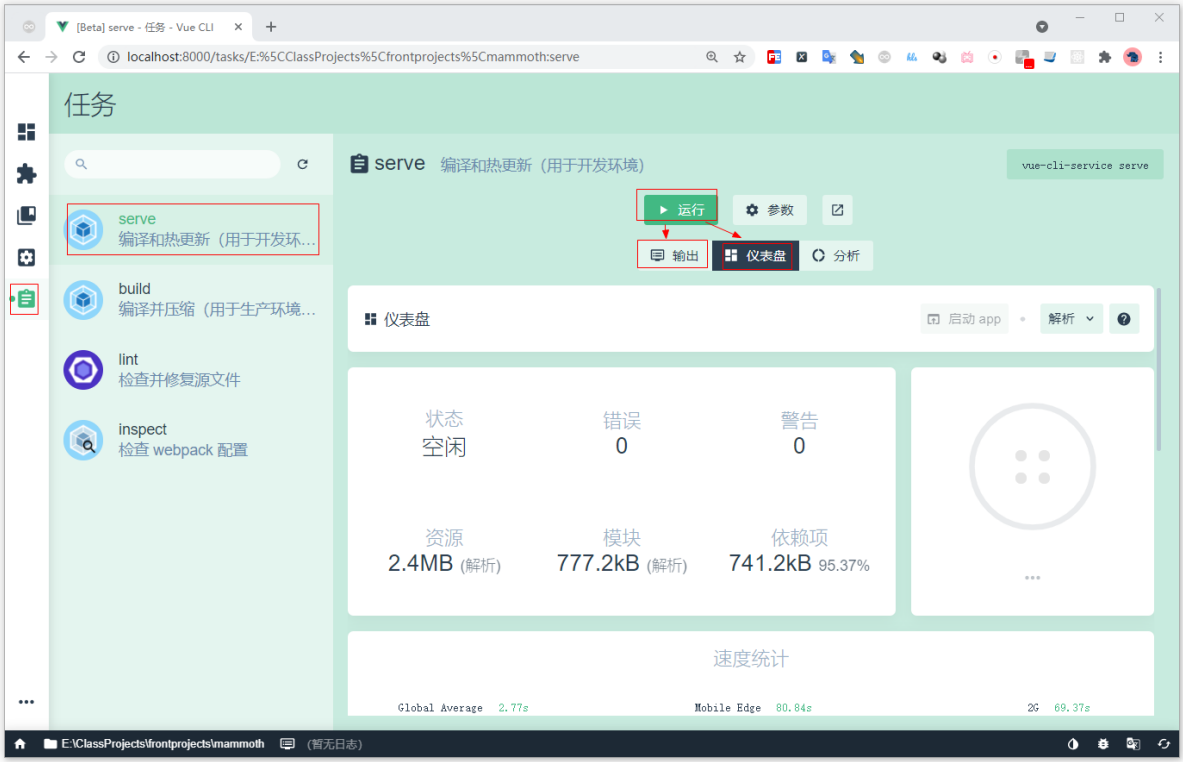
登录功能

分支

增加一个login分支检出，开发完成后合并到master，并提交到Gitee仓库。

```
在项目根目录下
wayne@wayne-pc MINGW64 /e/ClassProjects/frontprojects/mammoth (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

wayne@wayne-pc MINGW64 /e/ClassProjects/frontprojects/mammoth (master)
$ git checkout -b login
Switched to a new branch 'login'

注意下面分支的变化，已近变成login分支了
wayne@wayne-pc MINGW64 /e/ClassProjects/frontprojects/mammoth (login)
$ git status
On branch login
nothing to commit, working tree clean

wayne@wayne-pc MINGW64 /e/ClassProjects/frontprojects/mammoth (login)
$ git branch
* login
  master
```

开发用Server



点击`运行`，编译信息出现在`输出`中，如果有编译错误可以看到，`仪表盘`查看统计信息。

启动成功后，在`仪表盘`中点击`启动APP`。终于看到脚手架生成的第一个页面啦。http://localhost:8080/#/

看到到这个页面其实现实的是项目中src/main.js，而里面渲染的是App组件（App.vue）。

初识组件

`.vue`文件是单文件组件（single-file component）。

参考https://cn.vuejs.org/v2/guide/single-file-components.html

Vue文件分为3个部分

- template：结构，负责页面组件呈现
  - 每个组件必须**只有一个根元素**
- script：行为，数据和代码
  - data：数据，必须是个函数，这个函数必须返回一个对象，对象的属性就是组件可以用的数据
  - props：数组，组件的属性attr名称的数组
  - methods：对象，内部是函数名和函数的键值对
- style：样式，控制组件样式

data参考 https://cn.vuejs.org/v2/guide/components.html#data-%E5%BF%85%E9%A1%BB%E6%98%AF%E4%B8%80%E4%B8%AA%E5%87%BD%E6%95%B0

