

# Table of Contents

介绍	1.1
开始	1.2
环境准备	1.2.1
下载项目	1.2.2
项目配置	1.2.3
启动项目	1.2.4
基础	1.3
路由	1.3.1
过滤器	1.3.2
表单验证	1.3.3
接口数据格式	1.3.4
页面样式管理	1.3.5
展示列表数据	1.3.6
展示详情数据	1.3.7
添加新数据	1.3.8
更新数据	1.3.9
组件	1.4
图片上传	1.4.1
图片预览	1.4.2
关联数据列表选择	1.4.3
列表分页组件	1.4.4

- [1. 介绍](#)
  - [1.1. 用到的相关工具](#)

## 1. 介绍

diboot-mobile 是diboot开发平台为适应移动端开发，而研发的一套开发效率高，复用性好，交互性好的一个基础移动端基础项目。

### 1.1. 用到的相关工具

- vue-cli 辅助开发流程，提高开发效率。
- stylus 整合样式代码，减少代码量，提高复用性。
- es-lint 检查代码语法，并自动优化相关写法，使用的标准为airbnb标准（有些选项有所更改）。
- vue-router 控制页面路由，便于构建移动端单页应用，提高用户体验。
- axios 与后端的数据交互，在diboot-mobile中我们还集成了token授权，使得登录授权，以及OAuth授权等开发更加便捷，开箱即用。
- vee-validate 表单校验工具，diboot-mobile中已经配置了vee-validate，只需要按照文档中的示例来使用即可。
- moment 对于日期时间的处理。
- weixin-js-sdk 如果是开发与微信平台相关的移动应用，可以使用该sdk提供的接口调用js-sdk。

- 1. 开始

## 1. 开始

- [1. 环境准备](#)
  - [1.1. 环境要求](#)
  - [1.2. 安装Node](#)
  - [1.3. 安装vue-cli](#)

# 1. 环境准备

## 1.1. 环境要求

- node: 8.x
- vue-cli: 3.x
- npm: 6.2.x
- yarn: 1.x

## 1.2. 安装Node

- 到 [node官网下载](#) 相关版本进行安装。
- 或者使用 [nvm工具](#) 安装并管理node相关版本
- 安装node后，将源设置为阿里的淘宝源，下载包的速度会加快。

## 1.3. 安装vue-cli

- [vue-cli](#) 是vue开发中常用的开发工具，使用vue-cli我们可以很轻易的完成一些vue开发中的一些事情。
- 全局安装方法如下，在命令行中输入以下的一条命令，等待安装完成即可 ([MacOS下全局安装可能需要在下面命令前面加 sudo](#)) 。

```
npm install -g @vue/cli  
# OR  
yarn global add @vue/cli
```

- 1. 下载项目

## 1. 下载项目

- [1. 项目配置](#)
  - [1.1. vue-cli开发环境配置](#)
  - [1.2. 内置功能配置](#)

# 1. 项目配置

## 1.1. vue-cli开发环境配置

- vue开发环境相关配置在项目根目录下的 `vue.config.js` 配置文件中。
- 这里的相关配置项，可以根据自己的需求按照vue-cli官网的相关方法进行配置。
- `devServer` 为开发环境中对数据接口的代理设置。

```
module.exports = {
  baseUrl: '/',
  outputDir: 'dist',
  assetsDir: 'static',
  devServer: {
    disableHostCheck: true,
    port: 80,
    proxy: {
      '/rest': {
        target: 'http://localhost:8080/rest',
        changeOrigin: true,
        pathRewrite: {
          '^/rest': '/',
        },
      },
    },
  },
};
```

## 1.2. 内置功能配置

内置功能配置在 `src/config/index.js` 文件中

- `authType` 认证方式，即登录方式，目前支持微信授权和用户名密码登录，也可以自定义。
- `AUTH_TYPE` 为认证方式引用对象，如果有自定义的登录方式，可以增加到该对象中。
- `authHeaderKey` 为作为请求头中携带token的字段名，默认为 `authtoken` 。
- `AUTH_REFRESH_INTERVAL` 为token刷新间隔，为了防止页面获取数据提交等出现授权事变，会定时刷新 token，确保页面不过期。默认为30分钟，可自行设置，毫秒为单位。
- `requestTimeout` 请求数据时的超时时间设置，默认为10秒。
- `TOKEN_API` 使用用户名密码登录时登录接口地址。
- `WECHAT_TOKEN_API` 使用微信授权登录时通过code和state进行token认证的接口地址。
- `GET_WEIXIN_JSSDK_API` 获取JS\_SDK初始化信息的接口地址。
- `noneAuthList` 不需要登录就可以获取数据的路由列表（填写路由名即可）。

- [1. 启动项目](#)
  - [1.1. 命令行启动](#)
    - [1.1.1. npm](#)
    - [1.1.2. yarn](#)
  - [1.2. UI界面启动](#)

## 1. 启动项目

### 1.1. 命令行启动

打开命令行，并进入项目目录下，运行命令即可启动开发环境的项目。

#### 1.1.1. npm

```
npm install  
npm run serve
```

#### 1.1.2. yarn

```
yarn  
yarn serve
```

### 1.2. UI界面启动

- 打开命令行，在任意目录下，运行一下命令，即可打开vue的UI管理界面。
- UI界面中先导入当前项目，然后可以在UI界面上进行提供的相关功能的操作等。常用功能包括vue项目管理，启动项目，打包，eslint语法检查，插件管理，依赖管理等。

```
vue ui
```

- 1. 基础

## 1. 基础

- **1. 路由**
  - **1.1. 路由配置**
    - **1.1.1. 路由模式**
    - **1.1.2. 路由列表**
  - **1.2. 切换动画**

# 1. 路由

- 路由使用了 [vue-router](#) 进行管理和控制。
- 对于 [vue-router](#) 的更多属性和配置可参考 [vue-router官方文档](#)。

