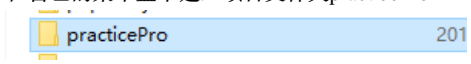
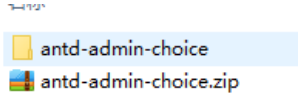
雏鹰一班dva项目实战（antd-admin-choice）

一、战前准备

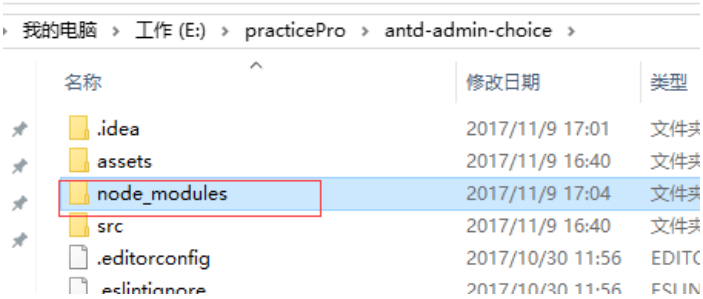
1、在自己的某个盘下建立项目文件夹practicePro



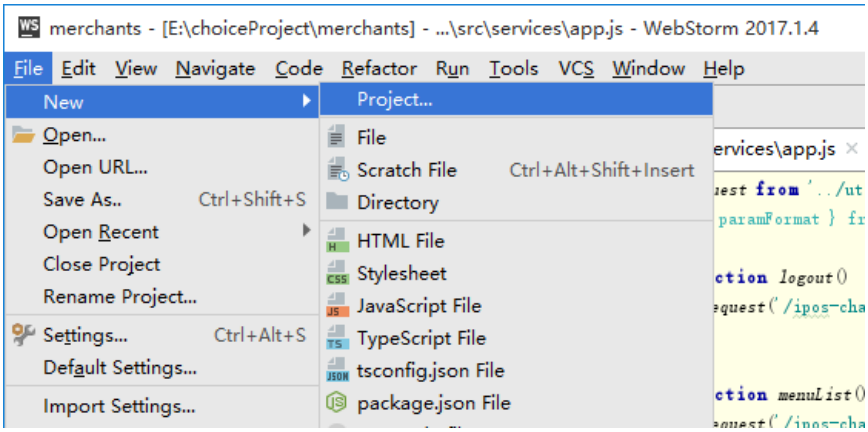
2、解压当前包到该文件夹

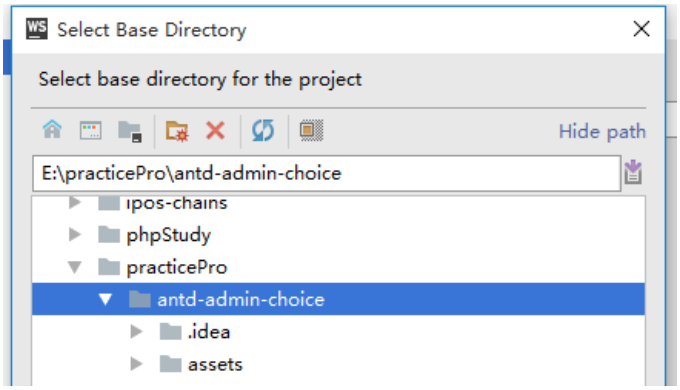


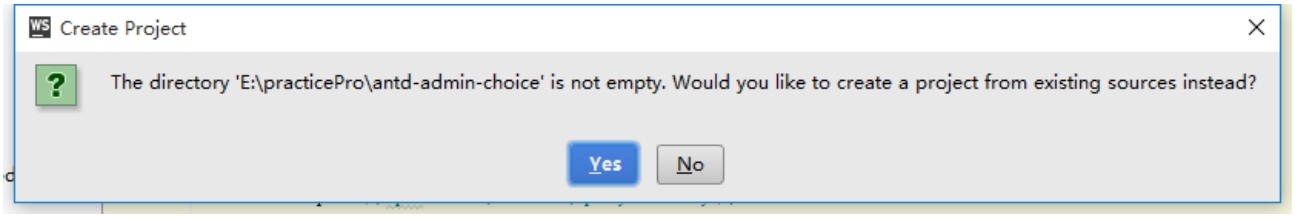
3、进入antd-admin-choice文件夹，建空文件夹node_modules（如果先安装依赖插件的话，webstorm难以加载）



