前端组件化说明文档

1.Build Setup

```bash
# 安装对应的依赖包
npm install

# 启动服务在8080端口 localhost:8080
npm run dev

# 构建文件，生成html文件
npm run build

# 打印生成日志
npm run build --report
```

2.常见设置

1. 找到config下的index.js，找到dev下的port选项，更改端口号，详情见前端脚手架搭建指南
2. 如果出现 npm XXXXX install 的错误，请下载对应依赖包

3.表单组件Form.vue

- 对应文件引入form.vue

```
<tabform :searchIpt="searChform" :searchFunc="ThisSearchFunc" :fromClass="firstFormClass">
</tabform>
```

- 传递三个对应参数 searchIpt，searchFunc，fromClass,参数结构大致如下,**form**:整个form表单的样式，**formItem**:每一个item的样式，**label**:每一个label标签的样式，**iptClass**: 每一个input框的样式，**buttonClass**: button的样式, **formSize**: input 框的大小，有三个参数可选small，medium，mini

```
fromClass: {
        form: 'firForm',
        formItem: 'firFormItem',
        label: 'firformLabel',
        iptClass: 'firIptClass',
        buttonClass: 'firbutton'
      },
```

- form表单支持的表单控件有**input输入框,password输入框，textarea，单选select，多选select2，日期选择datetime，周选择week，月份选择month，年份选择year，日期范围选择datetimerange，时间选择timepicker，单选框radio，复选框checkbox**
  注意：下文json中带有this的为函数，this.$t('xxx')的为国际化，true或false为布尔类型，其他均为string类型
- 统一表单控件支持功能 input控件 即type='text'，对应参数见json

```
{
          getLabel: this.$t('form.input'), // 标签名称
          models: '', // 输入内容
          placeHolder: '请输入文本', // placeholder提示
          type: 'text',
          disabled: false, // 是否禁用
          maxlength: 10, // 最大长度限制
          endIcon: 'el-icon-date', // 尾部图标
          StartIcon: 'el-icon-date', // 头部图标
          name: 'adcs', // 对应input框的name函数
          blur: this.iptbur // 失去焦点函数
          focus: this.focus, // 获得焦点函数
          change: this.change // 发生改变函数
        },
```

- 统一表单控件支持功能 password控件 即type='password'，对应参数见json

```
{
          getLabel: this.$t('form.input'), // 标签名称
          models: '', // 输入内容
          placeHolder: '请输入文本', // placeholder提示
          type: 'password',
          disabled: false, // 是否禁用
          maxlength: 10, // 最大长度限制
          endIcon: 'el-icon-date', // 尾部图标
          StartIcon: 'el-icon-date', // 头部图标
          name: 'adcs', // 对应input框的name函数
          blur: this.iptbur // 失去焦点函数
          focus: this.focus, // 获得焦点函数
          change: this.change // 发生改变函数
        },
```

- 统一表单控件支持功能 单选select控件 即type='select'，对应参数见json

```
{
          getLabel: this.$t('form.select'), // 标签名称
```

```
            models: '', // 输入内容
            placeHolder: '只能选择一个', // placeholder提示
            disabled: false, // 是否禁用
            clearable: true, // 是否有清空按钮（尾部清空）
            type: 'select',
            filterable: true, // 是否可以快速搜索
            name: 'select', // 源生name
            blur: this.iptbur, // 失去焦点函数
            clear: this.clear, // 清除按钮函数
            focus: this.focus, // 获得焦点函数
            change: this.change // 发生改变函数
            option: [
              {
                value: '1', // 获取值
                label: '选择1', // 显示label
                disabled: false // 是否可以禁用
              },
              {
                value: '2',
                label: '选择2',
                disabled: true
              }
            ]
          },
```

- 统一表单控件支持功能 多选select2控件 即type='select2'，对应参数见json

```
  {
            getLabel: this.$t('form.select'), // 标签名称
            models: '', // 输入内容
            placeHolder: '只能选择一个', // placeholder提示
            disabled: false, // 是否禁用
            type: 'select2',
            name: 'select2', // 源生name
            blur: this.iptbur, // 失去焦点函数
            focus: this.focus, // 获得焦点函数
            change: this.change // 发生改变函数
            removeTag: this.removeTag, // 多选移除时触发
            option: [
              {
                value: '1', // 获取值
                label: '选择1', // 显示label
                disabled: false // 是否可以禁用
              },
              {
                value: '2',
                label: '选择2',
                disabled: true
              }
            ]
          },
```

- 统一表单控件支持功能 日期选择datetime，周选择week，月份选择month，年份选择year 即type='datetime | week | month | year'，对应参数见json

```
  {
            getLabel: this.$t('form.datetime'),// 标签名称
            models: '',// 输入内容
            placeHolder: '请选择时间日期', // placeholder提示
            name: 'pickers',//源生名字
            readonly：ture, //是否只读
            disabled：ture, //是否禁用
            clearable: true, // 是否有清空按钮（尾部清空）
            blur: this.iptbur, // 失去焦点函数
            focus: this.focus, // 获得焦点函数
            change: this.change, //选择日期后触发
            format: 'MM.dd.yyyy HH:mm:ss', //显示的日期格式
            valueFormat: 'yyyy-MM-dd HH:mm:ss', // 存入日期格式
            type: 'datetime',
          },
```

- 统一表单控件支持功能 时间日期选择datetimerange 即type='datetimerange'，对应参数见json

```
  {
            getLabel: this.$t('form.datetimerange'),// 标签名称
            models: '',// 输入内容
            rangeSeparator: '到', //范围过滤字符
            readonly：ture, //是否只读
            disabled：ture, //是否禁用
            clearable: true, // 是否有清空按钮（尾部清空）
            blur: this.iptbur, // 失去焦点函数
            focus: this.focus, // 获得焦点函数
            change: this.change, //选择日期后触发
            name: 'llll',//源生名字
            format: 'MM.dd.yyyy HH:mm:ss',//显示的日期格式
            valueFormat: 'yyyy-MM-dd HH:mm:ss', // 存入日期格式
            type: 'datetimerange',
            startPlace: '开始时间',//开始时间文字提示
            endPlace: '结束时间'//结束时间文字提示
          },
```

- 统一表单控件支持功能 时间选择timepicker 即type='timepicker'，对应参数见json

```
  {
            getLabel: this.$t('form.timepicker'),// 标签名称
            models: '',// 输入内容
```

```
            placeHolder: '请选择时间',//文字提示
            type: 'timepicker',
            pickerOptions: {
              selectableRange: '18:30:00 - 20:30:00' //可选时间范围
            },
            format: 'HH小时mm分ss秒',//显示的日期格式
            valueFormat: 'HH小时mm分ss秒', // 存入日期格式
            readonly: ture, //是否只读
            disabled: ture, //是否禁用
            clearable: true, // 是否有清空按钮（尾部清空）
            blur: this.iptbur, // 失去焦点函数
            focus: this.focus, // 获得焦点函数
            change: this.change, //选择日期后触发
          },
```

