ReactJS简介

- React 起源于 Facebook 的内部项目，因为该公司对市场上所有 JavaScript MVC 框架，都不满意，就决定自己写一套，用来架设 Instagram 的网站。做出来以后，发现这套东西很好用，**就在2013年5月开源了。**
- 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。
- 中文文档：https://react.docschina.org/

前端三大主流框架

- Angular.js：出来最早的一套前端框架，学习曲线比较陡峭，1.x学起来比较麻烦，2.x进行了一系列的改革，启动了组件化思想，在NGJS中还支持TS（typescript）进行编写逻辑，Typescript支持JS的ES3-ES6的语法，然后JS并不支持TS的语法
- Vue.js：最火的一套框架，它是中国人开发的，对我们来说，文档方面比较友好（作者：尤雨溪）
- React.js：最流行的一套框架，它的设计思想很优秀（渲染数据用虚拟DOM方式实现）

分析React

开发团队方面

- React是由FaceBook前端官方团队进行维护和更新的；因此，React的维护开发团队，技术实力比较雄厚；

社区方面

- 在社区方面，React由于诞生的较早，所以社区比较强大，一些常见的问题、坑、最优解决方案，文档、博客在社区中都是可以很方便就能找到的；

移动APP开发体验方面

- React，结合 ReactNative，也提供了无缝迁移到 移动App的开发体验（RN用的最多，也是最火最流行的）；

为什么要学习React

1.设计优秀，是基于组件化，方便我们UI代码的重用

2.开发团队实力比较强悍，不用担心断更的问题

3.社区比较强大，在很多问题上都能找到对应得解决方案

4,提供了无缝迁移到ReactNactive上的开发，让我们技术能力得到了扩展，曾强了核心竞争力

React中几个核心的概念

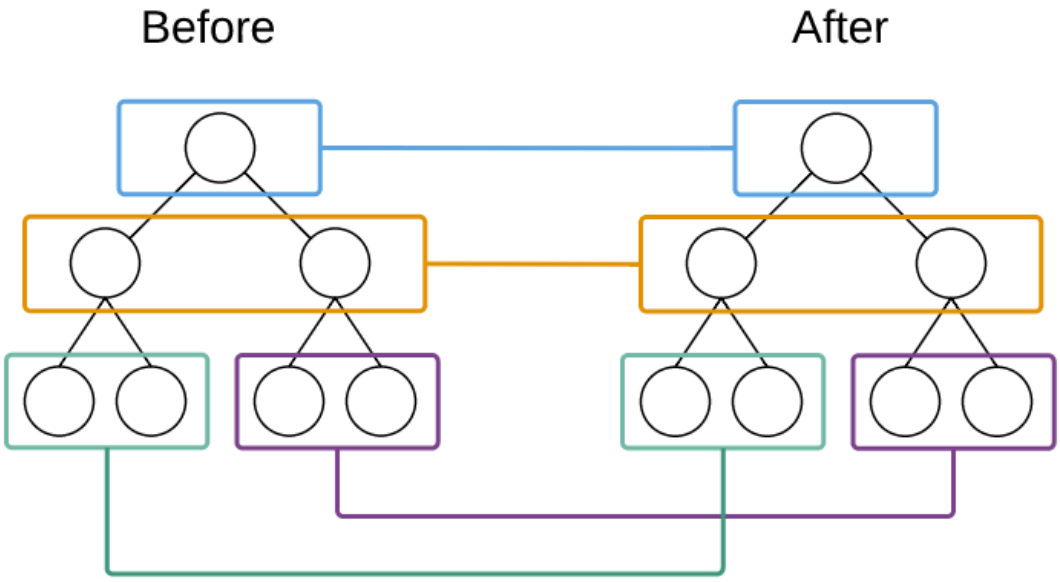
虚拟DOM

- DOM的本质是什么：就是用JS表示的UI元素（dom节点）
- DOM和虚拟DOM的区别：
  - DOM是浏览器提供的JS功能，所以我们人为的使用，只能通过提供的固定的API来进行操作DOM对象
  - 虚拟DOM：并不是由浏览器提供的，而是由程序员手动模拟实现的，类似于浏览器中的DOM，但是有本质的区别
- 为什么要实现虚拟DOM：
- 什么是React中的虚拟DOM：
- 虚拟DOM的目的：

