基本知识

1. 区分Real DOM和Virtual DOM

Real DOM	Virtual DOM
更新缓慢。	更新更快。
可以直接更新 HTML。	无法直接更新 HTML。
如果元素更新，则创建新DOM。	如果元素更新，则更新 JSX 。
DOM操作代价很高。	DOM 操作非常简单。
消耗的内存较多。	很少的内存消耗。

2. 什么是React?

- React 是 Facebook 在 2011 年开发的前端 JavaScript 库。
- 它遵循基于组件的方法，有助于构建可重用的UI组件。
- 它用于开发复杂和交互式的 Web 和移动 UI。
- 尽管它仅在 2015 年开源，但有一个很大的支持社区。

3. React有什么特点?

React的主要功能如下：

1. 它使用虚拟DOM 而不是真正的DOM。
2. 它可以用服务器端渲染。
3. 它遵循单向数据流或数据绑定。

4. 列出React的一些主要优点。

React的一些主要优点是：

1）它提高了应用的性能

2）可以方便地在客户端和服务器端使用

3）由于 JSX，代码的可读性很好

4）React 很容易与 Meteor，Angular 等其他框架集成

5）使用React，编写UI测试用例变得非常容易

5. React有哪些限制?

React的限制如下:

1）React 只是一个库，而不是一个完整的框架

2）它的库非常庞大，需要时间来理解

3）新手程序员可能很难理解

4）编码变得复杂，因为它使用内联模板和 JSX

6. 什么是JSX?

JSX 是JavaScript XML 的简写。是 React 使用的一种文件，它利用 JavaScript 的表现力和类似 HTML 的模板语法。这使得 HTML 文件非常容易理解。此文件能使应用非常可靠，并能够提高其性能。下面是JSX的一个例子:

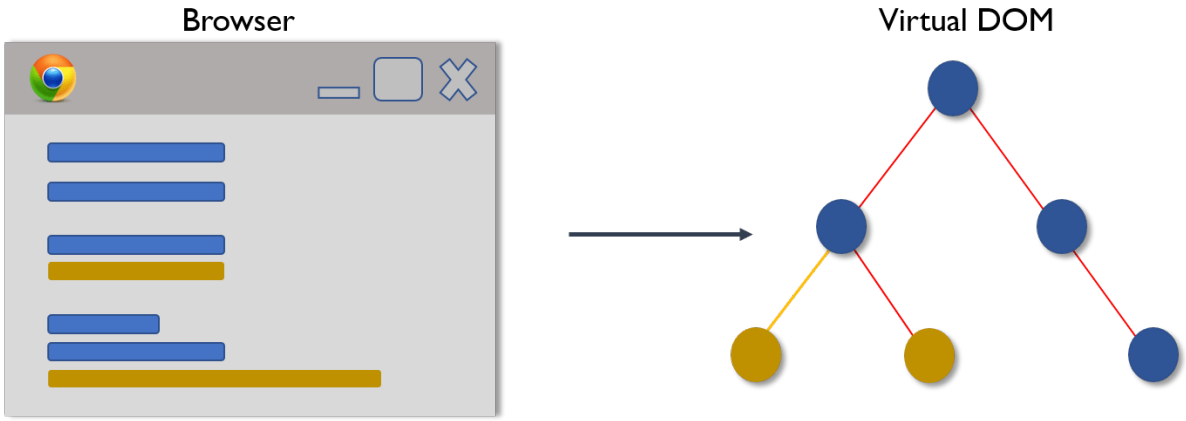
```
render(){
    return(
        <div>
            <h1> Hello World from Edureka!!</h1>
        </div>
    );
}
```

7. 你了解 Virtual DOM 吗？解释一下它的工作原理。

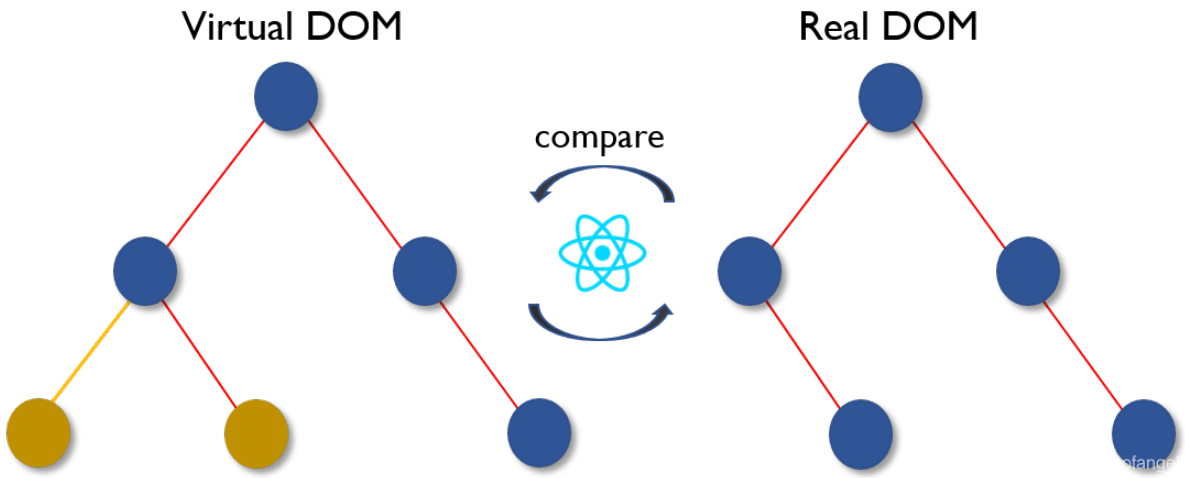
Virtual DOM 是一个轻量级的 JavaScript 对象，它最初只是 real DOM 的副本。它是一个节点树，它将元素、它们的属性和内容作为对象及其属性。 React 的渲染函数从 React 组件中创建一个节点树。然后它响应数据模型中的变化来更新该树，该变化是由用户或系统完成的各种动作引起的。