使用VSCode打开项目根目录，打开`src/App.vue`，清空内容，**输入vue**自动完成生产如下代码，保存后开始自动检查。删除最后style部分，**输入less**，选择`less-scoped.vue`，那么样式就只能用在当前组件，不会全局可用。

从src/main.js可知，App.vue是第一个组件文件，一切都从这里开始。

删除原有脚手架生成的src/views/*.vue和src/components/HelloWorld.vue，以后组件我们自己写。

```
<template>
  <div id="app">
    一起从这里开始
  </div>
</template>

<script>
export default {

}
</script>

<style lang="less" scoped>

</style>

```

Newline问题

保存vue文件后，ES检查，如果出现下面错误

```
error  Newline required at end of file but not found  eol-last
```

在.vue文件最后增加一个新空行，保存即可（上面代码第16行，`</style>`之后，是一个空行）。

这样就看到了第一个自己写的组件。

登录组件

`src/components/Login.vue`

```
<template>
  <div>登录组件</div>
</template>

<script>
export default {

}
</script>

<style lang="less" scoped>

</style>

```

路由

Vue Router 是 Vue.js 官方的路由管理器，参考 https://router.vuejs.org/zh/guide/#html

在组件中，在需要的地方增加`<router-view>`， 它是路由匹配到的组件显示的地方。

`src/router/index.js`如下

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Login from '../components/Login.vue'

Vue.use(VueRouter)

const routes = [
  { path: '/', redirect: '/login' },
  { path: '/login', component: Login }
]

const router = new VueRouter({
  routes
})

export default router
```

- path：路径
- redirect：重定向，https://router.vuejs.org/zh/guide/essentials/redirect-and-alias.html

`src/App.vue`如下

```
<template>
  <div id="app">
    一起从这里开始
    <!-- 路由匹配到的组件将渲染在这里 -->
    <router-view></router-view>
  </div>
</template>

<script>
export default {}
</script>

<style lang="less" scoped>
</style>
```

`<router-view>`组件是一个 functional 组件，渲染路径匹配到的视图组件。

- name属性，默认为'default'

less

CSS好处简单易学，但是坏处是没有模块化、复用的概念，因为它不是语言。

LESS是一门CSS的预处理语言，扩展了CSS，增加了变量、Mixin、函数等开发语言的特性，从而简化了CSS的编写。
LESS本身是一套语法规则及解析器，将写好的LESS解析成CSS。LESS可以使用在浏览器端和服务器端。

```
@color: #001529;

#header {
    color: @color;
    .menu {
        background-color: @color;
    }
}
```

可以使用解析成如下的CSS

```
#header {
  color: #001529;
}
#header .menu {
  background-color: #001529;
}
```

目前本项目缺少对less的支持，安装到**开发依赖**。这个插件需要**重启**server

可以使用Vue UI图形化安装，也可以用下面命令手动安装

```
$ yarn add -D less less-loader
```

安装后如果出现下面错误

```
 error  in ./src/components/Login.vue?vue&type=style&index=0&id=ef68022e&lang=less&scoped=true&

Syntax Error: TypeError: this.getOptions is not a function
```

原因是less-loader版本太高，和webpack兼容问题。目前默认为10.x.x版本，删除，安装低版本

```
$ yarn add -D less-loader@7.3.0
```

```
编译输出到控制台
$ npx lessc test.less

编译输出到文件
$ npx lessc test.less test.css
```

全局样式

`src/assets/css/main.css`

```
/* 全局样式表 */
html, body, #app {
    height:100%;
    margin: 0;
    padding: 0;
    min-height: 200px;
}
```

在`src/main.js`中导入它，作为全局样式

```
import './assets/css/main.css'
```

登录框居中样式

下面是一个测试盒子居中的页面，仅供测试参考。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        .box {
            width: 200px;
            height: 200px;
            background-color: #f0f0f0;
            /* 绝对定位，左边偏移宽度50%，顶偏移高度50% */
            position: absolute;
            left: 50%;
            top: 50%;
            transform: translate(-50%, -50%); /* 相对自己宽度x、y轴减去自己宽度高度的一半 */
        }
    </style>
</head>
<body>
    <div class="box"></div>
</body>
</html>
```

