基础知识
监控运维
React 基础

中国联通软件研究院

2020.03

课程简介

《监控运维-React 基础》课程为具备一定前端基础（HTML、JS、CSS）并且打算学习 React 的学员所编写。整个课程分为了两个部分，React 简介与 React 基础概念。简介部分可以使学员对 React 有一个初步认识以及 React 与传统前端开发的区别和优势。基本概念部分为学员讲解了一些 React 中最基础的概念，例如虚拟 DOM、组件化、JSX 语法、生命周期。

React 作为主流的前端开发框架，目前广泛应用在了研发中台的前端开发，在后续中高级课程中天眼运维工具孵化前端主要使用的是 React 技术栈包括 AntDesign Pro 框架等。通过本课程的学习可以使学员对前端技术有一定了解，为学员日后深入学习 React、提升前端开发技术奠定了基础。

目录

第一章 React 简介

1.1 前端发展历史

1、静态页面阶段

1990 年的 12 月 25 日，恰是西方的圣诞节，Tim Berners-Lee 在他的 NeXT 电脑上部署了第一套“主机-网站-浏览器”构成的 Web 系统，这标志 BS 架构的网站应用软件的开端，也是前端工程的开端。

1993 年 4 月 Mosaic 浏览器作为第一款正式的浏览器发布。1994 年 11 月，鼎鼎大名的 Navigator 浏览器发布发布了，到年底 W3C 在 Berners-Lee 的主持下成立，标志着万维网进入了标准化发展的阶段。

这个阶段的网页还非常的原始，主要以 HTML 为主，是纯静态的只读网页。

2、动态页面的发展

Javascript 的诞生之初，就给网页带来了一些跑马灯、浮动广告之类的特效和应用，让网页动了起来。但是网页真正开始向动态交互发展的开端，却是 PHP、JSP 和 ASP 为代表的后端动态页面技术的出现。

这些服务器端的动态页面技术使得网页可以获取服务器的数据信息并保持更新，推动了 Google 为代表的搜索引擎和各种论坛的出现，万维网开始快速发展。

服务器端网页动态交互功能的不断丰富，伴随的是后端逻辑的复杂度快速上升，代码越来越复杂。为了更好的管理后端逻辑，出现了大量后端的 MVC 框架。

3、Ajax 的流行开启 Web2.0 时代

2004 年前的动态页面都是由后端技术驱动的，虽然实现了动态交互和数据即时存取，但是每一次的数据交互都需要刷新一次浏览器。频繁的页面刷新非常影响用户的体验，这个问题直到谷歌在 04 年应用 Ajax 技术开发的 Gmail 和谷歌地图的发布，才得到了解决。

这背后的秘密就是 Ajax 技术中实现的异步 HTTP 请求，这让页面无需刷新就可以发起 HTTP 请求，用户也不用专门等待请求的响应，而是可以继续网页的浏览或操作。

Ajax 开启了 web2.0 的时代。

4、前端兼容性框架的出现

NetScape 在第一次浏览器之战中败给了 IE 之后，创办了 Mozilla 技术社区，该社区之后发布了遵循 W3C 标准的 firefox 浏览器，和 Opera 浏览器一起代表 W3C 阵营和 IE 开始了第二次浏览器战争。

不同的浏览器技术标准有不小的差异，不利于兼容开发，这催生了 Dojo、Mooltools、YUIExtJS、jQuery 等前端兼容框架，其中 jQuery 应用最为广泛。

5、前端 MV*架构及 SPA 时代的开启

随着各大浏览器纷纷开始支持 HTML5，前端能够实现的交互功能越多，相应的代码复杂度也快速提高，以前用于后端的 MV*框架也开始出现在前端部分。

从 2010 年 10 月出现的 Backbone 开始，Knockout、Angular、Ember、Meteor、React、Vue 相继出现。

这些框架的运用，使得网站从 Web Site 进化成了 Web App，开启了网站应用的 SPA（Single Page Application）的时代。

6、ECMAScript6

2015 年 6 月，ECMAScript 6.0 发布，该版本增加了很多新的语法，极大的拓展了 javascript 的开发潜力。由于浏览器 ES6 语法的支持滞后，出现了 Babel 和 TypeScript 来把 ES6 代码编译成 ES5 等现有浏览器支持的代码。

