Vue前端小作业说明文档

小作业目标

使用Vue框架设计一个简单的留言板web页面，通过完成该任务，同学们主要学习掌握以下技能：

- 使用Vue框架学会编写前端组件
- 前端与后端通信获取并展示
- 通过前端弹出框等交互方式学习组件的回调和数据交互
- 修改和删除cookie实现前端持久化存储
- 学会通过mock、devServer等方式实现前端开发时调试（不要求编码）

实验环境版本信息

```
"@vue/cli": "4.4.6"
"core-js": "^3.6.5",
"element-ui": "^2.13.2"
"vue": "^2.6.11"
```

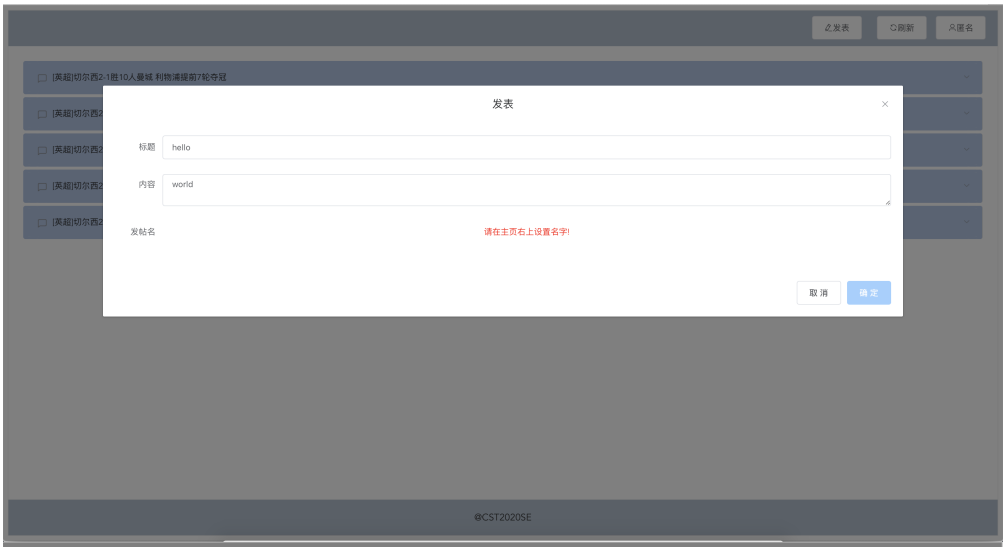
项目需求与示例

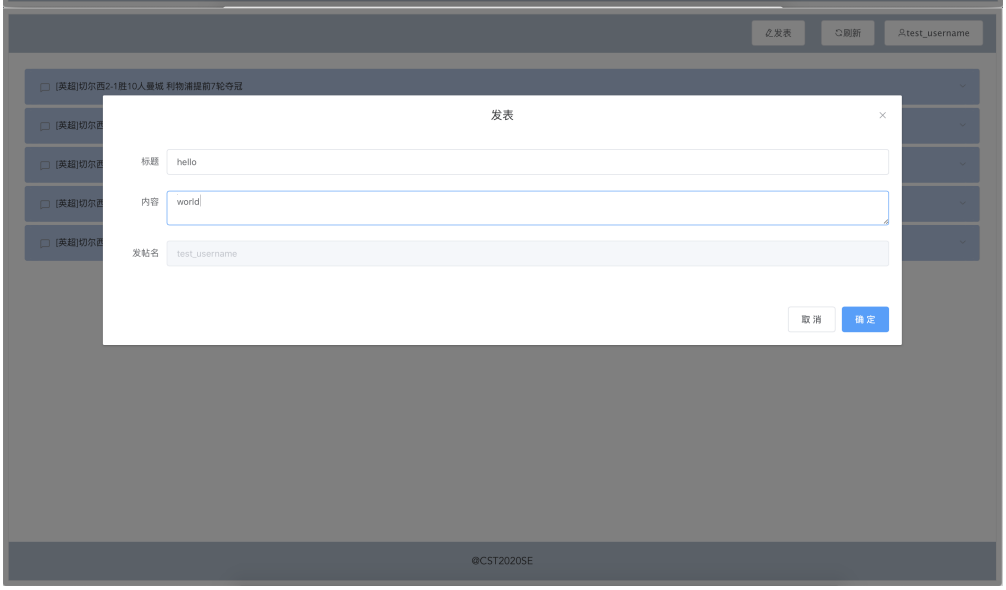
在该次小作业任务中，我们需要设计一个简单的留言板web页面，需求如下：

- 实现一个包含导航栏和信息板的留言板，如下所示（这里消息默认折叠到标题，点击时展开）：

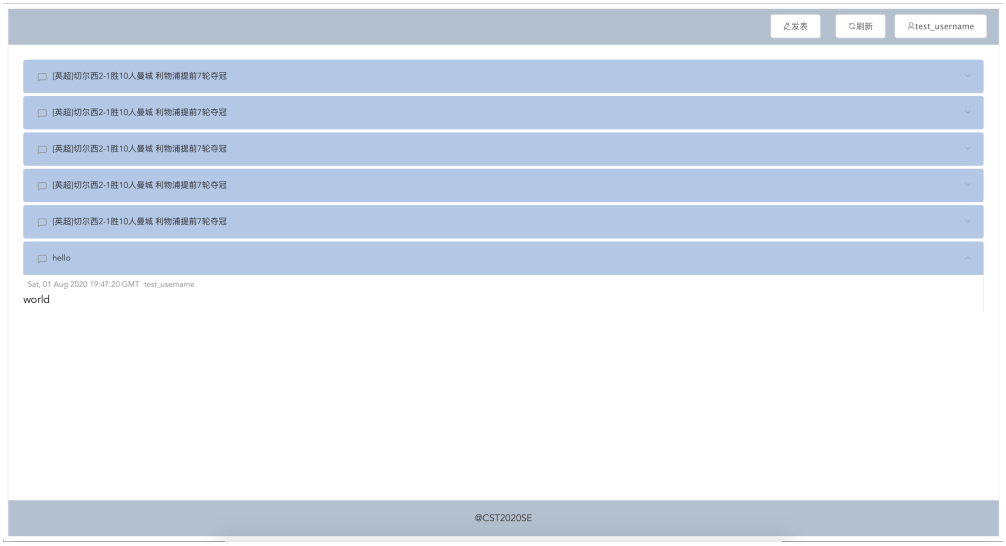
- 通过导航栏的按钮触发弹出对话框，控制发表留言和用户名保存
    - 用户名保存：
        - 请通过修改cookie实现页面中用户名的持续存储，使得保存用户名后刷新页面，用户名仍然保存
        - 本次前端项目不做用户登录校验，本质上是一个匿名的留言板。Alice和Bob同时以Cindy的名字发帖，也是合法的

- 发表留言：
  - 点击留言，匿名状态下禁止发帖，存在用户名的情况下才允许发帖





  - 发帖成功后可以看到发帖内容



- 信息栏：
  - 通过获取的消息列表逐条显示即可，要求显示标题、内容、用户名和时间
