1.什么是React，谈谈你对React 的看法

- 用来构建UI的 JavaScript库
- React 不是一个 MVC 框架，仅仅是视图（V）层的库

2.React的主要特性包含哪些

- 组件

React 应用都是构建在组件之上，两个核心概念props(属性）state(状态)

- JSX

将 HTML 直接嵌入了 JS 代码里面

- Virtual DOM

当组件状态 state 有更改的时候，React 会自动调用组件的 render 方法重新渲染整个组件的 UI，但是操作的只是虚拟DOM,React通过 diff 算法进行对比寻找到要变更的 DOM 节点，再把这个修改更新到浏览器实际的 DOM 节点，这个 Virtual DOM 是一个纯粹的 JS 数据结构，所以性能会比原生 DOM 快很多

- Data Flow

“单向数据绑定”

3.React的优势与不足

优势:

(1)React不是一个MVC框架，只是其中一个层次，理解学习就变得简单;

(2)由于轻巧与VDOM的概念，使得React速度非常的快(虚拟DOM）；

(3)模块化理念；（AMD/CMD/COMMON JSà模块定义、接口暴露、模块引入、模块调用)

(4)单向数据绑定;

(5)能实现服务器端的渲染，便于搜索引擎优化；(Server Side Render SEO)

(6)与其它框架/库兼容性好;

不足:

(1)本身内容比较新，API变化可能存在变化大的风险;

(2)它仅仅是一个V而已，处理大型项目的时候还需要引入FLUX和Routing等相关的内容

4. 什么是JSX，浏览器是否能够渲染jsx

JSX就是将 HTML 直接嵌入了 JS 代码里面,可以通过 React.createElement 来构造组件的 DOM 树。第一个参数是标签名，第二个参数是属性对象，第三个参数是子元素

```
<a href="http://facebook.github.io/react/">Hello!</a>
```

用 JS 代码来写就成这样了：

```
React.createElement('a', {href: 'http://facebook.github.io/react/'}, 'Hello!')
```

你可以通过 `React.createElement` 来构造组件的 DOM 树。第一个参数是标签名，第二个参数是属性对象，第三个参数是子元素。

一个包含子元素的例子：

```
var child = React.createElement('li', null, 'Text Content');
var root = React.createElement('ul', { className: 'my-list' }, child);
React.render(root, document.body);
```

对于常见的 HTML 标签，React 已经内置了工厂方法：

```
var root = React.DOM.ul({ className: 'my-list' },
             React.DOM.li(null, 'Text Content')
           );
```

使用HTML 标签， class 在 JSX 里要写成 className，因为 class 在 JS 里是保留关键字。同理某些属性比如 for 要写成 htmlFor

使用JavaScript 表达式

属性值使用表达式，只要用 {} 替换 ""

自定义 HTML 属性

如果在 JSX 中使用的属性不存在于 HTML 的规范中，这个属性会被忽略。如果要使用自定义属性，可以用 data- 前缀

属性扩散(要给组件设置多个属性，不想一个个写下这些属性)

使用展开式参数传递 ...