时间	新关注人数	取消关注人数	净增关注人数	累积关注人数
2015-11-21	0	0	0	0
2015-11-20	0	0	0	285
2015-11-19	1	0	1	285
2015-11-18	0	1	-1	284
2015-11-17	2	0	2	284
2015-11-16	1	3	-2	282
2015-11-15	0	1	-1	285
2015-11-14	0	0	0	286

Diff算法

- tree diff:新旧DOM树，逐层对比的方式，就叫做 tree diff,每当我们从前到后，把所有层的节点对比完后，必然能够找到那些 需要被更新的元素；
- component diff：在对比每一层的时候，组件之间的对比，叫做 component diff;当对比组件的时候，如果两个组件的类型相同，则暂时认为这个组件不需要被更新，如果组件的类型不同，则立即将旧组件移除，新建一个组件，替换到被移除的位置；
- element diff:在组件中，每个元素之间也要进行对比，那么，元素级别的对比，叫做 element diff；
- key：key这个属性，可以把 页面上的 DOM节点 和 虚拟DOM中的对象，做一层关联关系；



React初体验

- CDN方式引入类库
- 本地下载引入
- React基本使用

JSX语法

1. 如要要使用 JSX 语法，必须使用babel语法转换器；
2. JSX语法的本质：还是以 React.createElement 的形式来实现的，并没有直接把 用户写的 HTML代码，渲染到页面上；
3. 如果要在 JSX 语法内部，书写 JS 代码了，那么，所有的JS代码，必须写到 {} 内部；
4. 当 编译引擎，在编译JSX代码的时候，如果遇到了 `<` 那么就把它当作 HTML代码去编译，如果遇到了 `{}` 就把 花括号内部的代码当作 普通JS代码去编译；
5. 在{}内部，可以写任何符合JS规范的代码；
6. 在JSX中，如果要为元素添加 `class` 属性了，那么，必须写成 `className`，因为 `class` 在ES6中是一个关键字；和 `class` 类似，label标签的 `for` 属性需要替换为 `htmlFor`.
7. 在JSX创建DOM的时候，所有的节点，必须有唯一的根元素进行包裹；
8. 如果要写注释了，注释必须放到 {} 内部

组件化方面

1. 什么是模块化：一个js文件就是一个模块，从js代码角度去分析，把我们编程时所写的业务逻辑代码，分割到不同的模块中进行调用，这样方便代码的重用
2. 什么是组件化：从UI角度分析（HTML结构），把一个页面，拆分成互不相干的小组件，随着我们项目的代码开发，我们的组件就会越来越多，最后，如果要实现一个页面，可以把能用到的小组件直接通过调用的方式，进行页面拼接，然后就可以得到一个完整页面
3. 组件化的好处：减少代码体积，方便UI视图的元素重用，组件也是元素的集合体；
4. React如何实现组件化：直接使用JS代码的形式，去创建任何你想要的组件

组件的创建方式

1. 基本理解和使用

```
1). 自定义的标签: 组件类(函数)/标签
2). 创建组件类
    //方式1: 无状态函数(简单组件, 推荐使用)
    function MyComponent1(props) {
        return <h1>自定义组件标题11111</h1>
    }
    //方式2: ES6类语法(复杂组件, 推荐使用)
    class MyComponent3 extends React.Component {
        render () {
          return <h1>自定义组件标题33333</h1>
        }
    }
3). 渲染组件标签
    ReactDOM.render(<MyComp />,  cotainerEle)
4). ReactDOM.render()渲染组件标签的基本流程
    React内部会创建组件实例对象/调用组件函数, 得到虚拟DOM对象
    将虚拟DOM并解析为真实DOM
    插入到指定的页面元素内部
```

两种创建组件方式的对比

1. 用构造函数创建出来的组件：专业的名字叫做“无状态组件”
2. 用class关键字创建出来的组件：专业的名字叫做“有状态组件”

> 用构造函数创建出来的组件，和用class创建出来的组件，这两种不同的组件之间的**本质区别就是**：有无state属性！！！
> 有状态组件和无状态组件之间的本质区别就是：有无state属性!

