react100道面试题-1（https://github.com/minsion）

面试题 1. 简述什么是React （ 概念 ）？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

1、React是Facebook开发的一款JS库。
2、React一般被用来作为MVC中的V层，它不依赖其他任何的库，因此开发中，可以与任何其他的库集成使用，包括Jquery、Backbone等。
3、它可以在浏览器端运行，也可以通过nodejs在服务端渲染。
4、React的思想非常独特，性能出众，可以写出重复代码少，逻辑清晰的前端代码。
5、React的语法是jsx，通过使用这种语法，可以在react代码中直接混合使用js和html来编写代码，这样代码的逻辑就非常清晰，当然也意味着，需要将jsx代码编译成普通的javascript代码，才能在浏览器中运行，这个过程根据实际项目情况，可以选择多种不同的思路，或者在服务器端通过webpack进行编译

面试题 2. 简述React有什么特点?

推荐指数： ★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

1.声明式设计：React 使创建交互式 UI 变得轻而易举。为你应用的每一个状态设计简洁的视图，当数据变动时 React能高效更新并渲染合适的组件。
2.组件化: 构建管理自身状态的封装组件，然后对其组合以构成复杂的 UI。
3.高效：React通过对DOM的模拟，最大限度地减少与DOM的交互。
4.灵活：无论你现在使用什么技术栈，在无需重写现有代码的前提下，通过引入React来开发新功能。

React 是一个声明式，高效且灵活的用于构建用户界面的 JavaScript 库。使用 React 可以将一些简短、独立的代码片段组合成复杂的 UI 界面，这些代码片段被称作“组件”。
由于React的设计思想极其独特，属于革命性创新，性能出众，代码逻辑却非常简单。所以，越来越多的人开始关注和使用，认为它可能是将来Web开发的主流工具。
这个项目本身也越滚越大，从最早的UI引擎变成了一整套前后端通吃的Web App解决方案。衍生的React Native 项目，目标更是宏伟，希望用写Web App的方式去写Native App。如果能够实现，整个互联网行业都会被颠覆，因为同一组人只需要写一次UI ，就能同时运行在服务器、浏览器和手机。

面试题 3. 请说明什么是JSX ？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

JSX是一种JavaScript的语法扩展，运用于React架构中，其格式比较像是模版语言，但事实上完全是在JavaScript内部实现的。元素是构成React应用的最小单位，JSX就是用来声明React当中的元素，React使用JSX来描述用户界面。
可以通过以下三个方面了解JSX：
1）JSX 是一种 JS 扩展的表达式
2）JSX 是带有逻辑的标记语法，有别于 HTML 模版
3）并且支持样式、逻辑表达式和事件

面试题 4. 简述虚拟DOM的概念和机制 ？

推荐指数： ★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

传统的 DOM 操作是直接在 DOM 上操作的，当需要修改一系列元素中的值时，就会直接对 DOM 进行操作。而采用 Virtual DOM 则会对需要修改的 DOM 进行比较（DIFF），从而只选择需要修改的部分。也因此对于不需要大量修改 DOM 的应用来说，采用 Virtual DOM 并不会有优势。开发者就可以创建出可交互的 UI。

在React中，render执行的结果得到的并不是真正的DOM节点，结果仅仅是轻量级的JavaScript对象，我们称之为virtual DOM。

虚拟DOM是React的一大亮点，具有batching(批处理)和高效的Diff算法。这让我们可以无需担心性能问题而”毫无顾忌”的随时“刷新”整个页面，由虚拟 DOM来确保只对界面上真正变化的部分进行实际的DOM操作。在实际开发中基本无需关心虚拟DOM是如何运作的，但是理解其运行机制不仅有助于更好的理解React组件的生命周期，而且对于进一步优化 React程序也会有很大帮助。

虚拟的DOM的核心思想是：对复杂的文档DOM结构，提供一种方便的工具，进行最小化地DOM操作。这句话，也许过于抽象，却基本概况了虚拟DOM的设计思想

面试题 5. 简述React有什么优缺点?