5. jsx中如何写注释

在 JSX 里使用注释也很简单，就是沿用 JavaScript，唯一要注意的是在一个组件的子元素位置使用注释要用 {} 包起来。

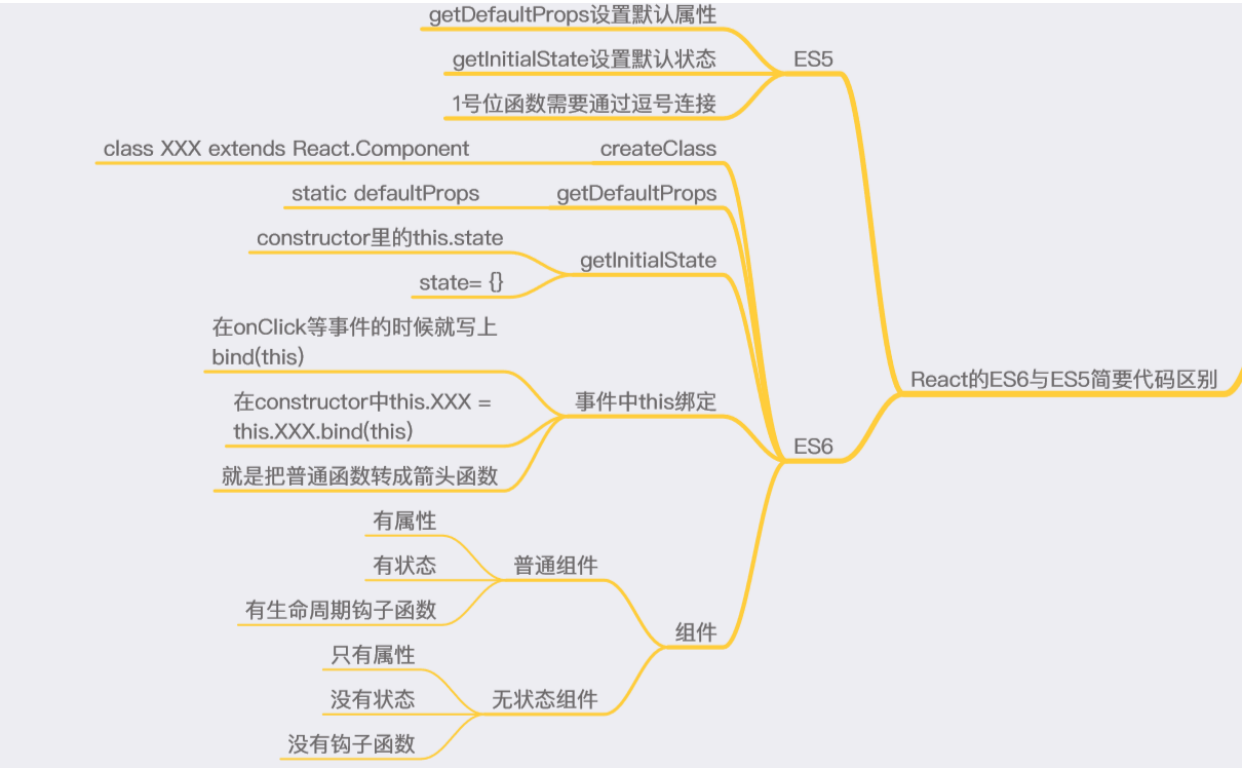
6. 如何理解虚拟DOM？解释一下它的工作原理

虚拟DOM其实就是用一个对象来描述DOM，通过对比前后两个对象的差异，最终只把变化的部分重新渲染，提高渲染的效率

VituralDOM的处理方式

- 用 JavaScript 对象结构表示 DOM 树的结构，然后用这个树构建一个真正的 DOM 树，插到文档当中
- 当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异
- 把2所记录的差异应用到步骤1所构建的真正的DOM树上，视图就更新了

7.React中ES5与ES6的语法结构有哪些变化，举例说明



8.React与Vue、Angular等MVVM 框架的比较

9.单向数据流是什么

整个流程如下:

- 首先要有 action，定义一些 *action creator* 方法，提供给 dispatcher
- View 层通过用户交互（比如 onClick）会触发 Action
- Dispatcher 会分发触发的 Action 给所有注册的 Store 的回调函数
- Store 回调函数根据接收的 Action 更新自身数据之后会触发一个 *change* 事件通 知 View 数据更改了

- View 会监听这个 *change* 事件，拿到对应的新数据并调用 setState 更新组件 UI

所有的状态都由 Store 来维护，通过 Action 传递数据，构成了如上所述的单向数据流循环，所以应用中的各部分分工就相当明确，高度解耦了。

组件

10. 如何将两个或多个组件合并成一个组件

在 React 组件中要包含其他组件作为子组件，只需要把组件当作一个 DOM 元素引入就可以了；

循环插入子元素，渲染插入组件时要设置key值

this.props.children，组件标签里面包含的子元素会通过 props.children 传递进来

props.children 通常是一个组件对象的数组，但是当只有一个子元素的时候，props.children 将是这个唯一的子元素，而不是数组了

React.createClass() 方法总结

1. 所有组件类都必须有自己的 render 方法，用于输出组件。
2. 组件类的第一个字母必须大写，否则会报错
3. 组件类只能包含一个顶层标签，否则也会报错。
4. 组件的属性可以在组件类的 this.props 对象上获取
5. 添加组件属性，有一个地方需要注意，就是 class 属性需要写成 className ， for 属性需要写成 htmlFor ， 这是因为 class 和 for 是 JavaScript 的保留字。

11. react中定义样式有哪些方式

三种:

1.行内样式：使用{{ }}，与正常jsx中插入js代码不一样，这里需要两个括号

```
<div style={ { float: 'right',} }>
        { this.renderButton() }
```

```
</div>
```

