React + typescript + antd开发前端应用（三）todo list

ep76 关注

上一篇：React + typescript + antd开发前端应用（二）Hello world

本次将开发一个具备添加和删除的待办列表功能，主要学习React的state和触发事件的代码编写知识。

1、功能描述

该功能的基本界面包括一个输入框、一个添加按钮、一个删除按钮和一个显示任务列表的控件。在输入框中输入任务名称，点击添加按钮后将刚刚输入的任务加入到列表中；勾选已完成的任务，点击删除按钮删除选中的任务。

2、界面设计

可以用ppt画出一个草图，设计这个todo list的UI，草图如下：

界面设计草图

3、创建静态UI

根据草图，使用antd的栅格系统将页面划分成对应的区域，并在相应位置存放页面元素，修改后的DemoApp.tsx文件内容如下：

```
import React from 'react';
import { Input, Button, Row, Col, Checkbox } from 'antd';

function DemoApp() {
    return (
        <React.Fragment>
            <h1 style={{"textAlign": "center"}}>待办列表</h1>
            <hr></hr>
            <Row align='midd
```

```
                    <Col span={1}>任务标题</Col>
                    <Col span={11}><Input></Input></Col>
                    <Col span={12}><Button type='primary'>添加</Button></Col>
                </Row>
                <Row>
                    <Col span={24}>
                        <ul>
                            <li><Checkbox></Checkbox>任务一</li>
                            <li><Checkbox></Checkbox>任务二</li>
                        </ul>
                    </Col>
                </Row>
                <Row gutter={5}>
                    <Col span={12}></Col>
                    <Col span={12}>
                        <Button type='primary'>删除</Button>
                    </Col>
                </Row>
            </React.Fragment>
        );
    }

    export default DemoApp;
```

静态页面效果

仔细阅读源码，在第二行以解构赋值的形式导入了antd的基础组件，在函数式组件DemoApp函数的返回值中根据设计，返回对应的页面元素。相关源码细节比如align、gutter等属性的作用，可参考antd相关文档。这里需要说明的是，TSX使用大括号语法来嵌入值，在大括号中可以书写的是任意合法的typescript变量或合法的typescript表达式。在TSX的页面标签style元素中直接内嵌css表达式时，需要使用{{css表达式}}的形式，注意这里css的属性不是css的属性，而是去掉css属性中的-，且将-后面的第一个字符变为大小，例如：css属性text-align需要写成textAlign，css属性background-color需要携程backgroundColor。

4、组件状态

React组件状态就是state，是一个组件所包含的状态信息。所谓“状态”，就是该组件任意时刻都存放的各种数据。 state最重要的一点就是，当组件更新state时，会触发组件的重新渲染，可以认为发生了局部刷新。根据需求，todo list有两个state：一个是输入框对应的新增任务名称，另外一个是保存已添加的任务列表。使用React.useState函数，定义组件状态，在函数DemoApp中，return语句前增加state的定义代码：

```
const [dataInfo, setDataInfo] = React.useState({
    taskName: '新任务名称',
    taskList: ["编写React教程", "吃饭", "睡觉"]
});
```

这段代码调用React.useState函数
对象，第二个元素是修改state数

面元素重新渲染的动作。这时React在函数组件中初始化state对象的固有写法。

然后再将taskName和输入框绑定，并在页面上显示已有的任务列表，修改后DemoApp.tsx文件内容如下：

```
import React from 'react';
import { Input, Button, Row, Col, Checkbox } from 'antd';

function DemoApp() {
    const [dataInfo, setDataInfo] = React.useState({
        taskName: '新任务名称',
        taskList: ["编写React教程", "吃饭", "睡觉"]
    });
    return (
        <React.Fragment>
            <h1 style={{"textAlign": "center"}}>待办列表</h1>
            <hr></hr>
            <Row align='middle' gutter={5}>
                <Col span={1}>任务标题</Col>
                {/** value属性与state对象中的taskName绑定 */}
                <Col span={11}><Input value={dataInfo.taskName}></Input></Col>
                <Col span={12}><Button type='primary'>添加</Button></Col>
            </Row>
            <Row>
                <Col span={24}>
                    <ul>
                        {/** 将原来的静态内容修改为根据taskList内容输出列表 */}
                        {
                            dataInfo.taskList.map(oneItem => <li key={oneItem}><Checkb
                        }
                    </ul>
                </Col>
            </Row>
            <Row gutter={5}>
                <Col span={12}></Col>
                <Col span={12}>
                    <Button type='primary'>删除</Button>
                </Col>
            </Row>
        </React.Fragment>
    );
}

export default DemoApp;
```

