1.

前端的三大框架angular （Google Supported）， react （Facebook Supported）， vue。

React的优势：

使用虚拟DOM （操作React虚拟DOM, 比操作实际DOM花费少）

性能性很高。

解决各终端的问题包括PC，移动端问题

React的缺点：

需要插件的支持。

学习曲线高

P.s 网页的发展从以前的单网页开始，当数据进行变化或者用户点击进行交互以后，以前的网页需要整体的整份的网页进行更新。 后来为了简化，我们想要网页进行部分更新， 例如Jquery的插件的引入。 再后来随着交互的程度越来越高，我们想当数据变化的时候以最少代价自动更新部门网页，此时React技术出现。我们通过将网页进行控件化，并且和数据进行绑定，并且使用React的虚拟DOM技术，可以最快花费最少的改变网页。

2.

JavaScript库和同框架的区别。库是封装了原生的Javascript，还是使用原js的思想，例如JQuery, Zepto等。而框架则是从本质上修改了思想，以不同的方式去操作。

3. React有自己自定义的语法JSX，JSX语言可以允许将HTML，XML等一些其他语言混合Javascript一起写。并且JSX语言不能直接被识别，需要最终转化为Javascript语言才可以。（并且我们在JSX语法中写入的形似html标签，在经过编译以后，将变为javascript语法给浏览器，而不会因为像html语言而不变）

这个转化的过程，如果我们只是随手写一个小型的React,我们可以通过外置引入额外的JS文件来帮助我们完成这个过程，例如babel/browser.js.

如果是比较有规模的react app, 或者是标准做法应该是通过特定的构建工具，例如React推荐的webpack来有结构的进行打包转化处理。(其中Webpack配置中，我们便会配置jsx通过babel来进行转化为Javascript)

4.一些React的基本规则：

定义html标签的时候不需要引号， 即 let a = <div>hellow react!</div> 便可以。

在书写多便签的时候必须包含根元素，并且允许自由缩进，即是对缩进没有要求。

例如：

let a = <div>AAA</div><span>BBB</span> 🡪不合法，必须有根元素

let a = <div>

<div>AAA</div>

<span>BBB</span>

</div> 🡪 合法并且缩进被允许， 并且可以在根元素的前后加括号表示整体

书写**单标签**的时候，必须在结尾加上/

<img>不合法 一定要<img/>

<input>不合法 一定要<input/>

<br>不合法 一定要<br/>

指定class类型的时候， 要使用 className关键字， <div class=’aaa’></div>不合法，<div className=’aaa’></div>才合法

***在jsx中使用单花括号{}，将会认为花括号中的语法为javascript语法。JSX会对其进行渲染，运行求值。***

***例如 let b= <div>{a}</div> 🡪 a为js变量. 并且这个b所代表的不是普通的html标签，是React所构建的虚拟DOM.我们通常只在html标签中加入{}***

5.React开发模式

第一种是直接引入React.js启动React。并且引入react-dom.js启用虚拟DOM， 引入babel使得可以在<script>中写入jsx语言。在Html中引入的顺序必须为react.js, react-dom.js, babel里面的browser.js

另一种一种是脚手架模式基于webpack.

6. Bower是一个管理所有JS库和框架的工具，通过Bower可以下载所有的JS框架以及各个版本。Bower的使用需要Node.js的支持。所以要先安装Node.js然后再安装bower。 当成功安装Node.js以及bower.js以后在命令行中使用bower,或者node都可以被系统识别。

7.Bower的简明使用

Bower Info 展示包的各个版本信息

Bower Install 安装包#版本号

8. 使用Bower安装React.js的，将会将React.js框架需要的所有文件以文件夹的形式下载到路径中，而不是安装到系统中。并且React框架一般需要bower install react, bower install babel这两个包。

9.