2.在jsx文件中定义样式变量，驼峰命名

```
let buttonStyle = {  //定义style变量
        backgroundColor: 'blue',
        float: 'left' as 'left', //哎，找了半天源码,好神奇。
        width:'230px',
        border: '1px solid blue',
        padding:'5px',
        margin: '10px',
        marginLeft: '213'
    }

//jsx调用
<div style={ buttonStyle }>  //此时使用一个花括号
     { this.renderButton() }
</div>
```

3.外部写css文件，然后引入，className

```
import './style.css';
<div className="sty1">看背景颜色和文字颜色</div>
```

12.react中组件的通信方式都有哪些

父组件向子组件通信

这种情况下很简单，就是通过 props 属性传递，在父组件给子组件设置 props，然后子组件就可以通过 props 访问到父组件的数据 / 方法，这样就搭建起了父子组件间通信的桥梁。

子组件向父组件通信

利用回调函数，可以实现子组件向父组件通信：父组件将一个函数作为 props 传递给子组件，子组件调用该回调函数，便可以向父组件通信。

跨级组件的通信

- 中间组件层层传递 props
- 使用 context 对象

使用 context 也很简单，需要满足两个条件：

- 上级组件要声明自己支持 context，并提供一个函数来返回相应的 context 对象
- 子组件要声明自己需要使用 context

非嵌套组件间通信

使用自定义事件方式，在 componentDidMount 里面订阅事件，

在 componentWillUnmount 里面取消订阅，当收到事件触发的时候调

用 setState 更新 UI

13.react中一般使用什么方式进行数据绑定，如何使用

1.绑定的数据我们一般要放在构造函数中，通过this.state定义我们要绑定的数据，然后在组件的属性值位置我们可以通过用花括号来实现属性数据的绑定

2.通过event.target事件，首先设置一个初始状态，给input添加一个onChange事件，编写事件代码

React数据绑定

(1)React通过state、props进行数据绑定

●

2.数据绑定的第一种方式：基于jsx绑定

(1)在componentWillMount中获取ajax数据

(2)将得到的数据存储state中

(3)在render中直接将state中的数据循环遍历，放在一个jsx的数据模板中

(4)循环叠加这个jsx模板，通过{}嵌入到页面

●

3.数据绑定的第二种方式：基于父子组件嵌套

(1)在componentWillMount中获取ajax数据

(2)将得到的数据存储state中

(3)在render中循环叠加子组件

循环叠加这个子组件放到一个空数组里，通过{}嵌入到页面中

首先初始化状态

```
this.state={"username":""}
```

给输入框添加onChange事件

```
<input type="text"  onChange={this.handleChange} />
<p>{this.state.username}</p>
```

编写事件代码

```
handleChange = (event) => {
        this.setState({
            username: event.target.value
        })
    }
```

14.什么是属性，什么是状态，属性与状态的比较关系是怎么样的

属性：props 就是组件的属性，由外部通过 JSX 属性传入设置，一旦初始设置完成，就可以认为 this.props 是不可更改的；

状态：当更改这个状态（数据）需要更新组件 UI 的就可以认为是 state

一旦状态（数据）更改，组件就会自动调用 render 重新渲染 UI，这个更改的动作会通过 this.setState方法来触发。

	属性	状态
能否从父组件获取初始值	√	×
能否由父组件修改	√	×
能否在组件内部设置默认值	√	√
能否在组件内部修改	×	√
能否设置子组件的初始值	√	×
能否修改子组件的值	√	×

15.通过什么方式更新一个状态，有哪些方法

1.this.setState()

2.replaceState（）

replaceState()方法与setState()类似，但是方法只会保留nextState中状态，原state不在nextState中的状态都会被删除。

使用语法：

replaceState(object nextState[, function callback])

nextState，将要设置的新状态，该状态会替换当前的state。

callback，可选参数，回调函数。该函数会在replaceState设置成功，且组件重新渲染后调用。

16.箭头函数在React中的应用是怎么样的，特别是在事件绑定操作的时候

react绑定事件函数的三种方式：

1.直接用bind（）方法绑定

```
<div onClick={this.handleClick.bind(this,param)}></div>
```

param 代表要传递的参数

2.用箭头函数绑定

```
<div onClick={(data)=>{this.handleClick}></div>
```

data 代表要传递的参数

3.构造器内声明，优点为仅需要一次绑定，不需要每次调用去执行绑定。

```
class Mian extends Component{
    constructor(props){
        super(props)
        this.clickHandle = this.clickHandle.bind(this)
    }
}
```

