React与TypeScript

开发环境创建

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

基于Vite创建开发环境

Vite是一个框架无关的前端工具链，可以快速的生成一个 React + TS 的开发环境，并且可以提供快速的开发体验

```
npm create vite@latest react-ts-pro -- --template react-ts
```

说明：

1. npm create vite@latest 固定写法 （使用最新版本vite初始化项目）
2. react-ts-pro 项目名称 （可以自定义）
3. -- --template react-ts 指定项目模版位react+ts

useState与TypeScript

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

useState-自动推导

通常React会根据传入useState的默认值来自动推导类型,不需要显式标注类型

```
const [value, toggle] = useState(false)
```

说明：

1. value：类型为boolean
2. toggle: 参数类型为boolean

useState-传递泛型参数

useState本身是一个泛型函数，可以传入具体的自定义类型

```ts
type User = {
  name: string
  age: number
}

const [user, setUser] = useState<User>()
```

说明：

1. 限制useState函数参数的初始值必须满足类型为： User | （）=> User
2. 限制setUser函数的参数必须满足类型为：User | （）=> User | undefined
3. user状态数据具备User类型相关的类型提示

useState-初始值为null

当我们不知道状态的初始值是什么，将useState的初始值为null是一个常见的做法，可以通过具体类型联合null来做显式注解

```ts
type User = {
  name: string
  age: number
}

const [user, setUser] = useState<User | null>(null)
```

说明：

1. 限制useState函数参数的初始值可以是 User | null
2. 限制setUser函数的参数类型可以是 User | null

事件与TypeScript

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

事件与TypeScript - 为事件回调添加类型

为事件回调添加类型约束需要使用React内置的泛型函数来做，比如最常见的鼠标点击事件和表单输入事件：

```tsx
function App() {
  const changeHandler: React.ChangeEventHandler<HTMLInputElement> = (e) => {
    console.log(e.target.value)
  }

  const clickHandler: React.MouseEventHandler<HTMLButtonElement> = (e) => {
    console.log(e.target)
  }

  return (
    <>
      <input type="text" onChange={changeHandler} />
      <button onClick={clickHandler}>click me!</button>
    </>
  )
}
```

说明：通过泛型函数约束了整个事件回调函数的类型，主要是为了约束事件参数e的类型

Props与TypeScript

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

props与TypeScript - 基础使用

为组件prop添加类型，本质是给函数的参数做类型注解，可以使用type对象类型或者interface接口来做注解

```tsx
type Props = {
  className: string
}

function Button(props: Props) {
  const { className } = props
  return <button className={className}>click me</button>
}
```

说明：Button组件只能传入名称为className的prop参数，类型为string，且为必填

props与TypeScript - 为children添加类型

children是一个比较特殊的prop, 支持多种不同类型数据的传入，需要通过一个内置的ReactNode类型来做注解

```tsx
type Props = {
  className: string
  children: React.ReactNode
}

function Button(props: Props) {
  const { className, children } = props
  return <button className={className}>{children}</button>
}
```

说明：注解之后，children可以是多种类型，包括：React.ReactElement 、string、number、React.ReactFragment 、React.ReactPortal 、boolean、 null 、undefined

props与TypeScript - 为事件prop添加类型

组件经常执行类型为函数的prop实现子传父，这类prop重点在于函数参数类型的注解

```tsx
type Props = {
  onGetMsg?: (msg: string) => void
}

function Son(props: Props) {
  const { onGetMsg } = props

  const clickHandler = () => {
    onGetMsg?.('this is msg')
  }

  return <button onClick={clickHandler}>sendMsg</button>
}
```

```tsx
return (
  <>
    <Son onGetMsg={(msg) => console.log(msg)} />
  </>
```

(parameter) msg: string

```tsx
const getMsgHandler = (msg: string) => {
  console.log(msg)
}
return (
  <>
    <Son onGetMsg={getMsgHandler} />
  </>
)
```

说明：

1. 在组件内部调用时需要遵守类型的约束，参数传递需要满足要求
2. 绑定prop时如果绑定内联函数直接可以推断出参数类型，否则需要单独注解匹配的参数类型

useRef与TypeScript

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

useRef与TypeScript - 获取dom

获取dom的场景，可以直接把要获取的dom元素的类型当成泛型参数传递给useRef,可以推导出.current属性的类型

```tsx
function App() {
  const domRef = useRef<HTMLInputElement>(null)

  useEffect(() => {
    domRef.current?.focus()
  }, [])

  return (
    <>
      <input ref={domRef} />
    </>
  )
}
```

useRef与TypeScript - 引用稳定的存储器

把useRef当成引用稳定的存储器使用的场景可以通过泛型传入联合类型来做，比如定时器的场景：

```tsx
function App() {
  const timerRef = useRef<number | undefined>(undefined)

  useEffect(() => {
    timerRef.current = setInterval(() => {
      console.log('1')
    }, 1000)

    return () => clearInterval(timerRef.current)
  }, [])

  return <>this is div</>
}
```