登录框居中效果，就可以参考这个样式

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="avatar_box">
        <img src="../assets/logo.png" alt="logo" />
      </div>
    </div>
  </div>
</template>

<script>
export default {}
</script>

<style lang="less" scoped>
.login_container {
  background-color: #2b4b6b;
  height: 100%;
}

.login_box {
  width: 450px;
  height: 300px;
  background-color: #fff;
```

```
  border-radius: 5px;
  position: absolute;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%);

  .avatar_box {
    height: 130px;
    width: 130px;
    padding: 10px;
    position: absolute;
    left: 50%;
    transform: translate(-50%, -50%);
    img {
      width: 100%;
      height: 100%;
      border-radius: 35%;
    }
  }

  .login_form {
    position: absolute;
    bottom: 0;
    width: 100%;
    padding: 0 20px;
    box-sizing: border-box;
  }
}
</style>

```

logo和Login.vue在提供的资源中有。

格式化问题

`Alt + Shift + F`是格式化代码快捷键，但是会带来一个问题，会在js代码后加上分号，这使得代码验证出错。

```
<script>
export default {}; // 就是这个分号导致代码校验失败，
</script>
```

```
error  Extra semicolon  semi 意思是额外的分哈
```

目前格式化后，手动删除这个分号即可

按需引用

直接使用Element-UI表单组件，加入表单组件后，发现没有效果，甚至还要报错，原因是没有引入这些组件

`src/plugins/element.js`中引入组件的同时，还要引入样式，否则非常难看。

按需引入参考：https://element.eleme.cn/#/zh-CN/component/quickstart#an-xu-yin-ru

```
import Vue from 'vue'
import { Form, FormItem, Input, Button } from 'element-ui'
import 'element-ui/lib/theme-chalk/index.css' // UI组件样式

Vue.use(Form)
Vue.use(FormItem)
Vue.use(Input)
Vue.use(Button)
```

表单组件*

使用Element-UI组件，饿了么推出的基于Vue的UI框架。

带验证表单参考：https://element.eleme.cn/#/zh-CN/component/form#biao-dan-yan-zheng

表单由表单组件el-form和表单项组件el-form-item组成。每一个el-form-item中可以有文本框、密码框、按钮等组件。

Input

- type和HTML类型一样，如文本框、密码框
- placeholder提示
- 需要和v-model双向绑定
- 可以使用prefix-icon 和 suffix-icon增加图标，图标可以参考Icon组件

Button

- type表示样式，primary就是主按钮颜色

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="avatar_box">
        <img src="../assets/logo.png" alt="logo" />
      </div>
      <el-form class="login_form">
        <el-form-item>
          <el-input
            placeholder="请输入用户名"
            prefix-icon="el-icon-user"
          ></el-input>
        </el-form-item>
        <el-form-item>
          <el-input
            type="password"
            placeholder="请输入密码"
            prefix-icon="el-icon-s-custom"
          ></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary">登录</el-button>
          <el-button type="info">取消</el-button>
```

```
        </el-form-item>
      </el-form>
    </div>
  </div>
</template>

<script>
export default {}
</script>

<style lang="less" scoped>
.login_container {
  background-color: #2b4b6b;
  height: 100%;
}

.login_box {
  width: 450px;
  height: 300px;
  background-color: #fff;
  border-radius: 5px;
  position: absolute;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%);

  .avatar_box {
    height: 130px;
    width: 130px;
    padding: 10px;
    position: absolute;
    left: 50%;
    transform: translate(-50%, -50%);
    img {
      width: 100%;
      height: 100%;
      border-radius: 35%;
    }
  }

  .login_form {
    position: absolute;
    bottom: 0;
    width: 100%;
    padding: 0 20px;
    box-sizing: border-box;
  }
}
</style>

```

数据绑定*

v-bind指令：缩写为`:`冒号，为属性attr绑定数据，可以动态的更新这个数据。v-bind指令参考：https://cn.vuejs.org/v2/api/#v-bind