- 统一表单控件支持功能 单选框radio 即type='radio'，对应参数见json

```
  {
            getLabel: this.$t('form.timepicker'),// 标签名称
            models: '',// 输入内容
            type: 'radio',
            disabled: true //raido组是否禁用
            change: this.change, //radio组发生改变时，回调函数
            option: [
              {
                value: '1',//radio的value
                name: 'rass', //单个raido名字
                label: '单选1',//radio的显示值
                disabled: true //是否禁用
              },
              {
                value: '2',
                change: this.change, //单个radio发生改变时，回调函数
                label: '单选2'
              },
              {
                value: '3',
                label: '单选3'
              }
            ]
          },
```

- 统一表单控件支持功能 多选框checkbox 即type='checkbox'，对应参数见json **注意：因为为多选，所以此models为数组类型，并非字符串类型，如果添加min参数，可能会出现勾选后，不可取消的状态因为初始勾选与min的值不同，请谨慎使用**

```
  {
            getLabel: this.$t('form.timepicker'),// 标签名称
            models: [],// 输入内容
            type: 'checkbox',
            disabled: true //raido组是否禁用
            change: this.change, //checkbox组发生改变时，回调函数
            min: '2', //checkbox组最少选择数量
            max: '2', //checkbox组最多选择数量
            option: [
              {
                value: '1',//checkbox的value
                name: 'rass', //单个checkbox名字
                label: '单选1',//checkbox的显示值
                disabled: true //是否禁用
                checked:  true //是否选择
                change: this.change, //checkbox发生改变时，回调函数
              }
            ]
          },
```

- 统一表单控件支持功能 文本域textarea 即type='textarea'，对应参数见json

```
  {
            getLabel: this.$t('form.textarea'),//label文字
            models: '',//输入内容
            placeHolder: '请输入文本',//提示内容
            type: 'textarea',
            name: 'adcs', // 对应textarea框的name函数
            blur: this.iptbur, // 失去焦点函数
            focus: this.focus, // 获得焦点函数
            change: this.change, // 发生改变函数
            resize: 'none'//是否可以拉伸，源生css，resize属性
            //both:'均可拉伸',
            //horizontal: '纵向拉伸',
            //vertical: '横向拉伸',
            //none: '不拉伸'
          }
```

4.表格组件table.vue

- 表格组件 引入对应的table.vue

```
      <getTable :tableData="listTableData" :tableGroup="listTableGroup"></getTable>
```

- 其对应两个参数 tableData 用于存放数据等，tableGroup 用于存放列属性
- tableGroup参数：tableGroup为Array类型
- **目前支持的功能有**
  - 国际化显示

- 文本显示
- 空白文本显示
- 时间显示
- 表格数据多选
- 图片显示
- 链接显示
- 可编辑文本显示
- 可编辑数字显示
- 可选择文本显示
- 可选择时间显示
- 自定义下标显示
- 自定义操作显示
- 划过显示

```
tableGroup: [
    {
      label: 'ID',// 用于标题头显示
      model: 'ID', // 用于该列内容存放
      type: 'text' //该列为文本显示
    },
    {
      label: '时间',
      model: 'times',
      type: 'time', //该列为日期显示
      formatter: this.getFormat
    },
    {
      label: 'name',
      model: 'nametype',
      type: 'locale', //该列为国际化显示
      formatter: this.getLocale
    },
    {
      label: 'operation',
      model: 'operation',
      type: 'operation' // 该列为操作显示
    }
  ]
```

- tableData参数为Object类型

```
tableData: {
        data: [   //data参数用于存放数据，其key值等同于tableGroup中的model
          {
            ID: 'zzg',
            nametype: 'title',
            times: '1512028314000',
            operation: [
              {
                func: this.getDetail, //func为函数类型
                //operation中的label为html类型
                label: '<i class="iconfont iconOptration">&#xe612;</i>',
                title: this.$t('table.operate.detail'),//title为操作所对应的title
                show: true //show为是否显示当前操作，通过后台status去书写改属性
              },
              {
                func: this.getEnable,
                label: '<i class="iconfont iconOptration">&#xe626;</i>',
                title: this.$t('table.operate.enable'),
                show: true
              }
            ]
          }
        ],
        selection: this.selection, //如果有多选选项对应的函数回掉
        refName: 'firRef', //vue对应的ref属性名字
        isBorder: true, //是否显示border表格
        isStripe: true, //是否显示斑马条纹
        isPage: true, //是否有分页
        isExcel: true, //是否有excel导出功能
        ExcelTitle: '表格123', //导出excel的标题
        PdfTitle: 'table导出', //导出pdf的标题
        isPdf: true, // 是否有pdf导出功能
        localgroup: 'table.', //国际化资源拼接前缀
        id: 'abc', //改该table所对应的id名字
        MaxHeight: '200', //改table所对应的最大高度
        page: {
          pageGroup: [10, 20, 50, 100], //分页数据
          CurrentPage: 1, //当前页码
          pagesize: 20, //当前分页条数
          Allpage: 15, //总数据
          handleSizeChange: this.handleSizeChange, //分页条数改变触发该函数
          handleCurrentChange: this.handleCurrentChange //当前页码改变触发改函数
        }
      },
```

- 当**type = img**时,既图片显示table 参数如下

```
tableGroup: [
    {
      label: '图片显示',// 用于标题头显示
      model: 'imgs', // 用于该列内容存放
      type: 'img', //该列为文本显示
      height: '80px', //图片显示高度
      width: '80px', //该列宽度
```

```
        //表格出现滚动条以后是否会定位到左边 或右边
        fixed: ture //定位方式 （可选参数，left，right，true，false）
      }
    ]
```

- 当**type = link**时,既链接显示table 参数如下

```
  tableGroup: [
      {
        label: '链接显示',// 用于标题头显示
        model: 'linkss', // 用于该列内容存放
        type: 'link', //该列为文本显示
        width: '80px', //该列宽度
        //表格出现滚动条以后是否会定位到左边 或右边
        fixed: ture //定位方式 （可选参数，left，right，true，false）
      }
    ]
  tableData: [
      data: [
          {
              linkss: {
                  link: 'https://www.baidu.com', //跳转链接
                  label: '百度一下', //显示名称
                  title: 'abcs' //显示title
                  style: {   // 整体样式
                      'color': 'blue',
                      'text-decoration': 'none'
                    }
                }
            }
      ]
  ]
```

- 当**type = text**时,既文本显示table 参数如下

```
  tableGroup: [
      {
        label: '文本显示',// 用于标题头显示
        model: 'texts', // 用于该列内容存放
        type: 'text', //该列为文本显示
        width: '80px', //该列宽度
        //表格出现滚动条以后是否会定位到左边 或右边
        fixed: ture //定位方式 （可选参数，left，right，true，false）
      }
    ]
```

