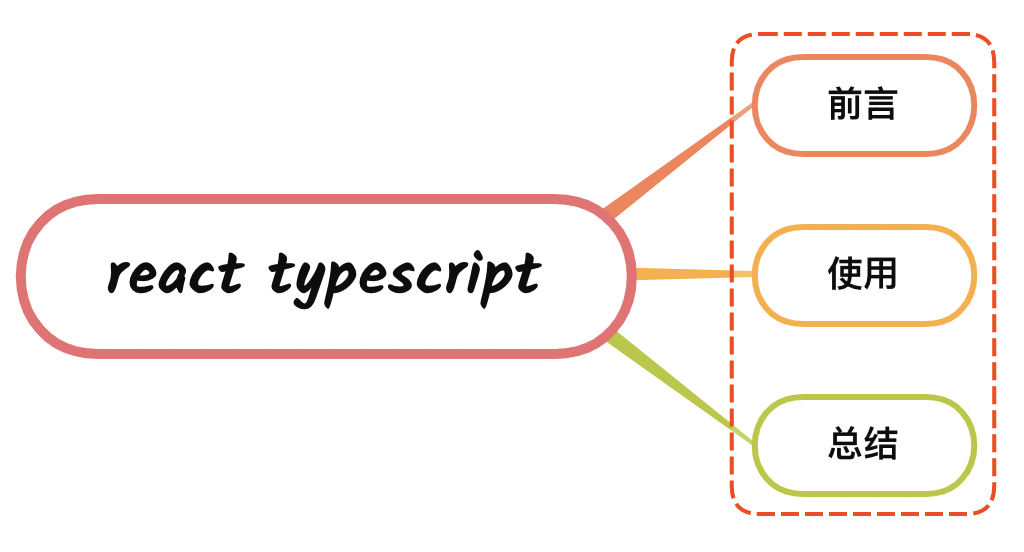
# 面试官：说说如何在 React 项目中应用 TypeScript？



## 一、前言

单独的使用 TypeScript 并不会导致学习成本很高，但是绝大部分前端开发者的项目都是依赖于框架的

例如与 Vue、React 这些框架结合使用的时候，会有一定的门槛

使用 TypeScript 编写 React 代码，除了需要 TypeScript 这个库之外，还需要安装 @types/react、@types/react-dom

npm i @types/react -s  
  
npm i @types/react-dom -s

至于上述使用 @types 的库的原因在于，目前非常多的 JavaScript 库并没有提供自己关于 TypeScript 的声明文件

所以，ts 并不知道这些库的类型以及对应导出的内容，这里 @types 实际就是社区中的 DefinitelyTyped 库，定义了目前市面上绝大多数的 JavaScript 库的声明

所以下载相关的 JavaScript 对应的 @types 声明时，就能够使用使用该库对应的类型定义

## 二、使用方式

在编写 React 项目的时候，最常见的使用的组件就是：

* 无状态组件
* 有状态组件
* 受控组件

### 无状态组件

主要作用是用于展示 UI，如果使用 js 声明，则如下所示：

import \* as React from "React";  
  
export const Logo = (props) => {  
 const { logo, className, alt } = props;  
  
 return <img src={logo} className={className} alt={alt} />;  
};

但这时候 ts 会出现报错提示，原因在于没有定义 porps 类型，这时候就可以使用 interface 接口去定义 porps 即可，如下：

import \* as React from "React";  
  
interface IProps {  
 logo?: string;  
 className?: string;  
 alt?: string;  
}  
  
export const Logo = (props: IProps) => {  
 const { logo, className, alt } = props;  
  
 return <img src={logo} className={className} alt={alt} />;  
};

但是我们都知道 props 里面存在 children 属性，我们不可能每个 porps 接口里面定义多一个 children，如下：

interface IProps {  
 logo?: string;  
 className?: string;  
 alt?: string;  
 children?: ReactNode;  
}

更加规范的写法是使用 React 里面定义好的 FC 属性，里面已经定义好 children 类型，如下：

export const Logo: React.FC<IProps> = (props) => {  
 const { logo, className, alt } = props;  
  
 return <img src={logo} className={className} alt={alt} />;  
};

* React.FC 显式地定义了返回类型，其他方式是隐式推导的
* React.FC 对静态属性：displayName、propTypes、defaultProps 提供了类型检查和自动补全
* React.FC 为 children 提供了隐式的类型（ReactElement | null）

### 有状态组件

可以是一个类组件且存在 props 和 state 属性

如果使用 TypeScript 声明则如下所示：

import \* as React from "React";  
  
interface IProps {  
 color: string;  
 size?: string;  
}  
interface IState {  
 count: number;  
}  
class App extends React.Component<IProps, IState> {  
 public state = {  
 count: 1,  
 };  
 public render() {  
 return <div>Hello world</div>;  
 }  
}

上述通过泛型对 props、state 进行类型定义，然后在使用的时候就可以在编译器中获取更好的智能提示

关于 Component 泛型类的定义，可以参考下 React 的类型定义文件 node\_modules/@types/React/index.d.ts，如下所示：

class Component<P, S> {  
 readonly props: Readonly<{ children?: ReactNode }> & Readonly<P>;  
  
 state: Readonly<S>;  
}

从上述可以看到，state 属性也定义了可读类型，目的是为了防止直接调用 this.state 更新状态

### 受控组件

受控组件的特性在于元素的内容通过组件的状态 state 进行控制

由于组件内部的事件是合成事件，不等同于原生事件，

例如一个 input 组件修改内部的状态，常见的定义的时候如下所示：

private updateValue(e: React.ChangeEvent<HTMLInputElement>) {  
 this.setState({ itemText: e.target.value })  
}

常用 Event 事件对象类型：

* ClipboardEvent 剪贴板事件对象
* DragEvent 拖拽事件对象
* ChangeEvent Change 事件对象
* KeyboardEvent 键盘事件对象
* MouseEvent 鼠标事件对象
* TouchEvent 触摸事件对象
* WheelEvent 滚轮事件对象
* AnimationEvent 动画事件对象
* TransitionEvent 过渡事件对象

T 接收一个 DOM 元素类型

## 三、总结

上述只是简单的在 React 项目使用 TypeScript，但在编写 React 项目的时候，还存在 hooks、默认参数、以及 store 等等......

TypeScript 在框架中使用的学习成本相对会更高，需要不断编写才能熟练

## 参考文献

* <https://juejin.cn/post/6952696734078369828>
* <https://juejin.cn/post/6844903684422254606>