# IT慕课吧 - www.itmk8.com 搜集整理

# day01

## 1. 项目开发准备

1). 描述项目

2). 技术选型

3). API接口/接口文档/测试接口

## 2. 启动项目开发

1). 使用react脚手架创建项目

2). 开发环境运行: npm start

3). 生产环境打包运行: npm run build serve build

## 3. git管理项目

1). 创建远程仓库

2). 创建本地仓库

a. 配置.gitignore

b. git init

c. git add .

d. git commit -m "init"

3). 将本地仓库推送到远程仓库

git remote add origin url

git push origin master

4). 在本地创建dev分支, 并推送到远程

git checkout -b dev

git push origin dev

5). 如果本地有修改

git add .

git commit -m "xxx"

git push origin dev

6). 新的同事: 克隆仓库

git clone url

git checkout -b dev origin/dev

git pull origin dev

7). 如果远程修改

git pull origin dev

## 4. 创建项目的基本结构

api: ajax请求的模块

components: 非路由组件

pages: 路由组件

App.js: 应用的根组件

index.js: 入口js

## 5 引入antd

下载antd的包

按需打包: 只打包import引入组件的js/css

下载工具包

config-overrides.js

package.json

自定义主题

下载工具包

config-overrides.js

使用antd的组件

根据antd的文档编写

## 6. 引入路由

下载包: react-router-dom

拆分应用路由:

Login: 登陆

Admin: 后台管理界面

注册路由:

<BrowserRouter>

<Switch>

<Route path='' component={}/>

## 7. Login的静态组件

1). 自定义了一部分样式布局

2). 使用antd的组件实现登陆表单界面

Form / Form.Item

Input

Icon

Button

## 8. 收集表单数据和表单的前台验证

1). form对象

如何让包含<Form>的组件得到form对象? WrapLoginForm = Form.create()(LoginForm)

WrapLoginForm是LoginForm的父组件, 它给LoginForm传入form属性

用到了高阶函数和高阶组件的技术

2). 操作表单数据

form.getFieldDecorator('标识名称', {initialValue: 初始值, rules: []})(<Input/>)包装表单项组件标签

form.getFieldsValue(): 得到包含所有输入数据的对象

form.getFieldValue(id): 根据标识得到对应字段输入的数据

3). 前台表单验证

a. 声明式实时表单验证:

form.getFieldDecorator('标识名称', {rules: [{min: 4, message: '错误提示信息'}]})(<Input/>)

b. 自定义表单验证

form.getFieldDecorator('标识名称', {rules: [{validator: this.validatePwd}]})(<Input/>)

validatePwd = (rule, value, callback) => {

if(有问题) callback('错误提示信息') else callack()

}

c. 点击提示时统一验证

form.validateFields((error, values) => {

if(!error) {通过了验证, 发送ajax请求}

})

## 9. 高阶函数与高阶组件

1. 高阶函数

1). 一类特别的函数

a. 接受函数类型的参数

b. 返回值是函数

2). 常见

a. 定时器: setTimeout()/setInterval()

b. Promise: Promise(() => {}) then(value => {}, reason => {})

c. 数组遍历相关的方法: forEach()/filter()/map()/reduce()/find()/findIndex()

d. 函数对象的bind()

e. Form.create()() / getFieldDecorator()()

3). 高阶函数更新动态, 更加具有扩展性

2. 高阶组件

1). 本质就是一个函数

2). 接收一个组件(被包装组件), 返回一个新的组件(包装组件), 包装组件会向被包装组件传入特定属性

3). 作用: 扩展组件的功能

3. 高阶组件与高阶函数的关系

高阶组件是特别的高阶函数

接收一个组件函数, 返回是一个新的组件函数

# day02

## 1. 后台应用

启动后台应用: mongodb服务必须启动

使用postman测试接口(根据接口文档):

访问测试: post请求的参数在body中设置

保存测试接口

导出/导入所有测试接口

## 2. 编写ajax代码

1). ajax请求函数模块: api/ajax.js

封装axios + Promise

函数的返回值是promise对象 ===> 后面用上async/await

自己创建Promise

1. 内部统一处理请求异常: 外部的调用都不用使用try..catch来处理请求异常

2. 异步返回是响应数据(而不是响应对象): 外部的调用异步得到的就直接是数据了(response --> response.data)

2). 接口请求函数模块: api/index.js

根据接口文档编写(一定要具备这个能力)

接口请求函数: 使用ajax(), 返回值promise对象

3). 解决ajax跨域请求问题(开发时)

办法: 配置代理 ==> 只能解决开发环境

编码: package.json: proxy: "http://localhost:5000"

4). 对代理的理解

1). 是什么?

