CSS：

1.Border-box把border和padding都纳入总的大小之中（没有margin）IE盒子模型

2.position：absolute会把元素变成块级元素，换行

3.按钮的cursor要设置为pointer

4.transform-style：preserve-3D会保留和呈现子元素的3D位置

5.transform-origin是设置旋转的基点

6.perspective是视距，决定用户与Z轴之间的距离，perspective-origin决定视线的左右和上下位置

7.@font-face自定义自己的字体

IE浏览器：EOT Mozilla浏览器：OTF，TTF  
Safari浏览器：OTF，TTF，SVG  
Chrome浏览器：TTF，SVG

1. -webkit-:chrome safari -moz-:firefox -ms-:IE -o-:opera 兼容按这个顺序最后在写标准的CSS属性
2. cubic-bezier(x1,y1,x2,y2)用于animation自定义自己的速度曲线（cubic-bezier(.81, -0.62, 0.23, 1.92)实现蓄力回弹的速度曲线）
3. animation-fill-mode控制元素在动画执行前和执行后的样式
4. translate（100%）相对元素自身的宽高在X轴和Y轴上面移动
5. .icon::before{content:’\f015’;font-family:FontAwesome;}在CSS中直接使用fontawesome的图标
6. img{width:100% height:100%}可以将img整个放入div中
7. box-shadow:0px 0px 1px rgba(0,0,0,1)可以使得阴影四周分布
8. opacity会使内容也透明，rgba可以解决这个问题
9. Localstorage只存储string
10. position: fixed;left: 50%;top: 50%;transform: translate(-50%, -50%);可以宽高自适应居中
11. $("#zkdiv").on("click",".zk",function(){ console.log("on 点击一次"); });解决后append的元素没有监听事件
12. Font: 12px/1.5中的12px是字体大小，1.5\*12px是行高
13. :nth-child(even)偶数子元素，nth-child(odd)奇数子元素
14. ::selection可以自定被选中文字的样式
15. div::before {content:"";position: absolute;border-style: solid;left: -15px;border-width: 8px 16px 8px 0;border-color: transparent red transparent transparent;}在div前面画出小三角形
16. getBoundingClientRect用于获取某个元素相对于视窗的位置集合。集合中有top, right, bottom, left等属性。
17. bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y):添加(cp1x,cp1y)、(cp2x,cp2y)两个控制点使得曲线弯曲
18. <Link>标签的样式受<a>标签影响
19. 父级元素设置了transform会导致子级元素的fixed失效
20. JQ的animate()是不支持backgrColor这些样式的，得使用JQ-UI插件

24.获取窗口宽度w.width = window.innerWidth || document.documentElement.clientWidth || document.body.clientWidth;（兼容）

JS:

1.if(arry)不能判断数组为空，必须用if(array === undefined || array.length == 0)

2.$(‘#my-canvas’)在jq中拿不到canvas元素，要$('#my-canvas').get(0)

3.jq中不能使用isrequired属性

React:

1.this.setState：this.state通常是用来初始化state的，this.setstate是用来修改state值的。如果你初始化了state之后再使用this.state，之前的state会被覆盖掉，如果使用this.setState,只会替换掉相应的state值

2.所谓的JSX其实就是JS对象，JSX经过编译（Babel编译+React.js构造）编程JS对象，在经过ReactDom.render变成DOM元素然后插入页面。不直接从 JSX 直接渲染构造 DOM 结构，第一个原因是，当我们拿到一个表示 UI 的结构和信息的对象以后，不一定会把元素渲染到浏览器的普通页面上，我们有可能把这个结构渲染到 canvas 上，或者是手机 App 上。所以这也是为什么会要把react-dom单独抽离出来的原因。第二个原因是，有了这样一个对象。当数据变化，需要更新组件的时候，就可以用比较快的算法操作这个 JavaScript 对象，而不用直接操作页面上的 DOM，这样可以尽量少的减少浏览器重排，极大地优化性能。

3.render方法要返回一个JSX元素，并且必须要用一个外层JSX元素包裹所有内容。

4.用{}在JSX中插入任何JS代码（也可以插在标签内部如标签的属性上），包括变量、表达式计算、函数执行以及JSX等等。Render方法会把这些代码返回的内容如实地渲染到页面上。