ES6 现已更名为 ES2015，以后每年会发布新的 ES 标准，这标志着 javascript 的发展将会更快。

7、今天的前端

今天的前端技术已经形成了一个大的技术系统。

以 Github 为代表的代码管理仓库；NPM 和 Yarn 为代表的包管理工具；ES6 及 Babel 和 TypeScript 构成的脚本体系；HTML5；CSS3 和相应的处理技术；React、Vue、Anjular 为代表的框架；Webpack 为代表的打包工具；Node.js 为基础的 Express 和 KOA 后端框架；

1.2 什么是 React

1.2.1 React 简介

React 是 Facebook 开发的一款 JS 库。React 一般被用来作为 MVC 中的 V 层，它不依赖其他任何的库，因此开发中，可以与任何其他的库集成使用，包括 Jquery、Backbone 等。它可以在浏览器端运行，也可以通过 nodejs 在服务端渲染。React 的思想非常独特，性能出众，可以写出重复代码少，逻辑清晰的前端代码。React 的语法是 jsx，通过使用这种语法，可以在 react 代码中直接混合使用 js 和 html 来编写代码，这样代码的逻辑就非常清晰，当然也意味着，需要将 jsx 代码编译成普通的 javascript 代码，才能在浏览器中运行，这个过程根据实际项目情况，可以选择多种不同的思路，或者在服务器端通过 webpack 进行编译。

1.2.2 React 较传统开发模式的区别与优势

1、 React 速度很快

与其它框架相比，React 采取了一种特立独行的操作 DOM 的方式。
它并不直接对 DOM 进行操作。
它引入了一个叫做虚拟 DOM 的概念，安插在 JavaScript 逻辑和实际的 DOM 之间。

这一概念提高了 Web 性能。在 UI 渲染过程中，React 通过在虚拟 DOM 中的微操作来实对现实际 DOM 的局部更新。

2、跨浏览器兼容

虚拟 DOM 帮助我们解决了跨浏览器问题，它为我们提供了标准化的 API，甚至在 IE8 中都是没问题的。

3、 组件化

为你程序编写独立的模块化 UI 组件，这样当某个或某些组件出现问题时，可以方便地进行隔离。每个组件都可以进行独立的开发和测试，并且它们可以引入其它组件。这等同于提高了代码的可维护性。

4、单向数据流

Flux 是一个用于在 JavaScript 应用中创建单向数据层的架构，它随着 React 视图库的开发而被 Facebook 概念化。它只是一个概念，而非特定工具的实现。它可以被其它框架吸纳。例如，Alex Rattray 有一个很好的 Flux 实例，在 React 中使用了 Backbone 的集合和模型。

第二章 React 基本概念

2.1 虚拟 DOM

在 Web 开发中，我们总需要将变化的数据实时反应到 UI 上，这时就需要对 DOM 进行操作。而复杂或频繁的 DOM 操作通常是性能瓶颈产生的原因（如何进行高性能的复杂 DOM 操作通常是衡量一个前端开发人员技能的重要指标）。

React 为此引入了虚拟 DOM（Virtual DOM）的机制：在浏览器端用 Javascript 实现了一套 DOM API。基于 React 进行开发时所有的 DOM 构造都是通过虚拟 DOM 进行，每当数据变化时，React 都会重新构建整个 DOM 树，然后 React 将当前整个 DOM 树和上一次的 DOM 树进行对比，得到 DOM 结构的区别，然后仅仅将需要变化的部分进行实际的浏览器 DOM 更新。而且 React 能够批处理虚拟 DOM 的刷新，在一个事件循环（Event Loop）内的两次数据变化会被合并，例如你连续的先将节点内容从 A 变成 B，然后又从 B 变成 A，React 会认为 UI 不发生任何变化，而如果通过手动控制，这种逻辑通常是极其复杂的。尽管每一次都需要构造完整的虚拟 DOM 树，但是因为虚拟 DOM 是内存数据，性能是极高的，而对实际 DOM 进行操作的仅仅是 Diff 部分，因而能达到提高性能的目的。这样，在保证性能的同时，开发者将不再需要关注某个数据的变化如何更新到一个或多个具体的 DOM 元素，而只需要关心在任意一个数据状态下，整个界面是如何 Render 的。

