1.

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

React的优势：

使用虚拟DOM

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

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，而不用原生的JS语法规则。

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语法里(即是<script></script>的范围中),在html标签里面使用js语法， 使用单花括号{}***

***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.***

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()

23.

React的表单元素分为受控表单和非受控表单

--受控表单是被控制的表单元素，在React中渲染以后无法改变值。对input元素指定value以后，将变为受控表单. 基本原则为，表单的值如果直接指定都将成为受控表单， 需要指定默认值时，在前面加入default.

例如<input type=’text’ value=’example’/> 🡪无法改变值。

<input type=’checkbox’ checked /> 🡪只能被选定，无法改变

--非受控表单可以改变值，如果要初始化值，则在React中使用defaultvalue

例如<input type=”text” defaultvalue=’example’/> 🡪 则值为example并且还可以改变值。

<input type=’checkbox’ defaultChecked />

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.