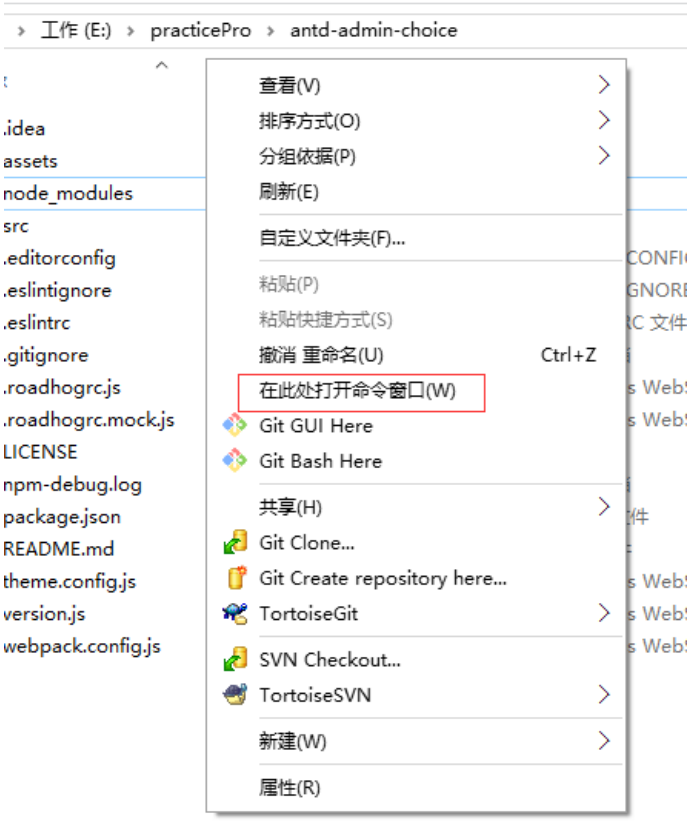
4、打开webstorm新建项目



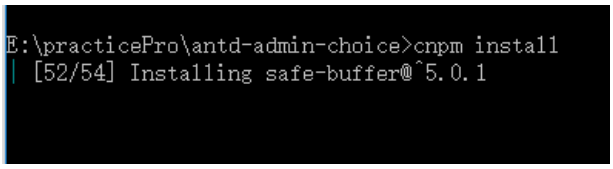


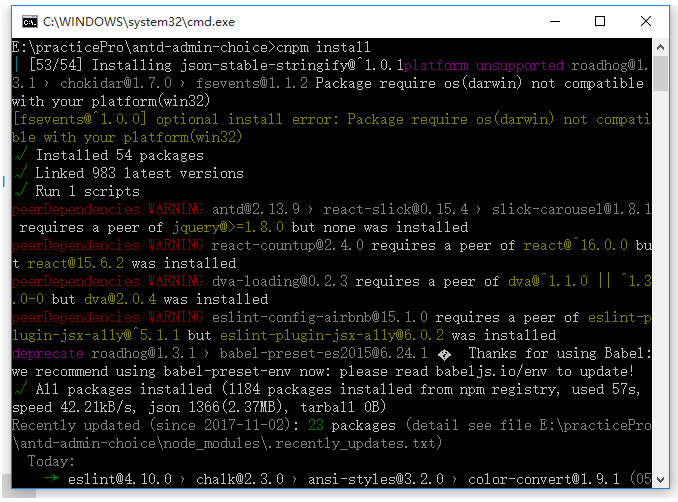


5、到该目录下shift加右键打开命令窗口

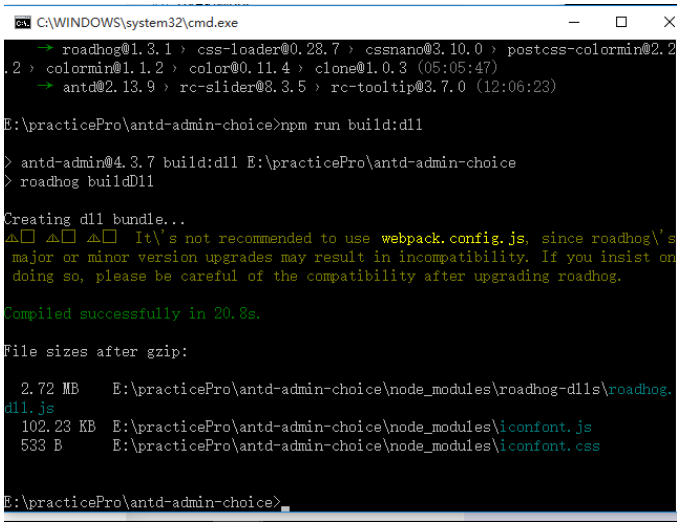


6、运行cnpm install安装依赖包

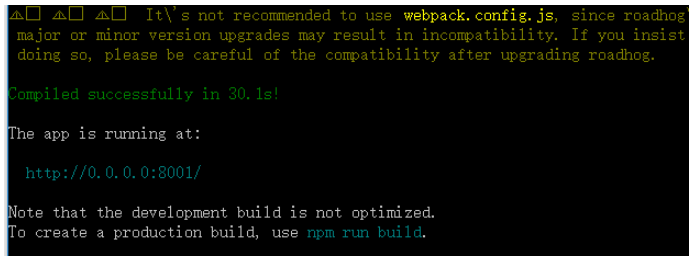




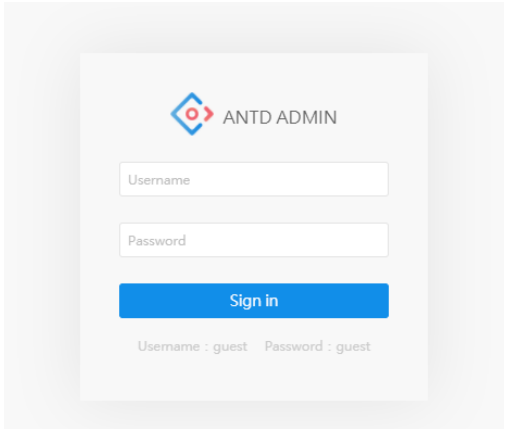
7、npm run build:dll #第一次npm run dev时需运行此命令，使开发时编译更快