如果你像在 90 年代那样写过服务器端 Render 的纯 Web 页面那么应该知道，服务器端所要做的就是根据数据 Render 出 HTML 送到浏览器端。如果这时因为用户的一个点击需要改变某个状态文字，那么也是通过刷新整个页面来完成的。服务器端并不需要知道是哪一小段 HTML 发生了变化，而只需要根据数据刷新整个页面。换句话说，任何 UI 的变化都是通过整体刷新来完成的。而 React 将这种开发模式以高性能的方式带到了前端，每做一点界面的更新，你都可以认为刷新了整个页面。至于如何进行局部更新以保证性能，则是 React 框架要完成的事情。

借用 Facebook 介绍 React 的视频中聊天应用的例子，当一条新的消息过来时，你的开发过程需要知道哪条数据过来了，如何将新的 DOM 结点添加到当前 DOM 树上；而基于 React 的开发思路，你永远只需要关心数据整体，两次数据

之间的 UI 如何变化，则完全交给框架去做。可以看到，使用 React 大大降低了逻辑复杂性，意味着开发难度降低，可能产生 Bug 的机会也更少。

2.2 组件化

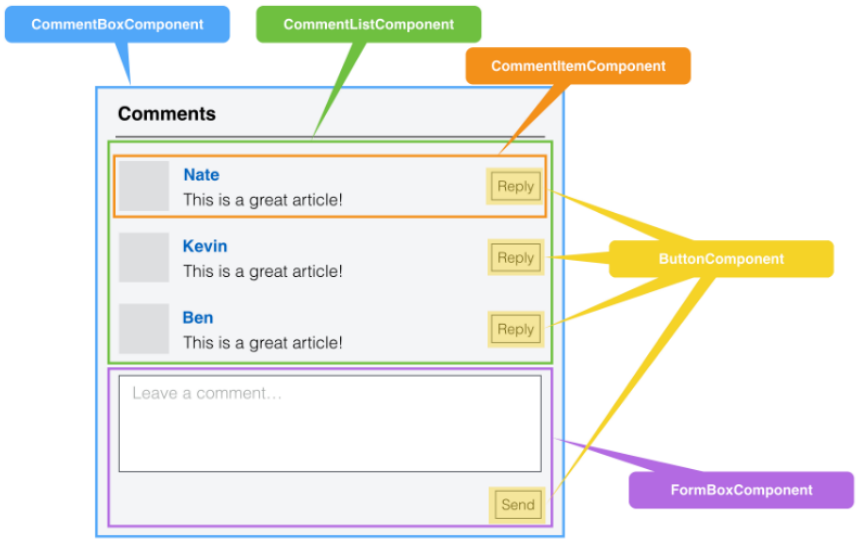
虚拟 DOM(virtual-dom)不仅带来了简单的 UI 开发逻辑，同时也带来了组件化开发的思想，所谓组件，即封装起来的具有独立功能的 UI 部件。React 推荐以组件的方式去重新思考 UI 构成，将 UI 上每一个功能相对独立的模块定义成组件，然后将小的组件通过组合或者嵌套的方式构成大的组件，最终完成整体 UI 的构建。例如，Facebook 的 instagram.com 整站都采用了 React 来开发，整个页面就是一个大的组件，其中包含了嵌套的大量其它组件。

如果说 MVC 的思想让你做到视图-数据-控制器的分离，那么组件化的思考方式则是带来了 UI 功能模块之间的分离。我们通过一个典型的 Blog 评论界面来看 MVC 和组件化开发思路的区别。

对于 MVC 开发模式来说，开发者将三者定义成不同的类，实现了表现，数据，控制的分离。开发者更多的是从技术的角度来对 UI 进行拆分，实现松耦合。

对于 React 而言，则完全是一个新的思路，开发者从功能的角度出发，将 UI 分成不同的组件，每个组件都独立封装。

在 React 中，你按照界面模块自然划分的方式来组织和编写你的代码，对于评论界面而言，整个 UI 是一个通过小组件构成的大组件，每个组件只关心自己部分的逻辑，彼此独立。



