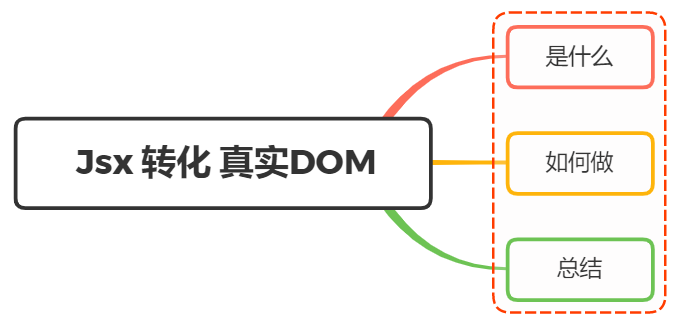
# 面试官：说说React Jsx转换成真实DOM过程？



## 一、是什么

react通过将组件编写的JSX映射到屏幕，以及组件中的状态发生了变化之后 React会将这些「变化」更新到屏幕上

在前面文章了解中，JSX通过babel最终转化成React.createElement这种形式，例如：

<div>  
 < img src="avatar.png" className="profile" />  
 <Hello />  
</div>

会被bebel转化成如下：

React.createElement(  
 "div",  
 null,  
 React.createElement("img", {  
 src: "avatar.png",  
 className: "profile"  
 }),  
 React.createElement(Hello, null)  
);

在转化过程中，babel在编译时会判断 JSX 中组件的首字母：

* 当首字母为小写时，其被认定为原生 DOM 标签，createElement 的第一个变量被编译为字符串
* 当首字母为大写时，其被认定为自定义组件，createElement 的第一个变量被编译为对象

最终都会通过RenderDOM.render(...)方法进行挂载，如下：

ReactDOM.render(<App />, document.getElementById("root"));

## 二、过程

在react中，节点大致可以分成四个类别：

* 原生标签节点
* 文本节点
* 函数组件
* 类组件

如下所示：

class ClassComponent extends Component {  
 static defaultProps = {  
 color: "pink"  
 };  
 render() {  
 return (  
 <div className="border">  
 <h3>ClassComponent</h3>  
 <p className={this.props.color}>{this.props.name}</p >  
 </div>  
 );  
 }  
}  
  
function FunctionComponent(props) {  
 return (  
 <div className="border">  
 FunctionComponent  
 <p>{props.name}</p >  
 </div>  
 );  
}  
  
const jsx = (  
 <div className="border">  
 <p>xx</p >  
 < a href=" ">xxx</ a>  
 <FunctionComponent name="函数组件" />  
 <ClassComponent name="类组件" color="red" />  
 </div>  
);

这些类别最终都会被转化成React.createElement这种形式

React.createElement其被调用时会传⼊标签类型type，标签属性props及若干子元素children，作用是生成一个虚拟Dom对象，如下所示：

function createElement(type, config, ...children) {  
 if (config) {  
 delete config.\_\_self;  
 delete config.\_\_source;  
 }  
 // ! 源码中做了详细处理，⽐如过滤掉key、ref等  
 const props = {  
 ...config,  
 children: children.map(child =>  
 typeof child === "object" ? child : createTextNode(child)  
 )  
 };  
 return {  
 type,  
 props  
 };  
}  
function createTextNode(text) {  
 return {  
 type: TEXT,  
 props: {  
 children: [],  
 nodeValue: text  
 }  
 };  
}  
export default {  
 createElement  
};

createElement会根据传入的节点信息进行一个判断：

* 如果是原生标签节点， type 是字符串，如div、span
* 如果是文本节点， type就没有，这里是 TEXT
* 如果是函数组件，type 是函数名
* 如果是类组件，type 是类名

虚拟DOM会通过ReactDOM.render进行渲染成真实DOM，使用方法如下：

ReactDOM.render(element, container[, callback])