5.用className代替标签中的class属性，用htmlFor代替label标签中的for属性。

6.自定义组件都必须使用大写字母开头，普通html标签使用小写字母。

7.一个组件的显示形态是可以由它数据状态和配置参数决定的。

8.事件监听：React.js 帮我们封装好了一系列的 on\* 的属性，当你需要为某个元素监听某个事件的时候，只需要简单地给它加上 on\* 就可以了（**这些 on\* 的事件监听只能用在普通的 HTML 的标签上，而不能用在组件标签上**）。而且你不需要考虑不同浏览器兼容性的问题，React.js 都帮我们封装好这些细节了。要注意的是，这些事件属性名都必须要用驼峰命名法。

9.event对象：事件监听函数会被自动传入一个 event 对象，但React.js 中的 event 对象并不是浏览器提供的，而是它自己内部所构建的。React.js 将浏览器原生的 event 对象封装了一下，对外提供统一的 API 和属性，这样你就不用考虑不同浏览器的兼容性问题。

10.事件this：手动地将实例方法bind(this)到当前实例上再传入给 React.js，就可以在事件函数当中使用当前的实例。

11.setState()：**它接受一个对象或者函数作为参数，**当我们调用这个函数的时候，React.js 会更新组件的状态state，并且重新调用render方法，然后再把render方法所渲染的最新的内容显示到页面上。（不能直接用 this.state = xxx 这种方式来修改）

12.setState(fn(prevState))：当你调用 setState 的时候，**React.js 并不会马上修改 state**。而是把这个对象放到一个更新队列里面，稍后才会从队列当中把新的状态提取出来合并到 state 当中，然后再触发组件更新。解决这问题就是在setState中传入函数，React.js 会把上**一个 setState 的结果**传入这个函数，你就可以使用该结果进行运算、操作，然后返回一个对象作为更新 state 的对象.

13.props:组件的props参数让组件能适应不同场景下的需求让组件具有一定的“可配置”性，每个组件都可以接受一个 props参数，它是一个对象，包含了所有你对这个组件的配置。**在使用一个组件的时候，可以把参数放在标签的属性当中，所有的属性都会作为 props 对象的键值。（props是不可修改的，避免**这个组件显示形态和行为变得不可预测，但是可以通过父组件主动重新渲染的方式来传入新的 props，从而达到更新的效果。**）**

14.defaultProps：在组件上添加static defaultProps={}（格式是固定的）就可以为组件设置默认的props，如果props没有传进来，会直接使用 defaultProps 中的默认属性，可以避免手动去判断props是否存在。

15.state VS props：**state 是让组件控制自己的状态，props 是让外部对组件自己进行配置**。使用原则：尽量少地用 state，尽量多地用 props。

16.无状态组件：没有 state 的组件叫无状态组件（stateless component），因为状态会带来管理的复杂性，所有尽量多地写无状态组件，尽量少地写有状态的组件。这样会降低代码维护的难度，也会在一定程度上增强组件的可复用性。

17.函数式组件：通常一个组件是通过继承 Component 来构建，一个子类就是一个组件。而用函数式的组件编写方式是一个函数就是一个组件，也通过 <HellWorld /> 使用该组件。但是函数式组件只能接受 props 而无法像跟类组件一样可以在 constructor 里面初始化 state。你可以理解函数式组件就是一种只能接受 props和提供 render 方法的类组件。（没有render，只会return JSX）

18.渲染JSX数组：React.js 把插入表达式数组里面的每一个 JSX 元素一个个罗列下来，渲染到页面上。也就是说**如果你往 {} 放一个数组，React.js 会帮你把数组里面一个个元素罗列并且渲染出来（**在 React.js 用 map 来处理、渲染数据列表）。

19.Key属性：**对于在数组中将要罗列渲染到页面上的元素，都要为每个元素加上 key 属性（** React.js 通过 key 来标识元素，从而尽量复用元素内部的结构，减少DOM操作**），这个 key 必须是每个元素唯一的标识（例如**循环计数器 i ） 。

20.受控组件：类似于 <input />、<select />、<textarea> 这些元素的 value 值被 React.js 所控制、渲染的组件，在 React.js 当中被称为受控组件（Controlled Component），需要设计事件监听来响应。