17.状态组件与无状态组件的差异与区别是怎么样的

无状态组件：无状态组件就是一个单纯的render函数，没有状态，没有生命周期，只有属性 ，使用纯函数定义。优点：可以通过减少继承Component而来的生命周期函数而达到性能优化的效果；缺点：因为它没有shouldComponentUpdate生命周期函数，所以每次state更新，它都会重新绘制render函数。

状体组件：有状态，生命周期、属性

18.组件生命周期的阶段分成哪几个，React生命周期的钩子函数具体哪些

装载组件触发

componentWillMount

只会在装载之前调用一次，在 render 之前调用，你可以在这个方法里面调用 setState 改变状态，并且不会导致额外调用一次 render

componentDidMount

只会在装载完成之后调用一次，在 render 之后调用，从这里开始可以通过 ReactDOM.findDOMNode(this)获取到组件的 DOM 节点

更新组件触发

这些方法不会在首次 render 组件的周期调用

- componentWillReceiveProps

在组件接收到一个新的 prop (更新后)时被调用。

- shouldComponentUpdate

返回一个布尔值。在组件接收到新的props或者state时被调用。在初始化时或者使用forceUpdate时不被调用。可以在你确认不需要更新组件时使用。

- componentWillUpdate

在组件接收到新的props或者state但还没有render时被调用

- componentDidUpdate

在组件完成更新后立即调用

卸载组件触发

- componentWillUnmount

在组件从 DOM 中移除之前立刻被调用

19.Input等内容，它们的value是否可以修改，如果想要让它们能够被修改

想要更改input的value，则需要监听onChange()事件，然后通过event.target.value来获取用户的输入，再通过设置一个名为value的state，。

```
onChange(event) {
this.setState({value: event.target.value});
}
<input value={this.state.value} onChange={this.onChange}/>
```

20.React生命周期钩子函数ES5与 ES6中有哪几个存在差异与区别，各自定义的方式是怎么样的

getInitialState

初始化 this.state 的值，只在组件装载之前调用一次。

如果是使用 ES6 的语法，你也可以在构造函数中初始化状态

```
class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: props.initialCount };
  }

  render() {
    // ...
  }
}
```

getDefaultProps

只在组件创建时调用一次并缓存返回的对象（即在 React.createClass 之后就会调用）

如果是使用 ES6 语法，可以直接定义 defaultProps 这个类属性来替代，这样能更直观的知道 default props 是预先定义好的对象值

```
Counter.defaultProps = { initialCount: 0 };
```

21.什么是React的事件，如何定义一个事件，如何调用一个事件

React 里面绑定事件的方式和在 HTML 中绑定事件类似，使用驼峰式命名指定要绑定的 onClick属性为组件定义的一个方法 {this.handleClick.bind(this)}

注意要显式调用 bind(this) 将事件函数上下文绑定在组件实例上

22.React中的Refs是什么，它的作用是怎么样的Refs的应用用场景有哪些

React 支持一种非常特殊的属性 Ref ，你可以用来绑定到 render() 输出的任何组件上。

这个特殊的属性允许你引用 render() 返回的相应的支撑实例

绑定一个 ref 属性到 render 的返回值上，在其它代码中，通过 this.refs 获取支撑实例

什么时候使用refs

- 管理焦点、文本选择或媒体回放。
- 触发命令动画。

- 与第三方DOM库集成

1、为DOM元素添加Ref

ref属性接收一个回调函数，这个回调函数在组件挂载或者卸载的时候被调用。当ref用于一个HTML元素的时候，ref指定的回调函数在调用的时候会接收一个参数，该参数就是指定的DOM元素

2.为组件Component添加Ref

当ref属性用于一个class指定的自定义组件的时候,ref回调函数会接收到一个挂载的组件实例作为参数。

使用方法

绑定一个 ref 属性到 render 的返回值上：

```
<input ref="myInput" />
```

在其它代码中，通过 this.refs 获取支撑实例：

```
var input = this.refs.myInput;
var inputValue = input.value;
var inputRect = input.getBoundingClientRect();
```

完整实例

你可以通过使用 this 来获取当前 React 组件，或使用 ref 来获取组件的引用，实例如下：

React 实例

```
class MyComponent extends React.Component {
  handleClick() {
    // 使用原生的 DOM API 获取焦点
    this.refs.myInput.focus();
  }
  render() {
    //  当组件插入到 DOM 后，ref 属性添加一个组件的引用于到 this.refs
    return (
      <div>
        <input type="text" ref="myInput" />
        <input
          type="button"
          value="点我输入框获取焦点"
          onClick={this.handleClick.bind(this)}
        />
      </div>
    );
  }
}

ReactDOM.render(
  <MyComponent />,
  document.getElementById('example')
);
```