- 当**type = index**时,既表格序号显示table 参数如下

```
  tableGroup: [
      {
        label: '序号显示',// 用于标题头显示
        model: 'index', // 用于该列内容存放
        type: 'index', //该列为文本显示
        width: '80px', //该列宽度
        //表格出现滚动条以后是否会定位到左边 或右边
        fixed: ture //定位方式 （可选参数，left，right，true，false）
      }
    ]
```

- 当**type = time**时,既时间显示table 参数如下

```
  tableGroup: [
      {
        label: '时间显示',// 用于标题头显示
        model: 'times', // 用于该列内容存放
        type: 'time', //该列为文本显示
        formatter: this.formatters //用于格式化时间的函数
        width: '80px', //该列宽度
        //表格出现滚动条以后是否会定位到左边 或右边
        fixed: ture //定位方式 （可选参数，left，right，true，false）
      }
    ]
    formatters (row, column, cellValue) {
        cellValue = this.$formatDate(cellValue, this.$t('language.format'))
        return cellValue
      },
      // this.$t('language.format')为国际化时间格式
```

- 当**type = emptyText**时,既空文本显示table 参数如下

```
  tableGroup: [
      {
        label: '空白文本替代显示',// 用于标题头显示
        model: 'emptyText', // 用于该列内容存放
        type: 'emptyText', //该列为文本显示
        formatter: this.formatters //用于格式化时间的函数
        width: '80px', //该列宽度
        //表格出现滚动条以后是否会定位到左边 或右边
        fixed: ture //定位方式 （可选参数，left，right，true，false）
      }
    ]
    formatters (row, column, cellValue) {
        if (cellValue === null || cellValue === undefined || cellValue === '') {
          return '-' //这里的空白文本被替换为 -
        } else {
```

```
          return cellValue
        }
      }
```

- 当**type = locale**时,既国际化显示table 参数如下

```
tableGroup: [
    {
      label: '国际化语言显示',// 用于标题头显示
      model: 'locale', // 用于该列内容存放
      type: 'locale', //该列为文本显示
      formatter: this.formatters //用于格式化时间的函数
      width: '80px', //该列宽度
      //表格出现滚动条以后是否会定位到左边 或右边
      fixed: ture //定位方式 （可选参数，left，right，true，false）
    }
  ]
  tableData: [
    localgroup:  'table.'  //用于存放国际化资源文件前缀
  ]
  formatters (row, column, cellValue) {
      var _this = this
      if (cellValue === null || cellValue === undefined || cellValue === '') {
        return ''
      } else {
        cellValue = this.$t(_this.listTableData.localgroup + cellValue)
        return cellValue
      }
    }
```

- 当**type = selection**时,既多选显示table 参数如下

```
tableGroup: [
    {
      type: 'selection', //该列为文本显示
      width: '80px', //该列宽度
      //表格出现滚动条以后是否会定位到左边 或右边
      fixed: ture //定位方式 （可选参数，left，right，true，false）
    }
  ]
```

- 当**type = operation**时,既操作显示table 参数如下

```
tableGroup: [
    {
      label: '操作列',// 用于标题头显示
      model: 'operation', // 用于该列内容存放
      type: 'operation', //该列为文本显示
      width: '80px', //该列宽度
      //表格出现滚动条以后是否会定位到左边 或右边
      fixed: ture //定位方式 （可选参数，left，right，true，false）
    }
  ]
tableData: [
    data: [
        {
          operation: [
              {
                func: this.getDetail, //对应的操作函数
                label: '<i class="iconfont iconOptration">&#xe612;</i>', // 对应的显示文本，可以为html
                title: this.$t('table.operate.detail'), //对应的title
                show: true //是否显示
              },
              {
                func: this.getEnable,
                label: '<i class="iconfont iconOptration">&#xe626;</i>',
                title: this.$t('table.operate.enable'),
                show: true
              }
            ]
        }
    ]
]
```

- 当**type = hover**时,既划过显示table 参数如下

```
tableGroup: [
    {
      label: '划过显示',// 用于标题头显示
      model: 'hover', // 用于该列内容存放
      type: 'hover', //该列为文本显示
      width: '80px', //该列宽度
      //表格出现滚动条以后是否会定位到左边 或右边
      fixed: ture //定位方式 （可选参数，left，right，true，false）
    }
  ]
tableData: [
    data: [
        {
           hover: {
              title: '<el-button size="medium">左边显示</el-button>', //可以为html代码显示
              hoverText: '<span style="color:red">abc</span>',//滑过显示的内容 可以为html代码显示
              position: 'left' //划过现实的方位 可以为left top right bottom
              class:  'avbc' // 划过显示层的class 样式
            },
        }
    ]
```

```
]
```

- 当**type = editInput**时,既可编辑文本table 参数如下

```
tableGroup: [
    {
      label: '可编辑文本',// 用于标题头显示
      model: 'editInput', // 用于该列内容存放
      type: 'editInput', //该列为文本显示
      width: '80px', //该列宽度
      //表格出现滚动条以后是否会定位到左边 或右边
      click: false, // 默认参数 必须填写
      fixed: ture //定位方式 （可选参数，left，right，true，false）
    }
  ]
```

- 当**type = editNumber**时,既可编辑数字table 参数如下

```
tableGroup: [
    {
      label: '可编辑数字',// 用于标题头显示
      model: 'editNumber', // 用于该列内容存放
      type: 'editNumber', //该列为文本显示
      width: '80px', //该列宽度
      //表格出现滚动条以后是否会定位到左边 或右边
      click: false, // 默认参数 必须填写
      fixed: ture //定位方式 （可选参数，left，right，true，false）
    }
  ]
```

- 当**type = editSelect**时,既可选择文本table 参数如下 （注:这里的option 是需要label和value相等的，暂时没有不等的情况）

```
tableGroup: [
    {
      label: '可选择文本',// 用于标题头显示
      model: 'editSelect', // 用于该列内容存放
      type: 'editSelect', //该列为文本显示
      width: '80px', //该列宽度
      //表格出现滚动条以后是否会定位到左边 或右边
      click: false, // 默认参数 必须填写
      fixed: ture //定位方式 （可选参数，left，right，true，false）
      option: [   //用于选择的select
            {
              value: 'a',
              label: 'a'
            },
            {
              value: 'b',
              label: 'b'
            },
            {
              value: 'c',
              label: 'c'
            },
            {
              value: 'd',
              label: 'd'
            }
          ]
    }
  ]
```