21.向父组件传数据：父组件只需要通过 props 给子组件传入一个回调函数。当子组件的点击事件发生的时候，子组件调用 props 中的回调函数并且将 state 传入该函数即可（组件之间就使用 props 通过父元素来传递数据）。

22.状态提升：当某个状态被多个组件**依赖**或者**影响**的时候，就把该状态提升到这些组件的最近公共父组件中去管理，用 props 传递**数据或者函数**来管理这种**依赖**或着**影响**的行为。

23.挂载：**React.js 将组件渲染，并且构造 DOM 元素然后塞入页面的过程称为组件的挂载，即**初始化组件 -> 挂载

24.生命周期：

挂载阶段：

* componentWillMount：组件挂载开始之前，也就是在组件调用 render 方法之前调用。
* componentDidMount：组件挂载完成以后，也就是 DOM 元素已经插入页面后调用。
* componentWillUnmount：组件对应的 DOM 元素从页面中删除之前调用。

利用生命周期函数把组件的 state 的初始化工作放在 constructor 里面去做；在 componentWillMount 进行组件的启动工作，例如 Ajax 数据拉取、定时器的启动；组件从页面上销毁的时候，有时候需要一些数据的清理，例如定时器的清理，就会放在 componentWillUnmount 里面去做。

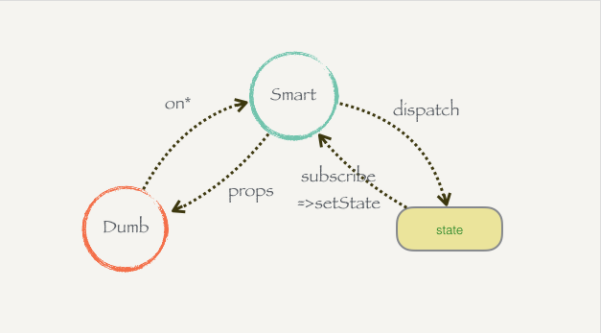
更新阶段：

1. shouldComponentUpdate(nextProps, nextState)：这个方法控制组件是否重新渲染。如果返回 false 组件就不会重新渲染。这个生命周期在 React.js 性能优化上非常有用。
2. componentWillReceiveProps(nextProps)：组件从父组件接收到新的 props 之前调用。
3. componentWillUpdate()：组件开始重新渲染之前调用。
4. componentDidUpdate()：组件重新渲染并且把更改变更到真实的 DOM 以后调用。
5. Ref属性： 用来帮助我们获取已经挂载的元素的 DOM 节点（例如<input ref={(input) => this.input = input} />），然后我们就可以在 componentDidMount 中使用这个 DOM 元素，并且调用该元素的 DOM API，但是**能不用 ref 就不用**。
6. dangerouslySetInnerHTML属性：出于安全考虑的原因（XSS 攻击），在 React.js 当中所有的表达式插入的任何 HTML 格式都会被转义掉，可以慎重使用dangerouslySetInnerHTML={{\_\_html: this.state.content}}动态渲染元素的 innerHTML 结构
7. Style属性：style 接受一个对象，这个对象里面是这个元素的 CSS 属性键值对，原来 CSS 属性中带 - 的元素都必须要去掉 - 换成驼峰命名，如 font-size 换成 fontSize，text-align 换成 textAlign，可以使用props 或者 state中的数据生成样式对象再传给元素。
8. propTypes库：通过 PropTypes 给组件的参数做类型限制，可以在帮助我们迅速定位错误，这在构建大型应用程序的时候特别有用；另外，给组件加上 propTypes，也让组件的开发、使用更加规范清晰（例如static propTypes = { comment: PropTypes.object.isRequired }）。
9. 组件的命名和方法的摆放顺序:

组件的私有方法都用 \_ 开头，所有事件监听的方法都用 handle 开头。把事件监听方法传给组件的时候，属性名用 on 开头。

组件的内容编写顺序：

1. static 开头的类属性，如 defaultProps、propTypes。
2. 构造函数，constructor。
3. getter/setter（还不了解的同学可以暂时忽略）。
4. 组件生命周期。
5. \_ 开头的私有方法。
6. 事件监听方法，handle\*。
7. render\*开头的方法，有时候 render() 方法里面的内容会分开到不同函数里面进行，这些函数都以 render\* 开头。
8. render() 方法。
9. 新的组件使用传入的组件作为子组件，**它的作用是用于代码复用**，可以把组件之间可复用的代码、逻辑抽离到高阶组件当中（**新的组件和传入的组件通过 props 传递信息**）。
10. Reducer编写顺序：action types =>reducer=>相关action creators
11. React-redux的类MVC结构