推荐指数： ★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

React优点：
1、React速度快、性能好
它并不直接对DOM进行操作，引入了一个叫做虚拟DOM的概念，安插在javascript逻辑和实际的DOM之间，性能好
2、跨浏览器兼容
虚拟DOM的原因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
3、单向数据流
Flux随着React视图库的开发而被Facebook概念化，是一个用于在JavaScript应用中创建单向数据层的架构
4、React兼容性好
使用RequireJS来加载和打包，而Browserify和Webpack适用于构建大型应用。

React缺点
1.React并不是一个单独完整的框架，React是目标是UI组件，通常可以和其它框架组合使用，目前并不适合单独做一个完整的框架

面试题 6. React 类组件和函数组件之间的区别是什么?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

主要要以下几个区别：

（1）语法不同、设计思想不同
（2）生命周期、状态变量
（3）复用性：
（4）优缺点

一、语法不同、设计思想不同
函数式组件是函数式编程思想，而类组件是面向对象编程思想。面向对象编程将属性和方法封装起来，屏蔽很多细节，不利于测试。

二、生命周期、状态变量
类式组件：使用state对象定义状态变量，有诸如componmentDidMount、shouldComponentUpdate等生命周期钩子函数；
函数式组件：没有this,使用一系列的内置hooks实现对应的功能，比如使用useState创建状态变量，使用useEffect实现类似于componmentDidMount、shouldComponentUpdate等生命周期钩子函数的功能。

三、复用性
类式组件：使用hoc（高阶组件）、render propss实现组件的逻辑复用、拓展组件的功能。
函数式组件：使用自定义hooks实现组件的逻辑复用。

四、优缺点
函数式组件：
优点：
相对于类式组件，一般情况而言，代码量更少，代码更简洁，可读性更强；
更易于拆分组件和测试；
缺点：
在业务逻辑巨复杂，状态依赖关系错乱的情况下，使用useEffect、useMemo等hooks，对其依赖项数组的思考为开发者带来了更大的心智负担；
不具备处理错误边界等业务情况的hooks；

类式组件：
优点：
功能完备，具有componentDidsCatch、getDerivedStateFromError等钩子函数处理边界错误；
缺点：

在复用性上，hoc组件等会出现诸如嵌套地狱、重名props被覆盖、难以拆分和测试等问题；

五、总结
类式组件和函数式组件各有其优点，关键是看自己的需求是什么；如果你开发的业务逻辑和状态并不复杂，那么类式组件可能会更合适；
但是如果你要处理错误边界或者是业务逻辑巨复杂的情况，那么类式组件更合适；
其实无论是什么技术开发，衡量代码是否优雅的标准无非是开发效率（复用性、易用性）、代码性能、是否易于测试和维护；当然三者常常不可兼得，关键看自己需求是什么；

面试题 7. 简述React 中 refs 的作用 ?

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

React Refs 提供了一种访问在render方法中创建的 DOM 节点或者 React 元素的方法。在典型的数据流中，props 是父子组件交互的唯一方式，想要修改子组件，需要使用新的pros重新渲染它。凡事有例外，某些情况下咱们需要在典型数据流外，强制修改子代，这个时候可以使用 Refs。
咱们可以在组件添加一个 ref 属性来使用，该属性的值是一个回调函数，接收作为其第一个参数的底层 DOM 元素或组件的挂载实例。

```
class UnControlledForm extends Component {
handleSubmit = () => {
console.log("Input Value: ", this.input.value);
};
render() {
return (

<br />
       <form onSubmit={this.handleSubmit}><br />
         <input type="text" ref={(input) => (this.input = input)} /><br />
         <button type="submit">Submit</button><br />
       </form><br />


);
}
}
```

请注意，input 元素有一个ref属性，它的值是一个函数。该函数接收输入的实际 DOM 元素，然后将其放在实例上，这样就可以在 handleSubmit 函数内部访问它。
经常被误解的只有在类组件中才能使用 refs，但是refs也可以通过利用 JS 中的闭包与函数组件一起使用。

```
function CustomForm({ handleSubmit }) {
let inputElement;
return (

<br />
    <form onSubmit={() => handleSubmit(inputElement.value)}><br />
      <input type="text" ref={(input) => (inputElement = input)} /><br />
      <button type="submit">Submit</button><br />
    </form><br />


);
}
```