## 1.1. 路由配置

- 路由配置文件为项目根目录下的 [src/router.js](#) 文件。
- 默认配置内容如下：

```
import Vue from 'vue';
import Router from 'vue-router';

import PageTransition from './components/common/PageTransition';
import Test from './views/test';

Vue.use(Router);

const mode = 'history';

const customRoutes = [];

const routes = [
  {
    path: '/',
    name: 'pageTransition',
    component: PageTransition,
    children: [
      {
        path: '/test',
        name: 'test',
        component: Test,
      }
    ].concat(customRoutes),
  },
];

export default new Router({ mode, routes });
```

### 1.1.1. 路由模式

- `mode` 设置路由模式，可以是 [hash模式](#) 和 [history模式](#)，如果需要在微信中使用OAuth回调授权，建议使用[history](#)模式。

### 1.1.2. 路由列表

- `customRoutes` 为新增的路由列表，如果使用[devtools](#)生成相关功能，新增页面路由将push到该变量中。
- 如果开发过程中手动新增路由，为了方便阅读，可以在routes中的`children`字段中添加相关路由对象。

## 1.2. 切换动画

- `PageTransition` 为`children`下相关路由页面切换的过渡动画组件，如果需要更改切换动画，可更改该组件相关内容。

- [1. 过滤器](#)
  - [1.1. 已有过滤器：](#)

## 1. 过滤器

- 在我们的diboot-mobile中，我们对过滤器做了统一的处理，在加载创建vue实例前，会在main.js中自动加载[src/filter/index.js](#)文件中export出的所有过滤器函数。
- 如果您需要新增或者更改我们内置的过滤器函数，可以直接更改[src/filter/index.js](#)文件。

### 1.1. 已有过滤器：

- [html2Text](#) html转文本，会去掉html标签。
- [formatNull](#) 格式化空置，如果为空字符串或者undefined会返回'-'，如果为true返回'是'，为false返回'否'。
- [date2string](#) 日期转换为字符串，可以传入两个参数，第一个参数为日期参数，第二个参数为格式参数，如'YYYY年MM月DD日'，不传参数默认为 'YYYY-MM-DD'格式。
- [datetime2string](#) 日期时间转换为字符串，也可以参数两个参数，第一个参数为日期时间参数，第二个参数为格式参数，如'YYYY年MM月DD日 h时mm分'，不传参数默认为'YYYY-MM-DD h:mm'格式。

- **1. 表单验证**
  - [1.1. 概述](#)
  - [1.2. vee-validate配置](#)
    - [1.2.1. 修改默认提示](#)
    - [1.2.2. 添加自定义校验方法](#)
  - [1.3. 校验方法列表](#)
    - [1.3.1. vee-validate内置校验方法](#)
    - [1.3.2. 自定义校验方法](#)

# 1. 表单验证

## 1.1. 概述

- 对表单的验证，我们引入了[vee-validate表单验证模块](#)。
- 为了表单验证格式支持中文提示，我们引入了[vue-i18n模块](#)。

## 1.2. vee-validate配置

- vee-validate的配置及入口文件为[src/utils/validator.js](#)。
- [src/utils/validator.js](#)文件可以配置vee-validate的一系列设置，更改相关校验项的默认提示，新增自定义的校验项等。
- [validator.js](#)默认内容如下：

```
import Vue from 'vue';
import VeeValidate, { Validator } from 'vee-validate';
import zh_CN from 'vee-validate/dist/locale/zh_CN';
import VueI18n from 'vue-i18n';

export default {
  init() {
    Vue.use(VueI18n);
    const i18n = new VueI18n({
      locale: 'zh_CN',
    });

    Vue.use(VeeValidate, {
      i18n,
      i18nRootKey: 'validation',
      dictionary: {
        zh_CN,
      },
    });

    this.changeMessage();
    this.addRules();
  },
  changeMessage() {
    const dict = {
      // name为标签值（默认为name，可以重命名为data-vv-as属性）
      zh_CN: {
        messages: {
          required: name => `${name} 不能为空`,
          length: (name, params) => `${name} 长度必须为${params[0]}位`,
          max: (name, params) => `${name} 长度最大为${params[0]}位`,
          min: (name, params) => `${name} 长度最小为${params[0]}位`,
        },
      },
    };
  },
}
```

```

    };
    Validator.localize(dict);
},
addRules() {
  Validator.extend('mobile', {
    getMessage: field => `${field} 请输入正确的手机号码`,
    validate: value => value.length === 11 && /^(13|14|15|17|18)[0-9]{1}\d{8}$/.test(value),
  });
  Validator.extend('Number', {
    getMessage: field => `${field} 请输入数字`,
    validate: value => /^[^-?\d+(\.\d+)?$/.test(value),
  });
  Validator.extend('phone', {
    getMessage: field => `${field} 请输入正确的手机号码`,
    validate: value => /^1\d{10}$/gi.test(value) || /^0\d{2,3}-?\d{7,8}$/.test(value),
  });
  Validator.extend('phonestr', {
    getMessage: field => `${field} 请输入正确的手机号码`,
    validate: value => /(^+|\d)\d{10,12}/.test(value),
  });
  Validator.extend('password', {
    getMessage: field => `${field} 不符合规则要求, 请输入6-20个字母、数字、下划线或减号, 以字母开头!`,
    validate: value => /(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[a-zA-Z\d]{6,32}$/.test(value),
  });
  Validator.extend('wechat', {
    getMessage: field => `${field} 不符合规则要求, 请输入6-20个字母、数字、下划线或减号, 以字母开头!`,
    validate: value => /^[a-zA-Z]{1}[-_a-zA-Z0-9]{5,19}$/.test(value),
  });
},
}

```

## 1.2.1. 修改默认提示

在`changeMessage()`方法中，我们已经更改了`required,length,max,min`等校验方法的默认提示，可以参照该配置，继续在下面添加相关方法以及提示消息等。