34.constructor(props){super(props)...}：在构造函数中需要用到props中的值时就需要传入props

Redux:

Store：保存数据的容器,createStore传入reducer和最初状态

State：包含所有数据的对象（getState()可以拿到值）

Action：View发出的通知（type属性和payload属性）

Dispatch（调度）：View发出Action的方法（会触发Reducer自动执行）

Reducer：作为state的处理过程，接受action和当前state作为参数，返回新的state（必须是纯函数）

Subscribe（订阅）：设置监听函数，当state改变时执行这个函数（返回函数可解除监听）

Middleware：对Dispatch进行重定义，在发出Action和执行Reducer这两步之间添加其他功能

applyMiddlewares：将所有中间件组成一个数组，增强Dispatch（logger要放最后）

Redux-thunk：使得dispatch可以接受函数为参数

Redux-promise：使得dispatch可以接受Promise对象为参数

异步：发出多种Action，改造State以反映不同的操作状态，用Action Creator来实现自动发送第二个Action，然后用redux-thunk使得dispatch可以接受函数为参数或用Redux-promise使得dispatch可以接受Promise对象为参数

流程：



React-Redux：

组件：将所有组件分为UI组件（负责UI的呈现）和容器组件（负责管理数据和逻辑）

Connect：接受mapStateToProps和mapDispatchToProps为参数，从UI组件生成容器组件

mapStateToProps：以state为参数，返回含有同名属性的对象，建立UI组件的props对象与store的State对象到的映射关系，在每次store的state发生变化的时候，都会被调用

mapDispatchToProps：以dispatch为参数，返回含有同名函数属性的对象，建立UI组件与store的dispatch的映射关系，定义UI组件如何发出Action

<provider>：让容器组件可以拿到state

Demo：[https://github.com/jackieli/simplest-redux-example/blob/master/index.js](https://github.com/jackielii/simplest-redux-example/blob/master/index.js)

React-Router：

<Router>：作为router组件的容器，拥有history属性（值hashHistory、browserHistory、createMemoryHistory）决定监听浏览器地址变化，解析URL为一个地址对象（browserHistory需要在package.json添加--history-api-fallback）

<Route>：path指定路由的匹配规则（两个规则，第二个规则不会生效），component指定在该地址下加载的组件

{this.props.children}:组件的所有子组件，若存在则加载

通配符：

1. :paramNamep匹配URL的一个部分，直到遇到/?#（this.props.params.paramName可以取出这个值）
2. ()标识URL的这部分可选
3. \*匹配任意字符，直到下一个字符
4. \*\*匹配任意字符，直到下一个/?#

<IndexRoute>：显式指定根路由的子组件，默认情况下加载的子组件（替代{this.props.children || <Home/>）

<Redirect>：用于路由的跳转，访问到一个路由会自动跳到另一个路由

<Link>：React版的<a>元素，生成一个连接，在用户点击后跳转到另一个路由，activeStyle属性设置:active的颜色值，activeClassName则指定当前路由的Class

<IndexLink>：链接要根路由不要使用<Link>（硬要使用则指定onlyActiveOnIdex={true}属性），要使用<IndexLink activeClassName=”active”>来避免链接一直处于active而带有activeStyle或activeClassName的颜色

表单处理：在handleSubmit()函数中实现跳转，第一种方法是browserHistory.push(path)，第二种方法是使用context对象，设置contextTypes:{router:React.propTypes.object}然后this.context.router.push(path)

路由钩子：每个路由都有Enter和Leave两个钩子，使用这两个钩子第一种方法是在onEnter和onLeave中指定一个函数，另一种方法是用withRouter(<App/>)（withRouter要从’react-router’导入）对组件进行“升级”，此时this.props.router有一个setRouterLeaveHooK(this.props.routes,function(){})方法来控制路由跳转，第一个参数是要控制的路由（this.props.routes很有可能是数组），第二个参数是一个函数（返回false不跳转，true直接跳转，返回字符串则浏览器弹出该字符串的confirm，确认跳转，取消不跳转）