面试题 8. 简述React store的概念？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

React Store 就是把它们联系到一起的对象。Store 有以下职责：
const store = createStore(reducer)
1:维持应用的 state；
2:提供 getState() 方法获取 state；
3:提供 dispatch(action) 方法更新 state；
4:通过 subscribe(listener)注册监听器;
5:通过 subscribe(listener)返回的函数注销监听器

面试题 9. 解释为什么浏览器不能读取 JSX?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

浏览器只能读取 JavaScript 对象，但不能读取常规 JavaScript 对象中的 JSX。因此，为了让浏览器能够读取 JSX，首先，我们需要使用 Babel 等 JSX 转换器将 JSX 文件转换为 JavaScript 对象，然后将其传递给浏览器

面试题 10. 请列举ES5 相比，React 的 ES6 语法有何不同?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

从 ES5 到 ES6 的语法有以下几个方面的变化：
1）require vs import

```
// ES5
var React = require('react');
// ES6
import React from 'react';
```

2) export vs exports

```
// ES5
module.exports = Component;
// ES6
export default Component;
```

3) component and function

```
// ES5
var MyComponent = React.createClass({
render: function() {
return
```

Hello Edureka!

```
;
}
});
// ES6
class MyComponent extends React.Component {
render() {
return
```

Hello Edureka!

```
;
}
}
```

4) props

```
// ES5
var App = React.createClass({
propTypes: { name: React.PropTypes.string },
render: function() {
return
```

Hello, {this.props.name}!

```
;
}
});

// ES6
class App extends React.Component {
render() {
return
```

Hello, {this.props.name}!

```
;
}
}
```

5) state

```
// ES5
var App = React.createClass({
getInitialState: function() {
return { name: 'world' };
},
render: function() {
return
```

Hello, {this.state.name}!

```
;
}
});
// ES6
class App extends React.Component {
```

```
constructor() {
super();
this.state = { name: 'world' };
}
render() {
return


Hello, {this.state.name}!


;
}
}
```

面试题 11. 简述React中引入css的方式?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

组件中引入 .module.css 文件
使用.module.css文件来为组件引入样式，这种方式也被称为CSS模块化。
在.module.css文件中定义的样式只能作用于当前组件内部，不会影响到其他组件或全局样式，这样可以避免样式冲突的问题。
CSS in JS
CSS in JS 是一种前端开发技术，它将 CSS 样式表的定义和 JS 代码紧密结合在一起，以实现更高效、更灵活的样式控制。在 CSS in JS 中，开发者可以使用 JS 来编写 CSS 样式，可以在代码中通过变量或函数等方式来动态生成样式。这种方式可以避免传统 CSS 中的一些问题，如全局作用域、选择器嵌套、命名冲突等，同时也提供了更高的可重用性和可维护性。

在 React 中，有多种支持 CSS in JS 的第三方库，比较常用的有 styled-components、Emotion、JSS 等。这些库都提供了方便的 API 来定义和应用样式，并且可以自动管理 CSS 的引入和组件的封装。使用 CSS in JS 可以更好地与组件化开发思想结合，提高代码的可复用性和可维护性

面试题 12. 请介绍React中的key有什么作用?