要进行DOM操作

在要引用的 DOM 元素上面设置一个 ref 属性指定一个名称，然后通过 this.refs.name 来访问对应的 DOM 元素

- 可以使用 ref 找到组件定义的任何公共方法，比如 this.refs.myTypeahead.reset()
- Refs 是访问到组件内部 DOM 节点唯一可靠的方法
- Refs 会自动销毁对子组件的引用（当子组件删除时）

23.数据请求时在哪个钩子函数中进行

constructor() 》componentWillMount() 》render() 》componentDidMount()

同步componentWillMount()　或者　异步 componentDidMount() 中

24.React中如何进行模块化开发

借助前端构建工具webpack

1、webpack是facebook为react量身打造的构建工具

2、主要作用是实现模块化，代码整合，代码分割的作用

3、使用webpack整合以后 也不需要使用browser进行将jsx转成js了

webpack实现模块化---什么是模块化

1、模块指的是一组具有同等属性和功能的集合叫做模块和类的概念相似

2、react模块化指的是一个js中存放一个或多个组件，这些组件通过commonjs规范对外提供接口

3、在其他组件当中可以调用这些对外提供成接口的组件

---

commonjs规范

1、一个js内部的函数或者变量想要被外部调用

2、想在另一个函数中调用这些接口 需要通过require（js路径）引入组件，这样实现了一个js调用另外一个js里面的内容

25.React中如何操作表单，如果有个注册用户的表单，我们操作流程是怎么样　　的

26.什么是HOC ，什么时候可以应用HOC

高阶组件就是一个函数，且该函数接受一个组件作为参数，并返回一个新的组件。

属性代理：它通过做一些操作，将被包裹组件的props和新生成的props一起传递给此组件，这称之为属性代理。

反向继承：这种方式返回的React组件继承了被传入的组件，所以它能够访问到的区域、权限更多，相比属性代理方式，它更像打入组织内部，对其进行修改。

实例：

实现Loading组件时，发现需要去拦截它的渲染过程，故使用了**反向继承**的方式来完成。

方式：在通过装饰器调用时，需要传入一个函数作为入参，函数可以获取到props，随后返回一个Boolean对象，来决定组件是否需要显示Loading态

实现copy组件的时候，我们发现不需要去改变组件内部的展示方式，只是为其在外围增加一个功能，并不会侵入被传入的组件，故使用了**属性代理**的方式。

27. 什么是React的纯组件

纯组件是通过控制shouldComponentUpdate生命周期函数，减少render调用次数来减少性能损耗的。这相对于Component来说，减少了手动判断state变化的繁琐操作，但该组件也具有一定的缺陷，因为它只能进行一层浅比较，简单来说，它只比较props和state的内存地址，如果内存地址相同，则shouldComponentUpdate生命周期就返回false。PureComponent的使用场景应该是局部数据发生改变的场景，比如带有输入框、switch开关等的UI组件就可以使用PureComponent组件封装。PureComponent中如果有数据操作最好配合一个第三方组件——Immutable一起使用，Immutable需要使用npm安装该插件才可以使用，因为Immutable可以保证数据的不变性。

28. React中key的存在意义

在 DOM 中的某些元素被增加或删除的时候，帮助 React 识别哪些元素发生了变化

29. 何为受控组件，受控与非受控组件的差异区别

表单的状态属性

- value，对应 <input> 和 <textarea> 所有
- checked，对应类型为 checkbox 和 radio 的 <input> 所有
- selected，对应 <option> 所有

表单元素包含以上任意一种状态属性都支持 onChange 事件监听状态值的更改。

对于设置了“状态属性“值的表单元素就是受控表单组件。

表单元素没有设置自己的“状态属性”，或者属性值设置为 null，就是非受控组件。

如果你想要给“状态属性”设置默认值，就要用 React 提供的特殊属性 defaultValue，对于 checked 会有 defaultChecked，<option> 也是使用 defaultValue

<select>

React 修改成统一使用 value指定选中项

30.为什么建议传递给setState 的参数是一个 callback 而不是一个对象

React 通过setState方法来更新组件的内部状态，当setState方法被调用时，React 会根据新的state来重新渲染组件（并不是每次setState都会触发render，React可能会合并操作，再一次性 render）。

React将setState设置为批次更新，从而避免了频繁地重新渲染组件，即setState是不能保证同步更新状态的。因此，如果我们在setState之后，直接使用this.state.key很可能得到错误的结果。那么当我们需要同步获得 state 更新后的值时，如何做呢？React提供了两种方式来实现该功能。