`v-bind:model`，即`:model="form"`，表单中填写的数据同步到form标识符定义的对象上。注意没有冒号，这个`"form"`就是字符串了。

v-model指令：表单组件input、textarea、select是让用户录入的地方，这些数据必须有个对应的值，例如文本框、密码框等，录入变量值变化，这个值变它们显示的文字也变。这就需要v-model实现双向绑定。v-model指令参考：https://cn.vuejs.org/v2/api/#v-model

`v-model="form.username"`绑定到form对象的user属性上。

需要：

1. 为el-form组件增加`:model="loginForm"`
2. 为input等增加`v-model="loginForm.username"`，做双向数据绑定
3. 在行为区中增加data:function() { return {form:{username:'xxx'}} }

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="avatar_box">
        <img src="../assets/logo.png" alt="logo" />
      </div>
      <el-form class="login_form" :model="loginForm">
        <el-form-item>
          <el-input
            v-model="loginForm.username"
            placeholder="请输入用户名"
            prefix-icon="el-icon-user"
```

```vue
          ></el-input>
        </el-form-item>
        <el-form-item>
          <el-input
            type="password"
            v-model="loginForm.password"
            placeholder="请输入密码"
            prefix-icon="el-icon-s-custom"
          ></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary">登录</el-button>
          <el-button type="info">取消</el-button>
        </el-form-item>
      </el-form>
    </div>
  </div>
</template>

<script>
export default {
  data () {
    return {
      loginForm: {
        username: '',
        password: ''
      }
    }
  }
}
</script>

<style lang="less" scoped>
没变化，省略，不重复贴代码
</style>
```

格式化后出错

格式化Alt + Shift + F使用Vetur格式化，会格式化顺便加分号、逗号、该单引号，导致ESLint报错，编译失败。

```
\ClassProjects\frontprojects\mammoth\src\components\Login.vue
34:7   error  Missing space before function parentheses  space-before-function-paren
37:19  error  Strings must use singlequote               quotes
38:19  error  Strings must use singlequote               quotes
38:21  error  Unexpected trailing comma                  comma-dangle
39:8   error  Unexpected trailing comma                  comma-dangle
40:6   error  Extra semicolon                            semi
41:4   error  Unexpected trailing comma                  comma-dangle
42:2   error  Extra semicolon                            semi
```

34行，圆括号前应有空格

37、38行，字符串应该使用单引号

38、39、41行，多余的逗号

40、42行，多余的分号

```vue
<script>
export default {
  data() {
    return {
      form: {
        username: "",
        password: "",
      },
    };
  },
};
</script>
```

Vetur内部使用prettier。

参考：

- https://vuejs.github.io/vetur/guide/formatting.html#settings

- https://prettier.io/docs/en/configuration.html

在项目根目录下，创建prettier的配置文件.prettierrc，使用json格式

```
{
  "trailingComma": "none",
  "semi": false,
  "singleQuote": true
}
```

- semi
  - **false**取消自动加分号
- singleQuote
  - **true** 开启使用单引号，就不会转为双引号
- trailingComma
  - **"none"**关闭结尾自动加逗号

如果觉得格式化代码后，折行很严重，可以使用`"printWidth": 200`，默认是80。

ESLint规则

ESLint规则，参考 https://eslint.org/docs/rules/。规则改了需要重启服务，ESLint要重新读配置。

ESLint的配置文件，可以是js、yaml、json格式，在项目根目录下新建文件`.eslintrc`或`.eslintrc.js`。

根目录下已经有`.eslintrc.js`，直接把选项键值对键入到rules中。

配置规则如下