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

在 React 中，key 是用来给每个组件的元素（Element）做一个唯一的标识。当 React 更新组件的时候，它会对比新旧两个组件的 key 是否一致，如果一致，则说明是同一个组件，直接更新它的内容即可。如果不一致，则说明是不同的组件，需要先删除旧组件，再新建一个新的组件并插入到 DOM 树中。

因此，key 的作用是帮助 React 快速判断出哪些元素发生了变化，从而提高性能，避免不必要的 DOM 操作。同时，key 也可以用来保证数组渲染时每个元素的稳定性，避免出现类似于数组元素位置发生变化但是内容没变的情况

面试题 13. 简述类组件和函数式组件的区别？

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

1 语法：类组件使用ES6的class语法创建组件，而函数式组件使用函数声明来创建组件。

2 状态管理：类组件可以使用state来管理组件的内部状态，而函数式组件则通常使用userState Hook来管理状态。

3 生命周期：类组件可以使用生命周期方法，如componentDidMount、componentDidUpdate等来管理组件的生命周期，而函数式组件则使用useEffect Hook来管理。

4 调用方式：如果是一个函数组件，调用则是执行函数即可，如果是一个类组件，则需要将组件进行实例化，然后调用实例对象的render方法

5 性能：函数式组件通常比类组件更轻量级，因为类组件需要实例化，而函数式组件只是普通函数调用

面试题 14. 请列举常用的React Hooks？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

useState(): 允许在函数组件中使用状态。使用useState() 声明一个状态变量，并使用它来存储组件的状态。每次更改状态时，组件将重新渲染。

useEffect():用于处理副作用。副作用指在React组件之外进行的操作，例如从服务器获取数据，处理DOM元素等。使用useEffect() hook，您可以执行此类操作，而无需在类组件中编写生命周期方法。点击去学习

useContext(): 允许您在React中使用上下文。上下文是一种在组件树中传递数据的方法，可以避免通过Props一层层传递数据。使用useContext() hook，您可以访问整个应用程序中定义的上下文对象。点击去学习

useReducer(): 是useState() hook的替代品，用于管理更复杂的状态。它使用Reducer函数来管理组件状态，Reducer函数接收当前状态和要进行的操作，然后返回新状态。详细使用方式见此文章。点击去学习

useCallback(): 用于避免在每次渲染时重新创建回调函数。当您需要将回调函数传递给子组件时，这非常有用，因为它可以避免子组件不必要地重新渲染。点击去学习

useMemo(): 用于缓存计算结果，以避免在每次渲染时重新计算。这非常有用，特别是当计算成本很高时。点击去学习

useRef(): 用于创建对DOM元素的引用。它还可以用于存储组件之间共享的变量，这些变量不会在组件重新渲染时发生更改

面试题 15. 请列举React和vue.js的相似性和差异性？

推荐指数： ★★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

相似性如下。
（1）都是用于创建UI的 JavaScript库。
（2）都是快速和轻量级的代码库（这里指 React核心库）。
（3）都有基于组件的架构。
（4）都使用虚拟DOM。
（5）都可以放在单独的HTML文件中，或者放在 Webpack设置的一个更复杂的模块中。
（6）都有独立但常用的路由器和状态管理库。

它们最大的区别在于 Vue. js通常使用HTML模板文件，而 React完全使用 JavaScript创建虚拟DOM。 Vue. js还具有对于“可变状态”的“reactivity”的重新渲染的自动化检测系统

面试题 16. React中什么是受控组件和非控组件?

推荐指数： ★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

（1）受控组件 在使用表单来收集用户输入时，例如<input><select><textearea>等元素都要绑定一个change事件，当表单的状态发生变化，就会触发

受控组件更新state的流程：

可以通过初始state中设置表单的默认值
每当表单的值发生变化时，调用onChange事件处理器
事件处理器通过事件对象e拿到改变后的状态，并更新组件的state
一旦通过setState方法更新state，就会触发视图的重新渲染，完成表单组件的更新
受控组件缺陷： 表单元素的值都是由React组件进行管理，当有多个输入框，或者多个这种组件时，如果想同时获取到全部的值就必须每个都要编写事件处理函

（2）非受控组件 如果一个表单组件没有value props（单选和复选按钮对应的是checked props）时，就可以称为非受控组件。在非受控组件中，可以使


```
class NameForm extends React.Component {<br />
  constructor(props) {<br />
    super(props);<br />
    this.handleSubmit = this.handleSubmit.bind(this);<br />
  }<br />
  handleSubmit(event) {<br />
    alert('A name was submitted: ' + this.input.value);<br />
    event.preventDefault();<br />
  }<br />
  render() {<br />
    return (<br />
<br />
      <form@ onSubmit={this.handleSubmit}><br />
        <label><br />
          Name:          <input type="text" ref={(input) => this.input = input} />        </label><br />
        <input type="submit" value="Submit" /><br />
      </form@><br />
<br />
    );<br />
  }<br />
}
```