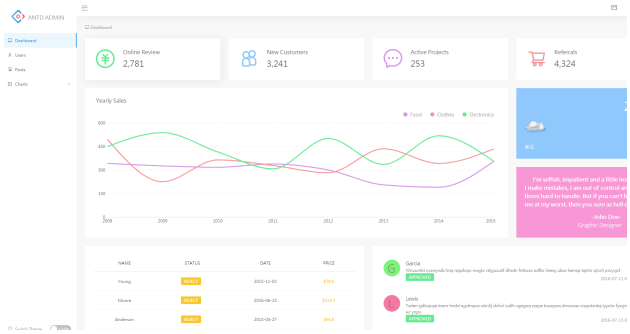
8、npm run dev运行命令



9、http://localhost:8000打开网站，登录名跟密码都是guest



10、登录上去了，是不是有点小激动啊，好戏才刚刚开始



二、实战背景

1、需求：参考good列表，实现一个菜品增加的菜品的增、删、改、查

2、字段：

编码（code）、缩写（abridge）、名称（name）、价格（price）、单位（unit）五个字段

id: '02029929',

code: '001',

name: '西红柿鸡蛋',

abridge: 'xhsjd',

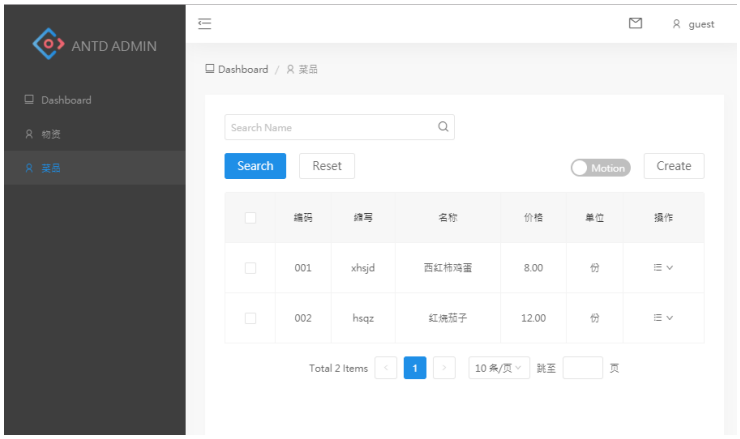
price: '8.00',

unit: '份',

3、接口：mock接口数据已经配置完毕，具体写法可以参考物资的例子

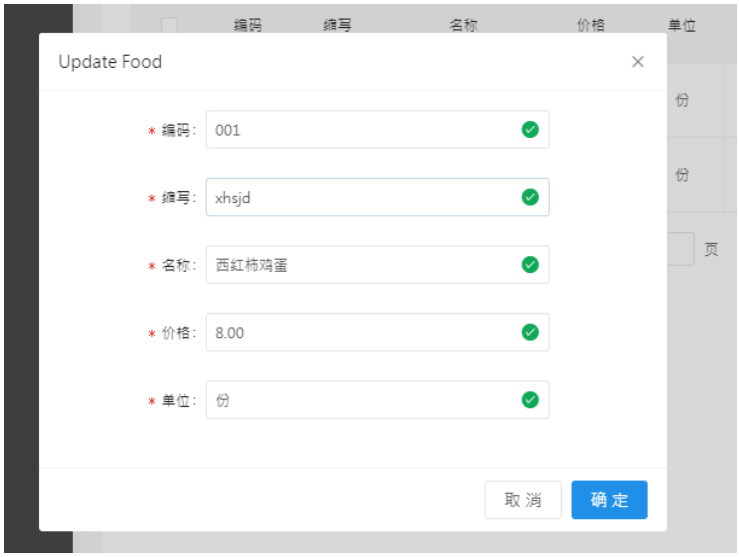
4、界面：

list:



add:

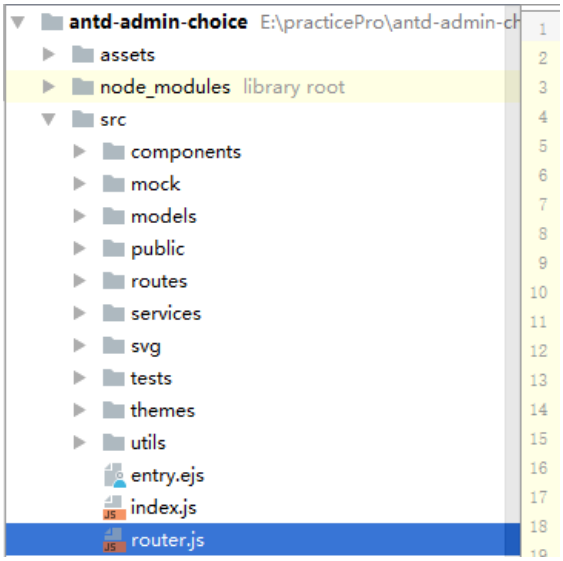
edit



三、实战演习

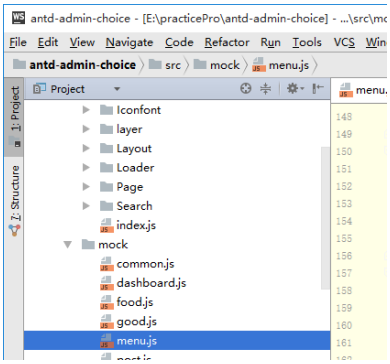