## 1.2.2. 添加自定义校验方法

- 在`addRules()`方法中，我们已经新增了相关的校验方法，可以参照方法继续在下面添加相关的自定义校验。
  - `getMessage`: 返回校验不通过时的消息通知，`field`为该字段的名称。
  - `validate`: 校验方法，结果返回`true`或`false`。

# 1.3. 校验方法列表

## 1.3.1. vee-validate内置校验方法

- `vee-validate`所有内置校验方法请查阅[vee-validate官网文档](#)
- [用法参考](#)

## 1.3.2. 自定义校验方法

- `mobile` 对11位以及8位电话号码做校验，不校验八位电话号码开始是否为0。
- `Number` 对数字做校验。
- `phone` 对11位以及8位电话号码做校验，八位电话号码开始必须为0，否则校验不通过。
- `phonestr` 仅对手机号码做校验，如果手机号码前面有+86等也将校验通过。
- `password` 对我们diboot开发平台默认的密码校验，校验规则为6位及以上且须包含大小写字母及数字。
- `wechat` 对企业微信账号的校验，校验规则为6-20个字母、数字、下划线或减号，以字母开头。



- **1. 接口数据格式**
  - [1.1. 数据请求格式](#)
  - [1.2. 数据接收格式](#)
  - **1.3. 请求方式**
    - [1.3.1. GET](#)
    - [1.3.2. POST](#)
    - [1.3.3. PUT](#)
    - [1.3.4. DELETE](#)

## 1. 接口数据格式

- 我们的diboot-mobile基础项目使用axios作为请求数据的工具。
- 我们也对axios做了一定封装，我们对axios封装的代码放在 [src/utils/axios.js](#) 中。
- 我们封装过的 [axios.js](#) 中的相关方法可以在请求头中添加token，可以对请求过程中的异常统一处理，统一处理条数据，并且对于搜索过程中频繁响应的搜索需求中可以自动取消前面搜索请求等。
- 封装后的 [\[axios.js\]](#) 会在页面打开时自动挂在到vue实例上，我们可以如下示例一样，在每个vue组件中调用该模块。

```
let id = this.$route.params.id;
const res = await this.$http.get(`/student/${id}`);
if (res.code === 0){
    this.merge2dist(res.data, this.model); // 将获取到的数据合并到model属性中
}
```

### 1.1. 数据请求格式

- 只有文本数据时，直接以对象方式提交数据即可。
- 如果含有文件图片等，先使用单独的方法新建一个FormData实例，将文件上传。

### 1.2. 数据接收格式

- 接收json格式的数据，格式如下：

```
{
  code: 0,
  data: [],
  msg: '获取数据成功'
}
```

- code属性表示获取数据的状态，0代表获取成功，其他状态依据约定即可。
- data属性为接口返回的数据，可以是单个对象，也可以是一个对象数组。
- msg属性为接口返回的消息文本。

### 1.3. 请求方式

- 常用的请求方式有GET, POST, PUT, DELETE
- 返回的数据格式为上述[数据接收格式](#)

#### 1.3.1. GET

- GET请求方式一般用于获取数据，包括获取列表数据与获取详情数据。
- 使用 this.\$http.get(url, data)获取get数据，例如：

```
let res = await this.$http.get(`/${this.name}/${id}`);
```

### 1.3.2. POST

- POST请求一般用于提交数据，或者需要将数据更加安全的提交后获取数据的场景。
- POST请求在diboot-mobile基础项目中一般用于新建数据的提交。
- 使用 this.\$http.post(url, data)提交数据，data为要提交的数据对象，例如：

```
let res = await this.$http.post('/student/', this.model);
```

### 1.3.3. PUT

- PUT请求在diboot-mobile中一般用于更新数据的提交。
- 使用 this.\$http.put(url, data)提交更新数据，例如：

```
let res = await this.$http.put('/student/${this.model.pk}', this.model);
```

### 1.3.4. DELETE

- DELETE在diboot-mobile中一般用于删除数据的提交。
- 使用 this.\$http.delete(url, data)删除数据，例如：

```
const id = this.$route.params.id;
let res = await this.$http.delete('/student/${id}');
```

- **1. 页面样式管理**
  - [1.1. stylus在项目中的配置](#)
  - [1.2. stylus文件](#)
  - [1.3. stylus在组件中的使用](#)
    - [1.3.1. 添加style标签](#)
    - [1.3.2. 引入stylus样式文件](#)

# 1. 页面样式管理

- diboot-mobile项目默认采用stylus作为css预处理工具，关于stylus的更多信息可参考[官方主页](#)及其他文档。
- stylus相较于sass与less更新，并且支持更多更加强大的功能。
- stylus与vue项目相结合，可以极大地减少代码量，并且将更加提高css样式的复用性。

## 1.1. stylus在项目中的配置

- 在package.json文件中的 `devDependencies` 配置项中，增加对 `stylus` 和 `stylus-loader` 的开发依赖即可。

## 1.2. stylus文件

- 对于项目中所有vue组件共用的stylus样式文件，放置在 `src/common/stylus` 文件夹下。
  - `base` 文件夹下的文件为整个项目的通用样式文件，其中`mixin.styl`为 mixin文件，可以在其他组件中引入该文件后，如调用函数一样调用这里面的一些mixin样式。如果您也用到了自己的mixin功能，可以将相关mixin添加到该文件中。
  - `custom` 文件夹下的文件为定制文件，目前diboot-mobile基础项目默认以mint-ui为UI框架，比如对mint-ui中的相关样式在整个应用中有所修改，建议放置到该文件夹下的`mint-ui.styl`文件中，APP组件中已经默认引入`mint-ui.styl`文件。
  - `views` 文件夹下的文件为某一类页面所具有的共同样式的stylus文件。比如`detailview.styl`为详情页共用的样式，`form.styl`为表单页共用的样式，`listview.styl`为列表页共用的样式。在相应组件中引入该文件，即可在该组件中生效。