2. 组件的3大属性: state

```
1. 组件被称为"状态机", 页面的显示是根据组件的state属性的数据来显示
2. 初始化指定:
    constructor() {
      super()
      this.state = {
        stateName1 : stateValue1,
        stateName2 : stateValue2
      }
    }
3. 读取显示:
    this.state.stateName1
4. 更新状态-->更新界面 :
    this.setState({stateName1 : newValue})
```

3. 组件的3大属性: props

```
所有组件标签的属性的集合对象
给标签指定属性, 保存外部数据(可能是一个function)
在组件内部读取属性: this.props.propertyName
作用: 从目标组件外部向组件内部传递数据
对props中的属性值进行类型限制和必要性限制
    Person.propTypes = {
        name: React.PropTypes.string.isRequired,
        age: React.PropTypes.number.isRequired
    }
扩展属性: 将对象的所有属性通过props传递
    <Person {...person}/>
```

4. 组件的3大属性: refs

```
组件内包含ref属性的标签元素的集合对象
给操作目标标签指定ref属性, 打一个标识
在组件内部获得标签对象: this.refs.refName(只是得到了标签元素对象)
作用: 找到组件内部的真实dom元素对象, 进而操作它
```

5. 组件中的事件处理

```
1. 给标签添加属性: onXxx={this.eventHandler}
2. 在组件中添加事件处理方法
    eventHandler(event) {

    }
3. 使自定义方法中的this为组件对象
    在constructor()中bind(this)
    使用箭头函数定义方法(ES6模块化编码时才能使用)
```

```
4. 事件监听
    绑定事件监听
        事件名
        回调函数
    触发事件
        用户对对应的界面做对应的操作
        编码
```

6. 组件的组合使用

```
1)拆分组件: 拆分界面,抽取组件
2)实现静态组件: 使用组件实现静态页面效果
3)实现动态组件
    ① 动态显示初始化数据
    ② 交互功能(从绑定事件监听开始)
```

7. 组件的生命周期

```
1. 组件的三个生命周期状态:
    Mount: 插入真实 DOM
    Update: 被重新渲染
    Unmount: 被移出真实 DOM
2. 生命周期流程:
    * 第一次初始化显示: ReactDOM.render(<Xxx/>, containDom)
        constructor()
        componentWillMount() : 将要插入回调
        render() : 用于插入虚拟DOM回调
        componentDidMount() : 已经插入回调
    * 每次更新state: this.setState({})
        componentWillReceiveProps(): 接收父组件新的属性
        componentWillUpdate() : 将要更新回调
        render() : 更新(重新渲染)
        componentDidUpdate() : 已经更新回调
    * 删除组件: ReactDOM.unmountComponentAtNode(div): 移除组件
        componentWillUnmount() : 组件将要被移除回调
3. 常用的方法
    render(): 必须重写, 返回一个自定义的虚拟DOM
    constructor(): 初始化状态, 绑定this(可以箭头函数代替)
    componentDidMount() : 只执行一次, 已经在dom树中, 适合启动/设置一些监听
```

8.ajax请求（fetch）

根据指定的关键字在github上搜索匹配的最受关注的库

测试地址： https://api.github.com/search/repositories?q=r&sort=stars

语法介绍

```
 ``fetch(``'http://localhost:3000/fdata'``).then(``function``(data) {
    ``return` `data.text(); ``// 通过调用text返回promise对象
  ``}).then(``function``(data) {
    ``console.log(data); ``// 得到真正的结果
  ``})
```

（1）get请求方式的参数传递 (传统方式)

```
fetch(``'http://localhost:3000/books?id=123'``, {
      ``method: ``'get'
  ``}).then(``function``(data) {
    ``return` `data.text();
  ``}).then(``function``(data) {
    ``console.log(data);
  ``})
```

（3）post请求方式的参数传递 （字符串类型参数）

```
fetch(``'http://localhost:3000/books'``, {
    ``method: ``'post'``,
    ``body: ``'uname=lisi&pwd=123'``,
    ``headers: {
      ``'Content-Type'``: ``'application/x-www-form-urlencoded'
    ``}
  ``}).then(``function``(data) {
    ``return` `data.text();
  ``}).then(``function``(data) {
    ``console.log(data);
  ``})
```