路由，service，model，展示组件、容器组件

1、打开./src/route.js配置路由，这里进行相应的配置即可



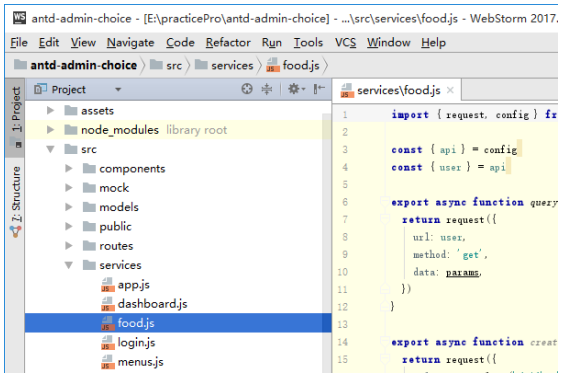

```
...
}, {
  path: '/post',
  models: () => [import('./models/post')],
  component: () => import('./routes/post/'),
},
{
  path: '/food',
  models: () => [import('./models/food')],
  component: () => import('./routes/food/'),
},
]
```

2、配置菜单，进入"./src/mock/menu.js"




```
{
  id: '800',
  bpid: '1',
  name: '物资',
  icon: 'user',
  route: '/good',
},
{
  id: '801',
  bpid: '1',
  name: '菜品',
  icon: 'user',
  route: '/food',
},
]
```