1、回调函数

setState方法可以传入第二个函数类型的参数作为回调函数，该回调函数将会在state完成更新之后被调用。

31.怎么阻止组件的渲染

shouldComponentUpdate，重渲染时render()函数调用前被调用的函数，有两个参数 nextProps和nextState ，分别表示下一个props和state的值。当函数返回false时，阻止接下来的render()函数的调用，阻止组件重渲染，返回true时，组件照常渲染，可以添加一个if条件判断，用return false阻止render调用

32.在构造函数中调用 super(props) 的目的是什么

ES6 语法中，super 指代父类的构造函数，React 里面就是指代 React.Component 的构造函数。在你调用 super() 之前，你无法在构造函数中使用 this，不传 props，只执行 super()，可以使用this.props；传入props，this.props就被定义了，就可以调用this.props.xxx

33.何为 Children

this.props.children是组件所有的子节点

this.props.Childern的值有三种可能：

a若此组件没有子节点就是undefined;

b若此组件有一个子节点则数据类型：object；若此组件有多个子节点，数据类型就是Array

34. Fragment是什么，它的缩写怎么样，作用是什么？

Fragments 可以聚合一个子元素列表，并且不在DOM中增加额外节点,为一个组件返回一个子元素列表,类似与vue的<template>,缩写<></>,也可以添加key值，并且key 是唯一可以传递给 Fragment 的属性

```
function Glossary(props) {
  return (
    <dl>
      {props.items.map(item => (
        // 没有`key`，将会触发一个key警告
        <React.Fragment key={item.id}>
          <dt>{item.term}</dt>
          <dd>{item.description}</dd>
        </React.Fragment>
      ))}
    </dl>
  );
}
```

Rudux

35.什么是redxu

可预测的状态容器

36.Redux流程中的三大基本原则是什么

- 整个应用只有唯一个可信数据源，也就是只有一个 Store
- State 只能通过触发 Action 来更改
- State 的更改必须写成纯函数，也就是每次更改总是返回一个新的State，在 Redux 里这种函数称为 Reducer

37.如何理解单一数据源

38. Redux的主要部分包含哪些

Actions

Action 很简单，就是一个单纯的包含 { type, payload } 的对象，type 是一个常量用来标示动作类型，payload 是这个动作携带的数据。

Action 需要通过 store.dispatch() 方法来发送

Reducers

Reducer 用来处理 Action 触发的对状态树的更改。

所以一个 reducer 函数会接受 oldState 和 action 两个参数，返回一个新的 state：(oldState, action) => newState

Redux 提供了一个工具函数 combineReducers 来简化这种 reducer 合并

Store

现在有了 Action 和 Reducer，Store 的作用就是连接这两者，Store 的作用有这么几个：

- Hold 住整个应用的 State 状态树
- 提供一个 getState() 方法获取 State
- 提供一个 dispatch() 方法发送 action 更改 State
- 提供一个 subscribe() 方法注册回调函数监听 State 的更改

39.简述一下redux数据流向的流程

store.dispatch(action) -> reducer(state, action) ->

store.getState() 其实就构成了一个“单向数据流”

1. 调用 store.dispatch(action)

Action 是一个包含 { type, payload } 的对象，它描述了“发生了什么”，比如：

```
{ type: 'LIKE_ARTICLE', articleID: 42 }
{ type: 'FETCH_USER_SUCCESS', response: { id: 3, name: 'Mary' } }
{ type: 'ADD_TODO', text: 'Read the Redux docs.' }
```

你可以在任何地方调用 store.dispatch(action)，比如组件内部，Ajax 回调函数里面等等

2. Action 会触发给 Store 指定的 root reducer

root reducer 会返回一个完整的状态树，State 对象上的各个字段值可以由各自的 reducer 函数处理并返回新的值。

- reducer 函数接受 (state, action) 两个参数
- reducer 函数判断 action.type 然后处理对应的 action.payload 数据来更新并返回一个新的 state

3. Store 会保存 root reducer 返回的状态树

新的 State 会替代旧的 State，然后所有 store.subscribe(listener) 注册的回调函数会被调用，在回调函数里面可以通过 store.getState() 拿到新的 State

40. 结构 Action层的主要工作是什么？

41. 如何定义Action方法

Action的创建:

- createActions
- generateActions

其中createActions：

- 接受的是一个定义了各种Actions的Class作为参数。
- 所定义的各个Action需要指定dispatch到store的数据。

最终返回是：

- 返回一个对象，该对象包含了所定义的所有Actions。

generateActions有这些特点：

- 接受一个由Action名称组成的列表作为参数，而不是一个Class。
- 不需要像createActions一样需要显式指定任何dispatch的数据，因为genreateActions内部已经帮我们处理好。

42. 简述一下reducer在redux中所起的角色

所有的数据都要放在reducer中，reducer是数据中心

43. 什么是Store，在Redux中的重要如何体现

仓库，万事开头从仓库，因为所有的数据操作都要放在仓库中进行统一管理

44. Flux与Redux的区别差异

一个组件所有的数据，必须由父组件传过来，而不能像flux中直接从store取。

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

Redux和Flux很像。主要区别在于Flux有多个可以改变应用状态的store，它通过事件来触发这些变化。组件可以订阅这些事件来和当前状态同步。

Redux没有分发器dispatcher，但在Flux中dispatcher被用来传递数据到注册的回调事件。另一个不同是Flux中有很多扩展是可用的，这也带来了一些混乱与矛盾。

45.Redux的优势是什么

redux把流程规范了，统一渲染根节点虽然对代码管理上规范了一些，只要有需要显示数据的组件，当相关数据更新时都会自动进行更新

46.展示组件(Presentational component)和容器组件(Container component)之间有何不同

展示组件（persentational components）

负责展示UI，也就是组件如何渲染，具有很强的内聚性。

只关心得到数据后如何渲染

容器组件（container components）

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

当展示组件的状态变更引起源数据变化时，展示组件通过调用容器组件提供的方法同步这些变化

注意

展示组件和容器组件是根据组件的意图划分组件，展示组件通常通过无状态组件实现，容器组件通过有状态组件实现

47.Redux Thunk的作用是什么

中间件，redux-thunk是用来做异步的，他允许你的action可以返回函数，带有dispatch和getState两个参数

48.何为纯函数(pure function)

同样的输入，必定得到同样的输出

- 不能调用系统 I/O 的API
- 不得改写参数
- 不能调用Date.now()或者Math.random()等不纯的方法，因为每次会得到不一样 的结果

49.redux中的异步操作需要使用什么方法

通过 dispatch(action) -> 中间件 -> reducer处理数据 -> 改变store -> 使用subscribe()监听store改变更新视图 的方式管理状态

50.mapStateToProps以及 mapDispatchToProps的用法与作用

mapStateToProps，将状态映射到UI组件的props

mapDispatchToProps，将dispatch方法映射到UI组件的props

51.provider和connect如何使用

通过Provider将store提供给所有组件，通过connect将mapStateToProps，mapDispatchToProps与视图组件进行连接

52.bindActionCreators是如何使用的

将dispatch直接和action creator结合好然后发出去的这一部分操作给封装成一个函数

同步action,把action绑定到了connect方法中

53.combineReducers是什么及使用方法

Redux 提供了一个工具函数 combineReducers 来简化这种 reducer 合并

54.redux中有多个dispatch，要如何处理

2 分钟了解 Redux 是如何运作的

关于 Store：

- 整个应用只有一个唯一的 Store

- Store 对应的状态树（State），由调用一个 reducer 函数（root reducer）生成
- 状态树上的每个字段都可以进一步由不同的 reducer 函数生成
- Store 包含了几个方法比如 dispatch, getState 来处理数据流
- Store 的状态树只能由 dispatch(action) 来触发更改

Redux 的数据流：

- action 是一个包含 { type, payload } 的对象
- reducer 函数通过 store.dispatch(action) 触发
- reducer 函数接受 (state, action) 两个参数，返回一个新的 state
- reducer 函数判断 action.type 然后处理对应的 action.payload 数据来更新状态树

路由

55.什么是React路由

通过管理 URL，实现组件的切换与状态的变化或者说是视图的导航。

56.Switch在路由4中的作用与意义

在路径相同的情况下，只匹配第一个，这个可以避免重复匹配

57. react路由跳转是怎么实现的

Link / NavLink

58. react路由怎样进行传参

方式 一：

通过params

1.路由表中

<Route path=’/sort/:id’ component={Sort}></Route>

2.Link处

HTML方式

<Link to={’/sort/’+’2’} activeClassName='active'>XXXX</Link>

JS方式

this.props.history.push('/sort/'+'2')

