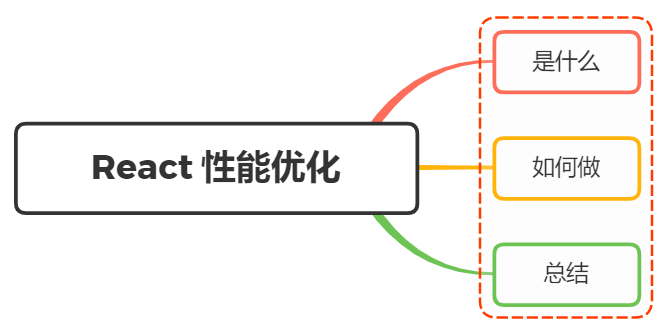
# 面试官：说说 React 性能优化的手段有哪些？

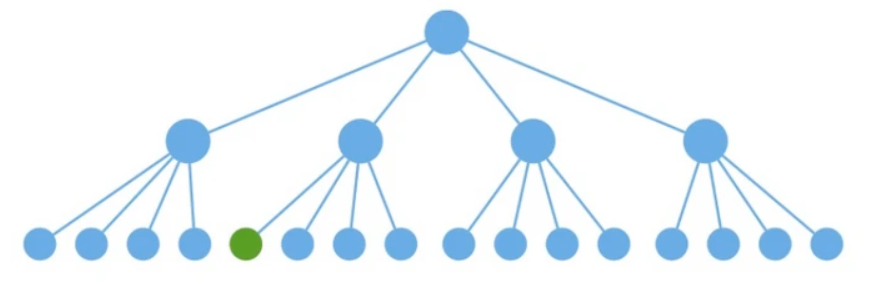


## 一、是什么

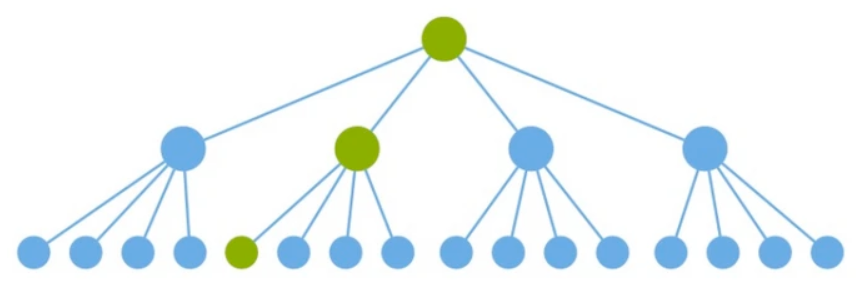
React凭借virtual DOM和diff算法拥有高效的性能，但是某些情况下，性能明显可以进一步提高

在前面文章中，我们了解到类组件通过调用setState方法， 就会导致render，父组件一旦发生render渲染，子组件一定也会执行render渲染

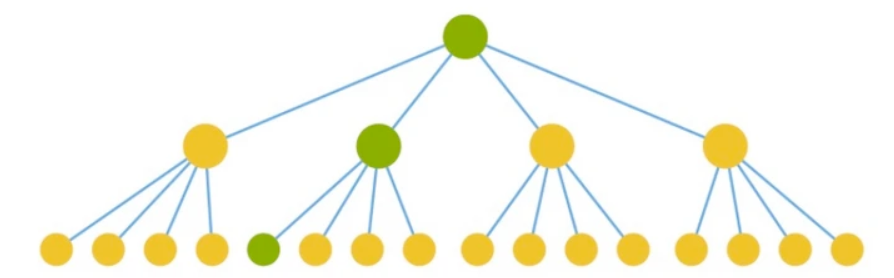
当我们想要更新一个子组件的时候，如下图绿色部分：



理想状态只调用该路径下的组件render：



但是react的默认做法是调用所有组件的render，再对生成的虚拟DOM进行对比（黄色部分），如不变则不进行更新



从上图可见，黄色部分diff算法对比是明显的性能浪费的情况

## 二、如何做

在[React中如何避免不必要的render](https://mp.weixin.qq.com/s/h4NX4Plr6TCjoIhlawiJTg)中，我们了解到如何避免不必要的render来应付上面的问题，主要手段是通过shouldComponentUpdate、PureComponent、React.memo，这三种形式这里就不再复述