- 当**type = editTime**时,既可编辑数字table 参数如下 （注:这里为了配合国际化的显示，组件内部设计到一个显示国际化时间的方式，vue的filter formatDate）

```
tableGroup: [
    {
      label: '可编辑数字',// 用于标题头显示
      model: 'editTime', // 用于该列内容存放
      type: 'editTime', //该列为文本显示
      width: '80px', //该列宽度
      //表格出现滚动条以后是否会定位到左边 或右边
      click: false, // 默认参数 必须填写
      fixed: ture //定位方式 （可选参数，left，right，true，false）
    }
  ]
  // 对应的时间戳转换
  Vue.filter('formatDate', function (value, format) {
      if (value === null || value === '' || value === undefined) {
        return
      }
      var paddNum = function (num) {
        num += ''
        return num.replace(/^(\d)$/, '0$1')
      }
      var date = new Date(parseInt(value))
      var cfg = {
        yyyy: date.getFullYear(),
        yy: date.getFullYear().toString().substring(2),
        M: date.getMonth() + 1,
        MM: paddNum(date.getMonth() + 1),
        m: date.getMonth() + 1,
        d: date.getDate(),
        dd: paddNum(date.getDate()),
        HH: paddNum(date.getHours()),
        mm: paddNum(date.getMinutes()),
```

```
            ss: paddNum(date.getSeconds())
          }
          return format.replace(/([a-z])(\1)*/ig, function (m) {
            return cfg[m]
          })
        })
```

5.轮播图组件swiper.vue

- **当前可配置功能**
  - 基础轮播
  - 宽度高度设置带单位
  - 自动轮播
  - 分页器
  - 分页器自定义
  - 前进后退按钮
  - 前进后退按钮自定义
  - loop循环
  - 懒加载
  - 多栏轮播
  - 多栏轮播间隔设置
  - 垂直、水平方向轮播
  - 卡片式轮播
  - 联动轮播
  - 增删栏目
  - 带文件轮播（展示为icon，点击预览）
  - 加载更多功能
- 引入

```
import Swiper from '../common/swipe'
import SwiperGroup from '../common/swipe-group' //需要联动功能时引入
```

- 基础示例

```
<c-swiper :propOption='options.swipeBoxNormal' :listData='listData'></c-swiper>
```

- 联动示例

```
<c-swiper-group :propOption='groupOptions.swipeGroupHon.option'>
  <c-swiper v-for="(item, key) in groupOptions.swipeGroupHon.items" :key="item.propId" :propOption='item.option':listData='item.listData'></c-sw
</c-swiper-group>
```

- 传入数据示例

```
listData: [
        {
          type: 'img',
          src: '../../static/1.png'
        },
        {
          type: 'img',
          src: '../../static/2.png'
        },
        {
          type: 'img',
          src: '../../static/3.png'
        },
        {
          type: 'audio',
          src: '../../static/4.mp3'
        },
        {
          type: 'video',
          src: '../../static/5.avi'
        },
        {
          type: 'txt',
          src: '../../static/6.txt'
        },
        {
          type: 'pdf',
          src: '../../static/7.pdf'
        }
      ]
```

- 传入配置示例

```
      options: {
        swipeBoxNormal: {
          propId: 'swipeBoxNormal',
          height: '30rem',
          width: '90rem'
        }
      }
```

- 配置项

attr Object	attr name	attr	description	default	Value	required
propOption	height		轮播组件容器高度	100%	String	false
	width		轮播组件容器宽度	100%	String	false
	propId		轮播组件容器ID，后续作为唯一标识	''	String	true
	listData		数据内容	[ ]	Array	false
	slidesPerView		当前容器内显示的可见栏数量	1	Number	false
	spaceBetween		分栏显示时，栏目之间的间隔	0	Number	false
	direction		轮播方向	horizontal	horizontal,vertical	false
	initialSlide		加载后显示的栏目索引	0	Number	false
	speed		轮播速度	300	Number	false
	preloadImages		是否预加载	true	Boolean	false
	effect		轮播效果	slide	slide(普通切换、默认)，fade(淡入)，cube(方块)，coverflow(3d流)，flip(3d翻转)	false
	loop		是否可以循环轮播	false	Boolean	false
	observer		监听器，是否可以动态添加栏目	false	Boolean	false
	autoplay		是否开启自动轮播	false	Boolean	false
	delay		自动轮播延迟	3000	String, Number	false
	pagination	type	分页器功能-分页器样式	bullets	‘bullets’ 圆点（默认） ‘fraction’ 分式 ‘progressbar’ 进度条 ‘custom’ 自定义	false
		dynamicBullets	分页器功能-动态分页器，数量多时会隐藏	false	Boolean	false
		clickable	分页器功能-分页器是否可点击	true	Boolean	false
	bulletsColor		分页样式pagination.type为bullets时，设置小点背景色	#007aff	String	false
	navigate	position	前进后退按钮在容器中位置	inside	inside, outside	false
		type	前进后退按钮的样式类型	default	default, circle, large, bold, shadow	false
	lazy		是否懒加载	false	Boolean	false

- 加载更多功能

```
<c-swiper :propOption='options.swipeBoxMore' @last='loadmore' :listData='listDataMore'></c-swiper>
```

last事件 关联于加载更多功能，当轮播组件滑动至最后一张时触发，需要执行当前引用组件的methods中的方法(loadmore)，对传入prop.listData的字段(listDataMore)二次赋值，继而更新视图

- swipe-group

Attr Object	attr name	description	default	Value	required
propOption	height	轮播组件容器高度	100%	String	false
	width	轮播组件容器宽度	100%	String	false
	direction	联动组件的排列方式	horizontal	String	horizontal，vertical
	align	相对于group容器，在内部排列的方式，flex布局中的justify-content	around	around，between	false

6.面包屑导航组件breadCrumb.vue

- 引入对应的breadCrumb.vue

```
<bread :breadcrumb="firstBread"></bread>
```

- 传递一个参数breadcrumb,breadcrumb为 Object类型,传递参数如下， 注：下文history模式详见vue-router

```
breadcrumb: {
        font: '/', // 中间分隔字符，可以是文字，也可以是符号
        class: 'el-icon', //icon标签传递 但不能和font属性同时存在
        data: [
          {
            label: '首页', // 面包屑导航显示文字
            replace: true, // 是否启动history模式
```

```
            path: '/index/form' // 跳转路径
          },
          {
            label: '当前页',
            replace: true,
            path: '/index/form'
          },
          {
            label: '页码3',
            replace: true,
            path: '/index/form'
          }
        ]
      },
```

- 如果更改非当前页面的导航颜色 请使用

```
.el-breadcrumb__inner,.el-breadcrumb__inner a{
    color:#ff0000
  }
```

- 如果更改非当前页面的hover颜色 请使用

```
.el-breadcrumb__inner:hover,.el-breadcrumb__inner a:hover{
    color:#00ff00
  }
```