修改代码的部分使用{/** */}的形式增加了文字注释。完成代码修改后，通过刷新浏览器中的页面（如果使用npm run start启动了前端服务，浏览器中的页面会自动刷新）。如下图所示：

可以看到输入框中显示了state中taskName属性的值，任务列表区也显示了taskList数组中的数据。这里因为使用循环输出了数据的值，所以为每项<li>标签增加了key属性，否则在浏览器的开发者工具中会出现元素缺少key属性的警告。

5、为Input绑定onChange事件

当我们尝试在输入框中输入新的taskName值时，发现输入框中的值并不会改变。这是因为React绑定数据时单项的。要产生修改state的效果，需要为Input绑定onChange事件。也就是说：当Input中的值发生改变时会触发onChange，在onChange事件的处理函数中，我们需要同步修改state中的taskName属性的值。修改后的DemoApp.tsx内容如下：

```
import React from 'react';
import { Input, Button, Row, Col, Checkbox } from 'antd';

function DemoApp() {
    const [dataInfo, setDataInfo] = React.useState({
        taskName: '新任务名称',
        taskList: ["编写React教程", "吃饭", "睡觉"]
    });
    /** 定义函数onChange处理函数，调用setDataInfo函数修改state的值 */
    function taskNameChanged(newTaskname: string) {
        let newStateInfo = {...dataInfo};
        newStateInfo.taskName = newTaskname;
        setDataInfo(newStateInfo);
    }
    return (
        <React.Fragment>
            <h1 style={{"textAlign": "center"}}>待办列表</h1>
            <hr></hr>
            <Row align='middle' gutter={5}>
                <Col span={1}>任务标题</Col>
                {/** 增加onChange事件处理 */}
                <Col span={11}><Input value={dataInfo.taskName} onChange={e => taskNam
                <Col span={12}><Button type='primary'>添加</Button></Col>
            </Row>
            <Row>
                <Col span={24}>
                    <ul>
                        {/** 将原来的静态内容修改为根据taskList内容输出列表 */}
                        {
                            dataInfo.taskList.map(oneItem => <li key={oneItem}><Checkb
                        }
                    </ul>
                </Col>
            </Row>
            <Row gutter={5}>
                <Col span={12}></Col>
                <Col span={12}>
                    <Button type='primary'>删除</Button>
                </Col>
            </Row>
        </React.Fragment>
    );
}

export default DemoApp;
```

在Input标签的onChange属性的值时一个Lambda表达式，传递参数为e，通过e.target.value将修改后的输入框的值作为参数传递
中，才有解构赋值的形式创建了–

state中的taskName值变更后，因为Input的value元素绑定了这个state中的taskName属性，在state更新后，页面Dom元素局部更新，因此更新了Input中的值。再次回到浏览器页面，现在可以在输入框中正常输入数据了，而且输入的数据会即时显示在输入框中。

这里需要注意：React中触发事件的名称，需要写成onXxx形式，这个事件名称的首字母必须大写。而普通的html元素的事件则全是小写。

根据需求：在输入新的任务名称后点击添加按钮，会将新的任务加入到任务列表中。当母亲点击添加按钮，页面没有任何变化。接下来我们就为按钮添加点击事件的处理逻辑。

6、为添加按钮绑定事件处理逻辑

根据需求，点击添加按钮后，需要将新的任务加入到任务列表中，也就是点击添加按钮后要在任务列表区显示新的任务列表。修改后的DemoApp.tsx文件内容如下：

```
import React from 'react';
import { Input, Button, Row, Col, Checkbox } from 'antd';

function DemoApp() {
    const [dataInfo, setDataInfo] = React.useState({
        taskName: '新任务名称',
        taskList: ["编写React教程", "吃饭", "睡觉"]
    });
    /** 定义函数onChange处理函数，调用setDataInfo函数修改state的值 */
    function taskNameChanged(newTaskname: string) {
        let newStateInfo = {...dataInfo};
        newStateInfo.taskName = newTaskname;
        setDataInfo(newStateInfo);
    }
    /** 添加按钮处理函数 */
    function addNewTaskToList() {
        let newStateInfo = {...dataInfo};
        newStateInfo.taskList = [...newStateInfo.taskList, newStateInfo.taskName];
        setDataInfo(newStateInfo);
    }
    return (
        <React.Fragment>
            <h1 style={{"textAlign": "center"}}>待办列表</h1>
            <hr></hr>
            <Row align='middle' gutter={5}>
                <Col span={1}>任务标题</Col>
                {/** 增加onChange事件处理 */}
                <Col span={11}><Input value={dataInfo.taskName} onChange={e => taskNam
                {/** 增加onClick事件处理 */}
                <Col span={12}><Button type='primary' onClick={addNewTaskToList}>添加<
            </Row>
            <Row>
                <Col span={24}>
                    <ul>
                        {/** 将原来的静态内容修改为根据taskList内容输出列表 */}
                        {
                            dataInfo.taskList.map(oneItem => <li key={oneItem}><Checkb
                        }
                    </ul>
                </Col>
            </Row>
            <Row gutter={5}>
                <Col span={12}></Col>
                <Col span={12}>
                    <Button
                </Col>
```

```
            </Row>
        </React.Fragment>
    );
}

export default DemoApp;
```