React 认为一个组件应该具有如下特征：

（1）可组合（Composeable）：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有（own）它创建的子组件，通过这个特性，一个复杂的 UI 可以拆分成多个简单的 UI 组件。

（2）可重用（Reusable）：每个组件都是具有独立功能的，它可以被使用在多个 UI 场景。
（3）可维护（Maintainable）：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护。

2.3 JSX 语法

HTML 语言直接写在 JavaScript 语言之中，不加任何引号，这就是 JSX（JavaScript and XML） 的语法，JSX,是一种 JavaScript 的语法扩展，它允许 HTML 与 JavaScript 的混写。JSX 是 facebook 为 React 框架开发的一套语法糖，语法糖又叫做糖衣语法，是指计算机语言中添加的某种语法，这种语法对语言的功能并没有影响，但是更方便程序员使用，它主要的目的是增加程序的可读性，从而减少程序代码错处的机会。JSX 就是 JS 的一种语法糖，类似的还有 CoffeeScript、TypeScript，最终它们都会被解析成 JS 才能被浏览器理解和执行，如果不解析浏览器是没有办法识别它们的，这也是所有语法糖略有不足的地方。

```
const element = <h1>Hello, world!</h1>;
```

上面这种看起来可能有些奇怪的标签语法既不是字符串也不是 HTML，被称为 JSX，JSX 带来的一大便利就是我们可以直接在 JS 里面写类 DOM 的结构，比我们用原生的 JS 去拼接字符串，然后再用正则替换等方式来渲染模板方便和简单太多了。推荐在 React 中使用 JSX 来描述用户界面。JSX 用来声明 React 当中的元素， 乍看起来可能比较像是模版语言，但事实上它完全是在 JavaScript 内部实现的。

你可以任意地在 JSX 当中使用 JavaScript 表达式，在 JSX 当中的表达式要包含在大括号里。例子如下：

```
const names = ['Jack', 'Tom', 'Alice'];
const element = (
  <div>
     { names.map(function (name) { return <div>Hello, {name}!</div>}) }
  </div>
);
```

在书写 JSX 的时候一般都会带上换行和缩进，这样可以增强代码的可读性。与此同时，推荐在 JSX 代码的外面扩上一个小括号，这样可以防止 分号自动插入 的 bug。

上面我们声明了一个 names 数组，然后遍历 names 数组在前面加上 Hello，生成了 element 数组。JSX 允许直接在模板插入 JavaScript 变量。如果这个变量是一个数组，则会展开这个数组的所有成员。JSX 本身其实也是一种表达式，在编译之后，JSX 其实会被转化为普通的 JavaScript 对象。代码如下：

```
import React from 'react';
import ReactDOM from 'react-dom';

const names = ['Jack', 'Tom', 'Alice'];
const element = names.map(function (name) { return <div>Hello, {name}!</div>});

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

显示结果如下：

JSX 属性：
你可以使用引号来定义以字符串为值的属性：