- 如果想更改当前页面的导航颜色 请使用

```
.el-breadcrumb__item:last-child .el-breadcrumb__inner,
.el-breadcrumb__item:last-child .el-breadcrumb__inner a,
.el-breadcrumb__item:last-child .el-breadcrumb__inner a:hover,
.el-breadcrumb__item:last-child .el-breadcrumb__inner:hover{
    color:#0000ff
  }
```

7.文件上传组件upload.vue

- 常规文件上传
- 预览上传成功文件的大小
- 限制上传文件大小
- 限制上传文件数量
- 限制上传文件格式
- 上传头像
- 上传图片显示成功列表
- 卡片式文件列表
- 拖拽上传
- 点击预览功能

```
//在当前vue文件中引入依赖文件
import Swiper from '../common/upload'

//html中的实例 propOption为配置项对象
<upload :propOption='options.normalUpload'></upload>
```

- 默认配置示例

```
    // 其中propClass和action为必填属性，本示例中的值均为不设置时的默认值
    propOption: {
        propId: '', //组件容器的id名，后续作为唯一标识 必填
        action: '', //上传的地址
        accept: '', //默认的上传文件格式，多类型时用逗号隔开 描述方式有两种
                    //1.audio/*,video/*,image/* 例image/jpg、audio/mp3
                    //2.后缀名 .png,.jpg
        fileList: [], //文件列表的接收数组
        limit: 3, //最大上传文件数量
        maxSize: 10, //最大上传文件的大小，单位为MB
        multiple: false, //是否支持多个文件一起上传
        buttonText: '', //选择文件按钮的文字内容
        submitText: '', //上传按钮的文字内容
        showFileList: true, //是否显示上传文件的列表
        listType: 'text', //文件列表的形式 text/picture/picture-card
        withCredentials: false, //支持发送 cookie 凭证信息
        autoUpload: false, //是否选择文件后自动上传
        headers: {}, //设置上传的请求头部
        data: {}, //上传时附带的额外参数
        name: 'file', //上传的文件字段名
        handleSuccess: function () {
          this.$message.success('OK')
        } //上传成功时的回调方法
      }
```

8.图像查看组件ImageLook.vue

- 引入此组件需要引入依赖一个包，命令如下
  npm install vuedraggable --save
- 将组件所必需的图片文件夹 imglook 拷贝到 \src\assets\ 目录下

- 该组件有两个参数 Imgdata 和 Imgtype
- 在对应 *.vue 文件引入ImageLook.vue

```
<imagelook :Imgdata="thisImgdata" :Imgtype="thisImgtype"></imagelook>
```

```
thisImgdata: [
  {
    imgSrc: '',       // 缩略图图片路径
    bigImgSrc: '',    // 大图图片路径
    isActive: true    // isActive 判断是否为选中状态, true 为选中, false 为未选中,默认第一张图片为选中状态
  },
  {
    imgSrc: '',
    bigImgSrc: '',
    isActive: false
  }
], // 图像资源配置
```

```
thisImgtype: [
  {
    operate: 'selectFile', // 打开文件
    showIpt: false
  },
  {
    operate: 'big', // 放大
    showIpt: true
  },
  {
    operate: 'small', // 缩小
    showIpt: true
  },
  {
    operate: 'showFull', // 1:1还原
    showIpt: true
  },
  {
    operate: 'widthAdaptive', // 宽度自适应
    showIpt: true
  },
  {
    operate: 'highAdaptive', // 高度自适应
    showIpt: true
  },
  {
    operate: 'leftRotate', // 左旋转
    showIpt: false
  },
  {
    operate: 'rightRotate', // 右旋转
    showIpt: false
  },
  {
    operate: 'sharpen', // 锐化
    showIpt: false
  },
  {
    operate: 'invertColor', // 反色
    showIpt: false
  },
  {
    operate: 'blur', // 模糊
    showIpt: false
  },
  {
    operate: 'gray', // 灰度
    showIpt: false
  },
  {
    operate: 'contrast', // 对比度
    showIpt: false
  },
  {
    operate: 'brightness', // 亮度
    showIpt: false
  },
  {
    operate: 'blackwhite', // 黑白
    showIpt: false
  },
  {
    operate: 'fudiao', // 浮雕
    showIpt: false
  },
  {
    operate: 'Histogram', // 直方图均衡
    showIpt: false
  },
  {
    operate: 'log', // 对数变换
    showIpt: false
  },
  {
    operate: 'linear', // 线性变换
    showIpt: false
  },
  {
    operate: 'PseudoColor', // 伪彩
    showIpt: false
  },
  {
```

```
      operate: 'organic',  // 有机物
      showIpt: false
    },
    {
      operate: 'inorganic',  // 无机物
      showIpt: false
    },
    {
      operate: 'metal',  // 重金属
      showIpt: false
    },
    {
      operate: 'mix',  // 混合物
      showIpt: false
    },
    {
      operate: 'EdgeEnhancement',  // 边缘增强
      showIpt: false
    },
    {
      operate: 'reduce',  // 还原
      showIpt: false
    },
    {
      operate: 'fullScreen',  // 全屏
      showIpt: false
    },
    {
      operate: 'ExitfullScreen',  // 退出全屏
      showIpt: false
    },
    {
      operate: 'save',  // 保存
      showIpt: false
    }
  ]  //  工具栏配置。operate 为工具按钮, showIpt 判断是否显示input比例框, true 为显示。默认 operate 设置为 big / small / showFull / widthAdaptive / hig
```

9.分页组件page.vue

- 该组件只有page一个参数

```
firstPage: {
        pageGroup: [10, 20, 50, 100],//分页组别
        CurrentPage: 1, //当前页码
        pagesize: 20,//当前分页条数
        Allpage: 15, //总条数
        prev: '前',// 前置图标
        next: '后', // 后置图标
        layout: 'total,sizes,prev,pager,next,jumper', //显示按钮
        background: true,// 是否显示背景色
        handleSizeChange: this.handleSizeChange, // 当分页条数改变时触发
        handleCurrentChange: this.handleCurrentChange // 当第...页改变时触发
      },
```

- layout 参数详解此参数用于展示组件布局 total:总条数 sizes:分页 prev:上一页 pager:页码 next:下一页 jumper:跳到/页
- 改变显示页码颜色(详情见 demo)

```
.el-pagination.is-background .el-pager li.active{
    background-color: #ff0000; // 选中背景
}
.el-pagination.is-background .el-pager li:hover{
    color: #ff0000 //划过颜色
}
.el-pagination.is-background  .btn-next,
.el-pagination.is-background  .btn-prev,
.el-pagination.is-background .el-pager li {
    background: yellow; //如果添加背景 更改默认背景色
}
```