Virtual DOM 工作过程有三个简单的步骤。

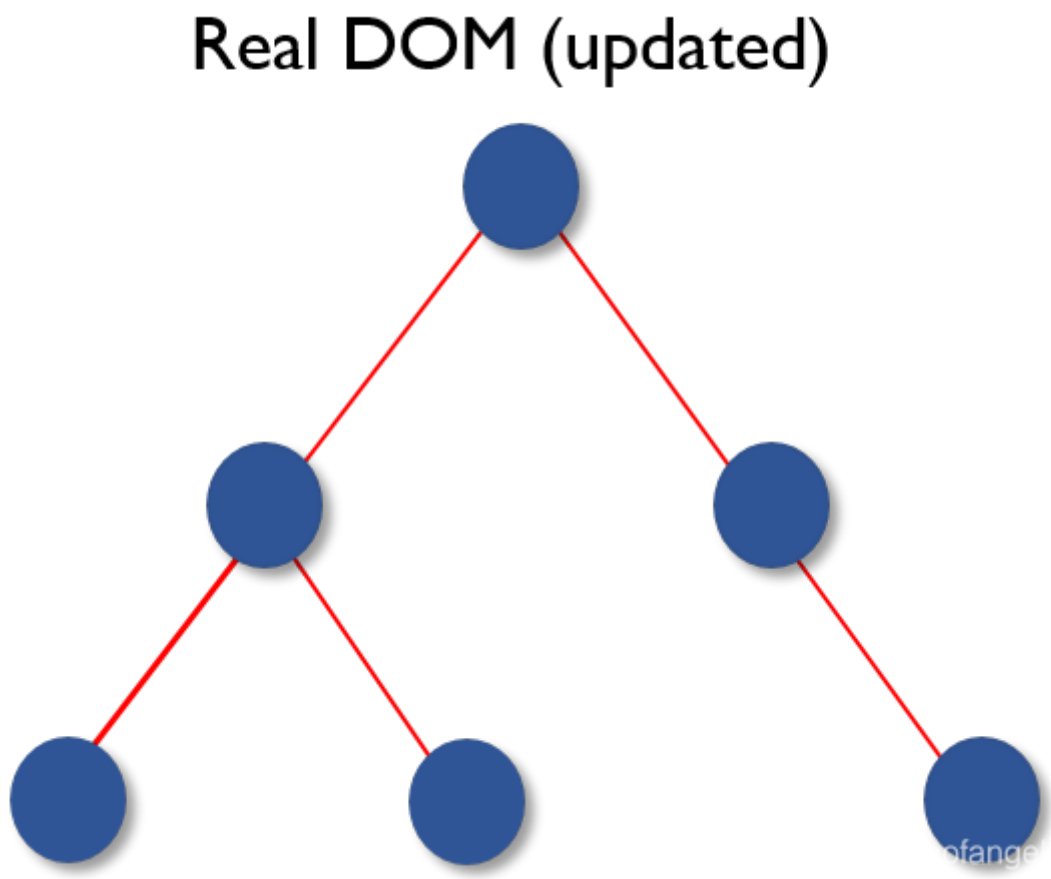
1）每当底层数据发生改变时，整个 UI 都将在 Virtual DOM 描述中重新渲染。



2）然后计算之前 DOM 表示与新表示的之间的差异。



3）完成计算后，将只用实际更改的内容更新 real DOM。



8. 为什么浏览器无法读取JSX?

浏览器只能处理 JavaScript 对象，而不能读取常规 JavaScript 对象中的 JSX。所以为了使浏览器能够读取 JSX，首先，需要用像 Babel 这样的 JSX 转换器将 JSX 文件转换为 JavaScript 对象，然后再将其传给浏览器。

9. 与 ES5 相比，React 的 ES6 语法有何不同?

以下语法是 ES5 与 ES6 中的区别：

1）require 与 import

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

// ES6
import React from 'react';
```

2）export 与 exports

```
// ES5
module.exports = Component;

// ES6
export default Component;
```

3）component 和 function

```
// ES5
var MyComponent = React.createClass({
    render: function() {
        return
            <h3>Hello Edureka!</h3>;
    }
});

// ES6
class MyComponent extends React.Component {
    render() {
        return
            <h3>Hello Edureka!</h3>;
    }
}
```

4）props

```
// ES5
var App = React.createClass({
    propTypes: { name: React.PropTypes.string },
    render: function() {
        return
            <h3>Hello, {this.props.name}!</h3>;
    }
});

// ES6
class App extends React.Component {
    render() {
        return
            <h3>Hello, {this.props.name}!</h3>;
    }
}
```

5）state

```
// ES5
var App = React.createClass({
    getInitialState: function() {
        return { name: 'world' };
    },
    render: function() {
        return
            <h3>Hello, {this.state.name}!</h3>;
    }
});

