1/15/16/18 react

相关概念（1/15）

vue

vue MVVM框架 渐进式（粘性弱，弱主张）

vue + vue-route + vuex + vue-cli

react

专门构建于用户界面的js库（顶多它是MVC中的V层框架）（更加重视数据）

优势：
react –> facebook（脸书）出品，背景大
生态比较丰富：安卓和ios通用。
据说大项目用react会特别舒服。

核心思想：通过数据操作视图。

官方推荐使用的脚手架：create-react-app

相对vue来说只是写法不一样（代码相对vue要多一点，基本上都是原生，自己写的功能多一些），思想都是相似的。

声明式：通过声明一个类的方式来创建UI的。

```
// 下面是声明组件的两种方式。

// 声明一个类来作为一个组件
class Person{
    render(){
        retrun(
            //写组件内容
        )
    }
}
// 声明一个函数来作为一个组件
function () {
    retrun(
```

```
            // 组件内容
        )
    }
```

组件化：创建好拥有各自状态的组件，再由组件构成更复杂的界面。

基础知识

安装、使用

安装：

1、安装create-react-app

- npx create-react-app ma-app
- cd my-app
- npm start/yarn start

create-react-app 官方推荐的脚手架

2、react全家桶：

react react-router-dom redux（mobx）第三方UI库antd

react-native 开发原生App

react：

- jsx语法
- state
- props
- 数据的通信

```
 安装
npx create-react-app +新建项目文件名


 使用

// 在index.js 中

// 第一步：引包
impor React from 'react'; // 主要库   不能去掉，必须要有
impor ReactDOM from 'react-dom'   // 把jsx转成能够让浏览器识别的工具。也必
```

```
须要。

// 挂载组件 （根节点的挂载）

/*r
ReacrDOM.render() 是个方法
三个参数：
第一个参数：模板、组件
第二个参数：挂载的根节点
第三个参数：挂载完成之后触发的回调函数
*/

// 下面是热更新的作用：
if(module.hot){
    module.hot.accept()
}

// 第二步
ReactDOM.render(
    <div>你好，世界！！</div>    // 此处的标签就是jsx语法。
    document.getElementById('root'),// index里面是root
        ()=>{
        挂载完成之后的代码
    }
)

// 第一个参数的顶层只能有一层。需要多层时，就用标签包裹，顶层只能有一个。

// 普通的render中的return中的retrun上面写逻辑，retrun下面是UI。
```

jsx语法

jsx –> javasript + xml 一种js和自定义写法的一种混合模式

jsx语法主要是为了更好的读、写模板或者组件，jsx语法浏览器是不识别的，通过babel的方式把jsx转成浏览器能识别的代码。

在react中，写`<div></div>`其实就等同于写了一个：React.createElement(‘div’, children: [], ‘click’, ‘active’)

jsx规则：

- 1、遇到样式中的class，直接在jsx语法中写成className
- 2、便签必须闭合，特别是单标签内必须带 ’ / ‘
- 3、{ }
    - { }内可以运行js代码；函数的话，只能声明不能调用

    - 默认帮你展开数组。[1, 2, 3, 4] –> “1” “2” “3” “4”
    - 或者[`<li />, <li />`] 如果数据组中的值为组件（列表），那么一定要给列表中的每项加key（唯一），为了优化
    - 注释 –> { /* 被注释的内容 */ } –> 注释的内容会被加上花括号
- 4、表单元素设置默认值value的时候，页面会显示，但是控制台会报错（因为它认为input是一个动态组件，是动态就会操作数据（value就一定会变化，一定会变化，就一定需要事件（onChange））所以会报错），
  `<input value="1" />` –> 会报错。
    - 解决方法1：要在后面加上 `onChange={()=>{}}` ，(推荐，此处注意事件函数内的this指向)

```
<input
    value='1'
    onChange={ () =>{}}
/>
```

-
    - 解决方法2：定义默认值value时，单词变成defaultValue

```
<input
    defaultValue='1'
>
```

- 5、设置style={ }
- 6、 `value={a}` 赋值某个元素的属性

```
<div style={{width:'200px',height:'200px',}}></div>
```

组件：

定义组件的方式：

两种方式，只是写法不同。

```
class App extends React.Component{ }

// 或者
import React,{component} from 'react'
class App extends component{ }

用类创建组件
```

```
class App extends Component {
        render(){
            return (
                <div>
                    <h1>组件</h1>
                </div>
            )
         }
}
// 类的声明方式，有状态、this、生命周期


用函数创建组件
function Fn(){
      return (
           <div>
                <h1>组件</h1>
           </div>
       )
}
```