addNewTaskToList函数的基本逻辑时：解构赋值创建新的state对象，然后将taskName通过解构赋值的方式加入到state的taskList的数组中。这里添加Array对象等还可以采用Array对象的push函数。

这里可能你已经注意到了：每次调用setDataInfo函数修改state时，都需要一个新的state对象，否则不会触发DOM元素的重新渲染。

截至到目前，我们已经完成了任务添加的功能，只是还有一些的瑕疵：

1、理论上，这个页面首次挂载的时候不应该有任务的列表，但是如果我直接修改定义state的taskList属性为空，如下代码所示：

```
const [dataInfo, setDataInfo] = React.useState({
    taskName: '新任务名称',
    taskList: []
});
```

则后续的添加代码编译不通过，会提示不能把string类型的数据赋值给never类型的数据。

2、点击添加后，未能清空数控中的内容，每次都要手工清除，比较麻烦。

3、点击添加按钮时，怎么知道用户当前选中了哪些任务呢？

7、改进后的todo list

1、定义state时指定state中各属性的数据类型

相对于javascript来说，typescript是一种强类型的语言，因此在数据赋值等过程中，需要对相关数据类型进行校验，目的是将Bug消灭在开发期（但这实际上不可能）。因此我们在定义state时，可以使用泛型的方式指定state对象的各属性值，修改state定义代码如下所示：

```
const [dataInfo, setDataInfo] = React.useState<{
    taskName: string,
    taskList: {
        taskName: string,
        checkedFlag: boolean
    }[]
}>({
    taskName: '',
    taskList: []
});
```

这代码看上去有点小复杂，但仔细分析一些还是很好理解的：本质上以上代码可以解释为React.useState<数据类型定义>(初始化对象的定义)。其中数据类型定义和初始化对象的定义都采用字面量的形式进行定义，都是“键-值”对的形式。类型定义对象中的值是类属性的数据类型，而初始化对象的值是类属性对应的数据值。这里taskList由原来的string[]类型变更成了类[]，增加了checkedFlag属性，目的是和任务列表中的Checkbox的checked属性对应，用来标识该项任务列表是否被中。由于列表数据类型发生了变化，因此循环显示任务列表的TSX代码也要做对应修改：

```
<ul>
    {/** 将原来的静态内容修改为根据taskList内容输出列表 */}
    {
        dataInfo.taskList.map(oneItem => <li key={oneItem.taskName}><Checkbox checked=
    }
</ul>
```

同理，添加任务列表的逻辑也需要调整：

```
/** 添加按钮处理函数 */
function addNewTaskToList() {
    let newStateInfo = {...dataInfo};
    newStateInfo.taskList = [...newStateInfo.taskList, {taskName: newStateInfo.taskNam
    newStateInfo.taskName = '';
    setDataInfo(newStateInfo);
}
```

现在添加列表的不是一个string，而是要给对象了。

2、添加任务后，情况输入框中的输入内容

其实就是在addNewTaskToList函数的setDataInfo调用前，增加了清空taskName的属性值的代码。

3、添加Checkbox的onChange事件处理逻辑

因为Checkbox的checked属性与任务列表中的checkedFlag属性绑定了，如果不添加onChange事件会出现和Input不绑定onChange事件的结果一样：无论你怎么点击Checkbox都不会处于选中状态。增加的Checkbox onChange事件处理逻辑如下：

```
/** Checkbox onChange处理逻辑 */
function checkedOrNotchecked(e: CheckboxChangeEvent) {
    let newStateInfo = {...dataInfo};
    newStateInfo.taskList.map(item => {
        if (item.taskName === e.target.value) {
            return item.checkedFlag = e.target.checked;
        } else {
            return false;
        }
    });
    setDataInfo(newStateInfo);
}
```

逻辑的基本处理思路就是：遍历taskList数组，将对应的任务项的checkFlag值变更为当前Checkbox的状态。记得最后一定要更新state，否则不会触发Dom元素的重新渲染动作。