具有特定功能的程序

2). 运行在哪?

前台应用端

只能在开发时使用

3). 作用?

解决开发时的ajax请求跨域问题

a. 监视并拦截请求(3000)

b. 转发请求(4000)

4). 配置代理

告诉代理服务器一些信息: 比如转发的目标地址

开发环境: 前端工程师

生产环境: 后端工程师

5). async和await

a. 作用?

简化promise对象的使用: 不用再使用then()来指定成功/失败的回调函数

以同步编码(没有回调函数了)方式实现异步流程

b. 哪里写await?

在返回promise的表达式左侧写await: 不想要promise, 想要promise异步执行的成功的value数据

c. 哪里写async?

await所在函数(最近的)定义的左侧写async

## 3. 实现登陆(包含自动登陆)

login.jsx

1). 调用登陆的接口请求

2). 如果失败, 显示错误提示信息

3). 如果成功了:

保存user到local/内存中

跳转到admin

4). 如果内存中的user有值, 自动跳转到admin

src/index.js

读取local中user到内存中保存

admin.jsx

判断如果内存中没有user(\_id没有值), 自动跳转到login

storageUtils.js

包含使用localStorage来保存user相关操作的工具模块

使用第三库store

简化编码

兼容不同的浏览器

memoryUtils.js

用来在内存中保存数据(user)的工具类

## 4. 搭建admin的整体界面结构

1). 整体布局使用antd的Layout组件

2). 拆分组件

LeftNav: 左侧导航

Header: 右侧头部

3). 子路由

定义路由组件

注册路由

## 5. LeftNav组件

1). 使用antd的组件

Menu / Item / SubMenu

2). 使用react-router

withRouter(): 包装非路由组件, 给其传入history/location/match属性

history: push()/replace()/goBack()

location: pathname属性

match: params属性

3). componentWillMount与componentDidMount的比较

componentWillMount: 在第一次render()前调用一次, 为第一次render()准备数据(同步)

componentDidMount: 在第一次render()之后调用一次, 启动异步任务, 后面异步更新状态重新render

4). 根据动态生成Item和SubMenu的数组

map() + 递归: 多级菜单列表

reduce() + 递归: 多级菜单列表

5). 2个问题?

刷新时如何选中对应的菜单项?

selectedKey是当前请求的path

刷新子菜单路径时, 自动打开子菜单列表?

openKey是 一级列表项的某个子菜单项是当前对应的菜单项

# day03

## 1. Header组件

1). 界面静态布局

三角形效果

2). 获取登陆用户的名称显示

MemoryUtils

3). 当前时间

循环定时器, 每隔1s更新当前时间状态

格式化指定时间: dateUtils

4). 天气预报

使用jsonp库发jsonp请求百度天气预报接口

对jsonp请求的理解

5). 当前导航项的标题

得到当前请求的路由path: withRouter()包装非路由组件

根据path在menuList中遍历查找对应的item的title

6). 退出登陆

Modal组件显示提示

清除保存的user

跳转到login

7). 抽取通用的类链接按钮组件

通过...透传所有接收的属性: <Button {...props} /> <LinkButton>xxxx</LinkButton>

组件标签的所有子节点都会成为组件的children属性

## 2. jsonp解决ajax跨域的原理

1). jsonp只能解决GET类型的ajax请求跨域问题

2). jsonp请求不是ajax请求, 而是一般的get请求

3). 基本原理

浏览器端:

动态生成<script>来请求后台接口(src就是接口的url)

定义好用于接收响应数据的函数(fn), 并将函数名通过请求参数提交给后台(如: callback=fn)

服务器端:

接收到请求处理产生结果数据后, 返回一个函数调用的js代码, 并将结果数据作为实参传入函数调用

浏览器端:

收到响应自动执行函数调用的js代码, 也就执行了提前定义好的回调函数, 并得到了需要的结果数据

# day04: Category组件

## 1. 使用antd组件构建分类列表界面

Card

Table

Button

Icon

## 2. 相关接口请求函数

获取一级/二级分类列表

添加分类

更新分类

## 3. 异步显示一级分类列表

设计一级分类列表的状态: categorys

异步获取一级分类列表: componentDidMount(){}

更新状态, 显示

## 4. 显示二级分类列表

设计状态: subCategorys / parentId / parentName

显示二级分类列表: 根据parentId状态值, 异步获取分类列表

setState()的问题

setState()更新状态是异步更新的, 直接读取状态值还是旧的状态值

setState({}, [callback]), 回调函数是在状态更新且界面更新之后执行, 可以在此获取最新的状态

## 5. 更新分类

1). 界面

antd组件: Modal, Form, Input

显示/隐藏: showStatus状态为2/0

2). 功能

父组(Category)件得到子组件(AddForm)的数据(form)

调用更新分类的接口

重新获取分类列表

# day05

## 1. 添加分类

1). 界面

antd组件: Modal, Form, Select, Input

显示/隐藏: showStatus状态为1/0

2). 功能

父组(Category)件得到子组件(AddForm)的数据(form)

调用添加分类的接口

重新获取分类列表

## 2. Product整体路由

1). 配置子路由:

ProductHome / ProductDetail / ProductAddUpdate

<Route> / <Switch> / <Redirect>

2). 匹配路由的逻辑:

默认: 逐层匹配 <Route path='/product' component={ProductHome}/>

exact属性: 完全匹配

## 3. 分页实现技术(2种)

1). 前台分页

请求获取数据: 一次获取所有数据, 翻页时不需要再发请求

请求接口:

不需要指定请求参数: 页码(pageNum)和每页数量(pageSize)

响应数据: 所有数据的数组

2). 基于后台的分页

请求获取数据: 每次只获取当前页的数据, 翻页时要发请求

请求接口:

需要指定请求参数: 页码(pageNum)和每页数量(pageSize)

响应数据: 当前页数据的数组 + 总记录数(total)

3). 如何选择?

基本根据数据多少来选择

## 4. ProductHome组件

1). 分页显示

界面: <Card> / <Table> / Select / Icon / Input / Button

状态: products / total

接口请求函数需要的数据: pageNum, pageSize

异步获取第一页数据显示

调用分页的接口请求函数, 获取到当前页的products和总记录数total

更新状态: products / total

翻页:

绑定翻页的监听, 监听回调需要得到pageNum

异步获取指定页码的数据显示

2). 搜索分页

接口请求函数需要的数据:

pageSize: 每页的条目数

pageNum: 当前请求第几页 (从1开始)

productDesc / productName: searchName 根据商品描述/名称搜索

状态: searchType / searchName / 在用户操作时实时收集数据

异步搜索显示分页列表

如果searchName有值, 调用搜索的接口请求函数获取数据并更新状态

3). 更新商品的状态

初始显示: 根据product的status属性来显示 status = 1/2

点击切换:

绑定点击监听

异步请求更新状态

4). 进入详情界面

history.push('/product/detail', {product})

5). 进入添加界面

history.push('/product/addupdate')

## 5. ProductDetail组件

1). 读取商品数据: this.props.location.state.product

2). 显示商品信息: <Card> / List

3). 异步显示商品所属分类的名称

pCategoryId==0 : 异步获取categoryId的分类名称

pCategoryId!=0: 异步获取 pCategoryId/categoryId的分类名称

4). Promise.all([promise1, promise2])

返回值是promise

异步得到的是所有promsie的结果的数组

特点: 一次发多个请求, 只有当所有请求都成功, 才成功, 并得到成功的数据,一旦有一个失败, 整个都失败

# day06

## 1. ProductAddUpdate

1). 基本界面

Card / Form / Input / TextArea / Button

FormItem的label标题和layout

2). 分类的级联列表

Cascader的基本使用

异步获取一级分类列表, 生成一级分类options

如果当前是更新二级分类的商品, 异步获取对应的二级分类列表, 生成二级分类options, 并添加为对应option的children

async函数返回值是一个新promise对象, promise的结果和值由async函数的结果决定

当选择某个一级分类项时, 异步获取对应的二级分类列表, 生成二级分类options, 并添加为当前option的children

3). 表单数据收集与表单验证

## 2. PicturesWall

1). antd组件

Upload / Modal / Icon

根据示例DEMO改造编写

2). 上传图片

在<Upload>上配置接口的path和请求参数名

监视文件状态的改变: 上传中 / 上传完成/ 删除

在上传成功时, 保存好相关信息: name / url

为父组件提供获取已上传图片文件名数组的方法

3). 删除图片

当文件状态变为删除时, 调用删除图片的接口删除上传到后台的图片

4). 父组件调用子组件对象的方法: 使用ref技术

1. 创建ref容器: thi.pw = React.createRef()