## 1.3. stylus在组件中的使用

- \* 在vue组件文件中添加好style标签并设置好属性后，在其中引入其他stylus文件，或者编写stylus样式代码即可。
- 保存后，即可自动编译并更新到页面上。

### 1.3.1. 添加style标签

```
<style scoped lang="stylus" rel="stylesheet/stylus">
</style>
```

### 1.3.2. 引入stylus样式文件

```
<style scoped lang="stylus" rel="stylesheet/stylus">
  @import "../../common/stylus/views/detailview.styl"
</style>
```



- **1. 展示列表数据**
  - **1.1. 新建列表组件**
  - **1.2. 添加列表页面路由**
  - **1.3. 引入mixins**
  - **1.4. 数据**
    - **1.4.1. 接口及参数**
    - **1.4.2. 接口数据格式**
  - **1.5. 样式**

## 1. 展示列表数据

- 对于列表数据的展示，在我们diboot-mobile基础项目里，已经在mixins中内置了获取列表数据，列表数据分页，列表数据搜索等功能，您只需要引入我们 [src/components/mixins/listview.js](#) 这个mixins文件，即可快速集成我们的解决方案。

### 1.1. 新建列表组件

- 建议在 [src/views](#) 文件夹下创建相关业务的文件夹，然后再改文件夹下创建用于路由引用的列表组件，比如下面示例中的list.vue组件。
- 对于列表的数据操作和展示等等建议放到 [listview.vue](#) 中，或者将其他需要展示该列表数据相同的部分放到 [listview.vue](#) 文件中，这样可以提高列表代码复用性
- 如果您遵循了上面的建议，那么只需要在list组件中引用listview组件即可。

```
views
└── student
    ├── list.vue
    └── listview.vue
```

### 1.2. 添加列表页面路由

- 在 [src/router.js](#) 文件中，[routes](#)的[children](#)字段中添加下面的对象：

```
import StudentList from './views/student/list'
const routes = [
  {
    path: '/',
    name: 'pageTransition',
    component: PageTransition,
    children: [
      {
        path: '/student/list',
        name: 'studentList',
        component: StudentList
      }
    ].concat(customRoutes),
  },
];
```

### 1.3. 引入mixins

- diboot-mobile基础项目中的mixins文件存放在 [src/components/mixins](#) 文件夹下。
- 在您创建的列表组件中引入 [src/components/mixins/listview.js](#) 到刚刚创建的列表组件中，添加到mixins属性中，列表组件script标签内的JavaScript代码示例如下：

```

import listview from '@/components/mixins/listview'
export default {
  data () {
    name: "studentListview",
    return {
      name: 'student',
      attachMore: true,
    }
  },
  mixins: [listview]
}

```

## 1.4. 数据

### 1.4.1. 接口及参数

- 如果使用 `my-infinite` 组件进行分页数据列表显示：
  - 组件加载完之后将会自动从该组件的 `infiniteUrl` 的属性所配置的接口中请求第一页数据。
  - `infiniteList` 属性会将获取到的列表数据添加到该数据中。
  - `pageSize` 属性可以设置一页加载的数据数量。
  - `query` 属性所传入的对象，将会以参数形式在请求列表数据时提交到接口中，查询数据时使用。
  - `fail2load` 为加载数据失败时执行的操作，在mixins中已经提供了默认方法，提示出错误信息。如果需要自定义，可以在该列表组件中添加相关方法，或者重写`fail2load`。

```

<my-infinite @fail2load="fail2load" :infiniteList='list' :infiniteUrl="infiniteUrl" ajaxType="GET" pageSize="20"
" :query="queryObj">

</my-infinite>

```

- 该组件的data方法中如果设置了`attachMore`为`true`，则会在组件加载完成后自动请求 `/${this.name}/attachMore` 接口，在上面就是 `/student/attachMore`，添加其他的关联信息，添加的关联信息，可以通过 `this.more.xxx` 获取到。

### 1.4.2. 接口数据格式

- 获取列表数据成功时的数据格式，`code`为0时认为获取数据成功：

```

{
  code: 0,
  data: [...],
  msg: '获取数据成功'
}

```

- 获取列表数据失败时的数据格式，`code`可以为约定好的其他格式：

```

{
  code: 4001,
  msg: '授权信息过期，请重新登录'
}

```

## 1.5. 样式

- 通过以下代码，可以引入列表页公用样式的`stylus`文件。
- 对于您的定制化系统，列表页的公用样式也可以写到 `src/common/stylus/views/listview.styl` 文件中。

- 对于该页面的私有样式，建议直接添加到 @import 这行代码下面，注意遵循stylus的语法规规范。

```
<style scoped lang="stylus" rel="stylesheet/stylus">
  @import "../../common/stylus/views/listview.styl"
</style>
```

- **1. 展示详情数据**
  - **1.1. 新建详情组件**
  - **1.2. 添加详情页路由**
  - **1.3. mixins**
    - **1.3.1. data属性**
    - **1.3.2. methods方法**
    - **1.3.3. 组件挂载**
  - **1.4. 详情数据获取**
    - **1.4.1. 详情数据接口**
    - **1.4.2. 数据格式**
  - **1.5. 样式**

## 1. 展示详情数据

对于详情数据的展示，我们提供了相应的通用流程，您可以参照该流程来快速构建您的详情数据展示页面。

### 1.1. 新建详情组件

- 新建detail.vue组件到 `src/views` 文件夹下相关业务文件夹内。
- 组件内，添加 `template`、`script`、`style`标签，并引入相关基础代码（以下示例是以student为例的，具体业务中可将 `student` 替换为自己的英文名称，遵循驼峰命名法）。

```
<template>
  <div class="studentDetail" v-show="showBox">
    <mt-cell title="姓名">{{ model.name | formatNull }}</mt-cell>
    <mt-cell title="性别">{{ model.gendar | formatNull }}</mt-cell>
  </div>
</template>
<script>
import detail from '@/components/mixins/detail';

export default {
  name: 'studentDetail',
  data() {
    return {
      name: 'student',
      attachMore: true,
    };
  },
  mixins: [detail],
  components: {
  },
};
</script>

<style scoped lang="stylus" rel="stylesheet/stylus">
  @import "../../common/stylus/views/detailview.styl"
</style>
```