3、在“./services/”下新增数据接口文件food.js,这里是前后端数据交互的窗口



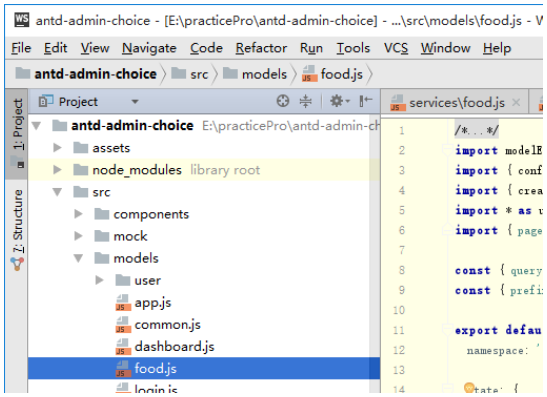
1）引入request文件，主要做数据的异步操作

```
import { request, config } from 'utils'

const { api } = config
const { food } = api
```

2）写具体的操作方法，这里的方法是数据模型model用到的

```
export async function query (params) {
 return request({
  url: food,
  method: 'get',
  data: params,
 })
}

export async function create (params) {
 return request({
  url: food.replace('/:id', ''),
  method: 'post',
  data: params,
 })
}

export async function remove (params) {
 return request({
  url: food,
  method: 'delete',
  data: params,
 })
}

export async function update (params) {
 return request({
  url: food,
  method: 'patch',
  data: params,
 })
}
```

4、建立数据模型food.js



1) 引入用到的组件跟连接的数据接口

```
/* global window */
import modelExtend from 'dva-model-extend'
import { config } from 'utils'
import { create, remove, update } from 'services/food'
import * as usersService from 'services/users'
import { pageModel } from './common'

const { query } = usersService
const { prefix } = config
```

2）定义namespace，这是数据在整个项目中的唯一标识

```
export default modelExtend(pageModel, {
  namespace: 'food',
```

3）定义state，这个就是该模块的数据结构

```
  state: {
    currentItem: {},
    modalVisible: false,
    modalType: 'create',
    selectedRowKeys: [],
    isMotion: window.localStorage.getItem(`${prefix}foodIsMotion`) === 'true',
  },
```

4）定义subscriptions，用于监听url变化时，调用effect中的query方法更新state，重新渲染页面

```
  subscriptions: {
    setup ({ dispatch, history }) {
      history.listen((location) => {
        if (location.pathname === '/food') {
          const payload = location.query || { current: 1, pageSize: 10 }
          dispatch({
            type: 'query',
            payload,
          })
        }
      })
    },
  },
```

5）定义effect中的异步方法，这些方法都是通过调用接口方法，进行增删改查

```
  effects: {

    * query ({ payload = {} }, { call, put }) {
      const data = yield call(query, payload)
      if (data) {
        yield put({
          type: 'querySuccess',
          payload: {
            list: data.data,
            。。。
          }
```

```
  },

  * delete ({ payload }, { call, put, select }) {
    const data = yield call(remove, { id: payload })
    const { selectedRowKeys } = yield select(_ => _.food)
    。。。
  },

  * multiDelete ({ payload }, { call, put }) {
    const data = yield call(usersService.remove, payload)
    。。。
  },

  * create ({ payload }, { call, put }) {
    const data = yield call(create, payload)
    。。。
  },

  * update ({ payload }, { select, call, put }) {
    const id = yield select(({ food }) => food.currentItem.id)
    。。。

},
```

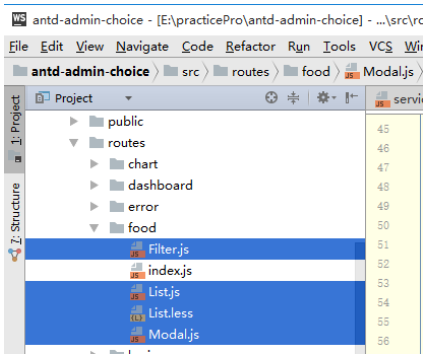
6）定义reducer里的方法，只有在这里才能更新state

```
  reducers: {

    showModal (state, { payload }) {
      return { ...state, ...payload, modalVisible: true }
    },

    hideModal (state) {
      return { ...state, modalVisible: false }
    },

    switchIsMotion (state) {
      window.localStorage.setItem(`${prefix}foodIsMotion`, !state.isMotion)
      return { ...state, isMotion: !state.isMotion }
    },

  },
})
```