2. 将ref容器交给需要获取的标签元素: <PicturesWall ref={this.pw} /> // 自动将将标签对象添加为pw对象的current属性

3. 通过ref容器读取标签元素: this.pw.current

# day07

## 1. RichTextEditor

1). 使用基于react的富文本编程器插件库: react-draft-wysiwyg

2). 参考库的DEMO和API文档编写

3). 如果还有不确定的, 百度搜索, 指定相对准确的关键字

## 2. 完成商品添加与修改功能

1). 收集输入数据

通过form收集: name/desc/price/pCategoryId/categoryId

通过ref收集: imgs/detail

如果是更新收集: \_id

将收集数据封装成product对象

2). 更新商品

定义添加和更新的接口请求函数

调用接口请求函数, 如果成功并返回商品列表界面

## 3. 角色管理

1). 角色前台分页显示

2). 添加角色

3). 给指定角色授权

界面: Tree

状态: checkedKeys, 根据传入的role的menus进行初始化

勾选某个Node时, 更新checkedKeys

点击OK时: 通过ref读取到子组件中的checkedKeys作为要更新product新的menus

发请求更新product

解决默认勾选不正常的bug: 利用组件的componentWillReceiveProps()

# day08

## 1. setState()的使用

1). setState(updater, [callback]),

updater为返回stateChange对象的函数: (state, props) => stateChange

接收的state和props被保证为最新的

2). setState(stateChange, [callback])

stateChange为对象,

callback是可选的回调函数, 在状态更新且界面更新后才执行

3). 总结:

对象方式是函数方式的简写方式

如果新状态不依赖于原状态 ===> 使用对象方式

如果新状态依赖于原状态 ===> 使用函数方式

如果需要在setState()后获取最新的状态数据, 在第二个callback函数中读取

## 2. setState()的异步与同步

1). setState()更新状态是异步还是同步的?

a. 执行setState()的位置?

在react控制的回调函数中: 生命周期勾子 / react事件监听回调

非react控制的异步回调函数中: 定时器回调 / 原生事件监听回调 / promise回调 /...

b. 异步 OR 同步?

react相关回调中: 异步

其它异步回调中: 同步

2). 关于异步的setState()

a. 多次调用, 如何处理?

setState({}): 合并更新一次状态, 只调用一次render()更新界面 ---状态更新和界面更新都合并了

setState(fn): 更新多次状态, 但只调用一次render()更新界面 ---状态更新没有合并, 但界面更新合并了

b. 如何得到异步更新后的状态数据?

在setState()的callback回调函数中

## 3. Component与PureComponent

1). Component存在的问题?

a. 父组件重新render(), 当前组件也会重新执行render(), 即使没有任何变化

b. 当前组件setState(), 重新执行render(), 即使state没有任何变化

2). 解决Component存在的问题

a. 原因: 组件的shouldcomponentUpdate()默认返回true, 即使数据没有变化render()都会重新执行

b. 办法1: 重写shouldComponentUpdate(), 判断如果数据有变化返回true, 否则返回false

c. 办法2: 使用PureComponent代替Component

d. 说明: 一般都使用PureComponent来优化组件性能

3). PureComponent的基本原理

a. 重写实现shouldComponentUpdate()

b. 对组件的新/旧state和props中的数据进行浅比较, 如果都没有变化, 返回false, 否则返回true

c. 一旦componentShouldUpdate()返回false不再执行用于更新的render()

4). 面试题:

组件的哪个生命周期勾子能实现组件优化?

PureComponent的原理?

区别Component与PureComponent?

## 4. 用户管理

1). 显示用户分页列表

2). 添加用户

3). 修改用户

4). 删除用户

## 5. 导航菜单权限控制

1). 基本思路(依赖于后台):

角色: 包含所拥有权限的所有菜单项key的数组: menus=[key1, key2, key3]

用户: 包含所属角色的ID: role\_id

当前登陆用户: user中已经包含了所属role对象

遍历显示菜单项时: 判断只有当有对应的权限才显示

2). 判断是否有权限的条件?

a. 如果当前用户是admin

b. 如果当前item是公开的

c. 当前用户有此item的权限: key有没有menus中

d. 如果当前用户有此item的某个子item的权限

# day09

## 1. redux理解

什么?: redux是专门做状态管理的独立第3方库, 不是react插件, 但一般都用在react项目中

作用?: 对应用中状态进行集中式的管理(写/读)