    - 不要求按照示例中将内容折叠到标题

接口说明

- 获取消息列表：
  - timestamp为13位时间戳

接口	方法	类型	示例
/api/message	GET	json	[{"title":"hello","content":"world","user":"Alice", "timestamp":1593133200000}, {"title":"hello2","content":"world2","user":"Bob", "timestamp":1593133200000}]

- 发表留言
  - user字段请通过cookie传递

接口	方法	类型	示例	其他
/api/message	POST	json	{"title":"hello","content":"world"}	cookie中传递user字段

- 返回码：
  - 200：成功
  - 其他：失败

简单了解Vue

目标

- 使用@vue/cli脚手架自动创建vue项目
- 了解webpack项目组成
- 了解Vue作用的方式和组件含义
- 简单了解Element-UI

具体介绍与步骤

- Vue是一个较为轻量的前端设计框架，我们使用组件化方法开发Vue前端项目，能非常方便插入各种项目中
  - 我们可以从官网https://cn.vuejs.org/v2/guide/了解更多
- 一般webpack项目使用npm管理，首先请安装npm
  - 链接：https://nodejs.org/en/download/
  - 配置好路径
- 使用npm安装@vue/cli，这是一个自动化创建vue项目的脚手架工具

```
npm install -g @vue/cli
```