```
{
    "rules": {
        'eol-last': 'off',
        'space-before-function-paren': 0
    }
}
```

规则值有3种：

1. `"off"`或0，表示不验证该规则
2. "warn"或1，验证但不满足时，发出警告
3. "error"或2，验证但不满足时，报错

可以写成`"eol-last": "off"`或者`eol-last: ["error", "always"]`

小括号问题

函数名后面小括号问题，prettier格式化后，这个空格被移除，导致经常看到它，所以也禁用吧。

参考 https://eslint.org/docs/rules/space-before-function-paren

配合在上方

新行问题

Newline问题，参考 https://eslint.org/docs/rules/eol-last

- "always"，缺省，要求非空文件结尾必须有LF换行符
- "never"，要求文件结尾一定不要有LF

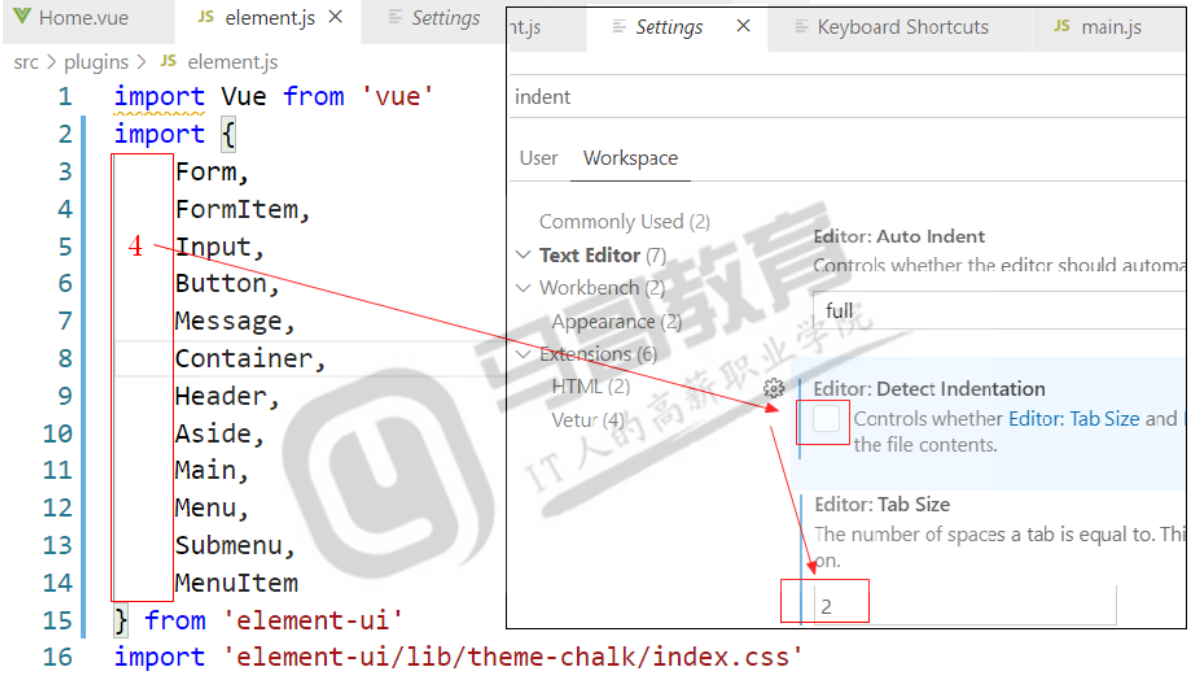
所以，我们选择关闭检验`"eol-last": "off"`，文件有没有换行符结尾无所谓，不需要验证。

配合在上方

js格式化问题

element.js等js文件格式化的时候，由于不是vue文件，格式化时tab使用4个空格，ESlint不能编译通过。解决办法如下

Detect Indentation不勾选，Tab Size改为2



表单验证规则***

参考：https://element.eleme.cn/#/zh-CN/component/form#biao-dan-yan-zheng

- 表单验证需要在el-form上，通过`rules`属性传入验证规则，需要v-bind到一个规则对象上
  - `<el-form class="login_form" :model="loginForm" :rules="loginRules">`
- 在el-form-item上，增加`prop`属性，提供验证字段的名字
  - `<el-form-item prop="username">`
- 在行为区，data中增加rules对象定义规则

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="avatar_box">
        <img src="../assets/logo.png" alt="logo" />
      </div>
      <el-form class="login_form" :model="loginForm" :rules="loginRules">
```

```
        <el-form-item prop="username">
          <el-input v-model="loginForm.username" placeholder="请输入用户名"
prefix-icon="el-icon-user"></el-input>
        </el-form-item>
        <el-form-item prop="password">
          <el-input type="password" v-model="loginForm.password"
placeholder="请输入密码" prefix-icon="el-icon-s-custom"></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary">登录</el-button>
          <el-button type="info">重置</el-button>
        </el-form-item>
      </el-form>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      loginForm: {
        username: '',
        password: ''
      },
      loginRules: {
        username: [
          { required: true, message: '请输入用户名', trigger: 'blur' },
          { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
        ],
        password: [
          { required: true, message: '请输入密码', trigger: 'blur' },
          { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
        ]
      }
    }
  }
}
</script>

样式省略
```

表单重置

重置

- 为表单提供ref属性`<el-form class="login_form" :model="loginForm" :rules="loginRules" ref="loginFormRef">`，将表单子组件的引用注册到当前组件Login上
- 按钮提供点击事件，`<el-button type="info" @click="resetLoginForm">重置</el-button>`。
    - v-on指令，简写为@
- 行为区，methods中定义事件回调函数

```
export default {
  data() {},
  methods: {
    resetLoginForm() {
      console.log('reset ~~~~', this) // 如何在这里操作Form呢?
    }
  }
}
```

如何在resetLoginForm中使用Form组件对应的对象loginForm呢?

