多页面的配置

有一个global的使用方式

```
glob.sync('./src/pages/**/main.js')
```

关于glob的使用方式 可以看这个链接 glob正则表达 匹配规则 这样子的话 pages下文件夹中的文件夹中的 `main.js` 也是会被匹配到的 但是如果加一个`**?` 就是只会匹配到一层文件夹下的main.js 关于vue.config.js中配置多页面的要求看下面的链接 配置pages

v-panel 的使用

它的宽度 是在属性width里面 默认是900px 所以 你在class

使用less

- 设置 属性值变量 对一个重复出现的css值 可以用@name:值 在下面的属性里直接用 @name这个值
- 选择器变成一个变量 或者属性变量 @{}加一个大括号

```
这样子的话  设置的pointer只会对wrap2下面的元素生效  因为&:伪类指的就是它的上层父元素
<style lang="less" scoped>
@color:#999;
.wrap-1 .wrap-2 {
  .wrap-3 {
    width: 20px;
    height: 20px;
    background-color: @color;
  }
  .wrap-4 {
    color: yellow;
  }
//   这样的话 也是它的父级下面的color为red
// 相当于 wrap1 wrap2 wrap4 -->color: yellow
// wrap1 wrap2--->color:red
// 所以还是上面的权重值比较高
  &{
    color: red;
  }
}
</style>
```

```
<div class="demo">
    <div class="demo-title"></div>
</div>

// 样式设置成下面这个样子
```

```
// 编译出来的是 demo-title....

.demo {
  &-title {
    width: 40px;
    height: 40px;
    background-color: aquamarine;
  }
}
// 如果改成 & &-title
// 这样子的话编译出来的就是 .demo .demo-title 有嵌套了
```

table的使用

mock在项目里面的用法 vea项目里面的解析

- path.normalize(path) 它会规范化给定的path 解析..和 .片段

express的使用

框架模板自动机的使用

1. npm install art-template --save

先理一下 vea项目里面对模板自动机使用的流程 有一个render.js 返回 artTemplate.render() 在viewPath 下面将对应的生成的文件放在了下面

在dev-server 在 app.get的时候 他是把数据先require到 再渲染到模板里面？？ 还是说直接是Mock.mock拦截了这个请求?

.render(source, data, options)

编译并返回渲染结果

```
// tpl是把这个文件里面的数据得到
// html = render(viewPath, data, withLocalVhtml);
// 懂了懂了 就是 TPL是一个模板的index.html 这个模板是在views下面的
//  然后里面要用到locals 和data的数据
// 把locals和data的数据进行传进去！！！
var html = artTemplate.render(tpl, { locals, data })
举个例子 下面这个样子
var html = artTemplate.render('hi, <%=value%>,<%=data.key%>,<%=data.code%>.', {
    value: 'aui',
    data: { key: 134, code: 345 }
})
打印出来就是下面这个样子
hi, aui,134,345.
static 下面的是最开始的原始模板
view下面的是生成的模板 这个模板是 在webpack.dev.conf.js的时候生成的吗？？？
```

在dev-server里面是这么做的 那么 如果是真的把 这个项目进行打包？？ 它又是怎么做的 我没有看到在build里面对模板自动框架做处理的代码

```
html = render(viewPath, data, withLocalVhtml);
```

dialog的使用

对于dialog使用的时候 点击确定按钮 要在达到某些条件的时候 才能出发这个确定的函数的执行

```
if (this.currentStep === 2) {
    this.flag = false
}else{
    e.preventDefault();
}
```

mock + express 的使用方法

只是单独用的mock 它的请求是不会出现在Network vue.config.js里面的.devServer 与webpack.config.js里面的devServer 是一样的 可以理解成他自己就是一个服务器 所以我之前 const app=express() 端口老是会+1

```
module.exports = {
  //...
  devServer: {
    before: function (app, server, compiler) {
      app.get('/some/path', function (req, res) {
        res.json({ custom: 'response' });
      });
    },
  },
};
或者可以使用中间件
我在mock.js 进行了定义 就是按照请求的路径 去对应的文件夹下面找对应的mock数据
在mock.js 定义了一个函数
把这个函数 写在 devServer.before里面
devServer: {
        before(app) {
            // 这个app就是一个express
            // 加载本地数据文件
            app.use(mock)
        }
}
这样子就可以了
```