  此时应该可以在命令行cmd/bash/terminal中找到vue命令

- 使用@vue/cli脚手架自动创建vue项目，在命令行中输入以下命令

```
vue create project-name
```

  其中`project-name`为你想要的项目名称

- 此时在project-name路径下就是你创建完毕的vue项目代码
  - 其中比较重要的有
    - `package.json`：项目名、项目版本、npm脚本、依赖与开发时依赖版本等信息的配置
      - `scripts`是npm运行的脚本，在@vue/cli初始创建项目中，有serve/build/lint等，对应可以执行npm run serve/build/lint命令，可以自己修改每个命令对应的具体指令以控制参数
      - `dependencies`与`devDependencies`为依赖与开发时依赖
    - `public/index.html`：代码的入口，其中有一个id为app的div元素（锚点）
    - `src/main.js`：启动执行的代码，初始完毕后，将Vue组件App渲染到了`public/index.html`中id为app的div上
    - `App.vue`：初始的一个全局组件，代表整个页面，其内部使用了其他组件实现渲染
- 组件
  - Vue使用组件作为每个单元的基本元素，每个组件有较好的封装，通过指标的绑定和传入参数的不同控制其各自的具体表现
  - 每个`.vue`组件一般主要包含template、script、style几部分，其中template部分为待渲染的内容；script部分根据外部参数、自身的数据和函数回调将template中的内容渲染，供外部使用；style部分为css样式
    - vue的组件之间可以嵌套，大的组件可以包含小的组件，例如初始项目内App内就使用了一个其他组件
  - 更多请参照
    - 模版：https://cn.vuejs.org/v2/guide/syntax.html
    - 组件基础：https://cn.vuejs.org/v2/guide/components.html
- Element-UI
  - 是一整套已经编写好对应样式的组件，在引入后可以直接使用
  - 使用它，我们可以不用编写大量繁杂的css样式，而且当需要调换主题风格时，只需要替换相关路径下的css即可
  - 引入：

- 在我们项目的`package.json`中的`dependencies`中添加`element-ui`

```
"dependencies":{
    ...,
    "element-ui": "^2.13.2"
}
```

并命令行执行`npm install`更新安装

- 在我们`src/main.js`中添加相关代码使之如下：

```
import Vue from 'vue'

//添加以下三行代码引入ElementUI的内容
import ElementUI from 'element-ui';
import 'element-ui/lib/theme-chalk/index.css';
Vue.use(ElementUI)

Vue.config.productionTip = false

import App from './App.vue'

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

此时我们就可以使用Element-UI的组件了，尝试修改一下我们的`App.vue`（或者其他组件）中的template的内容，参照https://element.eleme.cn/#/zh-CN/component/使用一下各类组件吧

  - 更多请见：
    - 完整引入和部分引入：https://element.eleme.cn/#/zh-CN/component/quickstart
    - 自定义主题：https://element.eleme.cn/#/zh-CN/component/custom-theme

Vue中的数据

目标

- 编写一个消息帖组件展示一条消息的时间、用户、标题和内容

说明与任务要求

- 每一个`.vue`组件都可以通过外部传入对应参数决定对应的渲染结果。例如以下是一个组件的内容，则指定了有一个名为`metric_str`的字符串类型传入参数和一个`metric_nmb`的实值参数，并分别渲染出来

```
<template>
    <div>
        {{metric_str}}
    </div>
        <div>
        {{metric_nmb}}
    </div>
</template>

```

```
<script>
    export default {
        name: 'test_component',
        props:{//传入参数列表
          metric_str:{  //参数名
                type:String,    //类型
                default: () => "unknown value"  //默认值
            },
          metric_nmb: { //参数名
                type:Number,    //类型
                default: () => 0    //默认值
            },
        }
    }
</script>
```

则此时我们在另一个父组件中可以如下指定传入参数的值，对应的子组件的内容也会随着绑定的值的不同而不同，绑定通过`v-bind:`实现