一个重要的react渲染方法ReactDom.render(插件，目标)

例如：

<script type=”text/babel”> 🡪不使用text/javascript, 如果申明javascript那么浏览器将认为是javascript语法，将会报错。使用babel表示这不是普通的js语法

ReactDOM.render( 🡪 React渲染方法

<h1>hello world!</h1> 🡪 插入的内容

Documents.getElementById(‘xx’)) 🡪插入的目标

10.

Jsx支持style里面使用Json， 但是需要2层花括号。

例如 {{‘background’:’red’}}， 第一层的花括号是告诉jsx这将会是一个js代码，第二层的花括号则是js中的字典定义中的花括号用于指定json。

11.

Jsx中对控件事件的指定使用的是驼峰命名法（也就是除了第一个单词，以后的每个单词的首字母大写）

例如 onClick, onMouseOver

12.

JS中的两种面向对象

第一种是ES6规则之前的：不是严格的OOP语言，定义类的时候，通过Function关键字来定义,通过原型链的方法来实现继承

第二种是ES6规则，也就是可以使用class来定义类，然后在class中定义有初始化方法constructor，以及定义各种方法。使用extends关键字来继承等（并且在子类的constructor函数中必须调用super（）父类方法先）。使用new来新建对象。需要注意的是ES6虽然采用了类似于JAVA, C++这些class的关键字，使得更加容易被接受，但是这些语法相同于语法糖，真正Javascript底层上的实现还是通过原型链来实现的。

例子：

class Person {

constructor(name, age){ 🡪定义初始变量

this.name = name;

this.age = age;

}

say(){ 🡪 申明方法的时候不需要function关键字。定义为实例方法，需要实例化以后使用

console.log('my name is ' + this.name + ', ' + this.age + ' years old');

}

static print(){ 🡪定义静态方法，不需要实例化类便可以使用

return 'hello world';

}

}

🡪ES6中，没有静态属性，只有静态方法

🡪类中的方法互相调用，需要使用this.方法名()

🡪extends可以继承方法和私有属性

🡪React.js推荐使用ES6规则编写组件

13.

基本的例子使用React的组件

<div id=’app’></div> 🡪定义一个HTML用于渲染的元素

<script type=”text/babel”>

Class MyComponents extends React.Component{ 🡪 定义一个组件，并且**组件名的首字母一定要大写**， 并且最主要是继承React中的组件件

render(){ 🡪 render方法用于定义此组件渲染的内容

return (<div>Hellp React</div>) 🡪用于返回渲染的内容

}

}

ReactDom.render(<MyComponents/>, app) 🡪使用此组件对id为app的html元素进行渲染，并且单标签闭合

</script>

14.

Javascript中this的多变性

--普通函数中的this表示调用此函数时的对象，即是指向调用者。

--this指向的可变性。

--如果javascript内嵌在html中，例如onClick=”…”那么此时的调用者是Window和在外部写是一样的。

当在全局作用域时，this指向全局；

当在某个对象中使用this时，this指向该对象；

当把某个对象的方法赋值给另外一个对象时，this会指向后一个对象。

例如:

var a = {

user:"管理员",

fn:function(){

console.log(this.user);

}

}

var b = a.fn;

b(); //undefined

当b=a.fn的时候，我们把a对象的方法fn赋值给方法对象b, 在执行的时候，由于是window对象调用的，所以this将向全局，也就是window对象，而window对象中没有user, 所以为undefined.

例子2:

var a =5;

var b={

a : 12,

show(){

alert(this.a);

}

};

b.show(); 🡪为12因为是b调用，那this就是b,所以为12

c = b.show;

c(); 🡪 现在是window调用，所以为5

c = b.show.bind(b);

c(); 🡪现在为window调用，但是改变了this为b,所以为12

--this的使用场合有：在全局环境中使用；在构造函数中使用，在对象的方法中使用。

--this的使用注意点，最重要的一点就是要避免多层嵌套使用this对象。

为了固定this的指定，我们可以使用call, apply 以及bind方法来规定this的指向。

Call🡪第一个参数为this所应该指向的对象，剩下的参数为函数所需要的参数，以形参的方式“x，x,x..”指定，并且马上执行。

Apply🡪第一个参数为this所应该指向的对象，剩下的参数为函数所需要的参数，以数组[]方式指定，并且马上执行。

Bind🡪**第一个参数为this所应该指向的对象**，剩下的参数才为函数所需要的参数，以形参的方式“x，x,x..”指定，并且不会马上调用函数，而是返回的是指定过this作用域的函数。

15.

