知乎

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

ep76

关注

上一篇：React + typescript + antd开发前端应用（三）

上一篇的内容比较多，建议初学者反复练习这个需求，这样能快速熟悉React及typescript的相关语法及API。通过练习，可以感觉到一个组件的代码如果写在一个文件中，对于一个稍微大一些的组件（大一些的组件，可以理解成页面标签比较多的组件），那么一个文件代码行数太多，维护起来就比较困难的。其实对于组件，React官方的认为一个组件和他的行为代码，通常都是紧耦合的，没有必要进行拆分。但笔者见过一个.tsx文件有上千行的代码，那的确是一个不太好维护的组件。

对于React的代码拆分，可以从两个方法来考虑：一是将组件拆解成父子组件，二是将组件的行为代码和TSX代码进行分离。

1、将组件拆分为父子组件

还是以todo list为例子，我们将显示组件的列表拆成另外一个TaskList.tsx组件。在src\demo目录中新建TaskList.tsx文件，编辑文件内容如下：

```
import React from 'react';
import { Checkbox } from 'antd';

//定义接收父组件传递过来的state中的业务数据类型
type ParamsType = {
    taskDataInfo: {
        taskName: string,
        taskList: {
            taskName: string,
            checkedFlag: boolean
        }[]
    }
};

//函数式组件中的参数，父组件传递过来的业务数据
function TaskList(props: ParamsType) {
    return (
        <React.Fragment>
            {
                {/** 遍历父组件state中的taskList数据，生成<li>标签 */}
                props.taskDataInfo.taskList.map(oneItem => {
                    return (
                        <li key={oneItem.taskName}>
                            <Checkbox checked={oneItem.checkedFlag} value={oneItem.tas
                                {oneItem.taskName}
                            </Checkbox>

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

知乎

TaskList.tsx首先利用关键值type定义了一个数据类型，这个type相当于C语言中的结构体，就是定义了数据父组件传递给子组件的数据的数据类型。修改DemoApp.tsx组件的代码，使用刚刚创建的子组件显示添加后的待办任务列表：

```
import React from 'react';
import { Input, Button, Row, Col, Checkbox } from 'antd';
import { CheckboxChangeEvent } from 'antd/es/checkbox';
//引入刚刚创建的子组件列表，以便在后面使用<TaskList></TaskList>组件
import TaskList from './TaskList';
//这里隐去了其他代码
......
function DemoApp() {
    return (
        <React.Fragment>
            {/** 这里隐去了其他代码 */}
            ......
            <Row>
                <Col span={24}>
                    <ul>
                        {/** 使用子组件显示待办任务列表 */}
                        <TaskList taskDataInfo={dataInfo}></TaskList>
                    </ul>
                </Col>
            </Row>
            ......
        </React.Fragment>
    );
}

export default DemoApp;
```

DemoApp.tsx组件的代码重点是添加了import TaskList from './TaskList'和后面使用组件<TaskList taskDataInfo={dataInfo}> </TaskList>，这两行代码。这里标签属性taskDataInfo和TaskList.tsx文件中中type定义中字面量的key值的大小写保持一致，值就是向子组件传递的state中的数据。

完成代码修改后，输入任务列表，点击添加按钮，可以正常添加待办列表，但是点击Checkbox，不能选中需要删除的待办任务项，这是怎么回事呢？

对了，这是因为没有为Checkbox绑定onChange事件处理函数，这个函数在父组件已经有定义了，我们只要在子组件中直接调用就可以了。我们说在typescript中，函数也是一种对象，可以作为参数值传递给其他函数或者是子组件。那么这个函数又怎么传递给子组件呢？想想我们是怎么传递state中的数据的，数据怎么传递，函数就怎么传递。修改TaskList.tsx文件代码如下：