组件中的方法、函数注意事项

写方法的时候，如果不对函数做处理，事件函数中的this默认指向undefined
解决方案：四种

```
第一种：使用create-app  （推荐）
click = () => {

}

第二种：bind，this
click () {

}
constructor(){
    this.click = this.click.bind(this);
}

第三种：
click () {  }
<button
    onClick = {this.click.bind(this)}
>
```

```
第四种：
click () {  }
<button
    onClick = {(e)=>this.click(e)}
>
```

循环生成li

```
let list = arr.map((e,i)=>{
  return <List {...{
    key:i,
    val:e
  }}/>
});

// 然后把list这个变量放到组件就行了：{list}

<ul>{list}</ul>

// map是一个方法，是一个遍历（循环每一项）的方法。和forEach差不多。
```

状态：state

state 状态
this.state = { } 状态初始化，初始化必须放在constructor下，只要更新state就会更新视图。
操作状态中的数据，就能操作视图。
操作数据不能用this.state.xx = ' '

改变数据状态只能用：this.setState ({ }) –》双向数据绑定

this.setState(对象(就是你的state数据), ()=>{当操作完数据之后的回调函数}) ==> 推荐使用
this.setState(函数(prev)=>{return prev+1}, ( )=>{当操作完数据之后的回调函数}) ==> 此方法更适合在某个值的基础上进行累计。

数据通信

数据之间的通信（单向数据流）
传递数据在组件身上使用属性的绑定

传递数据在组件身上使用属性绑定

属性中如果有2个重复的，后面会把前面的覆盖，

两种写法：

- 直接写在标签内
- {...{obj / 或者变量赋值}}

```
 在组件标签上写
 // 第一种写法：
<APP aaa="你好"/>

// 第二种写法：加省略号和花括号，或者定义一个对象obj 直接{...obj}
<App {...{
    aa="你好"
}}/>

// 子组件接受：
let {aa} = this.props
```

外部传进来的数据用：this.props.xxx 接收、使用
组件内部的数据用：this.state.xxx 直接使用，修改的话用this.setState.xxx

子级修改父级传进来的数据：

方法一：

- 父级有一个这样的修改数据的函数方法，
- 这个修改函数的方法，当做一个数据传递给子级，
- 接收这个修改函数，然后放在想要执行的地方。`执行:fn()`

方法二：（把父级的数据变成子级，子级的数据不就跟父级的数据有瓜葛了）

- 把父级的数据变成子级的
- 子级修改。

受控组件：表单元素如果设置一个默认值，此时表单元素就是一个受控组件（默认值：value、checked）（里面数据（val等）是可变的。两者是联动的。）
非受控组件：里面只是自己本身的东西。

小例子：两级之间传递数据和改变数据（知识点挺综合的。）

```
// app.js中写的。

import React,{Component} from 'react';

class PPa extends Component {
    constructor(props){                         // 数据初始化。
```

```
        super(props);
        this.state = {
            bb: 1
        }
    }

//constructor 和 render 中间的部分或者下面的部分，是写逻辑的空间，不能在里面拿取数据。
// 写函数的时候（逻辑），注意使用箭头函数。否则使用的时候会有this指向问题。

    ad = ()=> {                                    // 写函数时要用箭头函数。因为这里有this指向问题。
        let {addd} = this.props;
        addd()
        let {bb} = this.state;
        bb++
        this.setState({bb})
    }
    render(){
        return(
            <div>
                <button
                    onClick = {this.ad}
                >PPa的{this.state.bb}</button>
                <div>{this.props.aaa}</div>

            </div>
        )
    }
}

class App extends Component {
    constructor(props){
        super(props);
        this.state = {
            aa:0
        }
    }
    add = () => {
        let {aa} = this.state;
        aa += 2
        this.setState({aa})
    }

    render(){
        return (
            <div>
                <h1>2019/1/16</h1>
                <button
```

```
                    onClick = {this.add}
                >App的{this.state.aa}</button>
                <hr />
                <PPa
                    {...{aaa:this.state.aa,addd:this.add}}  // 此处
传参要注意写法
                ></PPa>
            </div>
        )
    }

}

export default App
```

todos例子自己总结：

- **一层一层之间传递事件函数时，注意每层都要接收然后继续传递。如果函数需要传参的话，那么在触发的那一层需要包一层箭头函数，然后传参。如果不用传参的话，直接等于这个函数就行（用花括号包裹），但是有参数时，传参时需要用括号包裹，会默认为执行这个函数。所以需要包裹一层箭头函数。**
- **传递数据时，子级接收数据时，解构赋值：简单类型就是新的变量了，复合类型是赋址关系。**
- **注意使用事件函数（函数名）**