3.sort页面

通过 this.props.match.params.id

就可以接受到传递过来的参数（id）

方式 二：

通过query

前提：必须由其他页面跳过来，参数才会被传递过来

注：不需要配置路由表。路由表中的内容照常：<Route path='/sort' component={Sort}></Route>

1.Link处

HTML方式

<Link to={{ path : ' /sort ' , query : { name : 'sunny' }}}>

JS方式

this.props.history.push({ path : '/sort' ,query : { name: ' sunny'} })

2.sort页面

this.props.location.query.name

方式 三：

通过state

同query差不多，只是属性不一样，而且state传的参数是加密的，query传的参数是公开的，在地址栏

1.Link 处

HTML方式：

<Link to={{ path : ' /sort ' , state : { name : 'sunny' }}}>

JS方式：

this.props.history.push({ pathname:'/sort',state:{name : 'sunny' } })

2.sort页面

this.props.location.state.name

59. react路由嵌套

路由2里面是通过标签嵌套

路由4在不同组件设置Route进行路由嵌套

60. React如何进行重定向

第一种是通过 <Redirect> 标签实现，第二种是通过编程式导航方式实现。

<Redirect to={'/default'}/>

this.props.history.push('/default')

61. link与Navlink的区别

a标签：如果使用锚点元素实现，在每次点击时，页面被重新加载

replace(bool)：为 true 时，点击链接后将使用新地址替换掉访问历史记录里面的原地址；为 false 时，点击链接后将在原有访问历史记录的基础上添加一个新的纪录。默认为 false

link：url会更新，组件会被重新渲染，但是页面不会重新加载，使用to参数来描述需要定位的页面。它的值既可是字符串，也可以是location对象（包含pathname、search、hash、与state属性）如果其值为字符串，将会被转换为location对象

<NavLink>是<Link>的一个特定版本，会在匹配上当前的url的时候给已经渲染的元素添加参数，组件的属性有

activeClassName(string)：设置选中样式，默认值为active

activeStyle(object)：当元素被选中时，为此元素添加样式

exact(bool)：为true时，只有当导致和完全匹配class和style才会应用

strict(bool)：为true时，在确定为位置是否与当前URL匹配时，将考虑位置pathname后的斜线

isActive(func)判断链接是否激活的额外逻辑的功能

62. react路由中exact是做什么的

精确匹配

63. 对react-router和react-router-dom的理解

react-router：实现了路由的核心功能，

react-router-dom：基于react-router，加入了在浏览器运行环境下的一些功能，

例如：Link组件，会渲染一个a标签，Link组件源码a标签行； BrowserRouter和HashRouter组件，前者使用pushState和popState事件构建路由，后者使用window.location.hash和hashchange事件构建路由。

功能	React-router-2.x	React-router-4.x	Vue-Router
静态路由表及分配地址	Router/Route/hashHistory 标签形式的嵌套	HashRouter/BrowserRouter as Router Route/Switch	设置数组，path和component对象内容的确认
首页的渲染	IndexRoute	Route的path为/，但是需要加上exact	router-view
链接形式1	a链接，但是要加#	a链接，但是要加#	a链接，但是要加#
链接形式2	Link链接，to，不需要加#	Link/NavLink链接，to，不需要加#	router-link,to，不需要加#
高亮显示	activeClassName	NavLink下的activeClassName/activeStyle	active-class
路由嵌套	Route标签的嵌套	是通过不同的组件中设置Route来进行路由的嵌套，嵌套操作被拆分了	children嵌套
路由嵌套的显示	this.props.children	Route既是路由，也是占位渲染显示	router-view
参数传递	设参在路由 :xxx 传参在地址 Link/NavLink to的设置 接参在组件 this.props.params.xxx	设参在路由 :xxx 传参在地址 Link/NavLink to的设置 接参在组件 this.props.match.params.xxx	设参在路由 :xxx 传参在地址 router-link的to的设置 接参在组件 this.$route.params.xxx/watch
首页的高亮显示	onlyActiveOnIndex IndexRoute+activeClassName	exact	exact
多层嵌套	Route标签的多层嵌套+this.props.children占位显示	是通过不同的组件中设置Route来进行路由的嵌套，嵌套操作被拆分了	children的多层嵌套+router-view
程序式导航	this.props.history.push(pathUrl) hashHistory.push(pathUrl)	this.props.history.push(pathUrl)	this.$router.push(pathUrl)
Miss与NoMatch		NoMatch不需要写path，直接写NoMatch	redirect与*号通配符

64. react性能优化方案

1. 重写shouldComponentUpdate来避免不必要的dom操作
2. 使用production版本的react.js
3. 使用key来帮助React识别列表中所有子组件的最小变化

65. 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View
2. View发出用户的Action
3. 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