在resetLoginForm中，观察this，发现它就是当前组件Login。它有$refs中保存着所有使用ref属性注册DOM元素或子组件实例。可以说就是一种便捷从当前组件上定位子组件或DOM元素的方法。

```
reset ~~~~                                                    Login.vue?7463:45
▼VueComponent {_uid: 17, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: VueCompo
 nent, …}
  ▶$attrs: Object
  ▶$children: [VueComponent]
  ▶$createElement: ƒ (a, b, c, d)
  ▶$el: div.login_container
   $listeners: (...)
  ▶$options: {parent: VueComponent, _parentVnode: VNode, propsData: undefined, _parentList…
  ▶$parent: VueComponent {_uid: 1, _isVue: true, $options: {…}, _renderProxy: Proxy, _sel…
  ▼$refs:
    ▶loginFormRef: VueComponent {_uid: 18, _isVue: true, $options: {…}, _renderProxy: Prox…
    ▶[[Prototype]]: Object
  ▶$root: Vue {_uid: 0, _isVue: true, $options: {…}, _renderProxy: Proxy, _self: Vue, …}
  ▶$scopedSlots: {$stable: true, $key: undefined, $hasNormal: false}
  ▶$slots: {}
  ▶$vnode: VNode {tag: "vue-component-11", data: {…}, children: undefined, text: undefine…
  ▶loginForm: Object
  ▶loginRules: Object
  ▶resetLoginForm: ƒ ()
```

那么通过`this.$refs.loginFormRef`就可以访问到表单了。

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="avatar_box">
        <img src="../assets/logo.png" alt="logo" />
      </div>
      <el-form class="login_form" :model="loginForm" :rules="loginRules"
ref="loginFormRef">
        <el-form-item prop="username">
          <el-input v-model="loginForm.username" placeholder="请输入用户名"
prefix-icon="el-icon-user"></el-input>
        </el-form-item>
        <el-form-item prop="password">
          <el-input type="password" v-model="loginForm.password"
placeholder="请输入密码" prefix-icon="el-icon-s-custom"></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary">登录</el-button>
          <el-button type="info" @click="resetLoginForm">重置</el-button>
        </el-form-item>
      </el-form>
    </div>
```

```
  </div>
</template>

<script>
export default {
  data() {
    return {
      loginForm: {
        username: '',
        password: ''
      },
      loginRules: {
        username: [
          { required: true, message: '请输入用户名', trigger: 'blur' },
          { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
        ],
        password: [
          { required: true, message: '请输入密码', trigger: 'blur' },
          { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
        ]
      }
    }
  },
  methods: {
    resetLoginForm() {
      this.$refs.loginFormRef.resetFields() // Form的resetFields方法重置
    }
  }
}
</script>

省略样式
```


键盘事件

为密码框增加keyup事件，响应什么键需要用到事件修饰符，修饰符可以串联。

https://cn.vuejs.org/v2/guide/events.html#%E6%8C%89%E9%94%AE%E4%BF%AE%E9%A5%B0%E7%AC%A6

需求是按键回车，所以应该是`@keyup.enter="login"`。可是增加对应代码发现，敲回车没反应。原因是，在el-input组件上，去监听它的原生事件，还要串联使用`.native`。

给登录按钮也增加一个`@click="login"`

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="avatar_box">
        <img src="../assets/logo.png" alt="logo" />
      </div>
      <el-form class="login_form" :model="loginForm" :rules="loginRules"
ref="loginFormRef">
        <el-form-item prop="username">
          <el-input
```

```
            v-model="loginForm.username"
            placeholder="请输入用户名"
            prefix-icon="el-icon-user"
          ></el-input>
        </el-form-item>
        <el-form-item prop="password">
          <el-input
            type="password"
            v-model="loginForm.password"
            placeholder="请输入密码"
            prefix-icon="el-icon-s-custom"
            @keyup.enter.native="login"
          ></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary" @click="login">登录</el-button>
          <el-button type="info" @click="resetLoginForm">重置</el-button>
        </el-form-item>
      </el-form>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      loginForm: {
        username: '',
        password: ''
      },
      loginRules: {
        username: [
          { required: true, message: '请输入用户名', trigger: 'blur' },
          { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
        ],
        password: [
          { required: true, message: '请输入密码', trigger: 'blur' },
          { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
        ]
      }
    }
  },
  methods: {
    resetLoginForm() {
      this.$refs.loginFormRef.resetFields() // Form的resetFields方法重置
    },
    login() {
      console.log('login~~~~~')
    }
  }
}
</script>

样式省略
```

