卸载期间是指组件被从DOM树中移除。componentWillUnmount()（只执行一次）：该方法会在组件被卸载之前被调用，你可以在这个函数中进行相关清理工作，比如删除定时器。
25. 错误捕获：componentDidCatch(error, info)
26. 常用到的声明周期函数：constructor: 初始化状态，进行函数绑定

componentDidMount: 进行DOM操作，进行异步调用初始化页面

coponentWillReceiveProps: 根据props更新状态

componentWillUnmount: 清理组件定时器，网络请求或者相关订阅等

* 1. 单元测试：可以使用umi中的jest()

1. 受控组件：表单项输入数据能自动收集成状态
2. 非受控组件：需要时才手动读取表单输入框中的数据
3. 非受控组件操作dom（react不推荐操作dom，但是代码简洁）：语法：ref={xxx=>this.aaa=xxx};调用时：const abc=this.aaa;
4. 非受控组件改变状态state（react推荐使用这个，但是代码太长）：思路：先在constructor()中声明状态，然后在组件中通过onchange()函数来调用某个函数，而该函数会改变状态，最后setstate()。
5. ref(refs)和受控组件、非受控组件：refs和受控组件类似，都是使用ref来操作dom，而非受控组件是改变状态
6. 定时器：setInterval与setTimeout的区别：setinterval可以循环调用多次，而settimeout是能调用一次
7. 如果要使用constructor里的某个属性的状态state：1要么直接调用:this.state.xxx。2如果不能用this.state则要先定义变量获取属性状态，再使用：const{abc}=this.state;abc=xxxxx;
8. 避免箭头函数和绑定bind(this)一起使用，会报错，解决方法：1将箭头函数改成function，然后function配合bind使用，如function(){...}.bind(this)。2直接使用箭头函数，不需要再bind如：()=>{.....}（推荐使用箭头函数）
9. Componentwillreceiveprops(nextprops)：当组件接受到新的属性时回调
10. 组件间的通信：1、通过props传递，2使用消息订阅（subscribe）-发布（publish）机制
11. 通过props传递：1、共同的数据放在父组件上，特有的数据放在自己组价内部（state），2、通过props可以传递一般数据和函数数据，只能一层一层传递：一般数据：父组件传递数据给子组件，子组件再读取数据。函数数据：子组件传递数据给父组件，子组件再调用函数
12. 使用消息订阅（subscribe）-发布（publish）机制：工具库：pubsubjs，下载cnpm install pubsub-js --save-dev，然后引用
13. 从29-35的路由没看！！
14. （UI）组件库：antd-design
15. Redux：一个独立专门用于做状态管理的js库（不是react插件库），作用：管理react应用中多个组件共享的状态（state）
16. Redux的三大核心：1、state，2action，3reducer（三大核心围绕state）
17. Redux的state：唯一改变state的方法时触发action。action是一个用于描述已发生事件的普通对象。Action把数据从应用传到store载体，它是store数据的唯一来源。一般来说，我们可以通过store.dispatch()将action传递给store。Action对象内部必须有一个type属性来表示要执行的动作，多数情况下type会被定义成字符串常量。Reducer：reducer是一个函数，用来响应发送过来的action，经过处理后把state发送给store。注意：在reducer函数中，需要return一个返回值，这样store才能接受接收到数据。Reducer会接受到两个参数，一个是初始化state，第二个参数是action。
18. Store就是把action和reducer联系到一起的对象。提供的几个方法：1getstate()：获取state，2dispatch()：发送action，3subscribe()：注册监听，4通过subscribe()返回值来注销监听
19. Redux固定模式小结：1构建action.jsx，通过创建一个函数然后返回一个对象，注意需要携带type属性。2构建reducer，用来响应action，然后通过return把数据传回给store。3利用createStore来构建store，构建的时候传递我们写好的reducer。4利用store.subscribe()注册监听。5我们利用store.dispatch()发送一个action的时候就能出发我们的监听了，在里面利用store.gerState()就能拿到值
20. react-redux：react的一个插件库，用于降低react和store的耦合度
21. react-redux两个重要的组成部分：provider组件、connect组件。Provider组件用于包裹整个结构，让组件能够获取到store。如果我们需要接受provider提供的store，就需要把我们组件通过connect加强

DVA:dva是一个基于redux和redux-saga的数据流方案，为了简化开发体验，dva还额外内置了react-router和fetch，所以也可以理解为一个轻量级的应用框架

1. 路由跳转：1引用dva/router中的link。2引用dva/router中的push方法。3引用dva/router中的routerRedux的push（dva/router中提供有push跳转方式）如：this.props.dispatch(routerRedux.push("www......"))
2. 异步操作：async
3. Dva mock（数据处理）
4. Dva网络请求
5. this.props.children:表示该组件的所有子节点（子节点就是子组件）
6. Class组件和函数组件的区别：1、使用class创建组件，有自己的私有数据（this.state）和声明周期。2、使用function创建的函数组件，只有props没有自己的私有数据和生命周期函数。3、用构造函数创建的组件叫无状态组件，用class创建的组件叫做有状态组件。4、组件中的pros和state/data之间的区别：1props的数据都是从外界传递过来的。2state/data中的数据都是组件私有的。3props中的数据都是只读的，不能重新赋值的。4state/data中的数据都是可读可写的。
7. 函数组件性能比类组件高，但是函数组件没有生命周期（因为没有继承react.component）,生命周期函数是React.Component类的方法实现

面试准备题：

Q1:constructor中super与props参数一起使用的目的是什么?答：在子类的 constructor 中必须先调用 super 才能引用 this 。在 constructor 中可以使用 this.props

Q2：非受控组件，即组件的状态不受React控制的组件，受控组件就是组件的状态受React控制。

Q3:为什么类方法需要绑定（bind）？答：this 的值取决于当前上下文。在React类的组件方法中，开发人员通常希望它引用组件的当前实例，因此有必要 将 这些方法 绑定 到该实例。通常，这是在构造函数中完成。

Q4:展示组件(Presentational component)(也叫 UI 组件)和容器组件(Container component)（负责应用逻辑处理）之间有何不同？答：UI组件：负责组件如何渲染，只关心的到数据后如何渲染。容器组件：负责应用逻辑处理，发送网络请求，处理返回数据，将处理过的数据传递给展示组件；也提供修改数据源的方法，通过展示组件的props传递给展示组件

Q5:高阶组件（HOC）：高阶组件其实就是一个函数而已，只不过参数是一个组件，返回了一个新的组件。复用组件的业务逻辑react-redux connect其实就是一个高阶组件。HOC是一个纯函数（纯函数没有副作用，输入确定，输出就一定确定）

Q6:React的优缺点：优：对虚拟dom的操作，使得速度快，提高了web性能；组件化、模块化，react里每一个模块都是一个组件，组件化开发可维护性高；跨浏览器兼容：虚拟DOM帮助我们解决了跨浏览器问题，它为我们提供了标准化的 API，甚至在 IE8 中都是没问题的。缺点：要依赖引入很多其他模块开发；当父组件进行重新渲染操作时，即使子组件的props或state没有做出任何改变，也会同样进行重新渲染