```
<template>
    ...
        <test_component v-bind:metric_str="test_str" v-bind:metric_nmb=0 />
    ...
</template>
...
```

```
<template>
    ...
        <test_component v-bind:metric_str="test_str2" v-bind:metric_nmb=3 />
    ...
</template>
...
```

由于`v-bind:`可以在vue中缩写为`:`,以上部分可以写为

```
<test_component :metric_str="test_str" :metric_nmb=0 />
```

当然，很可能父组件需要传入的是一个变量而非常量，那么假设父组件为

```
<template>
    <div>
            ...
        <test_component :metric_str="test_component_props.str"
                        :metric_nmb="test_component_props.nmb" />
        ...
    </div>
</template>

<script>
    import test_component from "@/test_component"
    export default {
```

```
        name: 'test_component_parent',
        components: {
            test_component //表明使用了test_component作为子组件
        },
        props:{
          ...       //父组件自己的上层的传入参数
        },
        data():return{
          ...,
          test_component_props:{
            str:"123",
            nmb:123
            }
            },
        methods:{
          ...       //组件封装的回调函数
        }
    }
</script>
```

那么此时就实现了动态传入参数，如果之后在运行中，`test_component_props`改变了，那么我们的组件仍然会动态实时地重新渲染

- vue支持一些简单的计算，所以上面父组件中如果写成一个简单表达式也是可以的（如下），但是如果太复杂，则需要通过回调函数处理