4、添加删除按钮的点击事件处理逻辑

因为任务列表不再是一个string，因此我们很容易通过遍历任务列表知道哪些待办任务被选中，从而删除选中的待办任务。删除函数处理逻辑如下：

```
/** 删除选中的待办任务 */
const deleteCheckedTaskName = () => {
    let newStateInfo = {...dataInfo};
    newStateInfo.taskList = newStateInfo.taskList.filter(item => !item.checkedFlag);
    setDataInfo(newStateInfo);
};
```

这里采用了定义一个Lambda表达式（也叫箭头函数）变量的形式定义额了一个函数，和采用关键字function定义一个函数没有本质上的区别。删除的基本逻辑就是：变量taskList，过滤那些被选中的项。记得最后一定要更新state，否则不会触发Dom元素的重新渲染动作。

8、完整代码

```
import React from 'react';
import { Input, Button, Row, Col, Checkbox } from 'antd';
import { CheckboxChangeEvent } from 'antd/es/checkbox';

function DemoApp() {
    const [dataInfo, setDataInfo] = React.useState<{
        taskName: string,
        taskList: {
            taskName: string,
            checkedFlag: boolean
        }[]
    }>({
        taskName: '',
        taskList: []
    });
    /** 定义函数onChange处理函数，调用setDataInfo函数修改state的值 */
    function taskNameChanged(newTaskname: string) {
        let newStateInfo = {...dataInfo};
        newStateInfo.taskName = newTaskname;
        setDataInfo(newStateInfo);
    }
    /** 添加按钮处理函数 */
    function addNewTaskToList() {
        let newStateInfo = {...dataInfo};
        newStateInfo.taskList = [...newStateInfo.taskList, {taskName: newStateInfo.tas
        newStateInfo.taskName = '';
        setDataInfo(newStateInfo);
    }
    /** Checkbox onChange处理逻辑 */
    function checkedOrNotchecked(e: CheckboxChangeEvent) {
        let newStateInfo = {...dataInfo};
        newStateInfo.taskList.map(item => {
            if (item.taskName === e.target.value) {
                return item.checkedFlag = e.target.checked;
            } else {
                return false;
            }
        });
        setDataInfo(newStateInfo);
    }
    /** 删除选中的待办任务 */
    const deleteCheckedTaskName = () => {
        let newStateInfo = {...dataInfo};
        newStateInfo.taskList = newStateInfo.taskList.filter(item => !item.checkedFlag
        setDataInfo(newStateInfo);
    };
    return (
        <React.Fragment>
            <h1 style={{"textAlign": "center"}}>待办列表</h1>
            <hr></hr>
            <Row align='middle' gutter={5}>
                <Col span={1}>任务标题</Col>
                {/** 增加onChange事件处理 */}
                <Col span={11}><Input value={dataInfo.taskName} onChange={e => taskNam
                {/** 增加onCl
```

```
                    <Col span={12}><Button type='primary' onClick={addNewTaskToList}>添加<,
                </Row>
                <Row>
                    <Col span={24}>
                        <ul>
                            {/** 绑定onChange事件，否则无法选中复选框；绑定value值的目的是知道选
                            {
                                dataInfo.taskList.map(oneItem => <li key={oneItem.taskName
                            }
                        </ul>
                    </Col>
                </Row>
                <Row gutter={5}>
                    <Col span={12}></Col>
                    <Col span={12}>
                        {/** 绑定点击事件函数，删除选中记录 */}
                        <Button type='primary' onClick={deleteCheckedTaskName}>删除</Buttor
                    </Col>
                </Row>
            </React.Fragment>
        );
    }

    export default DemoApp;
```

截至到目前，todo list的功能开发就基本就完成了。过程中可以看到涉及到多方面的typescript的基础知识，建议读者在阅读本文过程中，对于那些不理解的代码，通过自行查阅相关资料，或者通过搜索引擎解决相关问题，或者通过与具有相同兴趣爱好的同事、朋友交流，相互答疑解惑，大家共同进步。也可以在文章评论区与网友们讨论。

下一篇：React + typescript + antd开发前端应用（四）代码拆分

编辑于 2023-10-03 05:03 · IP 属地上海

Angular Ant Design TypeScript

欢迎参与讨论

还没有评论，发表第一个评论吧

Typescript & React 相关积累

介绍 Typescript 配合 React 的基本使用，和一些实用的小技巧。 主要目的在于完善基本认知的统一，并对一些十分常见且实用的功能能够有一定的深入学习。 如何寻找一个类型文件位置常见 node...

折木

TypeScript在react项目中的实践

慕课网 发表于猿论

React + TypeScript 实践

字节前端

为什
Fibe

前端