<Switch>:只找到第一个path被匹配的<Route>就立即停止继续匹配，并且只把第一个配到的<Route>渲染出来，带有exact属性的<Route>即使先被匹配到，path属性也必须完全和location相同才会渲染

ES6:

1. let声明的变量只在所在的代码块（块级作用域）里有效，形成TDZ，只能从声明开始之后的地方使用
2. Const 声明的变量同样只在块级作用域中有效，但是值无法改变且必须立即初始化。Const存复合类型只是所存储的地址不变
3. Let、const、class声明的变量不在属于全局对象（window）的属性
4. 解构赋值：从数组或对象中获取数据的方式更方便简单。可以设置默认值（严格为undefined才取默认值）；解构对象必须同名（使用模式）；已声明的变量进行解构必须用大括号包裹；交换数值、指定函数参数默认值、提取JSON数据很便捷
5. 字符Unicode表示法：\uxxxx；四字节字符使用codePointAt()、fromcodePoint()和at()方式，for(...of...)方法可以遍历
6. 字符串搜索：indexOf()、includes()、 startsWith()、endsWith();字符串重复：repeat();字符串补全:padStart()、padEnd()
7. 模板字符串：`<h1>hello ${str}!</h1>`，空格缩进会被保留，大括号中可以存变量、表达式、执行函数，<%...%>放置JS代码、<%=...%>输出JS表达式
8. 标签模板：过滤HTML字符串、多语言转换、嵌入其他语言，String.raw()对反斜杠转义\转义成\\
9. 正则表达式用u修饰符处理四字节UTF-16编码，大括号可以表示Unicode字符，但是也必须使用u修饰符
10. /^[a-z]$/只匹配a-z字母中的一个(^匹配开头 &匹配结尾)
11. y修饰符：后一次匹配从上一次匹配成功的下一个位置开始，相比于g修饰符隐含一个^头部匹配，有y修饰符就有sticky属性
12. 0b二进制简写，0o八进制简写
13. 传统的isFinit()和isNaN(n)会对n执行Number()函数，而Number.isFinite()和Number.isNaN()不会
14. JS数值存储为64位双精度格式，如果数值精度超过53位二进制，第54位后面的位会被抛弃，存在小误差（最小精度：Numer.EPSILON=pow（2，-52），上限：Number.MAX\_SAFE\_INTEGER=pow(2,53)-1,下限：Number.MAX\_SAFE\_INTEGER=-pow(2,53)+1,）
15. 指数运算符\*\*：2\*\*4=32
16. 默认参数：ES6中可以自定义参数Function f(x, y=1)，当参数的值为undefined时取得默认参数，默认参数应该尽量放在参数尾部（设置的默认参数的函数，参数自己形成块级作用域，外部是全局作用域）
17. length属性：函数的length属性只放回设置了默认参数的参数前面的没有设置默认参数的个数
18. rest参数：f(item1，...items)将item1之后多余的参数放入items数组中（rest必须是最后一个参数）
19. 如果函数使用了默认值、解构赋值或者拓展运算符就在不能再函数内部定义严格模式
20. 箭头函数没有自己的this，所以它会“借用”自己所在的作用域的this，从而使得在箭头函数中this的指向是固定的
21. 函数绑定运算符：::双冒号运算符可以代替call、apply、bind来显示绑定this对象
22. 尾调用优化：在函数的内部调用函数会不停地堆起调用栈，消耗内存，但是如果把外层函数调用放在一个函数的最后一步，不再使用外层函数的信息，则外层函数的信息不再需要保存，内存被释放，在尾调递归能很好地优化递归（尾调只在严格模式下生效）
23. 拓展运算符：将一个数组转为用逗号分隔的参数序列，能够替代函数的apply（传入this和数组），可以用来复制数组、合并数组（浅复制）、与解构复制结合、转字符串为数组
24. Array.form：将类似数组的对象（拓展运算符做不到）和可遍历对象（具有Iterator接口）转为真正的数组，第二个参数类似于map()函数
25. Array.of：总是返回参数值组成的数组
26. copyWithin:copyWith(a,b,c)将从b位置到c位置的这（c-b）个数字，从位置a开始覆盖过去
27. find(callback):返回回调函数的返回值为true的第一个数组的值
28. findIndex(n):返回数组第一个值为n的值的index，没有则返回-1