面试题 17. 请简述React中props.children和React.Children的区别？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

在React中，当涉及组件嵌套，在父组件中使用props.children把所有子组件显示出来。如下：

```
function ParentComponent(props){
return (


{props.children}

)
}
```

如果想把父组件中的属性传给所有的子组件，需要使用React.Children方法。

比如，把几个Radio组合起来，合成一个RadioGroup，这就要求所有的Radio具有同样的name属性值。可以这样：把Radio看做子组件，RadioGroup看做父组件，name的属性值在RadioGroup这个父组件中设置。

首先是子组件：

```
//子组件
function RadioOption(props) {
return (


<input type="radio" value={props.value} name={props.name} />


{props.label}
)
}
```

然后是父组件，不仅需要把它所有的子组件显示出来，还需要为每个子组件赋上name属性和值：

```
//父组件用,props是指父组件的props
function renderChildren(props) {

//遍历所有子组件
return React.Children.map(props.children, child => {
if (child.type === RadioOption)
return React.cloneElement(child, {
//把父组件的props.name赋值给每个子组件
name: props.name
})
else
return child
})
}
//父组件
function RadioGroup(props) {
return (

{renderChildren(props)}

)
}
function App() {
return (

 <RadioGroup name="hello"><br />
      <RadioOption label="选项一" value="1" /><br />
      <RadioOption label="选项二" value="2" /><br />
      <RadioOption label="选项三" value="3" /><br />
    </RadioGroup><br />



)
```

```
}
export default App;
```

面试题 18. Redux 中间件是怎么拿到store 和 action? 然后怎么处理?

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

redux中间件本质就是一个函数柯里化。redux applyMiddleware Api 源码中每个middleware 接受2个参数， Store 的getState 函数和dispatch 函数，分别获得store和action，最终返回一个函数。该函数会被传入 next 的下一个 middleware 的 dispatch 方法，并返回一个接收 action 的新函数，这个函数可以直接调用 next（action），或者在其他需要的时刻调用，甚至根本不去调用它。调用链中最后一个 middleware 会接受真实的 store的 dispatch 方法作为 next 参数，并借此结束调用链。所以，middleware 的函数签名是（{ getState，dispatch }）=> next => action

面试题 19. React Hook 的使用限制有哪些?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

React Hooks 的限制主要有两条：

不要在循环、条件或嵌套函数中调用 Hook；
在 React 的函数组件中调用 Hook。
那为什么会有这样的限制呢？Hooks 的设计初衷是为了改进 React 组件的开发模式。在旧有的开发模式下遇到了三个问题。

组件之间难以复用状态逻辑。过去常见的解决方案是高阶组件、render props 及状态管理框架。
复杂的组件变得难以理解。生命周期函数与业务逻辑耦合太深，导致关联部分难以拆分。
人和机器都很容易混淆类。常见的有 this 的问题，但在 React 团队中还有类难以优化的问题，希望在编译优化层面做出一些改进。
这三个问题在一定程度上阻碍了 React 的后续发展，所以为了解决这三个问题，Hooks 基于函数组件开始设计。然而第三个问题决定了 Hooks 只支持函数组件。

那为什么不要在循环、条件或嵌套函数中调用 Hook 呢？因为 Hooks 的设计是基于数组实现。在调用时按顺序加入数组中，如果使用循环、条件或嵌套函数很有可能导致数组取值错位，执行错误的 Hook。当然，实质上 React 的源码里不是数组，是链表。

这些限制会在编码上造成一定程度的心智负担，新手可能会写错，为了避免这样的情况，可以引入 ESLint 的 Hooks 检查插件进行预防

面试题 20. 使用 React Router时，如何获取当前页面的路由或浏览器中地址栏中的地址?