表单预验证

参考：https://element.eleme.cn/#/zh-CN/component/form#form-methods

和重置类似，要调用From的validate(callback: Function(boolean, object))方法。回调函数是两参：是否校验成功和未通过校验的字段。

也就是说，如果validate校验的时候，会调用这个回调函数，注入2个实参，通过第一参判断是否验证成功，第二参取验证失败的结果。

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="avatar_box">
        <img src="../assets/logo.png" alt="logo" />
      </div>
      <el-form class="login_form" :model="loginForm" :rules="loginRules"
ref="loginFormRef">
        <el-form-item prop="username">
          <el-input
            v-model="loginForm.username"
            placeholder="请输入用户名"
            prefix-icon="el-icon-user"
          ></el-input>
        </el-form-item>
        <el-form-item prop="password">
          <el-input
            type="password"
            v-model="loginForm.password"
            placeholder="请输入密码"
            prefix-icon="el-icon-s-custom"
            @keyup.enter.native="login"
          ></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary" @click="login">登录</el-button>
          <el-button type="info" @click="resetLoginForm">重置</el-button>
        </el-form-item>
      </el-form>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      loginForm: {
        username: '',
        password: ''
      },
      loginRules: {
        username: [
          { required: true, message: '请输入用户名', trigger: 'blur' },
          { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
        ],
        password: [
```

```
          { required: true, message: '请输入密码', trigger: 'blur' },
          { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
        ]
      }
    }
  },
  methods: {
    resetLoginForm() {
      this.$refs.loginFormRef.resetFields() // Form的resetFields方法重置
    },
    login() {
      this.$refs.loginFormRef.validate((valid, object) => {
        if (valid) {
          console.log('ok ++++++')
        } else {
          console.log(object)
        }
      })
    }
  }
}
</script>

样式省略
```

axios

axios实现浏览器和服务端的异步通信。参考：https://www.kancloud.cn/yunye/axios/234845

使用axios时，很多情况发起HTTP请求都一样，就可以使用通用的全局的配置，这些全局配置有很多方式。

```
// 全局的 axios 默认值
axios.defaults.baseURL = 'https://api.example.com';
```

axios.get()、axios.post()的返回值都是Promise对象。

```
axios.get('/user?ID=12345')
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });


axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
```

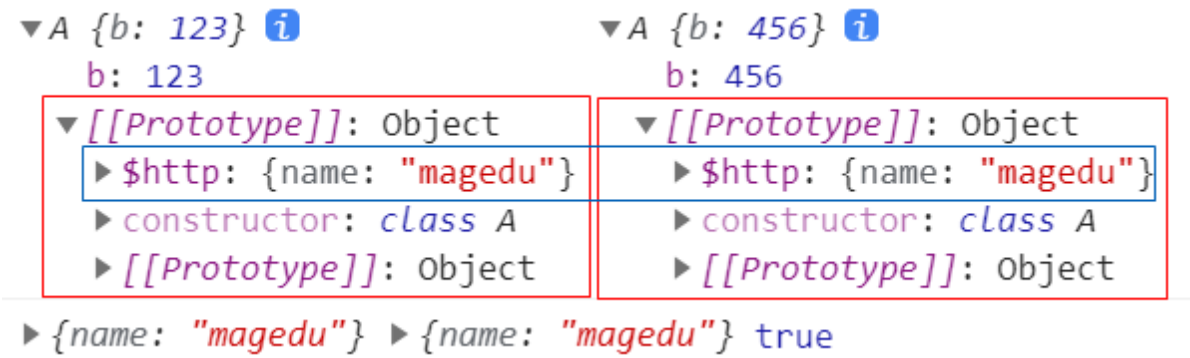
```
    console.log(error);
  });
```

全局属性

需求：每一个实例都可以获得的该类的一个属性

举例，假设有一个普通的类型A，得到它的实例t