UI2样式的使用

```
var route = []
route.push({
    path: '/customer/contacter',
    name: 'contacterRoute',
    meta: {
        title: '联系人 - 腾讯企点'
    },
    // component: r => require.ensure([], () =>
r(require('../../partner/contacter/route.vue')), 'contacter'),
    children: [{
        path: 'customer',
        name: 'fieldsCustomer',
        meta: {
            title: '联系人字段 - 腾讯企点'
        },
        // component: r => require.ensure([], () => {
        //     r(require('../../partner/contacter/fieldsCustomer.vue'));
        // }, 'contacter')
    }, {
        path: 'system',
        name: 'fieldsSystem',
        meta: {
            title: '联系人字段 - 腾讯企点'
        },
        // component: r => require.ensure([], () => {
        //     r(require('../../partner/contacter/fieldsSystem.vue'));
        // }, 'contacter')
    }, {
        path: 'uniq',
        name: 'uniqRule',
        meta: {
            title: '联系人字段 - 腾讯企点'
        },
        // component: r => require.ensure([], () => {
        //     r(require('../../partner/contacter/fieldsUniqRule.vue'));
        // }, 'contacter')
    }]
});
route.push({
    path: '/test',
    name: 'hhh'
})

// 这个函数 就是 你在这是route的时候 children的时候是直接写一个路径的 所以他就是在前面父
亲的路径下直接进行的拼接
var getList = function (pre, arr) {
    return arr.map(url => {
        if (url.children && url.children.length) {
            return getList((pre ? pre + '/' : pre) + url.path, url.children);
        }
        else {
            return (pre ? pre + '/' : pre) + url.path;
```

```
        }
    })
};
// 还要有一个flat的函数
var flat = function (arr) {
    var arr1 = []
    arr.forEach(item => {
        if (Array.isArray(item)) {
            arr1 = arr1.concat(item)
        } else {
            arr1.push(item)
        }
    })
    return arr1

}
// console.log(flat(getList("", route)))
// 这个意思就是 在U2里面设置的route 进行一个数组的形式
// 然后我现在请求的这个路径 如果是在U2路径里面的 就把我的
// checkUI2的标志置为true
// 就是意思是这个页面要用到UI2样式
flat(getList('', route)).forEach(url => {
    url = url.split(':')[0];
    console.log(url)
    // realReqUrl.indexOf(url) > -1 && (checkUI2 = true);
});
```

新的知识点 mixin

关于Vuex

Vue存在于window对象上时，Vuex会自动调用：Vue.use(Vuex)，因此在store中就不需要再显示调用 Vue 如何存在于window对象 就是script引进来vue的地址

自动化流程模块的解析

里面有个画布 是@tencent/qd-process-ui 这里学一下 QDPCanvas的使用方式 里面有一个menu的菜单 这些菜单就相当于你进入这个画板之前或者之后的配置页 本来奇怪为什么 进入工作流按钮点击了之后找不到路由跳转之后的函数 实际上就是这个菜单不显示了

```
单独一个节点的信息如下所示
{
        name: '模版消息',
        key: 'templateMessage',
        bpmnType: 'Task',//如果是完成这种 EndEvent
        // 还要再加上editable: false,
        // 就是字段不允许编辑 默认是true
        icon: 'icon-qd-bpmn-msg-tpl',
        connType: 3,
        actions: [
```

```
            'create',
            'connect',
            'remove',
        ],
    },
```

创建节点

除了start和end之外 其它的节点都是会被复用的 所以这里的逻辑就是 设置一个set 把节点的名字都加进去 令一个indexMap 的key值为这些节点的code 初始index=0 如果增加一个节点已经出现在这个set里面了 就把对应的index+1 并且更新节点 这个api

```
// 获取画布中所有节点
JSSDK.node.getAllNodes(); // 包含“开始”、“完成”节点

// 更新节点名称
JSSDK.node.update(<节点id 或 节点element对象>, { name: '新节点名称' }, () => {
  console.info('更新节点成功了');
});

// PS:  支持向节点对象注入自定义属性，方式如下
JSSDK.node.update(<节点id 或 节点element对象>, {
  foo: 1, // 支持自定义属性
  bar: 2,
});

在项目里面 每次添加或删除的时候他都用了一个 nextTick的方法重新获取节点列表
还有一点要注意的就是在data里面只是定义了
currentAllNodes:[]
所以它内部对象的变更无法检测到
用了 this.$set的方法
```

动态组件的用法

form表单

如果有一个v-form-item 中的v-button type="submit"那么就不用为这个button设置点击事件 直接对这个v-form @submit进行事件设置即可

vm.<img src="https://latex.codecogs.com/gif.latex?listeners

包含了父作用域中的%20(不含%20.native%20修饰器的)%20v-on%20事件监听器。它可以通过%20v-on=""/>listeners" 传入内部组件——在创建更高层次的组件时非常有用。

引入一个文件夹下面所有的.vue文件

父组件里面要做一个动态的子组件 可以把一个文件夹下面的子组件都引进来 这里要注意的一点就是 组件的名字这种 比如 `ComTest` 注册的时候 会是 `<com-test></com-test>`

```
// 引进来一组子组件 index.js
const req = require.context('./', false, /[^.]+\.vue/);
const components = req.keys().reduce((components, module) => {
    const mod = req(module).default;
    components[mod.name] = mod;
    return components;
}, {});

export default components;

// 在父组件里面调用这个子组件 index.js
// 这个nameCom 必须是符合 com-test 这种规范的名字
<component :is="nameCom"></component>
import side from './components/index.js'
components: {...side }
```