开发: 与react-redux, redux-thunk等插件配合使用

## 2. redux相关API

redux中包含: createStore(), applyMiddleware(), combineReducers()

store对象: getState(), dispatch(), subscribe()

react-redux:

<Provider store={store}>: 向所有的容器组件提供store

connect(

state => ({xxx: state.xxx}),

{actionCreator1, actionCreator2}

)(UI组件):

产生的就是容器组件, 负责向UI组件传递标签属性,

一般属性值从state中获取, 函数属性内部会执行dispatch分发action

## 3. redux核心概念(3个)

action:

默认是对象(同步action), {type: 'xxx', data: value}, 需要通过对应的actionCreator产生,

它的值也可以是函数(异步action), 需要引入redux-thunk才可以

reducer

根据老的state和指定的action, 返回一个新的state

不能修改老的state

store

redux最核心的管理对象

内部管理着: state和reducer

提供方法: getState(), dispatch(action), subscribe(listener)

## 4. redux工作流程

![](http://www.ruanyifeng.com/blogimg/asset/2016/bg2016091802.jpg)

## 5. 使用redux及相关库编码

需要引入的库:

redux

react-redux

redux-thunk

redux-devtools-extension(这个只在开发时需要)

redux文件夹:

action-types.js

actions.js

reducers.js

store.js

组件分2类:

ui组件(components): 不使用redux相关API

容器组件(containers): 通过connect()()生成的组件

# day10

## 1. 在项目中搭建redux整套环境

1). store.js

2). reducer.js

3). actions.js

4). action-types.js

5). index.js

6). 在需要与redux进行状态数据通信(读/写)的UI组件包装生成容器组件

## 2. 通过redux管理头部标题headTitle数据

1). action-types.js

2). actoins.js

3). reducer.js

4). 相关组件:

left-nav.js

header.js

## 3. 通过redux管理登陆用户信息user数据

1). action-types.js

2). actoin.js

3). reducer.js

4). 相关组件:

login.js

admin.js

left-nav.js

header.js

role.js

## 4. 自定义redux库

1). redux库向外暴露下面几个函数

createStore(): 接收的参数为reducer函数, 返回为store对象

combineReducers(): 接收包含n个reducer方法的对象, 返回一个新的reducer函数

applyMiddleware() // 暂不实现

2). store对象的内部结构

getState(): 返回值为内部保存的state数据

dispatch(): 参数为action对象

subscribe(): 参数为监听内部state更新的回调函数

3). combineReducers函数:

返回的总reducer函数内部会根据总的state和指定的action,

调用每个reducer函数得到对应的新的state, 并封装成一个新的总state对象返回

## 5. 自定义react-redux库

1). react-redux向外暴露了2个API

a. Provider组件类

b. connect函数

2). Provider组件

接收store属性

通过context将store暴露给所有的容器子组件

Provider原样渲染其所有标签子节点

3). connect函数

接收2个参数: mapStateToProps和mapDispatchToProps

connect()执行的返回值为一个高阶组件: 包装UI组件, 返回一个新的容器组件

mapStateToProps:

为一个函数, 返回包含n个一般属性对象,

容器组件中调用得到对象后, 初始化为容器组件的初始状态, 并指定为UI组件标签的一般属性

mapDispatchToProps:

如果为函数, 调用得到包含n个dispatch方法的对象

如果为对象, 遍历封装成包含n个dispatch方法的对象

将包含n个dispatch方法的对象分别作为函数属性传入UI组件

通过store绑定state变化的监听, 在回调函数中根据store中最新的state数据更新容器组件状态, 从而更新UI组件

# day11

## 1. 数据可视化

1). echarts(百度) ==> echarts-for-react

2). g2(阿里) ==> bizCharts

3). d3(国外)

## 2. 前台404界面

<Redirect from='/' to='/home' exact/>

<Route component={NotFound}/>

## 3. 打包应用运行

1). 解决生产环境ajax跨域问题

使用nginx的反向代理解决(一般由后台配置)

CORS: 允许浏览器端跨域

2). BrowserRouter模式刷新404的问题

a. 问题: 刷新某个路由路径时, 会出现404的错误

b. 原因: 项目根路径后的path路径会被当作后台路由路径, 去请求对应的后台路由, 但没有

c. 解决: 使用自定义中间件去读取返回index页面展现