// ES6
class App extends React.Component {
    constructor() {
        super();
        this.state = { name: 'world' };
    }
    render() {
        return
            <h3>Hello, {this.state.name}!</h3>;
    }
}
```

10. React与Angular有何不同?

主题	React	Angular
体系结构	只有 MVC 中的 View	完整的 MVC
渲染	可以在服务器端渲染	客户端渲染
DOM	使用 virtual DOM	使用 real DOM
数据绑定	单向数据绑定	双向数据绑定
调试	编译时调试	运行时调试
作者	Facebook	Google

11. React如何性能优化

- 充分利用shouldComponentUpdate函数，不过这需要你的组件尽量最小化，如果当前组件数据过于复杂，其实是很难优化的。
- 给你的DOM遍历上加上唯一的key,注意尽量不要用index,因为如果你新DOM中删了某一个节点，它会重新排列index，那跟原来同层级一比就都会完全不一样，而重新渲染了，所以最好使用id值什么的作key值。
- 能用const声明的就用const。

- DOM里少用箭头函数，当然其实要传参时也还是得用。再者，函数bind尽量写在constructor，避免每次render重新bind。
- 减少对真实DOM的操作。
- 如果是用webpack搭建环境的话，当一个包过大加载过慢时，可分打成多个包来优化。

12. react与vue的对比

相同点：

- 都用虚拟DOM实现快速渲染
- 都是轻量级框架
- 现在 vue 也在渐渐吸收 react 中的一些语法，比如 JSX 语法，类式声明写法等

不同点：

- React属于单向数据流——MVC模式，vue则属于双向——MVVM模式。
- react兼容性比vue好，vue不兼容IE8.
- react采用JSX语法，vue采用的则是html模板语法。
- vue的css可以有组件的私有作用域，react则没有。
- react比vue好的另一点是，它是团队维护，而vue属于个人，一般来说，大型项目更倾向于react，小型则用vue，当然这也不是绝对。

13. 使用React Hooks有什么优势?

hooks 是react 16.8 引入的特性，他允许你在不写class的情况下操作state 和react的其他特性。

hooks 只是多了一种写组件的方法，使编写一个组件更简单更方便，同时可以自定义hook把公共的逻辑提取出来，让逻辑在多个组件之间共享。

Hook 是什么

Hook 是什么？ Hook 是一个特殊的函数，它可以让你“钩入” React 的特性。例如，useState 是允许你在 React 函数组件中添加 state 的 Hook。稍后我们将学习其他 Hook。

什么时候我会用 Hook？ 如果你在编写函数组件并意识到需要向其添加一些 state，以前的做法是必须将其它转化为 class。现在你可以在现有的函数组件中使用 Hook。

ReactHooks的优点

- 无需复杂的DOM结构
- 简洁易懂

14. React中的useState?

案例:

```
import { useState } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  )
}
```

语法:

```
function useState<S>(initialState: S | (() => S)): [S,
Dispatch<SetStateAction<S>>];
```

其中 state 是他的值, setState 是用来设置值的函数, initialState 是初始值

useState-initialState

该初始值可以接受任何参数,但是记得当他接受为一个函数时,就变成了 `Lazy initialization` (延迟初始化)

该函数返回值即为initialState

```
const [count, setCount] = useState(0);

const [count, setCount] = useState(()=>0);
// 这两种初始化方式 是相等的,但是在函数为初始值时会被执行一次

const [count, setCount] = useState(()=>{
    console.log('这里只会在初始化的时候执行')
    // class 中的 constructor 的操作都可以移植到这里
    return 0
});
// 当第一次执行完毕后 就和另一句的代码是相同的效果了
```

useState-setState

也许很多人 在使用 class 的 setState 时候,会经常使用他的回调函数,

但是这里很遗憾,他只接受新的值,如果想要对应的回调,可以使用useEffect,这个问题等会会提供一个跳转链接

React 组件

1. 你理解“在React中，一切都是组件”这句话。

组件是 React 应用 UI 的构建块。这些组件将整个 UI 分成小的独立并可重用的部分。每个组件彼此独立，而不会影响 UI 的其余部分。

2. 解释 React 中 render() 的目的。

每个React组件强制要求必须有一个 **render()**。它返回一个 React 元素，是原生 DOM 组件的表示。如果需要渲染多个 HTML 元素，则必须将它们组合在一个封闭标记内，例如 `<form>`、`<group>`、`<div>` 等。此函数必须保持纯净，即必须每次调用时都返回相同的结果。

3. 如何将两个或多个组件嵌入到一个组件中?

可以通过以下方式将组件嵌入到一个组件中：

```
class MyComponent extends React.Component{
    render(){
        return(
            <div>
                <h1>Hello</h1>
                <Header/>
            </div>
        );
    }
}
class Header extends React.Component{
    render(){
        return
            <h1>Header Component</h1>
   };
}
ReactDOM.render(
    <MyComponent/>, document.getElementById('content')
);
```

4. 什么是 Props?

Props 是 React 中属性的简写。它们是只读组件，必须保持纯，即不可变。它们总是在整个应用中从父组件传递到子组件。子组件永远不能将 prop 送回父组件。这有助于维护单向数据流，通常用于呈现动态生成的数据。

5. state 和 props有什么区别?

state 和 props都是普通的JavaScript对象。尽管它们两者都具有影响渲染输出的信息，但它们在组件方面的功能不同。即

- `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。
- `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。

6. React中的状态是什么？它是如何使用的?

状态是 React 组件的核心，是数据的来源，必须尽可能简单。基本上状态是确定组件呈现和行为的对象。与props 不同，它们是可变的，并创建动态和交互式组件。可以通过 `this.state()` 访问它们。

7. 区分状态和 props

条件	State	Props
从父组件中接收初始值	Yes	Yes
父组件可以改变值	No	Yes
在组件中设置默认值	Yes	Yes
在组件的内部变化	Yes	No
设置子组件的初始值	Yes	Yes
在子组件的内部更改	No	Yes

8. 如何更新组件的状态?

可以用 `this.setState()` 更新组件的状态。

```
class MyComponent extends React.Component {
    constructor() {
        super();
        this.state = {
            name: 'Maxx',
            id: '101'
        }
    }
    render()
        {
            setTimeout(()=>{this.setState({name:'Jaeha', id:'222'})},2000)
            return (
                <div>
                    <h1>Hello {this.state.name}</h1>
                    <h2>Your Id is {this.state.id}</h2>
```

```
                </div>
            );
        }
    }
ReactDOM.render(
    <MyComponent/>, document.getElementById('content')
);
```

9. 为什么不直接更新state状态?

如果进行如下方式更新状态，那么它将不会重新渲染组件。