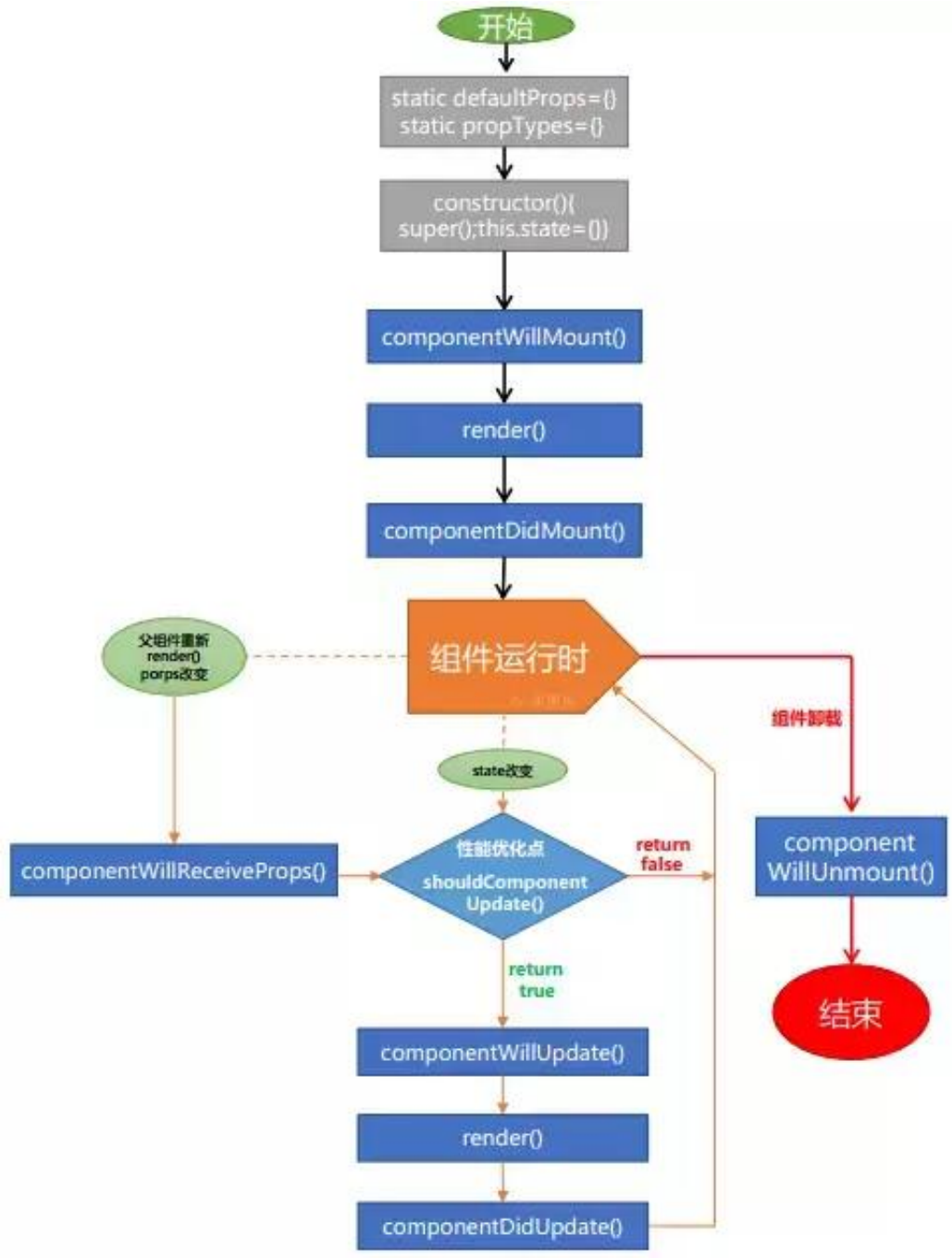
propTypes(检测传进来的数据类型)