当首次调用时，容器节点里的所有 DOM 元素都会被替换，后续的调用则会使用 React 的 diff算法进行高效的更新

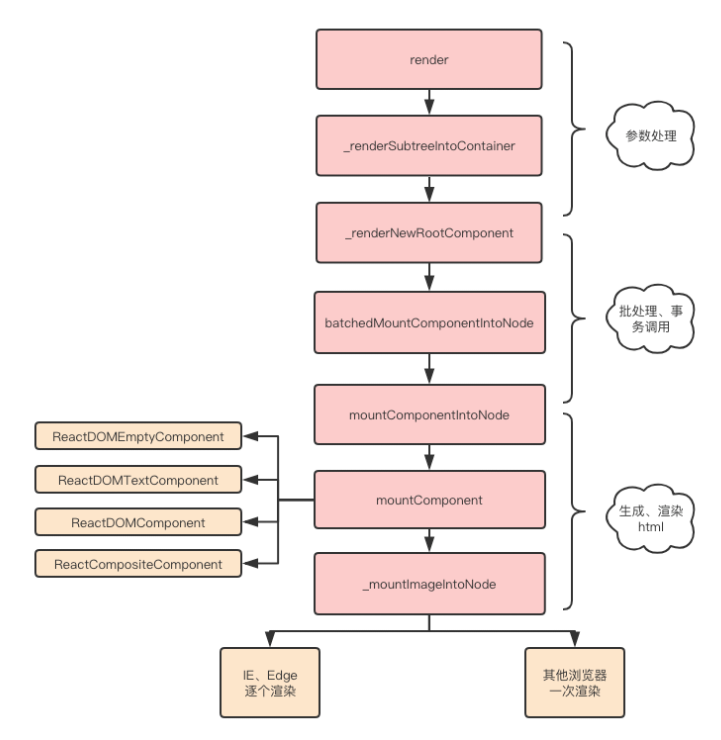
如果提供了可选的回调函数callback，该回调将在组件被渲染或更新之后被执行

render大致实现方法如下：

function render(vnode, container) {  
 console.log("vnode", vnode); // 虚拟DOM对象  
 // vnode \_> node  
 const node = createNode(vnode, container);  
 container.appendChild(node);  
}  
  
// 创建真实DOM节点  
function createNode(vnode, parentNode) {  
 let node = null;  
 const {type, props} = vnode;  
 if (type === TEXT) {  
 node = document.createTextNode("");  
 } else if (typeof type === "string") {  
 node = document.createElement(type);  
 } else if (typeof type === "function") {  
 node = type.isReactComponent  
 ? updateClassComponent(vnode, parentNode)  
 : updateFunctionComponent(vnode, parentNode);  
 } else {  
 node = document.createDocumentFragment();  
 }  
 reconcileChildren(props.children, node);  
 updateNode(node, props);  
 return node;  
}  
  
// 遍历下子vnode，然后把子vnode->真实DOM节点，再插入父node中  
function reconcileChildren(children, node) {  
 for (let i = 0; i < children.length; i++) {  
 let child = children[i];  
 if (Array.isArray(child)) {  
 for (let j = 0; j < child.length; j++) {  
 render(child[j], node);  
 }  
 } else {  
 render(child, node);  
 }  
 }  
}  
function updateNode(node, nextVal) {  
 Object.keys(nextVal)  
 .filter(k => k !== "children")  
 .forEach(k => {  
 if (k.slice(0, 2) === "on") {  
 let eventName = k.slice(2).toLocaleLowerCase();  
 node.addEventListener(eventName, nextVal[k]);  
 } else {  
 node[k] = nextVal[k];  
 }  
 });  
}  
  
// 返回真实dom节点  
// 执行函数  
function updateFunctionComponent(vnode, parentNode) {  
 const {type, props} = vnode;  
 let vvnode = type(props);  
 const node = createNode(vvnode, parentNode);  
 return node;  
}  
  
// 返回真实dom节点  
// 先实例化，再执行render函数  
function updateClassComponent(vnode, parentNode) {  
 const {type, props} = vnode;  
 let cmp = new type(props);  
 const vvnode = cmp.render();  
 const node = createNode(vvnode, parentNode);  
 return node;  
}  
export default {  
 render  
};

## 三、总结

在react源码中，虚拟Dom转化成真实Dom整体流程如下图所示：



其渲染流程如下所示：

* 使用React.createElement或JSX编写React组件，实际上所有的 JSX 代码最后都会转换成React.createElement(...) ，Babel帮助我们完成了这个转换的过程。
* createElement函数对key和ref等特殊的props进行处理，并获取defaultProps对默认props进行赋值，并且对传入的孩子节点进行处理，最终构造成一个虚拟DOM对象
* ReactDOM.render将生成好的虚拟DOM渲染到指定容器上，其中采用了批处理、事务等机制并且对特定浏览器进行了性能优化，最终转换为真实DOM

## 参考文献

* https://bbs.huaweicloud.com/blogs/265503)
* https://huang-qing.github.io/react/2019/05/29/React-VirDom/
* https://segmentfault.com/a/1190000018891454