```
const element = <div tabIndex="0"></div>;
```

也可以使用大括号来定义以 JavaScript 表达式为值的属性：

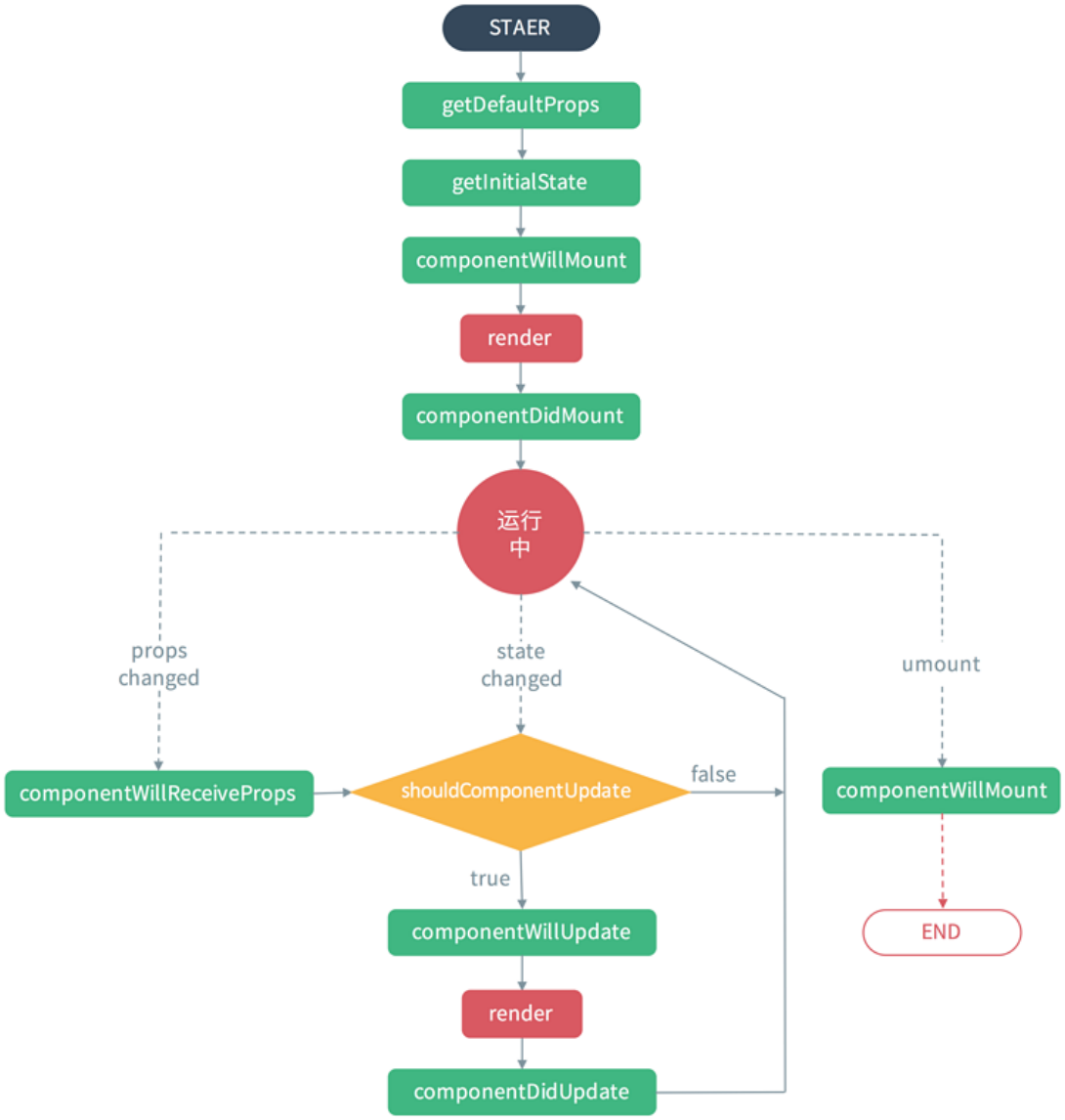
```
const element = <img src={user.avatarUrl}></img>;

```

切记当使用了大括号包裹的 JavaScript 表达式时就不要再到外面套引号了。JSX 会将引号当中的内容识别为字符串而不是表达式。

2.4 生命周期

React 的生命周期从广义上分为三个阶段：挂载、渲染、卸载。因此可以把 React 的生命周期分为两类：挂载卸载过程和更新过程。

2.4.1 React 的生命周期图:



2.4.2 挂载卸载过程

1、constructor()

constructor()中完成了 React 数据的初始化，它接受两个参数：props 和 context，当想在函数内部使用这两个参数时，需使用 super()传入这两个参数。注意：只要使用了 constructor()就必须写 super(),否则会导致 this 指向错误。

2、componentWillMount()

componentWillMount()一般用的比较少，它更多的是在服务端渲染时使用。它代表的过程是组件已经经历了 constructor()初始化数据后，但是还未渲染 DOM 时。

3、componentDidMount()

组件第一次渲染完成，此时 dom 节点已经生成，可以在这里调用 ajax 请求，返回数据 setState 后组件会重新渲染

4、componentWillUnmount ()

在此处完成组件的卸载和数据的销毁。

2.4.3 更新过程

1、componentWillReceiveProps (nextProps)

在接受父组件改变后的 props 需要重新渲染组件时用到的比较多，接受一个参数 nextProps，通过对比 nextProps 和 this.props，将 nextProps 的 state 为当前组件的 state，从而重新渲染组件。

2、shouldComponentUpdate(nextProps,nextState)