### 1.2. 添加详情页路由

在 `src/router.js` 文件中，`routes`的`children`字段中添加详情页面的路由对象：

```
import StudentDetail from './views/student/detail'
const routes = [
```

```
{
  path: '/',
  name: 'pageTransition',
  component: PageTransition,
  children: [
    {
      path: '/student/detail',
      name: 'studentDetail',
      component: StudentDetail
    }
  ].concat(customRoutes),
},
];

```

## 1.3. mixins

在上面的基础代码中，引入了detail的mixins，mixins已经添加了相关的属性、方法和处理流程等。

### 1.3.1. data属性

- **model** 该详情数据对象，在该组件的js代码中可通过 `this.model` 来访问，在该组件的html代码中可通过 `model` 来绑定，`xxx`为对应的字段。在组件挂载之后，会自动根据路由上的参数，来自动的获取数据到该model中。
- **roles** 存放为该页面可访问的角色列表，可以存放一个角色数组，用当前用户角色来判定是否有访问权限等。相关流程需要您根据具体业务实现。
- **currentUserld** 存放当前用户id，如果需要，需要根据您的具体业务实现。
- **showBox** 为当前dom的显示状态，默认状态为false，不显示当前页的dom，当加载成功数据后，该状态置为true，显示详情页面的dom，即显示出数据。
- **more** 存放关联数据，比如存放了teacher的关联数据对象，可以通过以下示例取得。
  - 如果您使用了我们的devtools，并设置了关联数据等，将在生成过程中自动添加对关联数据的获取。

```
let teacher = this.more.teacher;
```

### 1.3.2. methods方法

- **edit()** 如果有编辑页面，并且按照本文档的路由命名，将跳转到该详情数据的编辑页面，否则可能会出现错误。
- **showImage(imgUrl)** 显示图片的大图，在该基础项目中，默认是用mint-ui的弹出层对图片大图进行显示。是用该功能需要在 `template` 中添加一下html：

- **closeImgPanel()** 关闭大图显示，当 `showImage(imgUrl)` 方法被调用后，显示大图弹层之后，点击弹层左上角的返回按钮，将会出发该方法，退出大图显示。
- **remove()** 删除该条数据，当调用该方法后，将弹出删除确认框，确认后，将请求删除数据的接口，进行数据参数操作。并在数据删除后跳转到该业务的列表页下。

### 1.3.3. 组件挂载

- 在组件挂载之后，将会读取路由中的id参数，并通过此参数获取`this.model`的数据。
- 挂载之后，如果`this.more`为true，则会获取其他关联数据。

## 1.4. 详情数据获取

- 接口地址为该组件中的 `/${this.name}/${id}`，比如在student的业务中为 `'/student/${id}'`。

- 请求方法为GET请求。
- 请求失败将提示后端接口返回的msg字段中的消息提示，若msg没有消息提示，将返回获取数据失败提示。

### 1.4.1. 详情数据接口

- 接口地址为该组件中的 `/${this.name}/attachMore`，比如在student的业务中为 `'/student/attachMore'`。
- 请求方法为GET请求。

### 1.4.2. 数据格式

- 返回数据格式详见[接口数据格式](#)。
- 其中data为对象,赋值到this.model, 例如:

```
{
  "code":0,
  "data":{
    "active":true,
    "createBy":10001,
    "createTime":"2018-07-02 21:14",
    "gendar":"M",
    "id":1000000010,
    "modelName":"学生",
    "name":"杨朝",
    "pk":1000000010,
    "pkString":"1000000010",
    "pkType":"DBAI",
    "teacherId":100001,
    "teacherName":"孔夫子",
    "updateTime":"2018-07-02 23:30"
  },
  "msg":"操作成功"
}
```

## 1.5. 样式

- 在.vue的该组件中添加一下代码，将引入详情页的默认样式，以及对mint-ui的一些样式的优化。

```
<style scoped lang="stylus" rel="stylesheet/stylus">
  @import "../../common/stylus/views/detailview.styl"
</style>
```

- 其他样式可以在 `@import` 语句下面写入stylus样式代码进行调整。

- **1. 添加新数据**
  - **1.1. 新建相关组件**
  - **1.2. 添加新建页路由**
  - **1.3. mixins**
    - **1.3.1. data属性**
    - **1.3.2. methods方法**
    - **1.3.3. 组件挂载**

## 1. 添加新数据

如果需要在移动端进行数据的添加，则需要开发新建相关功能，基于我们的mixins中的form.js，将更加方便快捷地开发新增数据功能。

### 1.1. 新建相关组件

- 新建create.vue页面组件到src/views文件夹下相关业务文件夹内，内容示例如下：

```
<template>
    <form-view></form-view>
</template>

<script>
import formView from './form';

export default {
    name: 'studentCreate',
    components: {
        'form-view': formView,
    },
};
</script>

<style scoped lang="stylus" rel="stylesheet/stylus">
</style>
```

- 新建form.vue表单组件到src/views文件夹下相关业务文件夹内，内容示例如下：

```
<template>
    <div class="studentForm form-container">
        <mt-field v-validate="'required|max:100'" :state="validateStateObj['姓名']" name="姓名" label="姓名" placeholder="请输入姓名" v-model="model.name"></mt-field>
    </div>
</template>

<script>
import form from '@/components/mixins/form';

const moment = require('moment');

export default {
    name: 'studentForm',
    data() {
        return {
            name: 'student',
            model: {
                pk: undefined,
                name: '',
            },
        }
    }
}
</script>
```

```

        attachMore: true,
    },
},
methods: {

},
computed: {
}
mixins: [form]
};
</script>

<style scoped lang="stylus" rel="stylesheet/stylus">
    @import "../../common/stylus/views/form.styl"
</style>