ReactDom.render(<MyComponents a=’12’/>, app)

这样a=12将会传入组件当中，React将其看作是key=a, value=12的参数。存在参数中的props的字典当中。使用this.props.a获取。

16.

在react.js中自定义组件的时候，我们会在render写入代码，在组件中定义其他方法。需要注意的是在定义的方法中，和render当中，this默认的指向不同，有时候需要在调用其他定义方法的时候使用call,apply,bind方法改变指向。

17.

JavscriptScirpt的两大开发IDE, WebStrom由Jetbrains开发，付费软件以及Atom由Github上的开源软件。

在本地的IDE调试JS代码的时候，需要node.js的支持。

18.

React组件中props和state的区别：

--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。或者通过this.setstate=({})或者this.state.field = xxx 修改state中属性，但是这样不会重新渲染组件

state不同于props的一点是，state是可以被改变的。

19.

Javascript中的箭头函数主要有两个作用：

第一是箭头函数可以相当于申明一个默认函数，使其更加简洁的语法。

***第二是由箭头函数所申明的函数，不会改变this本来的绑定。***

20.

React获取元素的其中三种方法：

--获取事件源元素，使用事件对象e.target属性

--获取非事件源元素，通过React提供的ref机制。

例如，<input ref=”input\_ref” />, 然后通过this.refs返回的字典形式中，以this.refs.input\_ref获取

--使用原生的Javascript获取函数获取

例如函数，getElementbyId, getElementByTagName,等等

21.

React的生命周期：就是React组件通过组件生命周期函数所经过的生命周期。组件生命周期会调用对应的函数。

--componentWillMount() 组件挂载前 (此时元素还没有渲染，没有生成无法获取渲染元素，但属性，状态允许使用)

--componentDidMount() 组件挂载后(此时元素已经渲染，可以获取渲染元素，但属性，状态允许使用)

--componentWillUpdate() 在组件接收到新的props或者state但还没有render时被调用。在初始化时不会被调用。

--componentDidUpdate() 在组件完成更新后立即调用。在初始化时不会被调用。

--componentWillUnmount() 组件卸载之前

22.

事件冒泡：底层元素的事件冒泡将事件传到父类组件

首先在react中的事件对象e,是经过React封装过的，不是原生的事件对象。而e.nativeEvent才是原生的事件对象。所以阻止事件冒泡时，通过e.nativeEvent.stopImmediatePropagation()

24.

我们在JSX语法范围中创建的HTML元素，是经过React封装过的，不是最原生HTML DOM元素，是React的虚拟DOM.

例如：

在JSX范围中 <li>ABC</li> 在浏览器中可以看到这个元素是React元素(Symbol(react.element))，而不是原始的HTMLDOM.

25.

react render渲染，即是组件中的render方法将会执行的几种情况：

1. 首次加载

2. setState({})改变组件内部state。

注意： 此处是说通过setState({})方法改变。通过this.setState = ({}), 或者this.state.field都不会调用redner方法

3. 接受到新的props, 例如从父组件更新的子组件的props值

4. 调用this.forceUpdate

26.

当我们在componentWillMount以及componentDidUpdated中使用改变状态的方法时候，注意是否会引入死循环。

在这两个方法中使用this.state.field是可以的，因为不会调用render函数，使用this.setState = {}也是可以的因为也不会调用render方法。但是使用this.setState({})是不行的，因为产生死循环。

componentWillMount或者compoenentDidUpdated -> this.setState({})->render() -> componentWIllMount/compomentDidUpdated…..

27.