```
<template>
    <div>
            ...
        <test_component :metric_str="test_component_props.str"
                        :metric_nmb="test_component_props.nmb * 2 +
3" />
        ...
    </div>
</template>
```

- 任务要求：请参见`MessageBox`组件的内容，填上空缺部分，使之能正确显示消息的标题、内容、事件、用户名

更多

请参见声明式渲染、语法、计算属性、事件处理

Vue中的简单响应和控制(1)与前后端请求交互

目标

- 编写一个消息列表组件，根据从后端得到的消息列表，利用前一步中的消息帖组件，以列表方式展示
- 学习编写前后端交互

说明与任务要求

- vue中有一些控制属性可以控制组件是否显示渲染以及实现循环元素渲染等，请参照列表渲染与条件渲染为`MessageList`组件的对应代码填空，以完成`stage2`页面
- 获取消息列表需要与后端交互，请填写`communication.js`中`request_json`函数的内容
  - 前后端交互有原生xhr和axios等一些封装库，可以参照下面链接进行学习
    - https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest
    - https://www.w3school.com.cn/xml/xml_http.asp
    - http://www.axios-js.com/zh-cn/docs/

    注意： 使用库时请确认所传json是否序列化处理
  - 测试调试时如果直接访问后端地址，可能有跨域问题会导致浏览器禁止请求，请参照后文中`Vue项目的前端调试与测试`的内容，选取合适方法，在开发时通过mock或者devServer解决调试问题

Vue中的简单响应和控制(2)与前端cookie存储

目标

- 学习并编写一个消息弹出框，根据响应实现对应操作
- 学习cookie存储和修改、删除方法，并在前端实现设置、修改和删除user字段的值

说明与任务要求

- Element UI对话框
  - 请参照ElementUI对话框中的相关示例以及我们`MessageBoard`中的示例，学习对话框控制弹出和关闭的写法
- vue表单输入
  - vue使用v-model绑定表单数据，将用户的操作直接反映到对应数据组中
    - 参见vue表单输入绑定
- 前端cookie存储
  - 对一个web页面而言，有时需要进行一些信息的持久化存储，以避免刷新页面时用户重新输入相关信息，本次我们将会学习cookie的设置
  - 前端cookie的设置/修改/删除：
    - js中只需要形如如下进行一次字符串赋值操作即可完成cookie写入

      ```
      document.cookie = key + "=" + escape(value) + ";expires=" + GMTdatestring
      ```

      其中`key`,`value`,`GMTdatestring`分别为我们设置的键名、值、cookie过期时间字符串，浏览器会自动查找对应的键名，将对应的cookie修改（而不会影响其他已经设置好的cookie键值对）
      - `escape`为js原生函数，处理字符串转义
    - cookie的删除方式同上，只需要将cookie的过期时间设置为当前时间（或者之前）即可
  - 读取cookie
    - 在对应路径下，只要访问`document.cookie`即可获得在该页面下的所有有效存活cookie，其格式一般为形如`XXX=aaa; YYY=bbb; ZZZ=ccc`我们需要从中找出我们需要的键值对，这里可以推荐使用正则表达式
      - js正则表达式（直接搜索和使用RegExp均可）：https://www.runoob.com/js/js-regexp.html

- 任务要求：
  - 填写 `PostDialog` 的空缺代码，用户点击 发表 按钮后弹出发表留言对话框，要求没有用户名时不允许发表，用户有用户名且输入了标题和内容后发送相关请求到后端，并给出发送成功与否的反馈
  - 填写cookie操作相关函数，实现cookie的设置修改和删除

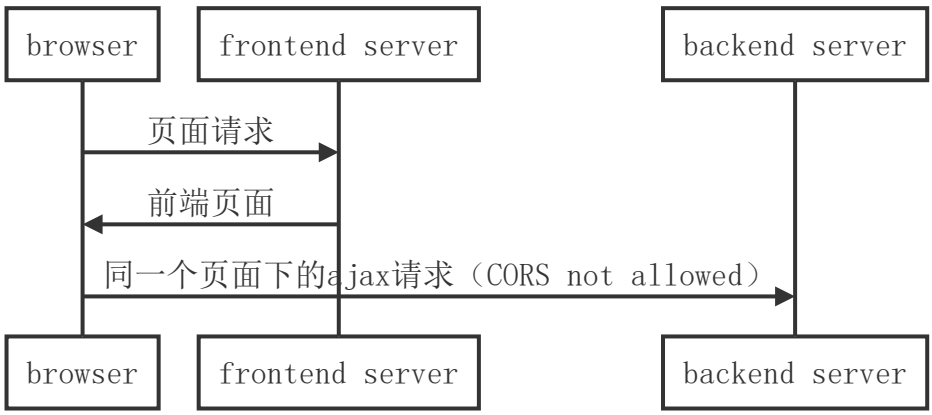
Vue项目的前端调试和测试

目标

- 了解mock调试方法
- 了解devServer调试方法
- 了解vue的简单组件调试

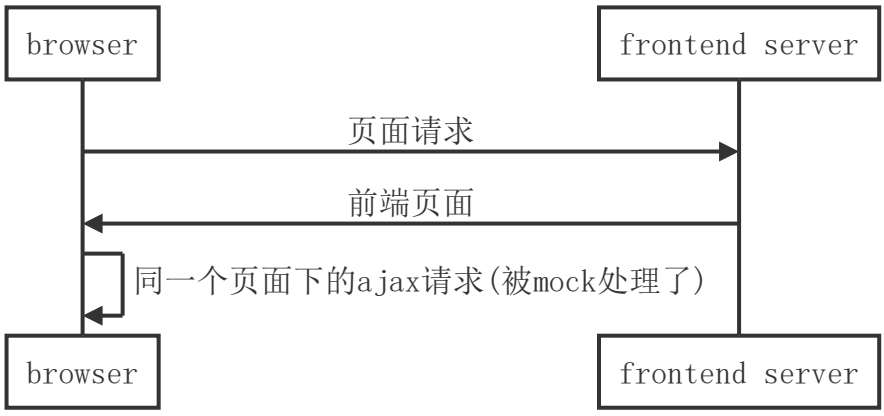
前端调试

跨域请求（也即前端页面和后端接口不属于同一个域）属于非常不安全的行为，因此一般后端都不会将CORS选项打开，在现在主流浏览器标准里，在没有后端CORS确认允许的情况下的一切跨域请求都会被浏览器拦截，这给我们直接使用前端访问后端接口以及前端代码开发调试带来了困难。在生产环境下，我们一般通过nignx代理前后端服务器或者将前端代码编译好后直接融入后端模版（这对于vue框架来说也是非常方便的），但是在开发环境下，尤其是前后端分离比较普遍的现在，这样的方法不够高效，不利于我们直接测试前端代码。下面介绍两类方法，分别可以针对不同情况实现测试。



前端调试（无后端）：使用mock

使用mock是一种比较自由简单的测试方法，其主要原理是将请求全部在浏览器前拦截（可以理解为，前端web页面尝试的xhr请求都被js代码处理了）



- 在`package.json`的`devDependencies`中添加`mockjs`