```

- 在组件的model属性中，我们需要添加该业务具有的相关字段等。
- 如果需要请求关联数据及数据列表但，将attachMore设置为true。

## 1.2. 添加新建页路由

在src/router.js文件中，在routes的children字段中添加新建页面的路由对象，如下：

```

import StudentCreate from './views/student/create'
const routes = [
{
    path: '/',
    name: 'pageTransition',
    component: PageTransition,
    children: [
        {
            path: '/student/create',
            name: 'studentCreate',
            component: StudentCreate
        }
    ].concat(customRoutes),
},
];

```

## 1.3. mixins

表单相关功能在form.js的mixins中已经内置了相关的默认处理方法，这些方法可以用于数据新建和更新。

### 1.3.1. data属性

- `showValidateState` 是否显示校验结果，当执行提交方法`submit()`之后改属性为true，当改属性为true，且有错误信息时，进行显示错误信息。
- `more` 为关联对象列表，供关联对象字段选择使用。
- `metadata` 为关联元数据相关的数据。`metadata.actionListObj`为元数据选择列表与字段相对应的对象。

### 1.3.2. methods方法

- `submit()` 提交表单，提交前会先进行表单校验。
- `validate` 校验表单。
- `getErrorMsg()` 获取表单校验失败的错误信息。

### 1.3.3. 组件挂载

- 获取路由中的id参数，如果该id参数有值，则根据该id参数请求相关数据，将表单作为更新数据的表单处理，如果没有相关id参数，则将该表单作为新建数据的表单处理。
- 请求 `/${this.name}/attachMore` 接口，获取关联数据列表等。

- **1. 更新数据**
  - [1.1. 新建更新页面组件](#)
  - [1.2. 新建更新页面路由](#)
  - [1.3. mixins](#)

## 1. 更新数据

如果需要在移动端更新已有数据，则需要数据更新功能，基于我们的mixins的form.js，将更加方便构建数据更新功能。

### 1.1. 新建更新页面组件

- 新建update.vue页面组件到src/views文件夹下相关文件夹内，内容示例如下：

```
<template>
  <form-view></form-view>
</template>

<script>
import formView from './form';

export default {
  name: 'studentUpdate',
  components: {
    'form-view': formView,
  },
};
</script>

<style scoped lang="stylus" rel="stylesheet/stylus">

</style>
```

### 1.2. 新建更新页面路由

在src/router.js文件中，在routes的children字段中添加新建页面的路由对象，如下：

```
import StudentUpdate from './views/student/update'
const routes = [
  {
    path: '/',
    name: 'pageTransition',
    component: PageTransition,
    children: [
      {
        path: '/student/update',
        name: 'studentUpdate',
        component: StudentUpdate
      }].concat(customRoutes),
  },
];
```

### 1.3. mixins

form.js详见添加新数据中的[form.js参考文档](#)。



- 1. 组件

## 1. 组件

- `diboot-mobile`内置了一些基础的功能组件，您也可以根据自己的业务来创建相关的功能组件。
- 内置的基础组件在[src/components/common](#)文件夹下，并根据相关功能场景以文件夹做了区分。
  - `image` 文件夹下为图片上传以及图片预览相关的功能组件。
  - `infinite` 为列表页滚动加载数据的组件，提供了加载数据上拉翻页等功能。
  - `select` 为关联数据列表选择组件，多用于表单中选择关联数据使用。

- **1. 图片上传**
  - **1.1. 文件方式上传图片**
    - **1.1.1. 引入图片文件上传组件**
    - **1.1.2. 设置组件相关参数**
  - **1.2. 微信js-sdk上传图片**
    - **1.2.1. js-sdk初始化**
    - **1.2.2. 引入微信上传图片组件**
    - **1.2.3. 设置组件相关参数**

## 1. 图片上传

- `src/components/common/image` 文件夹下提供了图片上传相关的组件，目前支持使用微信js-sdk上传图片，和以文件方式上传图片两种支持方式。
- `fileUploader.vue` 组件以文件方式上传图片。
- `wxUploader.vue` 组件以微信js-sdk上传图片。

### 1.1. 文件方式上传图片

#### 1.1.1. 引入图片文件上传组件

- 在`script`标签中引入`fileUploader.vue`组件，并在`components`属性中添加该组件，如下：

```
import form from '@/components/mixins/form';
import imgFileUploader from '@/components/common/image/fileUploader';
export default{
  name: 'studentForm',
  model: {
    pk: undefined,
    name: '',
    avatar: ''
  },
  components: {
    'img-file-uploader': imgFileUploader
  },
  mixins: [form]
}
```

- 在表单组件内，增加`img-file-uploader`标签，如下：

```
<div class="studentForm form-container">
  <mt-cell title="头像">
    <img-file-uploader v-model="model.avatar" maxCount="1" uploadUrl="/student/uploadImg"></img-file-uploader>
  </mt-cell>
</div>
```

#### 1.1.2. 设置组件相关参数

- `props`属性：
  - `v-model` 对于图片链接的双向绑定，如果是更新页面，`model.avatar`有值，该组件则会回显已有图片，上传成功之后，也将绑定到该属性绑定的参数中。如果有多张图片上传，则返回英文逗号分隔的图片链接字符串。
  - `maxCount` 设置最大上传图片张数，如果不设置该参数，则默认为3张，如果您使用了我们的`devtools`，自动生成的代码中这里的张数为1。
  - `uploadUrl` 图片上传接口，为服务器接收图片上传的上传接口，图片以`formData`的实例上传。