判断是否可以做为一个state的条件:

  1、变量如果是通过props从父组件中获取，就不是一个状态

  2、如果这个变量可以通过其他的状态state或者属性props 通过数据处理得到，就不是一个状态

　　3、如果变量在render中没有使用到，那就不是一个state

  4、变量在整个生命周期中都保持不变时，也不是一个状态

其实使用的时候最多的使用到的就是state和props，他们两个是有很大的区别的，最主要的区别就是：

　　State是可变的，是组件内部维护的一组用于反映组件UI变化的状态集合；

　　而Props对于使用它的组件来说，是只读的，要想修改Props，只能通过该组件的父组件修改。在组件状态上移的场景中，父组件正是通过子组件的Props,传递给子组件其所需要的状态。

　　在使用state的时候， 如果我们企图直接修改state中的某一个值之后立即直接打印（使用）他，就会发现，他其实并没有改变。

　　就像下面的例子，企图通过点击事件之后就使用修改之后的state的值，但是会发state中的并没有被立即修改，还是原先的值，那是因为setState就相当于是一个异步操作，不能立即被修改



28.

React在html元素中加入自定义的属性的时候，遵循html5的方法，需要全部小写，并且采用前缀为data-自定义属性名。

例如：

非法：<input myindex=”AAA” type=”button” value=”Click me”/>

合法：<input data-myindex=”AAA” type=”button” value=”Click me”/>

29.

在React中指定元素的style属性，采用Javascript的JSON形式指定的时候，如果有多个属性则中间以”,”分割，因为这是JSON的分隔。

30.

函数多次bind以后，this的指向只遵从第一次bind.例如：

function say() {

alert(this.x);

};

var a = say.bind({x: 1});

var b = a.bind({x: 2});

这里的输出将会是1，不是2. 在bind了第一次以后，第二次的bind中的this不会被设置。

31.

ReactJS中如果我们需要循环生成html元素，例如<a>, <li>等元素，我们需要为其制定key属性，这样在React才不会有警告each child in an array or iterator should have a unique "key" prop. 对于非循环的html元素，我们不需要制定其key值也可以。

32.

如果我们在html中，想要渲染Javascript内容，我们需要花括号{变量名}, 如果没有加入花括号则会报错SyntaxError: Unterminated JSX contents

33．

注意React的this.state={}和this.setState()的不同.this.state通常用于设置控件的初始化状态所以是将一个字典赋值给this.state={“A”:”B”}属性，而this.setState是更新状态，是调用setState方法，所以是将一个代表状态更新的字典赋值给setState方法, this.setState({“A”:”B”})。

34.

React的事件指派原理，当我们像React中添加时间指派的时候，React会将事件处理器安放在document的根节点当中，然后当事件被触发的时候，React将事件给对应的控件并且当控件unmount的时候，React也会自动的把事件触发器移除。

React的事件指派采用的是驼峰命名发例如onClick而不是html中的onclick, onMouseMove而不是html中的onmousemove等。

常用的方法是我们在html标签中制定onClick = {this.xxx.bind(this)}，然后在控件中定义xxx事件处理方法，而不是直接在onClick中写事件处理函数逻辑。

35.

JSX语言中需要注意的几点：

**--Tag Attributes are camel cased所有的标签属性都必须遵循驼峰命名法**

例如 HTML中<input type=”text” maxlength=”30” />

JSX中是<input type=”text” maxLength=”30” />

**--All elements must be balanced（所有的标签都必须闭合）**

例如 HTML中可以<br>. <img>

JSX中必须<br/>, <img/>

**--Atrribute Names are based on the DOM API(标签的属性名将根据JSX 虚拟DOM更改)**

例如 HTML中<div id=’box” class=”some-class”></div>

JSX中<div id=’box’ className=”some-class”></div>

**--JSX中值允许一个根节点**

例如：render方法中return (<h1>AAA</h1><h2>BBB</h2>)是非法的

Render方法中return (<div>(<h1>AAA</h1><h2>BBB</h2>)</div>)才是合法的，通过div包裹成只有一个根节点

**--不允许if关键字出现在jsx语言的html标签当中**

例如 <div className={if (condition) {“salutation”}}>Hello</div>

解决办法有两种：

--使用三元操作符号，如果逻辑并不复杂，那么我们可以使用三元操作符号，从而不出现if关键字

<div className= {condition? “salutation”:””}>Hello</div>

--将判断逻辑移除标签中判断。用于比较复杂的逻辑判断。

render () {

let className=””;