```
"devDependencies":{
    ...,
    "mockjs": "^1.1.0"
}
```

并在命令行中`npm install`更新

- 在`src`路径下加入`mock`文件夹，加入`index.js`文件，写入如下代码

```
var Mock = require('mockjs')
Mock.setup({timeout:"200-400"})
```

其中`timeout`为模拟时的模拟时延

- 编写对应URL的模拟代码，例如

```
//初始的消息列表
var messageList = Array(5).fill(
    {
        "user": "Alice",
        "title":"[英超]切尔西2-1胜10人曼城 利物浦提前7轮夺冠",
        "content":"北京时间6月26日03:15(英国当地时间25日20:15)，2019/20赛季英超第31轮一场焦点战在斯坦福桥球场展开争夺，切尔西主场2比1力克曼城，普利西奇和威廉进球，费尔南迪尼奥送出红点。切尔西3连胜，曼城客场连败送利物浦提前7轮夺冠。",
        "timestamp":1593133200000}
    )

//处理获取消息列表
Mock.mock(/\/api\/message[\s\S]+?/,"GET",(rqst)=>{
    return messageList
})

//处理消息上传
Mock.mock("/api/message","POST",(rqst)=>{
    console.log(rqst)
    rqst.body = JSON.parse(rqst.body)
    try{
        messageList.push({
            "user":rqst.body.user,  //注意！这里仅仅是为了调试，对原来请求的代码有所修改!
```

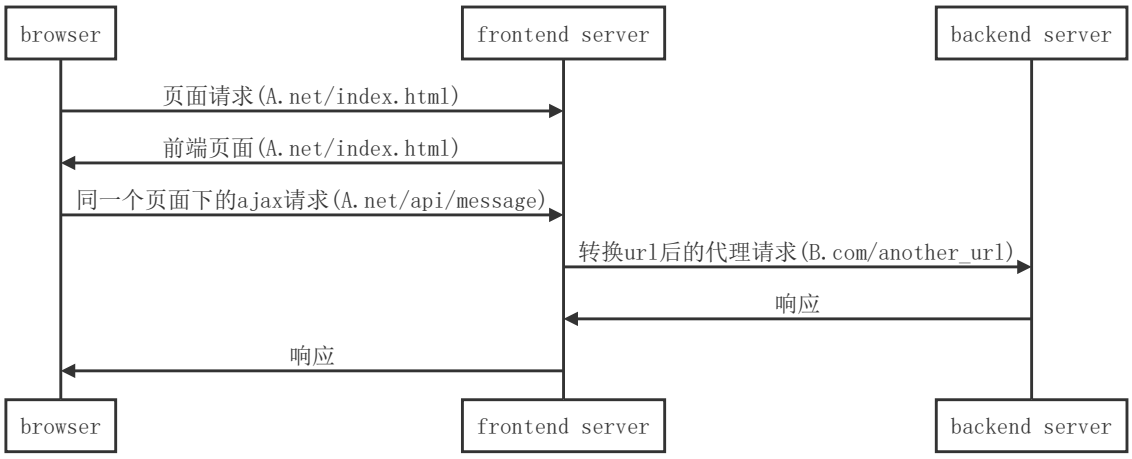
```
            "title":rqst.body.title,
            "content":rqst.body.content,
            "timestamp":new Date().getTime(),
        })
        return {"code":200,"message":"OK"}
    }catch (e) {
        console.log(e)
        return {"code":400,"message":e.toString()}
    }
})
```

注意！：mock方法模拟时，由于其实没有经过浏览器，所以无法获得对应的cookie字段，按照我们本项目中原来的设计，`user`是不应该放在请求的body中的，但是这里为了测试所以添加了冗余字段便于调试

- 最后将这段js代码被import或者require一次即可，因为所有请求都经过了`commnunication.js`所以我们可以在这个文件开头加上