```
import React from 'react';
import { Checkbox } from 'antd';
//引入Checkbox onChange事件的参数类型
import { CheckboxChangeEvent } from 'antd/es/checkbox';

type ParamsType = {
    //......
    //增加定义接收函数类型的参数，
    checkboxChange: (event: CheckboxChangeEvent) => void
};

function TaskList(props: ParamsType) {
    return (
```

```
                props.taskDataInfo.taskList.map(oneItem => {
                    return (
                        <li key={oneItem.taskName}>
                            {/** onChange={props.checkboxChange}就是绑定事件处理函数的代
                            <Checkbox checked={oneItem.checkedFlag} onChange={props.ch
                                {oneItem.taskName}
                            </Checkbox>

                        </li>
                    )
                })
            }
        </React.Fragment>
    );
}

export default TaskList;
```

这里需要注意，对于state中的业务数据是响应式的，如果要修改父组件的数据，需要借助父组件的函数实现，二不能直接在子组件中修改state中的数据。

修改父组件，可以通过<TaskList>标签，传递函数给子组件以便子组件Checkbox状态改变时修改state中对应项的值：

```
<TaskList taskDataInfo={dataInfo} checkboxChange={checkedOrNotchecked}></TaskList>
```

属性checkboxChange的值就是一个函数（注意这里时函数本身，而不是函数调用，函数调用需要一对小括号），这个函数会作为参数被传递给子组件。

完成这些修改后，回到浏览器页面进行验证，发现点击Checkbox后，待办项被选中或取消选中，功能又正常了。

我们将待办任务列表迁移到子组件TaskList.tsx中，但父组件的代码还是显得有点多。下面我们尝试把部分代码迁移到一个独立的.ts文件中，让父组件只保留必要的代码。

2、迁移行为代码到独立的文件中

在src\demo目录新建一个DemoFuncs.ts文件，用于保存行为代码。

1、在DemoFuncs.ts文件中新建state数据结构

```
//任务列表项数据结构
export type TaskListItem = {
    taskName: string,
    checkedFlag: boolean
};
//state中的业务数据的数据结构
export type DataInfoType = {
    taskName: string,
    taskList: TaskListItem[]
};
```

修改DemoApp.tsx文件，引入DemoFuncs.ts并定义state数据结构：

```
//引入DemoFuncs.ts文件
import { DataInfoType } from './DemoFuncs';
```

```
    //以泛型方式定义state的数据，名初始化state中的业务数据对象
    const [dataInfo, setDataInfo] = React.useState<DataInfoType>({
        taskName: '',
        taskList: []
    });
}
```

迁移后，再次回到浏览器打开页面，测试相关功能是否正常运行。而且强烈初学者采用这种重构一点代码，即刻运行测试的方式逐步迁移代码，以免最后最后测试功能时发现异常，却不知道哪一步迁移代码出现了问题。

2、移动函数taskNameChanged到DemoFuncs.ts文件中并修改：

```
/**
 * 输入框onChange事件处理函数
 * @param newTaskname 修改后的值
 * @param dataInfo state中的业务数据
 * @param setDataInfo 赋值修改值
 */
export function taskNameChanged(newTaskname: string, dataInfo: DataInfoType, setDataIn
    let newStateInfo = {...dataInfo};
    newStateInfo.taskName = newTaskname;
    setDataInfo(newStateInfo);
}
```

迁移到新的文件中的函数，部分函数用完定义在DemoApp.tsx中，因为函数中涉及到的参数不时在DemoFuncs.ts文件中声明的，因此需要以参数的新形势定义新的事件处理函数。这里newTaskname就是输入框中的输入值，dataInfo就是state中的业务数据，setDataInfo时定义在父组件中修改state业务数据的函数。

修改DemoApp.tsx中Input的onChange触发事件的代码，调用新的函数并传递相关业务数据：

```
<Input value={dataInfo.taskName} onChange={e => taskNameChanged(e.target.value, dataIn
```

这里onChange的值修改成一个Lambda表达式（也叫箭头函数），以完成新事件处理函数的绑定。

注意：定义在DemoFuncs.ts中的taskNameChanged函数type等需要在前面添加export关键字，这样才能在别的文件中使用import语句导入，导入之后才能使用：