```
//Wrong
    This.state.message =”Hello world”;
```

而是使用 `setState()` 方法。它计划对组件状态对象的更新。状态改变时，组件通过重新渲染做出响应

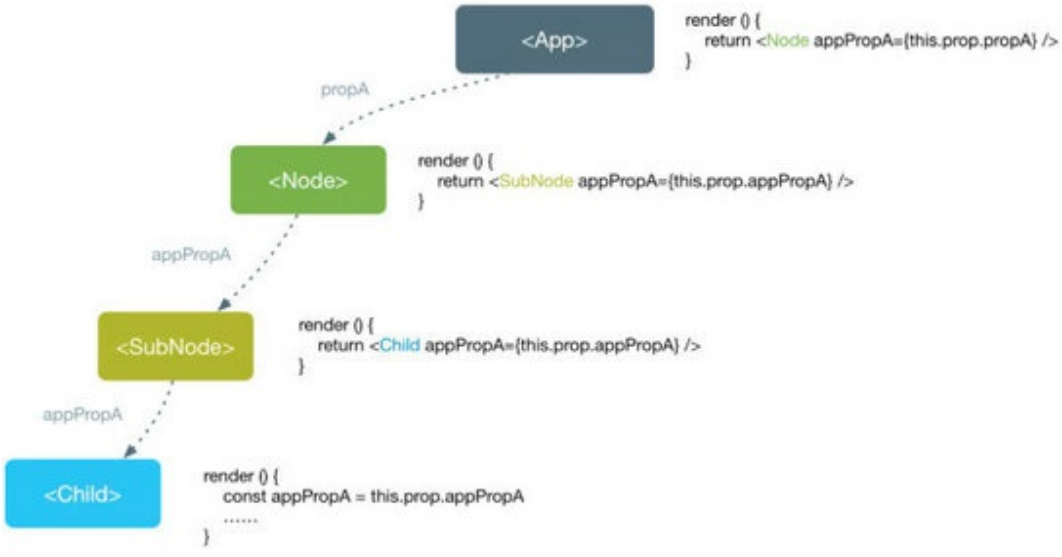
```
//Correct
 This.setState({message: ‘Hello World’});
```

注意：可以分配状态的唯一位置是构造函数。

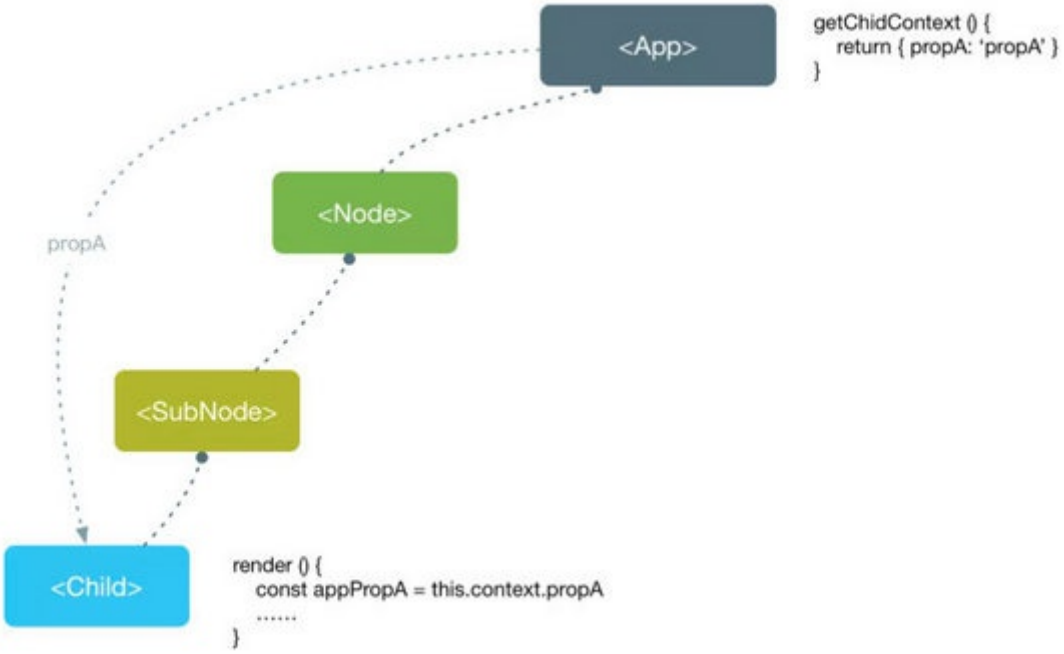
10. React context是什么?

React文档官网并未对 `Context` 给出“是什么”的定义，更多是描述使用的 `Context` 的场景，以及如何使用 `Context` 。

简单说就是，当你不想在组件树中通过逐层传递 `props` 或者 `state` 的方式来传递数据时，可以使用 `Context` 来实现 **跨层级** 的组件数据传递。



使用props或者state传递数据，数据自顶下流。



使用 `Context` ，可以跨越组件进行数据传递。

11. constructor中super与props参数一起使用的目的是什么?

在调用方法之前，子类构造函数无法使用 **this** 引用 `super()` 。

在ES6中，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`

使用props:

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        console.log(this.props);  // Prints { name: 'sudheer',age: 30 }
    }
}
```

不使用props:

```
class MyComponent extends React.Component {
    constructor(props) {
        super();
        console.log(this.props); // Prints undefined
        // But Props parameter is still available
        console.log(props); // Prints { name: 'sudheer',age: 30 }
    }

    render() {
        // No difference outside constructor
        console.log(this.props) // Prints { name: 'sudheer',age: 30 }
    }
}
```

上面的代码片段揭示了this.props行为仅在构造函数中有所不同。外部构造函数相同。

12. React 中的箭头函数是什么？怎么用?

箭头函数（=>）是用于编写函数表达式的简短语法。这些函数允许正确绑定组件的上下文，因为在 ES6 中默认下不能使用自动绑定。使用高阶函数时，箭头函数非常有用。

```
//General way
render() {
    return(
        <MyInput onChange = {this.handleChange.bind(this) } />
    );
}
//With Arrow Function
render() {
    return(
        <MyInput onChange = { (e)=>this.handleOnChange(e) } />
    );
}
```