- 该组件只上传gif,jpeg,jpg,png这四种格式的图片，如果需要上传其他格式的图片文件，建议更改fileUploader.vue组件中的属性。

```
data() {
  return {
    imgType: ['gif', 'jpeg', 'jpg', 'png']
  };
},
```

## 1.2. 微信js-sdk上传图片

### 1.2.1. js-sdk初始化

[js-sdk微信公众平台官方文档](#)

- 在src/config/index.js文件中配置获取js-sdk初始化参数的地址以及api列表。如下：

```
const GET_WEIXIN_JSSDK_API = '/wechat/mp/getJsSdkConfig';
const WEIXIN_JSSDK_API_LIST = ['uploadImage', 'chooseImage'];
export default {
  GET_WEIXIN_JSSDK_API,
  WEIXIN_JSSDK_API_LIST
};
```

- 获取js-sdk参数化参数需要返回js-sdk初始化参数，具体格式如下：

```
{
  appId: '', // 必填，公众号的唯一标识
  timestamp: '', // 必填，生成签名的时间戳
  nonceStr: '', // 必填，生成签名的随机串
  signature: '' // 必填，签名，见附录1
}
```

- 在main.js文件中调用加载sdk的方法，如下：

```
// 启动初始化
startUtils.init();
startUtils.applyToken().then(() => {
  new Vue({
    router,
    store,
    render: h => h(App),
  }).$mount('#app');
  startUtils.loadSdk();
});
```

### 1.2.2. 引入微信上传图片组件

- 在script标签中引入fileUploader.vue组件，并在components属性中添加该组件，如下：

```
import wxUploader from '@/components/common/image/wxUploader';
export default{
  name: 'studentForm',
  model: {
    pk: undefined,
    name: '',
    avatar: ''
  },
}
```

```
components: {
  'wx-uploader': wxUploader
},
mixins: [form]
}
```

- 在表单组件内，增加wx-uploader标签，如下：

```
<div class="studentForm form-container">
  <mt-cell title="头像">
    <wx-uploader v-model="model.avatar" maxCount="1" sourceType="['album', 'camera']" sizeType="['original']"></wx-uploader>
  </mt-cell>
</div>
```

### 1.2.3. 设置组件相关参数

- props属性：
  - v-model 对于图片链接或mediald的双向绑定，如果是更新页面，model.avatar有值，该组件则会回显已有图片，上传成功之后，也将绑定到该属性绑定的参数中。如果有多张图片上传，则返回英文逗号分隔的图片链接或者新上传的图片mediald。
  - maxCount 设置最大上传图片张数，如果不设置该参数，则默认为3张，如果您使用了我们的devtools，自动生成的代码中这里的张数为1。
  - sourceType 设置图片来源列表，album为相册选取图片，camera为相机拍照。
  - sizeType 设置图片上传格式列表，original为上传原图，compressed为上传压缩图。

- 1. 图片预览
  - 1.1. 引入图片预览组件

# 1. 图片预览

`src/components/common/image` 文件夹下提供了图片预览组件`imageView.vue`。按照本文档的方法引用，可对图片进行弹层预览。

## 1.1. 引入图片预览组件

- 在`script`标签中引入`imageView.vue`组件，并在`components`属性中添加该组件，以及引入`detail.js`的`mixins`，如下：

```
import detail from '@/components/mixins/detail';
import imageView from '@/components/common/image/imageView';
export default {
  name: 'studentDetail',
  data() {
    return {
      name: 'student',
      attachMore: true,
    };
  },
  mixins: [detail],
  components: {
    'image-view': imageView,
  },
};
```

- 在`template`标签内，增加`image-view`标签，如下：

```
<template>
  <div class="studentDetail">
    <mt-cell title="姓名">{{ model.name | formatNull }}</mt-cell>
    <mt-cell v-if="model.avatar" title="头像">
      <div class="cell-image-box" :style="'width:' + rightWidth + 'px;'>
        
      </div>
    </mt-cell>
    <image-view v-if="imageUrl" :imageUrl="imageUrl" @closePanel="imageUrl=''"></image-view>
  </div>
</template>
```

- 在小图中点击事件会调用`detail`的`mixins`的`showImage(img)`方法，改方法会显示出`image-view`的图片预览效果，`showImage(img)`方法如下：

```
showImage(imgurl) {
  this.imageUrl = imgurl;
}
```

- 图片预览弹层显示与否根据`this.imageUrl`是否为空判断。



- 1. 关联数据列表选择
  - 1.1. 引入组件
    - 1.1.1. 设置组件相关参数:

# 1. 关联数据列表选择

- `src/components/common/select/relationSelect.vue`组件为关联数据选择所需要的组件。
- 如果你使用了devtools，自动生成的代码将会在生成的表单组件中需要选择关联数据列表的地方自动引入该组件。

## 1.1. 引入组件

- 在`script`标签中引入`relationSelect.vue`组件，并在`components`属性中添加该组件，并添加`form.js`的mixins，如下：

```
import relationSelect from '@/components/common/select/relationSelect';
export default{
  name: 'studentForm',
  model: {
    pk: undefined,
    name: '',
    teacherId: '',
    teacherName: ''
  },
  components: {
    'relation-select': relationSelect
  },
  mixins: [form]
}
```

- 在表单组件内，最底部增加`relation-select`标签，如下：

```
<div class="studentForm form-container">
  <relation-select v-if="relationObj.name" @selectValue="selectValue" :name="relationObj.name" :id="relationObj.id" :fieldName="relationObj.fieldName" :targetValue="relationObj.targetValue" :targetName="relationObj.targetName"></relation-select>
</div>
```