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

在当前组件的 props中，包含 location属性对象，包含当前页面路由地址信息，在 match中存储当前路由的参数等数据信息。可以直接通过 this.props使用它们

面试题 21. React Hooks在平时开发中需要注意的问题和原因 ?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

```
<br />
（1）不要在循环，条件或嵌套函数中调用Hook，必须始终在 React函数的顶层使用Hook<br />
这是因为React需要利用调用顺序来正确更新相应的状态，以及调用相应的钩子函数。一旦在循环或条件分支语句中调用Hook，就容易导致调用顺序的不一致性
<br />
（2）使用useState时候，使用push，pop，splice等直接更改数组对象的坑<br />
使用push直接更改数组无法获取到新值，应该采用析构方式，但是在class里面不会有这个问题。代码示例：<br />
<br />
function Indicatorfilter() {<br />
  let [num,setNums] = useState([0,1,2,3])<br />
  const test = () => {<br />
    // 这里坑是直接采用push去更新num<br />
    // setNums(num)是无法更新num的<br />
    // 必须使用num = [...num ,1]<br />
    num.push(1)<br />
    // num = [...num ,1]<br />
    setNums(num)<br />
  }<br />
return (<br />
    <div className='filter'><br />
```

```
      <div onClick={test}>测试</div><br />
        <div><br />
          {num.map((item,index) => (<br />
<div key={index}>{item}</div><br />
          ))}      </div><br />
    </div><br />
  )<br />
}<br />
<br />
class Indicatorfilter extends React.Component<any,any>{<br />
  constructor(props:any){<br />
      super(props)<br />
      this.state = {<br />
          nums:[1,2,3]<br />
      }<br />
      this.test = this.test.bind(this)<br />
  }<br />
<br />
  test(){<br />
      // class采用同样的方式是没有问题的<br />
      this.state.nums.push(1)<br />
      this.setState({<br />
          nums: this.state.nums<br />
      })<br />
  }<br />
<br />
  render(){<br />
      let {nums} = this.state<br />
      return(<br />
          <div><br />
              <div onClick={this.test}>测试</div><br />
                  <div><br />
                      {nums.map((item:any,index:number) => (<br />
                         <div key={index}>{item}</div><br />
                      ))}<br />
              </div><br />
          </div><br />
<br />
      )<br />
  }<br />
}<br />
<br />
（3）useState设置状态的时候，只有第一次生效，后期需要更新状态，必须通过useEffect<br />
<br />
TableDeail是一个公共组件，在调用它的父组件里面，我们通过set改变columns的值，以为传递给TableDeail 的 columns是最新的值，所以tabCol
<br />
const TableDeail = ({    columns,}:TableData) => {<br />
    const [tabColumn, setTabColumn] = useState(columns) <br />
}<br />
<br />
// 正确的做法是通过useEffect改变这个值<br />
const TableDeail = ({    columns,}:TableData) => {<br />
    const [tabColumn, setTabColumn] = useState(columns) <br />
    useEffect(() =>{setTabColumn(columns)},[columns])<br />
}<br />
<br />
（4）善用useCallback<br />
父组件传递给子组件事件句柄时，如果我们没有任何参数变动可能会选用useMemo。但是每一次父组件渲染子组件即使没变化也会跟着渲染一次。<br />
<br />
（5）不要滥用useContext<br />
可以使用基于 useContext 封装的状态管理工具。<br />
<xmp></p></div>
                                            </div>
                                        </td>
                    </tr>
                        </table>
                                            <table width="100%" border="0" cellpadding="4" cellspacing="1" styl
                                    <tr >
                                        <td  colspan="2" >
                                          
                                        <span style="font-size:14px; font-style:normal; font-weight:600; colo
```

```
                                                    </td>
                                    </tr>
                                    <tr height="35px">
                                      <td width="66%"><div style=" margin-left:40px;margin-right:45px; color:#
                                          推荐指数：
                                                                                    <font color="#0066FF">中级<,
                                          原理题
                                    </tr>
                                   <tr>
                                      <td colspan="2"  height="10px;">
                                         <div  style="letter-spacing:1px; line-height:22px;text-align: justify
                                                      <div style=" padding-left:10px;padding-right:5px; paddi
<br />
（1）获取state<br />
connect 通过 context获取 Provider 中的 store，通过 store.getState() 获取整个store tree 上所有state<br />
<br />
（2）包装原组件<br />
将state和action通过props的方式传入到原组件内部 wrapWithConnect 返回一个 ReactComponent 对 象 Connect，Connect 重 新 render
<br />
（3）监听store tree变化<br />
connect缓存了store tree中state的状态，通过当前state状态 和变更前 state 状态进行比较，从而确定是否调用 this.setState()方法触发Co
                                          </div>
                                     </td>
                    </tr>
                         </table>
                                          <table width="100%" border="0" cellpadding="4" cellspacing="1" styl
                                    <tr >
                                         <td  colspan="2" >
                                         &amp;nbsp;&amp;nbsp;
                                         <span style="font-size:14px; font-style:normal; font-weight:600; colo
                                                                    </td>
                                    </tr>
                                    <tr height="35px">
                                      <td width="66%"><div style=" margin-left:40px;margin-right:45px; color:#
                                          推荐指数：
                                                                                    <font color="#0066FF">中级<,
                                          原理题
                                    </tr>
                                   <tr>
                                      <td colspan="2"  height="10px;">
                                         <div  style="letter-spacing:1px; line-height:22px;text-align: justify
                                                      <div style=" padding-left:10px;padding-right:5px; paddi
可以为应用程序的任何部分启用严格模式。例如：<br />
<br />
import React from 'react';<br />
function ExampleApplication() {<br />
  return (<br />
   <xmp> <div><br />
      <Header /><br />
      <React.StrictMode>        <br />
        <div><br />
          <ComponentOne /><br />
          <ComponentTwo /><br />
        </div><br />
      </React.StrictMode>      <br />
      <Footer /><br />
    </div>


);
}
```