10.拖拽组件dragg.vue

- 以为多个项目想要用户体验更好,所以增加了拖动组件demo，需要全局npm install awe-dnd 插件 (注：**新增 滚动条测试，兼容性无问题**)

```
data () {
      return {
        colors: [{
          text: '1',
          background: 'yellow'
        }, {
          text: '2',
          background: 'blue'
        }, {
          text: '3',
          background: 'cadetblue'
        }, {
          text: '4',
          background: 'lightblue'
        }, {
          text: '5',
          background: 'gray'
        }, {
          text: '6',
          background: 'wheat'
        }, {
          text: '7',
```

```
            background: 'gold'
          }, {
            text: '8',
            background: 'green'
          }, {
            text: '9',
            background: 'red'
          }]
        }
      }
```

```
<div class="color-list">
    <div
      class="color-item"
      v-for="color in colors" v-dragging="{ item: color, list: colors, group: 'color' }"
      :key="color.text" :style="{ background: color.background }"
    >{{color.text}}</div>
  </div>
```

11.自定义过滤器filter.vue

- 时间过滤器，使用方式 {{time | formatDate('yyyy-MM-dd HH:mm:ss')}}，其中yyyy为四位年份,yy为两位年，M为单一月，MM为前面补足单一月, d为日期，dd为前面补足日期， HH为小时，mm 为分钟，ss为秒。

```
Vue.filter('formatDate', function (value, format) {
  if (value === null || value === '' || value === undefined) {
    return
  }
  var paddNum = function (num) {
    num += ''
    return num.replace(/^(\d)$/, '0$1')
  }
  var date = new Date(parseInt(value))
  var cfg = {
    yyyy: date.getFullYear(),
    yy: date.getFullYear().toString().substring(2),
    M: date.getMonth() + 1,
    MM: paddNum(date.getMonth() + 1),
    d: date.getDate(),
    dd: paddNum(date.getDate()),
    HH: paddNum(date.getHours()),
    mm: paddNum(date.getMinutes()),
    ss: paddNum(date.getSeconds())
  }
  return format.replace(/([a-z])(\1)*/ig, function (m) {
    return cfg[m]
  })
})
```

- 文件大小过滤器 {{fileSize | FileSize}}

```
Vue.filter('FileSize', function (size) {
  var result
  switch (true) {
    case (size === null || size === '' || isNaN(size)):
      result = '-'
      break
    case (size >= 0 && size < 1024):
      result = size + ' B'
      break
    case (size >= 1024 && size < Math.pow(1024, 2)):
      result = Math.round(size / 1024 * 100) / 100 + ' K'
      break
    case (size >= Math.pow(1024, 2) && size < Math.pow(1024, 3)):
      result = Math.round(size / Math.pow(1024, 2) * 100) / 100 + ' M'
      break
    case (size >= Math.pow(1024, 3) && size < Math.pow(1024, 4)):
      result = Math.round(size / Math.pow(1024, 3) * 100) / 100 + ' G'
      break
    default:
      result = Math.round(size / Math.pow(1024, 4) * 100) / 100 + ' T'
  }
  return result
})
```

- 英文特殊字符过滤器 {{SpecialFont | SpecialFont}}

```
Vue.filter('SpecialFont', function (value) {
  return value.replace(/["\\/\b\f\n\r\t]/g, '').replace(/[-_,!|~`()#$%^&*{}:;"<>?《》""@+.]/g, '').replace('[', '').replace(']', '')
})
```

- MD5加密 {{GetMD5 | GetMD5（32）}} 这里的参数可以配置为 32位 或者16位 默认为16位,还需要额外引入MD5.js

```
Vue.filter('GetMD5', function (value, size) {
  return value.MD5(size)
})
```

- 字符串反转过滤器 {{reverse | reverse}}

```
Vue.filter('reverse', function (value) {
  return value.split('').reverse().join('')
})
```

- 数字转百分比 {{percents | numToPercent(1)}},这里的参数可以配置为 1- 10 为显示小数位数

```
Vue.filter('numToPercent', function (value, digits) {
  var result
  if (digits === null || digits === undefined) {
    digits = 2
  }
  digits = parseInt(digits)
  if (value === null || value === '' || value === undefined || isNaN(value)) {
    result = '-'
  } else {
    result = Math.round(value * Math.pow(10, digits) * 100) / Math.pow(10, digits) + '%'
  }
  return result
})
```

- 货币过滤器 {{formatAmount | formatAmount（'$'）}} 这里的参数为货币符号

```
Vue.filter('formatAmount', function (value, symbol) {
  var result
  if (value === null || value === '' || value === undefined || isNaN(value) || /\D+\.\D+/.test(value)) {
    return '-'
  }
  value = value.replace(/^0*[-+]*0*/, '')
  value = value.replace(/\s/g, '')
  value = value.replace(/^(\d*)$/, '$1.')
  value = (value + '00').replace(/(\d*\.\d\d)\d*/, '$1')
  value = value.replace('.', ',')
  var re = /(\d)(\d{3},)/
  while (re.test(value)) {
    value = value.replace(re, '$1,$2')
  }
  value = value.replace(/,(\d\d)$/, '.$1')
  result = symbol + value.replace(/^\./, '0.')
  return result
})
```

12.视频播放组件video.vue

- 引入此组件需要引入依赖两个包,命令如下npm install vue-video-player --save；npm install --save video.js;
- 该组件有两个参数 playerOptions 与 playerFunc

```
playerFunc: {
      play: this.onPlayerPlay, // 开始播放参数
      pause: this.onPlayerPause, // 暂停参数
      ended: this.onPlayerEnded,// 播放结束参数
      loadeddata: this.onPlayerLoadeddata, // 加载视频参数
      waiting: this.onPlayerWaiting, // 视频缓存
      playing: this.onPlayerPlaying, // 视频播放中
      timeupdate: this.onPlayerTimeupdate, // 播放进度条更新
      canplay: this.onPlayerCanplay,// 可以播放
      canplaythrough: this.onPlayerCanplaythrough, // 拖动进度触发
      statechanged: this.playerStateChanged, // 更新视频进度
      ready: this.playerReadied // 视频准备完成
},
```

```
playerOptions: {
      refName: 'abcs', // refname   相当于原先的唯一标识id
      muted: false, // 是否静音
      playbackRates: [0.7, 1.0, 1.5, 2.0], // 设置倍速
      language: 'zh-CN', // 插件语言
      sources: [{
        type: 'video/mp4', // 视频播放类型
        src: vids // 视频播放路径
      }],
      poster: imgs, //视频初始化封面
      preload: true, // 是否缓存在 页面初始化之时
      autoplay: true, // 是否自动播放
      controls: false // 是否显示控制条
},
```

- player.currentType() 获取当前文件格式
- player.currentTime(10) 快进10秒
- player.volume(0.75) 更改当前音量为75%
- player.currentSrc() 获取当前文件路径
- player.pause() 暂停
- player.play() 播放
- player.paused() 检查是否暂停
- player.src('path/to/video.mp4') 或者 player.src({type:'video/mp4',src:'path/to/video.mp4'}) 或者 player.src([{type:'video/mp4',src:'path/to/video.mp4'},{type:'video/mp4',src:'path/to/video.mp4'}]) 去设置不同的 src 路径