```
import { DataInfoType,taskNameChanged } from './DemoFuncs';
```

采用类似方式迁移其他函数。最终形成代码和TSX代码块分离的目的。

3、迁移后的代码

最终，这个todo list包括上个文件：DemoApp.tsx、DemoFuncs.ts、TaskList.tsx，则三个文件构成了todo list的所有资源。以下上三个文件的最终代码。

DemoApp.tsx：

```
import React from 'react';
import { Input, Button, Row, Col } from 'antd';
import TaskList from './TaskList';
```

知乎

```
function DemoApp() {
    /** 定义state数据类型 */
    const [dataInfo, setDataInfo] = React.useState<DataInfoType>({
        taskName: '',
        taskList: []
    });

    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(
            </Row>
            <Row>
                <Col span={24}>
                    <ul>
                        <TaskList taskDataInfo={dataInfo} changeFunc={(e) => checkedOr
                    </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(dataIn
                </Col>
            </Row>
        </React.Fragment>
    );
}

export default DemoApp;
```

DemoFuncs.ts：

```
import { CheckboxChangeEvent } from "antd/es/checkbox";

export type TaskListItem = {
    taskName: string,
    checkedFlag: boolean
};

export type DataInfoType = {
    taskName: string,
    taskList: TaskListItem[]
};
/**
 * 输入框onChange事件处理函数
 * @param newTaskname 修改后的值
 * @param dataInfo state中的业务数据
 * @param setDataInfo 赋值修改值
 */
export function taskNameChanged(newTaskname: string, dataInfo: DataInfoType, setDataIn
    let newStateInfo = {...dataInfo};
```

知乎

```
}
/**
 * 添加按钮点击事件处理函数
 * @param dataInfo state中的业务数据
 * @param setDataInfo state修改值函数
 */
export function addNewTaskToList(dataInfo: DataInfoType, setDataInfo: (value: DataInfo
    let newStateInfo = {...dataInfo};
    newStateInfo.taskList = [...newStateInfo.taskList, {taskName: newStateInfo.taskNam
    newStateInfo.taskName = '';
    setDataInfo(newStateInfo);
}
/**
 * Checkbox onChange事件绑定函数
 * @param e CheckboxOnChange事件对象
 * @param dataInfo
 * @param setDataInfo
 */
export function checkedOrNotchecked(e: CheckboxChangeEvent, dataInfo: DataInfoType, se
    let newStateInfo = {...dataInfo};
    newStateInfo.taskList.map(item => {
        if (item.taskName === e.target.value) {
            return item.checkedFlag = e.target.checked;
        } else {
            return false;
        }
    });
    setDataInfo(newStateInfo);
}
/**
 * 删除按钮处理函数
 * @param dataInfo state数据对象
 * @param setDataInfo state值修改函数
 */
export const deleteCheckedTaskName = (dataInfo: DataInfoType, setDataInfo: (value: Dat
    let newStateInfo = {...dataInfo};
    newStateInfo.taskList = newStateInfo.taskList.filter(item => !item.checkedFlag);
    setDataInfo(newStateInfo);
};
```

TaskList.tsx：

```
import { Checkbox } from 'antd';
import { CheckboxChangeEvent } from 'antd/es/checkbox';
import React from 'react';
import { DataInfoType } from './DemoFuncs';

type ParamsType = {
    taskDataInfo: DataInfoType,
    changeFunc: (e: CheckboxChangeEvent) => void
};

function TaskList(props: ParamsType) {
    return (
        <React.Fragment>
            {
                props.taskDataInfo.taskList.map(oneItem => {
                    return (
                        <li key={oneItem.taskName}>
                            <Checkbox checked={oneItem.checkedFlag} onChange={props.ch
```

知乎

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

export default TaskList;
```

完成代码迁移后，现在整个世界是不是清净了不少？几千行代码一个组件的情况建议开发人员还是要避免的，否则维护代码的时候，可能你都很想穿越到过去抽自己几个耳光。

后一篇：React + typescript + antd开发前端应用（五）React路由

编辑于 2023-10-04 05:39 · IP 属地上海

Ant Design TypeScript React