```
import "@/mock/index"
```

前端调试（有后端）：使用devServer配置



在开发到一定阶段时，如果我们已经有一个稳定后端而需要调试前端，或者前后端需要联调时，使用devServer配置会比较合理，如上图所示，devServer方法将后端链接反代理到自身的一个url，然后将代理的url解析后重新提交给后端（此时后端可以配置仅接受来自前端服务器的请求，此时对外部浏览器用户而言后端服务器的地址是不公开的）

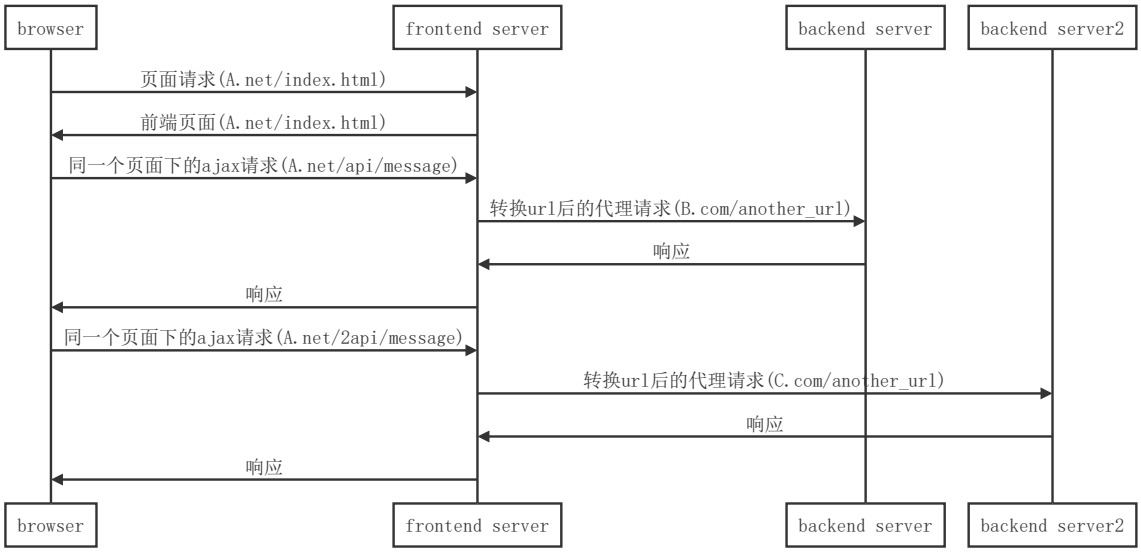
- 假设前端地址为`localhost:8000`，准备访问`localhost:9000/api_of_9000/message`的内容（两个不同端口，不是同一个域下），则在项目根目录下添加`vue.config.js`，加入以下内容：

```
module.exports = {
    publicPath: '/',
    outputDir: 'dist',       //和编译结果输出路径有关，开发阶段不用管
    devServer: {
        open: true,              //是否开启
        host: 'localhost',
        port: '8000',
        proxy: {
            '/api': {
```

```
                target: 'localhost:9000/api_of_9000', // 实际后端地址
                ws: true,
                changeOrigin: true,
                pathRewrite: {      //url转换
                    '^/api': ''
                }
            }
        }
    }
};
```

然后在前端只要访问`localhost:8000/api/message`即可

- 这里所做的转换，就是首先找到对应于`localhost:8000/api`的地址，如果发现这样的请求，就按照下面的`pathRewrite`，将其删去开头的`/api`，然后替换为`localhost:9000/api_of_9000`进行代理访问
- 可以看到，这里的proxy下可以有多个列表，所以可以配置实现不同的url转换规则或者配置转发到不同服务器



其他

- 检查：同学上传代码后，与助教提供的标准后端对接（以devServer模式），并进行检查