```
class A {
    constructor(x) {
        // this.$http = x // 打开这个属性试一试，可以看到实例自己的属性优先
    }
}

A.prototype.$http = {name:'magedu'}
// A.$http = 'aabbcc' // 不可以

let t1 = new A('t1')
t1.b = 123

let t2 = new A('t2')
t2.b = 456
console.log(t1.__proto__, t2.__proto__, A.prototype,
    t1.__proto__ === A.prototype, // true
    t2.__proto__ === A.prototype  // true
    );
console.log(A);
console.log(t1, t2);
console.log(t1.$http, t2.$http, t1.$http === t2.$http);
// 优先访问自己的$http，没有就顺着原型链找t1.__proto__里面的属性
```



src/main.js

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import './plugins/element.js'
import './assets/css/main.css'
import axios from 'axios'

```

```
// axios全局设置，baseURL指向后台服务
axios.defaults.baseURL = 'http://127.0.0.1:5000/api/v1/'
// 为Vue类增加全局属性$http，这样所有组件实例都可以使用该属性了
Vue.prototype.$http = axios

Vue.config.productionTip = false

new Vue({
  router,
  render: h => h(App)
}).$mount('#app')
```

async、await和Promise

async 函数中才能使用await，await后跟上一个Promise。一旦await，async函数会暂停执行（类似Python中的yield），去执行其它语句，直到等待的这个Promise被resolve或者reject才恢复执行该async函数。

```

function testPromise() {
    return new Promise(
        (resovle, reject) => {
            // 为了模拟延时，使用setTimeout
            setTimeout(() => {
                //resovle({ user: { id: 100 } })
                reject({ code: 1 })
            }, 5000);
        }
    )
}

function test1() {
    let t = testPromise()
    t.then(
        value => { // fulfilled
            console.log('fulfilled');
            console.log(value, '++++');
        },
        reason => { // rejected
            console.log('failed', reason);
        }
    )
}

test1()
console.log('~~~~~~~~~~~~~~~~~~~~')
```

改写为async function

```

function testPromise() {
    return new Promise(
        (resovle, reject) => {
            // 为了模拟延时，使用setTimeout
```

```
            setTimeout(() => {
                resovle({ user: { id: 100 } })
                //reject({ code: 1 })
            }, 5000);
        }
    )
}

async function test1() {
    let t = await testPromise() // 成功返回value，失败抛异常
    console.log('================')
    console.log(t); // 可以对t这个返回的对象进一步处理
}

test1();
console.log('~~~~~~~~~~~~~~~~~~~~')
```

异步请求处理

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="avatar_box">
        <img src="../assets/logo.png" alt="logo" />
      </div>
      <el-form class="login_form" :model="loginForm" :rules="loginRules"
ref="loginFormRef">
        <el-form-item prop="username">
          <el-input
            v-model="loginForm.username"
            placeholder="请输入用户名"
            prefix-icon="el-icon-user"
          ></el-input>
        </el-form-item>
        <el-form-item prop="password">
          <el-input
            type="password"
            v-model="loginForm.password"
            placeholder="请输入密码"
            prefix-icon="el-icon-s-custom"
            @keyup.enter.native="login"
          ></el-input>
        </el-form-item>
        <el-form-item>
          <el-button type="primary" @click="login">登录</el-button>
          <el-button type="info" @click="resetLoginForm">重置</el-button>
        </el-form-item>
      </el-form>
    </div>
  </div>
</template>

<script>
export default {
  // 省略其它，只保留修改的部分，便于分析代码
```

```
  methods: {
    login() {
      this.$refs.loginFormRef.validate(async (valid) => {
        if (valid) {
          const res = await this.$http.post('post', this.loginForm) // post
返回一个Promise
          console.log(res) // status状态码，data返回的数据
          const { data: response } = res // data解构出来
          // 返回的对象
          console.log(response)
          console.log(response.code)
          if (!response.code) {
            // 如果返回的对象没有code属性或不为0，说明成功了
            console.log('登录成功')
          } else {
            console.log('登录失败')
          }
        }
      })
    }
  }
}
</script>
```

由于没有后台，所以，例如下面网站测试

```
// src/main.js中
axios.defaults.baseURL = 'http://httpbin.org/'

// login函数中
const res = await this.$http.post('post', this.loginForm)
```

实际上POST到http://httpbin.org/post，它会返回提交的数据，便于测试。

如果一切成功，那么就需要一个后台了，下一节我们将开始Django之旅。