- 当调用`selectRelation('teacher', 'id', 'name', 'teacherId', 'teacherName')`方法时，将显示出该关联数据选择器，并加载关联数据列表，如下：

```
<div class="studentForm form-container">
  <div @click="selectRelation('teacher', 'id', 'name', 'teacherId', 'teacherName')">
    <mt-cell class="select-p" title="班主任">{{ model.teacherName }}</mt-cell>
  </div>
  <relation-select v-if="relationObj.name" @selectValue="selectValue" :name="relationObj.name" :id="relationObj.id" :fieldName="relationObj.fieldName" :targetValue="relationObj.targetValue" :targetName="relationObj.targetName"></relation-select>
</div>
```

### 1.1.1. 设置组件相关参数：

- `props`及相关属性：
  - `v-if` 绑定`relationObj.name`，该属性在`form.js`的mixins中已经添加好。
  - `name` 绑定业务名称，固定传入`relationObj.name`。
  - `id` 绑定需要复制给该表单的关联数据`id`字段的关联表单字段，默认为`id`，则固定传入`relationObj.id`。
  - `fieldName` 绑定字段名称，固定传入`relationObj.fieldName`。

- `targetValue` 绑定目标值，固定传入`relationObj.targetValue`。
- `targetName` 绑定目标名称，固定传入`relationObj.targetName`。
- `selectRelation()` 打开关联数据选择器，共有五个参数，都必须传入。
  - `name` 关联对象业务名称，此处为`teacher`。
  - `id` 关联对象设置到表单中的字段名称，此处为`id`，即`teacher.id`。
  - `fieldName` 关联对象中名称字段，此处为`name`，即`teacher.name`。
  - `targetValue` 设置到表单中的目标值字段，此处为`teacherId`，即选定的`teacher.id`设置到表单中的`this.model.teacherId`字段。
  - `targetName` 设置到表单中的目标名称字段，此处为`teacherName`，即选定的`teacher.name`设置到表单中的`this.model.teacherName`字段。
- 相关事件：
  - `selectValue()` 当选择完关联数据之后，将执行此方法，并将选择的关联数据`id`设置到开始调用`selectRelation()`方法中传入的`id`属性中，名称传入到最后一个属性中。

- **1. 列表分页组件**
  - [1.1. 引入列表分页组件](#)
  - [1.2. 组件相关参数](#)
    - [1.2.1. props属性:](#)
    - [1.2.2. 方法\]](#)
    - [1.2.3. 插槽](#)

# 1. 列表分页组件

- 列表分页组件，常用语数据列表中，具有加载列表数据，分页，搜索等功能。
- 组件文件为[src/components/common/infinite/infinite.vue](#)。

## 1.1. 引入列表分页组件

- 在`script`标签中引入`infinite.vue`组件，并在`components`属性中添加该组件。
- 引入`listview.js`的`mixins`，该`mixins`已内置了组件中对接该组件相关的属性和方法。
- 在`template`标签内使用该组件，如下：

```
<template>
    <my-infinite @fail2load="fail2load" :infiniteList='list' :infiniteUrl="infiniteUrl" ajaxType="GET" pageSize="20" :query="queryObj">
        <div slot="infinite">
            <div class="search-box">
                <mt-search v-model="query.FUZZY_SEARCH"></mt-search>
            </div>
            <div class="mint-cell-wrapper" v-for="item in list" :key="item.id">
                <router-link :to=`/${name}/detail/${item.id}` class="grid-container" tag="div">
                    <div class="grid-100 mobile-grid-50">
                        <div class="label-group">
                            <label>姓名: </label>
                            <span>{{ item.name }}</span>
                        </div>
                    </div>
                    <div class="grid-100 mobile-grid-50">
                        <div class="label-group">
                            <label>班主任: </label>
                            <span>{{ item.teacherName }}</span>
                        </div>
                    </div>
                </router-link>
            </div>
            <div class="mint-cell-wrapper">&nbsp;
            </div>
        </div>
        <!-- 提示区域(可自定义颜色位置 写在 tipposition) -->
        <p slot="loading" class="tipposition">加载中...</p>
        <p slot="ending" class="tipposition">没有数据了</p>
        <p slot="failed" class="tipposition">数据加载失败</p>
        <p slot="nodata" class="tipposition">暂无数据</p>
    </my-infinite>
</template>

<script>
    import listview from '@/components/mixins/listview'
    export default {
        data () {
            name: "studentListview",
            return {
                name: 'student',
                attachMore: true,

```

```

        },
        mixins: [listview]
    }
</script>

<style scoped lang="stylus" rel="stylesheet/stylus">
    @import "../../common/stylus/views/listview.styl"
</style>

```

## 1.2. 组件相关参数

### 1.2.1. props属性：

- `infiniteList` 绑定列表组件中列表数据。
- `infiniteUrl` 请求列表数据接口地址。
- `ajaxType` `ajaxType`请求方式，默认为GET请求。
- `pageSize` 每页数据条数，即每次请求数据条数。
- `query` 请求提交参数，用来搜索相关数据使用，传入一个js对象。

### 1.2.2. 方法】

- `fail2load()` 请求数据失败时执行该方法，如果需要自定义处理该方法，可在列表组件中添加该方法。

### 1.2.3. 插槽

- 滚动翻页插槽，如下：

```

<my-infinite @fail2load="fail2load" :infiniteList='list' :infiniteUrl="infiniteUrl" ajaxType="GET" pageSize="20
" :query="queryObj">
    <div slot="infinite">
    </div>
</my-infinite>

```

- 四个通知提示插槽，添加在该组件内的最后，当出现相关情况时，对应插槽的内容将显示到最底部，如下：

```

<my-infinite @fail2load="fail2load" :infiniteList='list' :infiniteUrl="infiniteUrl" ajaxType="GET" pageSize="20
" :query="queryObj">
    <div slot="infinite">
    </div>
    <p slot="loading" class="tipposition">加载中...</p>
    <p slot="ending" class="tipposition">没有数据了</p>
    <p slot="failed" class="tipposition">数据加载失败</p>
    <p slot="nodata" class="tipposition">暂无数据</p>
</my-infinite>

```