If(condition) { # 如果condition为false时候，React也不会产生空的className

className=” salutation”;

}

return (

<div className={className}>Hello</div>

)

}

**--使用javascript的字典来定义样式，同样的样式中的属性采用驼峰命名法则**

例如 <div style={{‘width’:100, ‘backgroundColor’:’#ee9900’}} >Hello World</div>

也可以

let styleDict = {

‘width’:100,

‘backgroundColor’: #ee9900

}

<div style={styleDict}>Hello World</div>

**--React对于表单元素的操作**

React的表单元素分为两种，受控表单元素以及非受控表单元素

**受控表单元素**是指在开始的时候指定了value以及checked属性的表单元素。这种表单元素一般情况下用户无法再改变它的值或者选定状态。

例如：

<input type=” search” value=”AAA”/>

对于这种受控元素如果要改变它的值，我们需要把初始值当作是state来处理，才可以改变例如

constructor () {

super();

this.state={searchTerm:”React”}

}

handleChange(event) {

this.setstate({searchTerm:event.target.value});

}

render () {

return(<div>

<input type=” search” value={this.state.searchTerm} onChange={this.handleChange.bind(this)}

</div>

}

**特殊的**对于textarea和select表单元素：

我们设置textarea的初始值时需要<textarea value=”xxxx” />而不是<textarea>xxx</textarea>

我们设置select的初始值时是

<select value=”xxx”>

<option value=”xxx”></option>

</select>

而不是

<select>

<option value=”xxx” selected></option>

</select>

**非受控表单元素**是指在开始的时候没有指定了value以及checked属性的表单元素。这种表单元素一般情况下用户可以随意改变它的值或者选定状态。（指定了defaultValue作为初始值的表单元素是非受控元素）

36. React的JSX语法中，行注释使用//注释的内容

37.

React中使用Markdown语言范例：

首先什么是Markdown语言：

Markdown是一种「标记语言」，通过在纯文本中加入特殊的标记，便可以让文字渲染出特定的格式。并且在跨平台上的支持度很高，可以快速的打包成HTML文档等格式。

例如规则是：

|  |  |
| --- | --- |
| #，##，###，####，#####，###### | 一级标题，二级标题，三级标题，四级标题，五级标题，六级标题 |
| ##  -  -  -  … | 无序列表  项1  项2  项3 |
| ##  1.。。。  2.。。。  3.。。。 | 有序列表  项1  项2  项3 |
| [显示文本](链接地址) | 插入超链接 |
| ！[](链接地址) | 插入超链接所指向的图片 |
| >引用 | 插入引用的内容 |
| \*文字 | 该文字显示斜体效果 |
| \*\*文字\*\* | 该文字为粗体效果 |
| | **Tables** | **Are** | **Cool** | | --- | --- | --- | | col 3 is | right-aligned | $1600 | | col 2 is | centered | $12 | | zebra stripes | are neat | $1 |   表格：  | Tables | Are | Cool |  | ------------- |:-------------:| -----:|  | col 3 is | right-aligned | $1600 | 成为  | col 2 is | centered | $12 |  | zebra stripes | are neat | $1 | | |

并且还可以进行流程图等高级图标的绘制。

在React中我们使用markdown语言时， 首先需要安装marked包，我们可以在package.json中安装marked包，然后在JSX中引入。最后需要注意的是，由于React处于安全性的考虑，默认是不允许例如markdown这样动态生成内容的语句，因为这样有可能遭到插入式语句的攻击。我们需要使用dangerouslySetInnerHtml={{\_\_html: markdown的内容}}来让React特殊允许的生成。

