这章我们正式开始搭建taro的开发框架

1. **搭建taro脚手架项目**

**第三方第三方**

首先,让我们搭建taro的脚手架,这个例子里我们假设项目名称为wos,则在命令行下运行  
taro init wos  
回答系列问题后,命令行会自动创建以wos为文件夹的脚手架项目.  
这个教程里,是否使用TypeScript选择了否,css预处理器选择了less,模板选择了redux  
如下所示:

Taro v1.2.4

Taro即将创建一个新项目!

Need help? Go and open issue: https://github.com/NervJS/taro/issues/new

? 请输入项目介绍！ wos

? 是否需要使用 TypeScript ？ No

? 请选择 CSS 预处理器（Sass/Less/Stylus） Less

? 请选择模板 Redux 模板

执行成功后,可以开始编译微信小程序:

cd wos

npm run dev:weapp

命令会执行一会儿,当出现

监听文件修改中...

说明编译完成,可以打开微信小程序开发工具,新建项目,选择wos文件夹后,可以看到小程序已经可以正常运行.

**2整合tara-ui**

在命令行下输入  
yarn add taro-ui  
现在,taro-ui已经整合成功,我们可以简单修改pages/indexs页面测试一下

+ import {aButton} from 'taro-ui'

render () {

return (

<View className='index'>

+<AtButton type='primary'>tryme</AtButton>

<Button className='add\_btn' onClick={this.props.add}>+</Button>

<Button className='dec\_btn' onClick={this.props.dec}>-</Button>

<Button className='dec\_btn' onClick={this.props.asyncAdd}>async</Button>

<View><Text>{this.props.counter.num}</Text></View>

<View><Text>Hello, World</Text></View>

</View>

)

}

再次运行  
npm run dev:weapp  
打开微信小程序开发工具,这时候你可以看到,首页里,已经多了一个蓝色的按钮.  
至此,我们已经搭建了taro的开发环境.

**3整合dva框架一点也不难**

下面的步骤,我们将整合dva框架.首先,我们先安装相关的依赖库  
yarn add dva-core dva-loading  
我们把所有的配置都集中在一起,在src目录下新建一个config文件夹,生成index.js,代码如下:

// 请求连接前缀

export const baseUrl = 'http://localhost:3721';

// 输出日志信息

export const noConsole = false;

dva会按照路由多model进行分层管理,在taro框架里,我们没有使用umi,所以需要统一管理model,让我们在/src目录下新建models文件夹,生成index.js文件,我们现在还没有model需要dva管理,因此,先空着.

export default [

]

我们让dva来管理我们的store,在src目录下新建util文件夹,生成dva.js文件,代码如下.

import Taro from '@tarojs/taro'

import { create } from 'dva-core'

import { createLogger } from 'redux-logger'

import createLoading from 'dva-loading'

let app;

let store;

let dispatch;

function createApp(opt) {

// redux日志

opt.onAction = [createLogger()];

app = create(opt);

app.use(createLoading({}));

// 适配支付宝小程序

if (Taro.getEnv() === Taro.ENV\_TYPE.ALIPAY) {

global = {};

}

if (!global.registered)

opt.models.forEach(

model =>{

app.model(model);

console.log(model)

});

global.registered = true;

app.start();

store = app.\_store;

app.getStore = () => store;

dispatch = store.dispatch;

app.dispatch = dispatch;

return app;

}

export default {

createApp:createApp,

getDispatch() {

return app.dispatch

}

}

让我们封装一下http请求你的工具类,同样,在src/util/目录下新建request.js文件,代码如下:

import Taro from '@tarojs/taro';

import { baseUrl, noConsole } from '../config';

const request\_data = {

platform: 'wap',

rent\_mode: 2,

};

export default (options = { method: 'GET', data: {} }) => {

if (!noConsole) {

console.log(`${new Date().toLocaleString()}【 M=${options.url} 】P=${JSON.stringify(options.data)}`);

}

return Taro.request({

url: baseUrl + options.url,

data: {

...request\_data,

...options.data

},

header: {

'Content-Type': 'application/json',

},

method: options.method.toUpperCase(),

}).then((res) => {

console.log("res:",res);

const { statusCode, data } = res;

if (statusCode >= 200 && statusCode < 300) {

if (!noConsole) {

console.log(`${new Date().toLocaleString()}【 M=${options.url} 】【接口响应：】`,res.data);

}

if (data.status !== 'ok') {

Taro.showToast({

title: `${res.data.error.message}~` || res.data.error.code,

icon: 'none',

mask: true,

});

console.error(`${res.data.error.message}~` || res.data.error.code);

}

return data;

} else {

throw new Error(`网络请求错误，状态码${statusCode}`);

}

})

}

典型的dva的开发目录如下:

每个文件件都需要4个文件,太复杂,所以我们在根目录下增加一个脚本template.js,自动生成文件:

/\*\*

\* pages模版快速生成脚本,执行命令 npm run tep `文件名`

\*/

const fs = require('fs');

const dirName = process.argv[2];

if (!dirName) {

console.log('文件夹名称不能为空！');

console.log('示例：npm run tep test');

process.exit(0);

}

// 页面模版

const indexTep = `import Taro, { Component } from '@tarojs/taro';

import { View } from '@tarojs/components';

import { connect } from '@tarojs/redux';

import './index.scss';

@connect(({${dirName}}) => ({

...${dirName},

}))

export default class ${titleCase(dirName)} extends Component {

config = {

navigationBarTitleText: '${dirName}',

};

componentDidMount = () => {

};

render() {

return (

<View className="${dirName}-page">

${dirName}

</View>

)

}

}

`;

// scss文件模版

const scssTep = `@import "../../styles/mixin";

.${dirName}-page {

@include wh(100%, 100%);

}

`;

// model文件模版

const modelTep = `import \* as ${dirName}Api from './service';

export default {

namespace: '${dirName}',

state: {

},

effects: {

\* effectsDemo(\_, { call, put }) {

const { status, data } = yield call(${dirName}Api.demo, {});

if (status === 'ok') {

yield put({ type: 'save',

payload: {

topData: data,

} });

}

},

},

reducers: {

save(state, { payload }) {

return { ...state, ...payload };

},

},

};

`;

// service页面模版

const serviceTep = `import Request from '../../utils/request';

export const demo = data => Request({

url: '路径',

method: 'POST',

data,

});

`;

fs.mkdirSync(`./src/pages/${dirName}`); // mkdir $1

process.chdir(`./src/pages/${dirName}`); // cd $1

fs.writeFileSync('index.js', indexTep);

fs.writeFileSync('index.scss', scssTep);

fs.writeFileSync('model.js', modelTep);

fs.writeFileSync('service.js', serviceTep);

console.log(`模版${dirName}已创建,请手动增加models`);

function titleCase(str) {

const array = str.toLowerCase().split(' ');

for (let i = 0; i < array.length; i++) {

array[i] = array[i][0].toUpperCase() + array[i].substring(1, array[i].length);

}

const string = array.join(' ');

return string;

}

process.exit(0);

作者：shtonyteng  
链接：https://www.jianshu.com/p/e5bbfb6ef3a9  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。