13. 区分有状态和无状态组件。

有状态组件	无状态组件
在内存中存储有关组件状态变化的信息	计算组件的内部的状态
有权改变状态	无权改变状态
包含过去、现在和未来可能的状态变化情况	不包含过去，现在和未来可能发生的状态变化情况
接受无状态组件状态变化要求的通知，然后将 props 发送给他们。	从有状态组件接收 props 并将其视为回调函数。

14. React组件生命周期的阶段是什么?

React 组件的生命周期有三个不同的阶段：

1）初始渲染阶段：这是组件即将开始其生命之旅并进入 DOM 的阶段。

2）更新阶段：一旦组件被添加到 DOM，它只有在 prop 或状态发生变化时才可能更新和重新渲染。这些只发生在这个阶段。

3）卸载阶段：这是组件生命周期的最后阶段，组件被销毁并从 DOM 中删除。

15. 详细解释 React 组件的生命周期方法。

一些最重要的生命周期方法是：

1）***componentWillMount*()** – 在渲染之前执行，在客户端和服务器端都会执行。

2）***componentDidMount*()** – 仅在第一次渲染后在客户端执行。

3）***componentWillReceiveProps*()** – 当从父类接收到 props 并且在调用另一个渲染器之前调用。

4）***shouldComponentUpdate*()** – 根据特定条件返回 true 或 false。如果你希望更新组件，请返回**true** 否则返回 **false**。默认情况下，它返回 true。

5）***componentWillUpdate*()** – 在 DOM 中进行渲染之前调用。

6）***componentDidUpdate*()** – 在渲染发生后立即调用。

7）***componentWillUnmount*()** – 从 DOM 卸载组件后调用。用于清理内存空间。

16. React中的事件是什么?

在 React 中，事件是对鼠标悬停、鼠标单击、按键等特定操作的触发反应。处理这些事件类似于处理 DOM 元素中的事件。但是有一些语法差异，如：

1）用驼峰命名法对事件命名而不是仅使用小写字母。

2）事件作为函数而不是字符串传递。

事件参数重包含一组特定于事件的属性。每个事件类型都包含自己的属性和行为，只能通过其事件处理程序访问。

17. 如何在React中创建一个事件?

```
class Display extends React.Component({
    show(evt) {
        // code
    }
    render() {
        // Render the div with an onClick prop (value is a function)
        return (
            <div onClick={this.show}>Click Me!</div>
        );
    }
});
```

18. React中的合成事件是什么?

合成事件是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。

19. 你对 React 的 refs 有什么了解?

Refs 是 React 中引用的简写。它是一个有助于存储对特定的 React 元素或组件的引用的属性，它将由组件渲染配置函数返回。用于对 render() 返回的特定元素或组件的引用。当需要进行 DOM 测量或向组件添加方法时，它们会派上用场。

```
class ReferenceDemo extends React.Component{
     display() {
         const name = this.inputDemo.value;
         document.getElementById('disp').innerHTML = name;
```

```
    }
render() {
    return(
          <div>
            Name: <input type="text" ref={input => this.inputDemo = input} />
            <button name="Click" onClick={this.display}>Click</button>

            <h2>Hello <span id="disp"></span> !!!</h2>
          </div>
    );
   }
 }
```

20. 列出一些应该使用 Refs 的情况。

以下是应该使用 refs 的情况：

- 需要管理焦点、选择文本或媒体播放时
- 触发式动画
- 与第三方 DOM 库集成

21. React中的refs作用是什么?

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。

我们可以为元素添加 ref 属性然后在回调函数中接受该元素在 DOM 树中的句柄，该值会作为回调函数的第一个参数返回：

```
class UnControlledForm extends Component {
  handleSubmit = () => {
    console.log("Input Value: ", this.input.value)
  }
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

上述代码中的 input 域包含了一个 ref 属性，该属性声明的回调函数会接收 input 对应的 DOM 元素，我们将其绑定到 this 指针以便在其他的类函数中使用。

另外值得一提的是，refs 并不是类组件的专属，函数式组件同样能够利用闭包暂存其值：

```
function CustomForm ({handleSubmit}) {
  let inputElement
  return (
    <form onSubmit={() => handleSubmit(inputElement.value)}>
      <input
        type='text'
        ref={(input) => inputElement = input} />
      <button type='submit'>Submit</button>
    </form>
  )
}
```

22. 如何创建refs?

Refs 是使用 **React.createRef()** 方法创建的，并通过 `ref` 属性添加到 React 元素上。为了在整个组件中使用 refs ， 只需将 *ref* 分配给构造函数中的实例属性

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

和:

```
class UserForm extends Component {
  handleSubmit = () => {
    console.log("Input Value is: ", this.input.value)
  }
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} /> // Access DOM input in handle
submit
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

我们还可以借助闭包在功能组件中使用它。

23. 如何模块化 React 中的代码?

可以使用 export 和 import 属性来模块化代码。它们有助于在不同的文件中单独编写组件。

```
//ChildComponent.jsx
export default class ChildComponent extends React.Component {
    render() {
        return(
              <div>
                <h1>This is a child component</h1>
              </div>
        );
    }
}

//ParentComponent.jsx
import ChildComponent from './childcomponent.js';
class ParentComponent extends React.Component {
    render() {
        return(
             <div>
                <App />
             </div>
        );
    }
}
```

24. 如何在 React 中创建表单

React 表单类似于 HTML 表单。但是在 React 中，状态包含在组件的 state 属性中，并且只能通过 `setState()` 更新。因此元素不能直接更新它们的状态，它们的提交是由 JavaScript 函数处理的。此函数可以完全访问用户输入到表单的数据。

```
handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
}

render() {
    return (
        <form onSubmit={this.handleSubmit}>
            <label>
                Name:
                <input type="text" value={this.state.value} onChange=
{this.handleSubmit} />
            </label>
            <input type="submit" value="Submit" />
        </form>
    );
}
```

25. 你对受控组件和非受控组件了解多少?