38.**React的Key和Ref**

**Key属性:**React在更新状态的时候，有React自己的算法让自己的虚拟DOM最快更新。但是为了达到最好的效果，我们需要在进行循环生成节点的时候，为每一个最外层节点提供key属性来帮助React识别(如果不提供，React也会给出警告)

例如：

render () {

let tasks = this.props.tasks.map(task=> (

<li key={task.id} ….> //为每一个最外层的节点提供key属性

<input type=” checkbox” defaultChecked=…. /> //非最外层可以不需要指定key属性

</li>

));

}

**Ref属性：**一般来说，我们都会使用React的虚拟DOM来操作DOM,在极少情况下我们会去操作原生的DOM.React使用this.refs.ref值来让我们获取原生的DOM对象。

例如：

<input ref=” myInput”>

我们可以使用this.ref.myInput来获取到input的原生DOM对象。

39.

组件的属性申明验证(Props Validation)

由于我们的组件是本着重用的原则，所以当被别人复用或者再次检测组件的时候，我们希望通过某些方法规定组件需要哪些属性以及类型，或者为组件某些属性在非指定的情况下提供默认值。

(在Reactv15.5以后propsTypes提供在props-types库中)

**propTypes（申明组件属性的类型以及要求）**

例如：

class Greeter extends Component {

render () {

return(<h1>{this.props.salutation}</h1>)

}

}

Greeter（组件类名）.propsTypes= {

salutation: PropTypes.string.isRequried

//表示这个组件需要salutation这个属性，类型为字符串并且必须指定，如果创建组件的时候没有指定将会有警告

salutation2: PropsTypes.string

//表示salutation2作为组件属性，类型为字符串但是非必要

}

一些常用的PropType规定

|  |  |
| --- | --- |
| PropTypes.array | 属性必须是数组 |
| PropTypes.bool | 属性必须是布尔值 |
| PropTypes.func | 属性必须是方法 |
| PropTypes.number | 属性必须是数字 |
| PropTypes.object | 属性必须是对象 |
| PropTypes.string | 属性必须是字符串 |
| PropTypes.oneofType | 属性必须是所列举的类型范围之一 |
| PropTypes.arrayOf | 属性必须是数组并且数组中的元素类型为指定的类型 |
| PropTypes.objectof | 属性必须是对象且其所含的属性必须是指定类型 |
| PropTypes.shape | 属性必须是对象，且必须包含所列举出的属性 |
| PropTypes.element | 属性必须是React元素 |
| PropTyoes.instanceof | 属性必须是所指定类的实例对象 |

除此之外，我们还可以自定义propTypes的检查,规则为将提供三个参数props为所有属性的字典， propName验证的属性名， componentName组件名。当符合规则的时候propTypes不需要返回，当不符合规则的时候，应该返回Error对象。

例如:

let titlePropType = (props, propName, componentName) => {

if (props[propName] {

let value = props[propName]

if (typeof value!== ‘string’ || value.length> 80){ //条件为字符串并且长度小于80

return new Error(`${proName} in ${componentName} is longer than 80`) // ES6的模板字符串写法

}

}

}

**defaultProps（申明组件属性的默认值，如果不提供值）**

Greeter.defaultProps = {

Salutation3: “Hello World” //如果不提供组件的Salutation3属性，那么默认值为Hello World

}

40.

React设计中其中很重要的一个方面是，我们将为控件设置多少个state,以及将这些state放在哪个空间当中。

**识别state:**

State将设置在引起变化的起点，它的变化将引起其他变化。例如一个变量变化，将引起显示的文字不同， 搜索出来的结果不同。那么改变量作为一个state.

**放置state:**

State通常放置在公共组件当中，放置在层次化的上层。例如state变量，将引起组件1，组件2的变化，那么该state放置在包裹组件1，组件2的父组件当中，通过props传给子组件1，子组件2.同时子组件1，子组件2也反向改变父组件中的这个state.