29. Entries()、keys()、values()和for...of结合分别遍历数组的键值对[index,elem]、键index和值elem
30. includes(n,s)：如果数值中包含值n则返回true，第二个参数设置搜索的起始位置（应该取代掉indexOf(n) === -1的检查方法）
31. [,,]：ES6将空位设为undefined，ES5和ES6对空位的处理很一致
32. 属性简写：在对象中，变量名以字符串的形式作为属性名，变量的值作为属性值，方法简写fn(){}
33. 属性名表达式：{[‘a’+’b’]:123}ES6可以在字面量中使用属性名表达式（[obj]会被转为[Object Object]）
34. Object.is()：相比于===，+0不等于-0，NaN等于NaN Object.getOwnPropertyDescriptor
35. Object.assign()：将源对象的可枚举属性复制到目标对象上，这种复制的浅复制，且只对对象和字符串有效（字符串转为了对象），同名属性取最后一个覆盖前面的值，复制函数则求值后复制。可用于为对象添加属性和方法、克隆和合并对象、为属性指定默认值
36. 可枚举性：Object.getOwnPropertyDescriptors（配合Object.defineProperties解决Object.assign不能拷贝get/set属性的问题）方法可以获取该属性的描述对象，描述对象有vaule、writable、enumerable（为false这该方法不可枚举）及configurable四个属性，目的是为避免toString()和length属性被for...in遍历到
37. \_\_proto\_\_：用来读取和设置当前对象的原型对象，但作为内部属性，应该避免使用这个属性
38. setPrototypeOf()：用来设置原型对象，getPrototypeOf()用来获取原型对象（如果参数不是对会被自动转为对象），create用来生成原型对象
39. Super()关键字：this在函数中指向当前对象，super则指向原型对象（只能在对象的简写方法中访问）
40. Object.keys(obj)、Object.values(obj)、Object.entries(obj)、分别返回对象的属性名数组、属性值数组和属性数组（非Symbol可枚举属性）
41. 对象也可以使用拓展运算符，用于取出参数对象的可遍历属性，拷贝到当前对象中（拓展运算符后的同名属性会覆盖前面的属性值）。对象的解构赋值是浅复制且不会拷贝原型对象的属性
42. Symbol:JS的第七种数据类型，由Symbol(d)函数生成的值是独一无二的,字符串参数d是Symbol实例的描述（打印时会显示，用于区分出Symbol）
43. Symbol值作为对象属性名必须包裹在[]中，且访问时不能使用点运算符
44. Symbol值不会被常规方法遍历到，要遍历可以使用getOwnPropertySymbols和Reflect.ownKyes方法（所以Symbol很适合为对象定义非似有但希望只用于内部的方法）
45. Symbol.for(key)会登记Symbol值，每次调用如果key存在则返回相同飞Symbol值
46. ES6提供了hasInstance、match、iterator、toStringTag等11个默认的Symbol值
47. Set：类似于数组，但是成员都是唯一的，没有重复值（[...new Set(array)]可以出去数组array中重复的值）
48. Set实例有constructor和size两个属性，add、delete、has和clear四个操作方法，keys、values、entries和forEach四个遍历方法（在Set实例中key和value是一样的，且遍历顺序就是插入顺序），Set中也可以使用数组的map和filter方法
49. WeakSet：与Set类似，但成员只能是对象且WeakSet中的对象都是弱引用，不被垃圾回收机制考虑，没有内存泄漏的烦恼，且WeakSet没有size属性，不能遍历
50. Map提供了“值-值”的hash结构，不像Object只能以字符串作为键值，Map中任何值都可以作为键值
51. Map有size属性，set(key,value)方法添加键值对，以及get(key)、has(key)、delete(key)及clear()操作方法，keys、values、entries和forEach四个遍历方法（遍历顺序就是插入顺序）,在将Map用拓展运算符转化为数组之后可以使用map、filter方法
52. WeakMap：键值只能是对象，与WeakSet一样，对象都是弱引用，不被垃圾回收机制考虑，没有内存泄漏的烦恼，同时没有size属性，不能遍历