受控组件	非受控组件
没有维持自己的状态	保持着自己的状态
数据由父组件控制	数据由 DOM 控制
通过 props 获取当前值，然后通过回调通知更改	Refs 用于获取其当前值

26. 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。

27. 你能用HOC做什么?

HOC可用于许多任务，例如：

- 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

28. 什么是纯组件?

纯（Pure）组件是可以编写的最简单、最快的组件。它们可以替换任何只有 **render()** 的组件。这些组件增强了代码的简单性和应用的性能。

29. React 中 key 的重要性是什么?

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。这些 key 必须是唯一的数字或字符串，React 只是重新排序元素而不是重新渲染它们。这可以提高应用程序的性能。

30. 类组件和函数组件之间有什么区别?

- 类组件（**Class components**）
    - 无论是使用函数或是类来声明一个组件，它决不能修改它自己的 `props` 。
    - 所有 React 组件都必须是纯函数，并禁止修改其自身 `props` 。
    - React是单项数据流，父组件改变了属性，那么子组件视图会更新。
        - 属性 `props` 是外界传递过来的，状态 `state` 是组件本身的，状态可以在组件中任意修改
        - 组件的属性和状态改变都会更新视图。

- **函数组件（functional component）**
    - 函数组件接收一个单一的 `props` 对象并返回了一个React元素

区别

函数组件的性能比类组件的性能要高，因为类组件使用的时候要实例化，而函数组件直接执行函数取返回结果即可。为了提高性能，尽量使用函数组件。

31. 为什么类方法需要绑定?

在JavaScript中， `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的，例如：

```
class SubmitButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      isFormSubmitted: false
    };
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleSubmit() {
    this.setState({
      isFormSubmitted: true
    });
  }

  render() {
    return (
      <button onClick={this.handleSubmit}>Submit</button>
    )
  }
}
```

32. React中的StrictMode是什么?

React的StrictMode是一种帮助程序组件，可以帮助您编写更好的react组件，您可以使用包装一些组件， `<StrictMode />` 并且基本上可以：

- 验证内部组件是否遵循某些推荐做法，如果不在控制台中，则会发出警告。
- 验证不赞成使用的方法，如果使用了严格模式，则会在控制台中警告您。
- 通过识别潜在风险来帮助您预防某些副作用。

React Redux

1. MVC框架的主要问题是什么?

以下是MVC框架的一些主要问题:

- 对 DOM 操作的代价非常高
- 程序运行缓慢且效率低下
- 内存浪费严重
- 由于循环依赖性，组件模型需要围绕 models 和 views 进行创建

2. 解释一下 Flux

Flux 是一种强制单向数据流的架构模式。它控制派生数据，并使用具有所有数据权限的中心 store 实现多个组件之间的通信。整个应用中的数据更新必须只能在此处进行。 Flux 为应用提供稳定性并减少运行时的错误。

3. 什么是Redux?

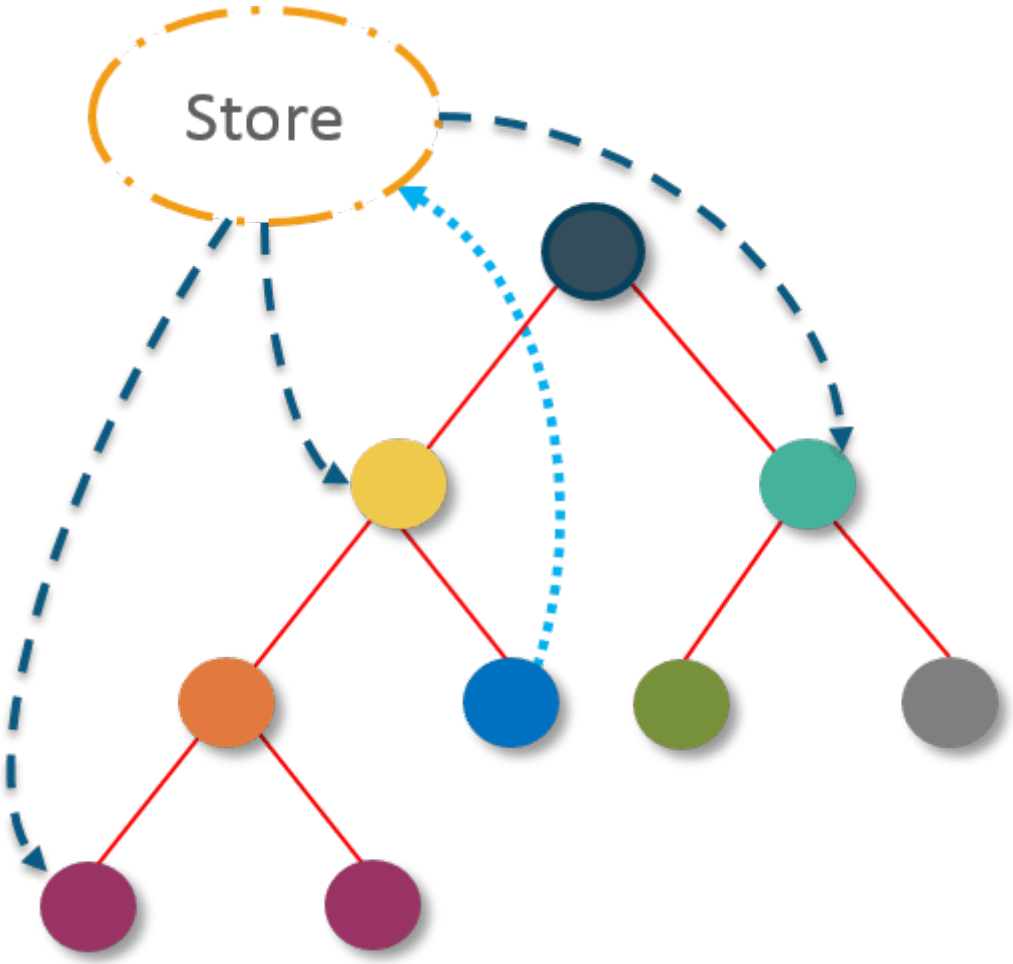
Redux 是当今最热门的前端开发库之一。它是 JavaScript 程序的可预测状态容器，用于整个应用的状态管理。使用 Redux 开发的应用易于测试，可以在不同环境中运行，并显示一致的行为。