13.loading组件loading.vue

- 需要引入loading.vue

```
    <GETloading :loadingOption="SecOption"></GETloading>
```

- 对应有8种loading样式，分别对应的type为music，squre，bounce，cube，point，Roundcircle，bettery，ring

- 当type为**music**的时候，可以传递以下参数。

```
firOption: {
  color: 'blue', // 对应loading组件颜色
  height: '60px', //单一组件最大高度
  width: '10px',  //单一组件最大宽度
  type: 'music'
}
```

- 当type为**squre**的时候，可以传递以下参数。

```
SecOption: {
  color: 'blue', // 对应loading组件颜色
  height: '60px', //单一组件最大高度
  width: '60px',  //单一组件最大宽度
  type: 'squre'
}
```

- 当type为**bounce**的时候，可以传递以下参数。

```
ThrOption: {
  color: 'blue', // 对应loading组件颜色
  height: '60px', //单一组件最大高度
  width: '60px',  //单一组件最大宽度
  type: 'bounce'
}
```

- 当type为**cube**的时候，可以传递以下参数。

```
ForOption: {
  color: 'pink',
  BoxWidth: '100px', // 外层盒子宽度
  BoxHeight: '100px', // 外层盒子高度
  height: '30px', // 内部元素高度
  width: '30px', // 内部元素宽度
  type: 'cube'
},
```

- 当type为**point**的时候，可以传递以下参数。

```
FifOption: {
  color: 'pink',
  BoxWidth: '100px', // 外层盒子宽度
  BoxHeight: '100px', // 外层盒子高度
  height: '30px', // 内部元素高度
  width: '30px', // 内部元素宽度
  type: 'point'
},
```

- 当type为**Roundcircle**的时候，可以传递以下参数。

```
SixOption: {
  color: 'red',
  ShadowColor: '-35px 0px 0px rgba(255,0,0,1)', // 注意 这里的阴影大小 一定要大于width 和 height
  BoxWidth: '200px', // 外层盒子宽度
  BoxHeight: '200px',// 外层盒子高度
  height: '20px', // 子元素高度
  width: '20px', // 子元素宽度
  type: 'Roundcircle'
},
```

- 当type为**bettery**的时候，可以传递以下参数。

```
SevenOption: {
  BoxWidth: '200px', // 外层盒子宽度
  BoxHeight: '200px', // 外层盒子高度
  height: '20px', // 子元素高度
  width: '20px', // 子元素宽度
  type: 'bettery'
},
```

- 当type为**ring**的时候，可以传递以下参数。

```
EightOption: { // 注意 这里的四个属性值为相等的
  BoxWidth: '200px',
  BoxHeight: '200px',
  height: '200px',
  width: '200px',
  type: 'ring'
}
```

14.倒计时组件Countdown.vue

- 需要引入CountDodwn.vue

```
<CountDodwn :CountOptions="firOptions"></CountDodwn>
```

- 对应有3种方法，分别对应的type为button，text，submit
  - 当type为**button**的时候，可以传递以下参数。业务场景为获取验证码倒计时，发送延迟提示等

```
    firOptions: {
      time: 5, // 延迟时间
      func: this.firFunc, // 调用函数
      text: '秒后获取', // 改变文本
      type: 'button'
    },
```

  - 当type为**text**的时候，可以传递以下参数。业务场景为定时提交表单等

```
    SecOptions: {
      time: 10,// 延迟时间
      func: this.secFunc, // 调用函数
      text: '', // text为引入文本·
      type: 'text'
    },
```

  - 当type为**submit**的时候, 可以传递以下参数。 业务场景为防止重复请求的button等。

```
    ThirdOptions: {
      time: 3,
      func: this.thirdFunc,
      type: 'submit'
    }
```

15.富文本编辑器组件editor.vue

- 引入此组件需要引入依赖两个包，命令如下
  npm install --save vueditor;
  npm install --save vuex;
- 在node_modules文件夹下，修改 vueditor \ dist \ language 下三个 *.js 文件，导出 lang，如下

```
const lang = {
...
}
export default lang
```

- 新建 vuex 文件。在 vuex 文件夹下新建 store.js 文件，并引入 vue 和 vuex ，创建 store 实例

```
import Vue from 'vue'
import Vuex from 'vuex'
Vue.use(Vuex)
const state = {
  num: 1
}
export default new Vuex.Store({
  state
})
```

- 在项目的main.js文件中引入 vueditor 、vueditor所需样式表 以及 新建的vuex文件，并在实例化Vue对象时加入 store 对象

```
import Vueditor from 'vueditor'
import 'vueditor/dist/style/vueditor.min.css'
import store from './vuex/store'
...
Vue.use(Vueditor)
...
new Vue({
...
  store,
...
})
```

- 该组件有一个参数 editorOptions
- 在对应的 *.vue 文件中引入 editor.vue

```
<editors :editorOptions="allConfig"></editors>
```

```
editorOptions: {
  toolbar: [
    'removeFormat',         // 清除格式
    'undo',                 // 撤销
    'redo',                 // 恢复
    '|',                    // 分割线
    'element',              // 段落/标题
    'fontName',             // 字体
    'fontSize',             // 字号
    'foreColor',            // 文字颜色
    'backColor',            // 背景颜色
    '|',
    'bold',                 // 加粗
    'italic',               // 斜体
    'underline',            // 下划线
    'strikeThrough',        // 中划线
    '|',
```

```
      'subscript',            // 下标
      'superscript',          // 上标
      '|',
      'justifyLeft',          // 左对齐
      'justifyCenter',        // 居中对齐
      'justifyRight',         // 右对齐
      'justifyFull',          // 两端对齐
      '|',
      'indent',               // 增加缩进
      'outdent',              // 减少缩进
      'insertOrderedList',    // 插入有序列表
      'insertUnorderedList',  // 插入无序列表
      '|',
      'emoji',                // 插入表情
      'picture',              // 插入图片
      'table',                // 插入表格
      'link',                 // 添加超链接
      'unLink',               // 取消超链接
      '|',
      'fullscreen',           // 全屏
      'sourceCode',           // 源码
      'markdown'
    ],    // 工具栏配置
    fontName: [
      {val: 'Arial'},
      {val: 'arial black'},
      {val: 'Helvetica'},
      {val: 'Tahoma'},
      {val: 'Verdana'},
      {val: 'Times New Roman'},
      {val: 'Georgia'},
      {val: 'Impact'},
      {val: 'Courier New'},
      {val: 'sans-serif'},
      {val: 'SimSun'},                  // 宋体
      {val: 'NSimSun'},                 // 新宋体
      {val: 'FangSong'},                // 仿宋
      {val: 'SimHei'},                  // 黑体
      {val: 'Microsoft YaHei'},         // 微软雅黑
      {val: 'Microsoft JhengHei'},      // 微软正黑体
      {val: 'KaiTi'}                    // 楷体
    ],    // 常用css字体样式配置
    fontSize: [
      '12px', '14px', '16px', '18px', '20px', '24px', '28px', '32px', '36px'
    ],    // 文字大小配置
    isInsertEmoji: true,    // 是否支持插入表情
    isBase64: true,         // 图片路径格式是否为base64
    language: 'zh-CN'       // 工具栏语言配置（zh-CN为中文，en-US为英文，ru-RU为俄语）
  }
```