在上述的示例中，不会对 Header 和 Footer 组件运行严格模式检查。但是，ComponentOne 和 ComponentTwo 以及它们的所有后代元素都将进行检查。

StrictMode 目前有助于：

识别不安全的生命周期
关于使用过时字符串 ref API 的警告
关于使用废弃的 findDOMNode 方法的警告
检测意外的副作用
检测过时的 context API

面试题 24. ReactNative中，如何解决8081端口号被占用而提示无法访问的问题?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

运行 react-native start时添加参数port 8082；在 package.json中修改“scripts”中的参数，添加端口号；修改项目下的 node_modules\react-native\local- cli\server\server.js文件配置中的 default端口

面试题 25. State 是怎么注入到组件的，从 reducer 到组件经历了什么样的过程？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

通过connect和mapStateToProps将state注入到组件中：

```
import { connect } from 'react-redux'
import { setVisibilityFilter } from '@/reducers/Todo/actions'
import Link from '@/containers/Todo/components/Link'

const mapStateToProps = (state, ownProps) => ({
active: ownProps.filter === state.visibilityFilter
})

const mapDispatchToProps = (dispatch, ownProps) => ({
setFilter: () => {
dispatch(setVisibilityFilter(ownProps.filter))
}
})

export default connect(
mapStateToProps,
mapDispatchToProps
)(Link)
```

上面代码中，active就是注入到Link组件中的状态。 mapStateToProps（state，ownProps）中带有两个参数，含义是：

state-store管理的全局状态对象，所有都组件状态数据都存储在该对象中。
ownProps 组件通过props传入的参数。
reducer 到组件经历的过程：

reducer对action对象处理，更新组件状态，并将新的状态值返回store。
通过connect（mapStateToProps，mapDispatchToProps）（Component）对组件 Component进行升级，此时将状态值从store取出并作为props参数传递到组件。

面试题 26. React中如何处理事件?