propsTypes：验证父级传递进来的数据类型，(为了报错，检测数据是否适用)

```
// 先在文件中引入 Prop-types
import React,{Component} from 'react';
import PropTypes from 'prop-types';

class PPa extends Component {
    render () {
        return (
            <div>
                <span>0</span>
                <span>{this.props.obj}</span>

            </div>
        )
    }
```

```
}

// 在导出之前， 在此处检测下从父级App传入的某一个数据类型
PPa.propTypes = {
    obj:PropTypes.number
    // 前面obj是父级传入的某一个数据名称， 后面的number 是此处规定的数据类型，此处是检测这个父级传入的数据，是否是这个规定的数据类型。 不是的话就在控制台报错，但是不影响视图的显示（测试时用的数字，暂不影响数字的显示）
}
export default PPa;


// 此处检测数据类型时，需要注意大小写的情况。  另外注意代码的位置，检测的代码写在导出组件的上面，以及组件的外面（尤其 注意不能在组件的里面）。
```

生命周期（三种生命状态）（1/18）★★★



mounting

mounting：只会执行一次，

- constructor –> 数据的初始化（实用）
- ComponentWillmount –> 组件的挂载之前
- render –> 解析jsx （实用）
- componentDidMount –> 全部挂载成功（子级之后，在最后）（实用 （数据的请求））

组件在挂载的时候，会先执行constructor（数据初始化，里面的数据都是子级本身的，全局只会执行一次），然后经过几个钩子函数之后，走到render
只要数据发生变化，就再次执行render，而comstructor第不会改变了。

componentDidmount 等同于mounted –> 组件挂载完成之后。
组件不仅能放进父级组件内，还可以放进变量里面。（例如：循环的时候。）

updating

updating：当数据改变的时候触发

- componentWillReceiveProps –> 如果有父级数据传递进来，当父级数据变化时才会执行。
  -
- shouldComponentUpdate –> 性能优化，极限的优化，接下来的组件要不要更新。（默认返回trun）
  - **注意千万不要在shouldComponentUpdate这里使用this.state，不然会死循环、**
  - 如果写了它就一定需要返回值，默认为trun，
  - 当为trun时，`componentWillUpdate、render、componentDidUpdate`才会执行。
  - 为false时不触发这三个。
  - **应用场景：组件放的父级a数据有变动，会重复执行render，（子级的render也会被重新执行）但是传进子级的父级b数据没有变动，子级无需重新执行，此时可用此函数。判断传进的数据是否变动，然后判断是否重新执行。**

```
// 优化：  解决子组件 不改变数据时多余的调用
shouldComponentUpdate(nextProps, nextState){
    return (nextProps === this.props)
}
```

- componentWillUpdate
- render ： 作用和 computed（vue中的）类型，先执行一次，改变后再次执行。
- componentDidUpdate ：作用和 watch（vue中的）类似，第一次不会执行，当数据改变时才会执行。
  当子级的更新完成之后，才会执行父级的componentDidUpdate。（父级的conponentDidUpdate是最后执行的。）

unmounting

unmounting：当组件卸载的时候。

- componentWillUnmount –> 关闭定时器，清除各种

> 回调函数就是钩子函数，生命周期函数。
> 当某个条件（事务）成立的时候触发的函数。

小知识

改变数据在 componentWillMount 和 componentDidMount 中的使用情况。
componentWillMount方法的调用
在constructor之后，在render之前，
在这方法里的代码调用setState方法不会触发重渲染，
所以它一般不会用来作加载数据之用，它也很少被使用到。

componentWillMount（组件挂载之前）：里面的数据改变是直接覆盖原数据，第一次已经是修改后的值了。（不能使用，会出问题）
componentDidMount（组件挂载之后）：里面的数据改变是后面改变的，第一次还是原来的数据。（推荐使用）

验证：在render中打印this.start中的数据。
如果在componentWillMount中调用setState，那么render中打印this.start，得到的是改变后得数据。

如果在componentDidMount中调用setStart，那么render中打印this.start,2次是不同的数据，第一次时原始数据，第二次是改变后的数据。
第一次render是为了挂载结构，第二次render是为了渲染数据。

> 一个月有多少天，这个月有多少天。（把时间设置为下个月的上个月）
> 当前月有几天：设置setDate（ ）就能拿到当前月有几天了；
> setDate（1）这个月的第一天；
> setDate（0） 上个月的最后一天；

```
let d = new Date;
d.setDate(1)  // 设置为本月的第一天。
d.setMonth(d.getmonth()+1);  // 下个月
d.setDate(0);   // 上个月有多少天
```