除此之外， 常见性能优化常见的手段有如下：

* 避免使用内联函数
* 使用 React Fragments 避免额外标记
* 使用 Immutable
* 懒加载组件
* 事件绑定方式
* 服务端渲染

#### 避免使用内联函数

如果我们使用内联函数，则每次调用render函数时都会创建一个新的函数实例，如下：

import React from "react";  
  
export default class InlineFunctionComponent extends React.Component {  
 render() {  
 return (  
 <div>  
 <h1>Welcome Guest</h1>  
 <input type="button" onClick={(e) => { this.setState({inputValue: e.target.value}) }} value="Click For Inline Function" />  
 </div>  
 )  
 }  
}

我们应该在组件内部创建一个函数，并将事件绑定到该函数本身。这样每次调用 render 时就不会创建单独的函数实例，如下：

import React from "react";  
  
export default class InlineFunctionComponent extends React.Component {  
   
 setNewStateData = (event) => {  
 this.setState({  
 inputValue: e.target.value  
 })  
 }  
   
 render() {  
 return (  
 <div>  
 <h1>Welcome Guest</h1>  
 <input type="button" onClick={this.setNewStateData} value="Click For Inline Function" />  
 </div>  
 )  
 }  
}

#### 使用 React Fragments 避免额外标记

用户创建新组件时，每个组件应具有单个父标签。父级不能有两个标签，所以顶部要有一个公共标签，所以我们经常在组件顶部添加额外标签div

这个额外标签除了充当父标签之外，并没有其他作用，这时候则可以使用fragement

其不会向组件引入任何额外标记，但它可以作为父级标签的作用，如下所示：

export default class NestedRoutingComponent extends React.Component {  
 render() {  
 return (  
 <>  
 <h1>This is the Header Component</h1>  
 <h2>Welcome To Demo Page</h2>  
 </>  
 )  
 }  
}

### 事件绑定方式

在[事件绑定方式](https://mp.weixin.qq.com/s/VfQ34ZEPXUXsimzMaJ_41A)中，我们了解到四种事假绑定的方式

从性能方面考虑，在render方法中使用bind和render方法中使用箭头函数这两种形式在每次组件render的时候都会生成新的方法实例，性能欠缺

而constructor中bind事件与定义阶段使用箭头函数绑定这两种形式只会生成一个方法实例，性能方面会有所改善

### 使用 Immutable

在[理解Immutable中](https://mp.weixin.qq.com/s/laYJ_KNa8M5JNBnIolMDAA)，我们了解到使用 Immutable可以给 React 应用带来性能的优化，主要体现在减少渲染的次数

在做react性能优化的时候，为了避免重复渲染，我们会在shouldComponentUpdate()中做对比，当返回true执行render方法

Immutable通过is方法则可以完成对比，而无需像一样通过深度比较的方式比较

### 懒加载组件

从工程方面考虑，webpack存在代码拆分能力，可以为应用创建多个包，并在运行时动态加载，减少初始包的大小

而在react中使用到了Suspense和 lazy组件实现代码拆分功能，基本使用如下：

const johanComponent = React.lazy(() => import(/\* webpackChunkName: "johanComponent" \*/ './myAwesome.component'));  
   
export const johanAsyncComponent = props => (  
 <React.Suspense fallback={<Spinner />}>  
 <johanComponent {...props} />  
 </React.Suspense>  
);

### 服务端渲染

采用服务端渲染端方式，可以使用户更快的看到渲染完成的页面

服务端渲染，需要起一个node服务，可以使用express、koa等，调用react的renderToString方法，将根组件渲染成字符串，再输出到响应中

例如：

import { renderToString } from "react-dom/server";  
import MyPage from "./MyPage";  
app.get("/", (req, res) => {  
 res.write("<!DOCTYPE html><html><head><title>My Page</title></head><body>");  
 res.write("<div id='content'>");   
 res.write(renderToString(<MyPage/>));  
 res.write("</div></body></html>");  
 res.end();  
});

客户端使用render方法来生成HTML

import ReactDOM from 'react-dom';  
import MyPage from "./MyPage";  
ReactDOM.render(<MyPage />, document.getElementById('app'));

### 其他

除此之外，还存在的优化手段有组件拆分、合理使用hooks等性能优化手段...

### 三、总结

通过上面初步学习，我们了解到react常见的性能优化可以分成三个层面：

* 代码层面
* 工程层面
* 框架机制层面

通过这三个层面的优化结合，能够使基于react项目的性能更上一层楼

## 参考文献

* https://zhuanlan.zhihu.com/p/108666350
* https://segmentfault.com/a/1190000007811296