若想获取编辑器中的代码，则使用 ref 为子组件指定一个引用 ID，通过 this.$refs.refName 访问子组件，然后直接调用子组件中的方法 getContent()，如下

```
<editors ref='editors' :editorOptions="allConfig"></editors>
<button @click='getCode'>提取代码</button>
```

```
methods: {
  /* 获取编辑器中的代码 */
  getCode () {
    let editor = this.$refs.editors
    let code = editor.getContent()
    console.log(code)
  }
}
```

16.图标使用文档

- import '@/assets/iconfont/iconfont.css' 引入对应iconfont.css
- 挑选相应图标并获取类名，应用于页面：

```
<i class="iconfont icon-xxx"></i>
```

- 或者使用unicode编码，应用于页面,推荐此方式，兼容性强：（各图标unicode编码详情见demo内部图标库 下 **demo_unicode.html**）

```
<i class="iconfont">&#xe645;</i>
```

17.axios使用交互,fetch使用交互

- axios和传统jq $ajax 类似,fetch为新兴技术，具体代码如下，见备注

```
 /**
 * 封装axios请求方法
 */
const axios  = require('axios')


const Axios = axios.create({
  baseURL: "/", // 此处存放请求地址前缀
  timeout: 10000, // 请求超时时间
  responseType: "json", // 一般一个网站的responseType 都是一样的
  withCredentials: true, // 是否允许带cookie这些
```

```
  headers: {
    "Content-Type": "application/x-www-form-urlencoded;charset=utf-8" //如果一个项目中涉及到不同的header
    // 请通过下文opts进行传递，但是一般很少见
  }
});

// 添加请求拦截器
Axios.interceptors.request.use(function (config) {
    // 在发送请求之前做些什么
    // 能做的事如下 检查权限 增加页面loading  网络状态判断等
    return config;
  }, function (error) {
    // 对请求错误做些什么
    return Promise.reject(error);
  });

// 添加响应拦截器
Axios.interceptors.response.use(function (response) {
    // 对响应数据做点什么
    return response;
  }, function (error) {
    // 对响应错误做点什么
    // 例如用户请求失效，返回登录页什么的
    return Promise.reject(error);
  });

function error(response) {
  console.log(response)
  // 如果http状态码正常，则直接返回数据
  if (response && (response.status === 200 || response.status === 304 || response.status === 400)) {
    return response
      // 如果不需要除了data之外的数据，可以直接 return response.data
  }else{
    // 这里做一些提示 根据不同的status  如果需要处理的细致的话
  }

}
function success(res) {
  //统一判断后端返回的错误码
  // 例如101
}
const $axios = (opts, data) => {
    let Public = {}  //用于存放公共参数，类似于当前用户id等
    let httpDefaultOpts = { //http默认配置
          method:opts.method,
          url: opts.url,
          params:Object.assign(Public, data),
          data:Object.assign(Public, data),
    }

    if(opts.method=='get'){
      delete httpDefaultOpts.data
    }else{
      delete httpDefaultOpts.params
    }

    let promise = new Promise(function(resolve, reject) {
      axios(httpDefaultOpts).then(
        (res) => {
          success(res)
          resolve(res)
        }
      ).catch(
        (response) => {
          error(response)
          reject(response)
        }
      )

    })
  return promise
}

export default $axios
```

fetch

```
/**
 * Created by zzg on 2017/12/5.
 */
import {
  baseUrl
} from './env'

export default async(url = '', data = {}, type = 'GET', method = 'fetch') => {
  type = type.toUpperCase(); //把字符串转换为大写
  url = baseUrl + url; //baseUrl为地址前缀

  if (type == 'GET') {
    let dataStr = ''; //数据拼接字符串
    Object.keys(data).forEach(key => {
      dataStr += key + '=' + data[key] + '&';
    })

    if (dataStr !== '') {
      dataStr = dataStr.substr(0, dataStr.lastIndexOf('&'));
      url = url + '?' + dataStr; //采用地址传参的方式传递参数
    }
  }

  if (window.fetch && method == 'fetch') {
    let requestConfig = {
```

```
        credentials: 'include',// 因为fetch发送请求默认是不发送cookie的，配置此参数cookie既可以同域发送，
        // 也可以跨域发送，配置为same-origin: 表示cookie只能同域发送，不能跨域发送
        method: type, // 请求方式
        headers: { // 请求头
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
        mode: "cors", // 是否可以进行跨域请求 cors代表不可以  no-cors代表可以
        cache: "force-cache" // 如何处理缓存的方式 force-cache表示fetch请求不顾一切的依赖缓存,
        //  即使缓存过期了，它依然从缓存中读取. 除非没有任何缓存，那么它将发送一个正常的request.
        // no-cache: 如果存在缓存，那么fetch将发送一个条件查询request和一个正常的request，拿到响应后，它会更新http缓存.
        // no-store: 表示fetch请求将完全忽略http缓存的存在. 这意味着请求之前将不再检查下http的缓存，拿到响应后，它也不会更新http缓存.
      }

      if (type == 'POST') {
        Object.defineProperty(requestConfig, 'body', { //运用方法 给requestConfig 追加属性
          value: JSON.stringify(data)
        })
      }

      try {
        const response = await fetch(url, requestConfig);
        const responseJson = await response.json(); //应答实例
        return responseJson
      } catch (error) {
        throw new Error(error)
      }
    } else {
      return new Promise((resolve, reject) => { //如果用不了fetch   就用源生ajax代替
        let requestObj;
        if (window.XMLHttpRequest) {
          requestObj = new XMLHttpRequest();
        } else {
          requestObj = new ActiveXObject;
        }

        let sendData = '';
        if (type == 'POST') {
          sendData = JSON.stringify(data);
        }

        requestObj.open(type, url, true);
        requestObj.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
        requestObj.send(sendData);

        requestObj.onreadystatechange = () => {
          if (requestObj.readyState == 4) {
            if (requestObj.status == 200) {
              let obj = requestObj.response
              if (typeof obj !== 'object') {
                obj = JSON.parse(obj);
              }
              resolve(obj)
            } else {
              reject(requestObj)
            }
          }
        }
      })
    }
  }
```