5、 在“./src/routes”下建立food的组件文件夹，并新建Filter.js、List.js、Modal.js展示组件这里一般都会带一个less文件，用于样式的小调整

当然，如果需要样式，那么需要增加相应的less文件，里面以列表的子组件list.jsx为例说明。

需要说明的是，我这里用的是无状态组件，也就是纯函数组件，这里的具体组件没有类跟实例的说法，具体的setState方法不可用，生命周期函数也不可用用，数据改变的唯一方式是通过props在路由容器的父组件里dispatch相关方法进行改变。需要了解更多，请参阅无状态组件的相关介绍。



1）引入相关组件和样式

```
import React from 'react'
import PropTypes from 'prop-types'
import { Table, Modal } from 'antd'
。。。

const confirm = Modal.confirm
```

2）定义组件的变量和配置

```
const confirm = Modal.confirm

const List = ({ onDeleteItem, onEditItem, isMotion, location, ...tableProps }) => {
 location.query = queryString.parse(location.search)

 const handleMenuClick = (record, e) => {
  if (e.key === '1') {
   onEditItem(record)
  } else if (e.key === '2') {
   confirm({
    title: 'Are you sure delete this record?',
    onOk () {
     onDeleteItem(record.id)
    },
   })
  }
 }

 const columns = [
  {
   title: '编码',
   dataIndex: 'code',
   key: 'code',
  。。。
 ]

 const getBodyWrapperProps = {
  page: location.query.page,
  current: tableProps.pagination.current,
 }

 const getBodyWrapper = (body) => { return isMotion ? <AnimTableBody {...getBodyWrapperProps} body={body} /> : body }
```

3）输出相关dom

```
return (
  <div>
   <Table
    {...tableProps}
    className={classnames({ [styles.table]: true, [styles.motion]: isMotion })}
    bordered
    scroll={{ x: 1250 }}
    columns={columns}
```

```
      simple
      rowKey={record => record.id}
      getBodyWrapper={getBodyWrapper}
    />
  </div>
 )
}
```

4）配置PropTypes，验证props

```
List.propTypes = {
 onDeleteItem: PropTypes.func,
 onEditItem: PropTypes.func,
 isMotion: PropTypes.bool,
 location: PropTypes.object,
}

export default List
```

6、最后“./src/routes/food”在建立容器组件index.js,以上准备那么多的东西都是零散的，我们需要一个“管理者”把大家联通起来

1)引入相关资源

```
import React from 'react'
import PropTypes from 'prop-types'
import { routerRedux } from 'dva/router'
import { connect } from 'dva'
import { Row, Col, Button, Popconfirm } from 'antd'
import { Page } from 'components'
import queryString from 'query-string'
```

2）引入相关展示组件

```
import List from './List'
import Filter from './Filter'
import Modal from './Modal'
```

3）定义容器组件，从state里析构相关对象，这里的数据在下面讲到的数据模型里有定义

```
const Food = ({ location, dispatch, food, loading }) => {
 location.query = queryString.parse(location.search)
 const { list, pagination, currentItem, modalVisible, modalType, isMotion, selectedRowKeys } = food
 const { pageSize } = pagination
```

4)定义子组件的props（包括数据跟方法），数据都是通过props传递的

```
const listProps = {
 dataSource: list,
 loading: loading.effects['food/query'],
 pagination,
 location,
 isMotion,
 。。。
}
```

5）组合页面

```
 return (
  <Page inner>
   <Filter {...filterProps} />
   {
    selectedRowKeys.length > 0 &&
    <Row style={{ marginBottom: 24, textAlign: 'right', fontSize: 13 }}>
     <Col>
```

```
        {`Selected ${selectedRowKeys.length} items `}
        <Popconfirm title={'Are you sure delete these items?'} placement="left" onConfirm={handleDeleteItems}>
          <Button type="primary" size="large" style={{ marginLeft: 8 }}>Remove</Button>
        </Popconfirm>
      </Col>
    </Row>
  }
  <List {...listProps} />
  {modalVisible && <Modal {...modalProps} />}
 </Page>
)
}
```

6）关联数据模型，并输出页面

```
export default connect(({ food, loading }) => ({ food, loading }))(Fooa)
```

至此，实战演习结束，不知道大家的战况如何呢？