4. Redux遵循的三个原则是什么?

1）***单一事实来源：***整个应用的状态存储在单个 store 中的对象/状态树里。单一状态树可以更容易地跟踪随时间的变化，并调试或检查应用程序。

2）***状态是只读的：***改变状态的唯一方法是去触发一个动作。动作是描述变化的普通 JS 对象。就像 state 是数据的最小表示一样，该操作是对数据更改的最小表示。

3）***使用纯函数进行更改：***为了指定状态树如何通过操作进行转换，你需要纯函数。纯函数是那些返回值仅取决于其参数值的函数。



5. 你对“单一事实来源”有什么理解?

Redux 使用 “Store” 将程序的整个状态存储在同一个地方。因此所有组件的状态都存储在 Store 中，并且它们从 Store 本身接收更新。单一状态树可以更容易地跟踪随时间的变化，并调试或检查程序。

6. 列出 Redux 的组件。

Redux 由以下组件组成：

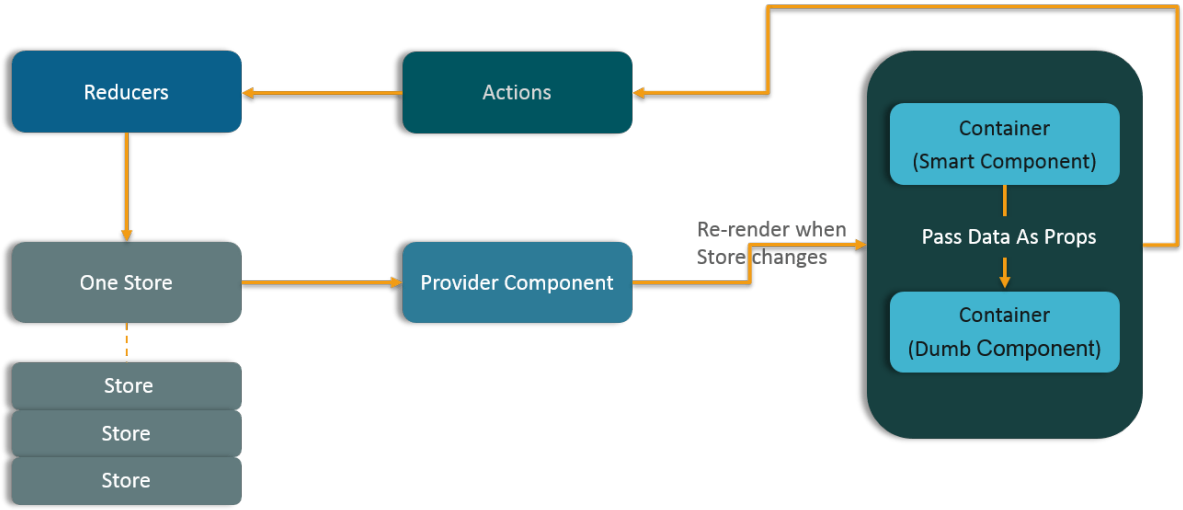
1）**Action** – 这是一个用来描述发生了什么事情的对象。

2）**Reducer** – 这是一个确定状态将如何变化的地方。

3）**Store** – 整个程序的状态/对象树保存在Store中。

4）**View** – 只显示 Store 提供的数据。

7. 数据如何通过 Redux 流动?



8. 如何在 Redux 中定义 Action?

React 中的 Action 必须具有 type 属性，该属性指示正在执行的 ACTION 的类型。必须将它们定义为字符串常量，并且还可以向其添加更多的属性。在 Redux 中，action 被名为 Action Creators 的函数所创建。以下是 Action 和Action Creator 的示例：

```
function addTodo(text) {
        return {
                type: ADD_TODO,
                 text
    }
}
```

9. 解释 Reducer 的作用。

Reducers 是纯函数，它规定应用程序的状态怎样因响应 ACTION 而改变。Reducers 通过接受先前的状态和 action 来工作，然后它返回一个新的状态。它根据操作的类型确定需要执行哪种更新，然后返回新的值。如果不需要完成任务，它会返回原来的状态。

10. Store 在 Redux 中的意义是什么?

Store 是一个 JavaScript 对象，它可以保存程序的状态，并提供一些方法来访问状态、调度操作和注册侦听器。应用程序的整个状态/对象树保存在单一存储中。因此，Redux 非常简单且是可预测的。我们可以将中间件传递到 store 来处理数据，并记录改变存储状态的各种操作。所有操作都通过 reducer 返回一个新状态。

11. Redux与Flux有何不同?

Flux	Redux
Store 包含状态和更改逻辑	Store 和更改逻辑是分开的
有多个 Store	只有一个 Store
所有 Store 都互不影响且是平级的	带有分层 reducer 的单一 Store
有单一调度器	没有调度器的概念
React 组件订阅 store	容器组件是有联系的
状态是可变的	状态是不可改变的

12. 简述flux 思想

Flux 的最大特点，就是数据的"单向流动"。

- 用户访问 View
- View发出用户的 Action
- Dispatcher 收到Action，要求 Store 进行相应的更新
- Store 更新后，发出一个"change"事件
- View 收到"change"事件后，更新页面