主要用于性能优化(部分更新)，一用于控制组件重新渲染的生命周期，由于在 react 中，setState 以后，state 发生变化，组件会进入重新渲染的流程，在这里 return false 可以阻止组件的更新，因为 react 父组件的重新渲染会导致其所有子组件的重新渲染，这个时候其实我们是不需要所有子组件都跟着重新渲染的，因此需要在子组件的该生命周期中做判断。

3、componentWillUpdate (nextProps,nextState)

shouldComponentUpdate 返回 true 以后，组件进入重新渲染的流程，进入 componentWillUpdate,这里同样可以拿到 nextProps 和 nextState。

4、componentDidUpdate(prevProps,prevState)

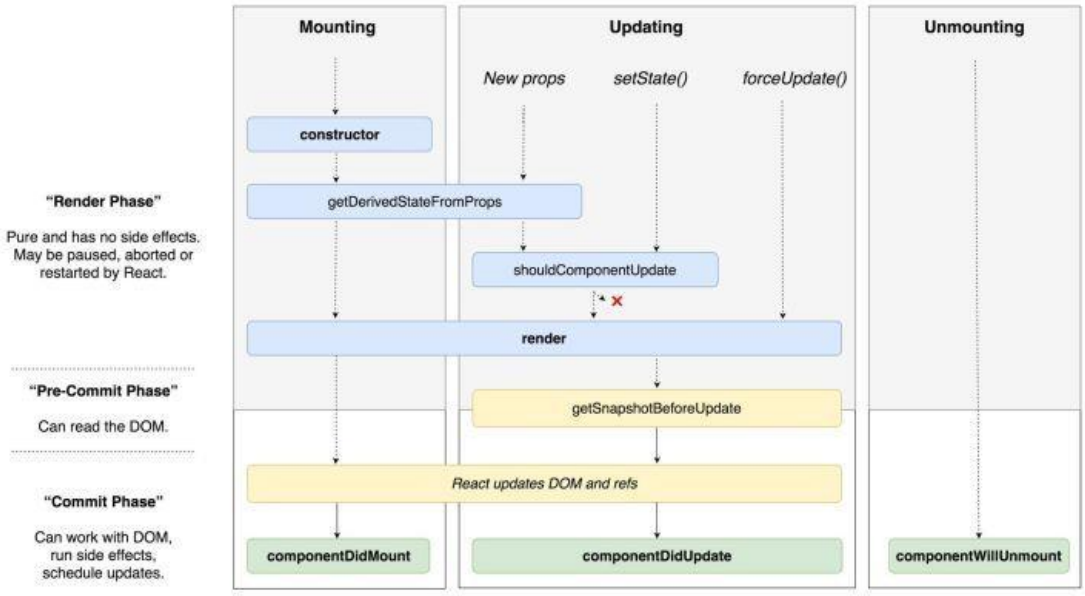
组件更新完毕后，react 只会在第一次初始化成功会进入 componentDidmount,之后每次重新渲染后都会进入这个生命周期，这里可以拿到 prevProps 和 prevState，即更新前的 props 和 state。

5、render()

render 函数会插入 jsx 生成的 dom 结构，react 会生成一份虚拟 dom 树，在每一次组件更新时，在此 react 会通过其 diff 算法比较更新前后的新旧 DOM 树，比较以后，找到最小的有差异的 DOM 节点，并重新渲染。

2.4.4 React16 版本

在 React 迎来了 16 版本的更新后，生命周期也有了不小的变化。



新的生命周期主要修改了以下几点：

1、弃用了 componentWillMount、componentWillReceivePorps，componentWillUpdate

2、新增了 getDerivedStateFromProps、getSnapshotBeforeUpdate 来代替弃用的三个钩子函数（componentWillMount、componentWillReceivePorps，componentWillUpdate）

3、React16 并没有删除这三个钩子函数，但是不能和新增的钩子函数（getDerivedStateFromProps、getSnapshotBeforeUpdate）混用，React17 将会删除 componentWillMount、componentWillReceivePorps，componentWillUpdate

4、新增了对错误的处理（componentDidCatch）