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

为了解决跨浏览器的兼容性问题，SyntheticEvent 实例将被传递给你的事件处理函数，SyntheticEvent是 React 跨浏览器的浏览器原生事件包装器，它还拥有和浏览器原生事件相同的接口，包括 stopPropagation() 和 preventDefault()。

比较有趣的是，React 实际上并不将事件附加到子节点本身。React 使用单个事件侦听器侦听顶层的所有事件。这对性能有好处，也意味着 React 在更新 DOM 时不需要跟踪事件监听器

面试题 27. React state和props区别是什么？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

（1）props
props是一个从外部传进组件的参数，主要作为就是从父组件向子组件传递数据，它具有可读性和不变性，只能通过外部组件主动传入新的props来重新渲染子组件，否则子组件的props以及展现形式不会改变。
（2）state
state的主要作用是用于组件保存、控制以及修改自己的状态，它只能在constructor中初始化，它算是组件的私有属性，不可通过外部访问和修改，只能通过组件内部的this.setState来修改，修改state属性会导致组件的重新渲染。
（3）区别
props 是传递给组件的（类似于函数的形参），而state 是在组件内被组件自己管理的（类似于在一个函数内声明的变量）。

props 是不可修改的，所有 React 组件都必须像纯函数一样保护它们的 props 不被更改。state 是在组件中创建的，一般在 constructor中初始化 state。state 是多变的、可以修改，每次setState都异步更新的

面试题 28. 简述什么是React 高阶组件?

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

一、定义
高阶组件（HOC）是 React 中用于复用组件逻辑的一种高级技巧。HOC 自身不是 React API 的一部分，它是一种基于 React 的组合特性而形成的设计模式。具体而言，高阶组件是参数为组件，返回值为新组件的函数，如：

```
const NewComponent = higherOrderComponent(OldComponent);
```

二、例子
高阶组件是一个函数（而不是组件），它接受一个组件作为参数，返回一个新的组件。这个新的组件会使用你传给它的组件作为子组件，我们可以看个例子来进一步理解一下。

假设在src/wrapWithLoadData.js 文件中写一个HOC，要求NewComponent 会根据第二个参数 name 在挂载阶段从 localStorage 加载数据，并且 setState 到自己的 state.data 中，而渲染的时候将 state.data 通过 props.data 传给 WrappedComponent，如下：

```
import React, { Component } from 'react';
export default (WrappedComponent, name) => {
class NewComponent extends Component {
constructor () {
super()
this.state = { data: null }
}

componentWillMount () {
let data = localStorage.getItem(name)
this.setState({ data })
}

render () {
return
}
}
return NewComponent;
}
```

假如有一个组件的需求是挂载的时候从 localStorage 里面加载 username 字段作为

```
<input />
```

的 value 值，现在有了 wrapWithLoadData，我们可以很容易地做到这件事情。只需要定义一个非常简单的 InputWithUserName，它会把 props.data 作为

```
<input />
```

的 value 值。然把这个组件和 'username' 传给 wrapWithLoadData，wrapWithLoadData 会返回一个新的组件，我们用这个新的组件覆盖原来的 InputWithUserName，然后再导出去模块，我们可以在src/inputWithUserName.js 文件中这样写：

```
import wrapWithLoadData from './wrapWithLoadData';

class InputWithUserName extends Component {
render () {
return {this.props.data}
}
}

InputWithUserName = wrapWithLoadData(InputWithUserName, 'username');
export default InputWithUserName;
```

这个新的组件挂载的时候会先去 localStorage 加载数据，渲染的时候再通过 props.data 传给真正的 InputWithUserName，别人用这个组件的时候实际是用了被加工过的组件：

```
import InputWithUserName from './InputWithUserName';

class Index extends Component {
render () {
return (

用户名：

)
}
}
```

如果现在我们需要另外一个文本输入框组件，它也需要 localStorage 加载 'content' 字段的数据。我们只需要定义一个新的 TextareaWithContent：

```
import wrapWithLoadData from './wrapWithLoadData';
class TextareaWithContent extends Component {
render () {
return <br />
  }<br />
```