13. Redux 有哪些优点?

Redux 的优点如下：

- **结果的可预测性 -** 由于总是存在一个真实来源，即 store ，因此不存在如何将当前状态与动作和应用的其他部分同步的问题。
- **可维护性 -** 代码变得更容易维护，具有可预测的结果和严格的结构。
- **服务器端渲染 -** 你只需将服务器上创建的 store 传到客户端即可。这对初始渲染非常有用，并且可以优化应用性能，从而提供更好的用户体验。
- **开发人员工具 -** 从操作到状态更改，开发人员可以实时跟踪应用中发生的所有事情。
- **社区和生态系统 -** Redux 背后有一个巨大的社区，这使得它更加迷人。一个由才华横溢的人组成的大型社区为库的改进做出了贡献，并开发了各种应用。
- **易于测试 -** Redux 的代码主要是小巧、纯粹和独立的功能。这使代码可测试且独立。
- **组织 -** Redux 准确地说明了代码的组织方式，这使得代码在团队使用时更加一致和简单。

14. Redux有什么缺点

- 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。
- 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。

15. Redux的实现流程

用户页面行为触发一个`Action`，然后，`Store`自动调用`Reducer`，并且传入两个参数：当前 State 和收到的 Action。Reducer 会返回新的 State 。每当state更新之后，`view`会根据state触发重新渲染。

16. redux中间件的理解，以及用过哪些中间件

理解：中间件就是要对redux的store.dispatch方法做一些改造，以实现其他的功能。

背景：Redux 的基本做法，是用户发出 Action，Reducer 函数立刻算出新的 State，View 重新渲染，但这是做同步。

而如果有异步请求时，那就不能知道什么时候获取的数据有存进store里面，因此此时需要在请求成功时返回一个标识或状态，并在此时再触发action给reducer传值。
因此，为了解决异步的问题，就引入了中间件的概念。

作用： redux-thunk 帮助你统一了异步和同步 action 的调用方式，把异步过程放在 action 级别解决，对 component 调用没有影响。

17. 比较redux和vuex的区别

相同点：

- 数据驱动视图，提供响应式的视图组件
- 都有virtual DOM， 组件化开发，通过props参数进行父子组件数据的传递，都实现webComponents规范
- 都支持服务端渲染
- 都有native解决方案，reactnative（facebook团队） vs weex（阿里团队）

不同点：

- vuex是一个针对VUE优化的状态管理系统，而redux仅是一个常规的状态管理系统（Redux）与React框架的结合版本。
- 开发模式：React本身，是严格的view层，MVC模式；Vue则是MVVM模式的一种方式实现
- 数据绑定：Vue借鉴了angular，采取双向数据绑定的方式；React，则采取单向数据流的方式
- 数据更新：Vue采取依赖追踪，默认是优化状态：按需更新；
  React在则有两种选择：
  1）手动添加shouldComponentUpdate，来避免冗余的vdom，re-render的情况
  2）Components 尽可能都用 pureRenderMixin，然后采用 redux 结构 + Immutable.js
- 社区：react相比来讲还是要大于vue，毕竟背后支撑团队不同。
  facebook vs 个人！当然目前vue的增长速度是高于react的增速，不知道未来的发展趋势是如何。

React 路由

1. 什么是React 路由?

React 路由是一个构建在 React 之上的强大的路由库，它有助于向应用程序添加新的屏幕和流。这使 URL 与网页上显示的数据保持同步。它负责维护标准化的结构和行为，并用于开发单页 Web 应用。React 路由有一个简单的API。

2. 为什么React Router v4中使用 switch 关键字 ?

虽然 `<div>` 用于封装 Router 中的多个路由，当你想要仅显示要在多个定义的路线中呈现的单个路线时，可以使用 “switch” 关键字。使用时，`<switch>` 标记会按顺序将已定义的 URL 与已定义的路由进行匹配。找到第一个匹配项后，它将渲染指定的路径。从而绕过其它路线。

3. 为什么需要 React 中的路由?

Router 用于定义多个路由，当用户定义特定的 URL 时，如果此 URL 与 Router 内定义的任何 “路由” 的路径匹配，则用户将重定向到该特定路由。所以基本上我们需要在自己的应用中添加一个 Router 库，允许创建多个路由，每个路由都会向我们提供一个独特的视图

```
<switch>
    <route exact path='/' component={Home}/>
    <route path='/posts/:id' component={Newpost}/>
    <route path='/posts'   component={Post}/>
</switch>
```

4. 列出 React Router 的优点。

几个优点是:

1. 就像 React 基于组件一样，在 React Router v4 中，API 是 *'All About Components'*。可以将 Router 可视化为单个根组件（`<BrowserRouter>`），其中我们将特定的子路由（`<route>`）包起来。
2. 无需手动设置历史值：在 React Router v4 中，我们要做的就是将路由包装在 `<BrowserRouter>` 组件中。
3. 包是分开的：共有三个包，分别用于 Web、Native 和 Core。这使我们应用更加紧凑。基于类似的编码风格很容易进行切换。

5. React Router与常规路由有何不同?

主题	常规路由	React 路由
参与的页面	每个视图对应一个新文件	只涉及单个HTML页面
URL 更改	HTTP 请求被发送到服务器并且接收相应的 HTML 页面	仅更改历史记录属性
体验	用户实际在每个视图的不同页面切换	用户认为自己正在不同的页面间切换

6. react-router的实现原理

原理：实现URL与UI界面的同步。其中在react-router中，URL对应Location对象，而UI是由react components来决定的，这样就转变成location与components之间的同步问题。

优点：

- 风格: 与React融为一体,专为react量身打造，编码风格与react保持一致，例如路由的配置可以通过component来实现
- 简单: 不需要手工维护路由state，使代码变得简单
- 强大: 强大的路由管理机制，体现在如下方面
- 路由配置: 可以通过组件、配置对象来进行路由的配置
- 路由切换: 可以通过`<Link>` Redirect进行路由的切换
- 路由加载: 可以同步记载，也可以异步加载，这样就可以实现按需加载
- 使用方式: 不仅可以在浏览器端的使用，而且可以在服务器端的使用

缺点：API不太稳定，在升级版本的时候需要进行代码变动。

7. react-router 里的`<Link>`标签和`<a>`标签有什么区别

对比`<a>`,`Link`组件避免了不必要的重渲染