

# Axios

axios是一个基于 promise 的 HTTP 库，可在浏览器和 node.js 中。

特性：

* 从浏览器中创建 XMLHttpRequest
* 从 node.js 发出 http 请求
* 支持 Promise API
* 拦截请求和响应
* 转换请求和响应数据
* 取消请求
* 自动转换JSON数据
* 客户端支持防止 [CSRF/XSRF](http://baike.baidu.com/link?url=iUceAfgyfJOacUtjPgT4ifaSOxDULAc_MzcLEOTySflAn5iLlHfMGsZMtthBm5sK4y6skrSvJ1HOO2qKtV1ej_" \t "https://ykloveyxk.github.io/2017/02/25/axios%E5%85%A8%E6%94%BB%E7%95%A5/_blank)

## 使用方式

* npm使用方式

|  |  |  |
| --- | --- | --- |
| npm install axios |  |  |
|  |  |  |

* cdn方式

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>

* 使用本地文件

<script src="js/axios.min.js"></script>

## 使用axios发起请求

例子1： get请求url中传递参数

服务器controller

@GetMapping("/api/v1/user/query")  
public SysUser getUser2(Integer uid){  
 SysUser user = new SysUser();  
 user.setId(uid);  
 user.setLoginPassword("hello");  
 user.setName("lisi");  
 user.setPhone("1360000000");  
 return user;  
}

前端：

在项目根目录下创建js目录，目录中添加axios.min.js文件

准备页面：testAxios.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <script type="text/javascript" src="js/axios.min.js"></script>  
</head>  
<body>  
 <button onclick="f1()">发起get请求例子1</button>  
 <script>  
  
 </script>  
</body>  
</html>

f1请求

<body>  
 <button onclick="f1()">发起get请求例子1</button>  
 <script>  
 function f1(){  
 let url="http://localhost:8000/ylb/api/v1/user/query?uid=1021";  
 axios.get(url).then( (resp)=>{  
 ***console***.log("请求成功，服务响应数据="+resp)  
 }).catch((err)=>{  
 ***console***.log("请求失败："+err)  
 })  
 }  
 </script>  
</body>

f2请求： 使用axios配置项 params，指定get请求参数列表

<button onclick="f2()">发起get请求例子2</button>

function f2(){  
 let url="http://localhost:8000/ylb/api/v1/user/query";  
 axios.get(url,{  
 params:{  
 uid:2126  
 }  
 }).then( (resp)=>{  
 ***console***.log("请求成功，服务响应数据="+resp)  
 ***console***.dir(resp)  
 }).catch((err)=>{  
 ***console***.log("请求失败："+err)  
 })  
}

例子2 ： post请求，参数名=值&参数名=值方式传递请求参数

服务器：

@PostMapping("/api/v1/users")  
public SysUser getUser2(String name,String passwd){  
 SysUser user = new SysUser();  
 user.setId(1001);  
 user.setLoginPassword(passwd);  
 user.setName(name);  
 user.setPhone("1360000000");  
 return user;  
}

前端

<button onclick="f3()">发起post请求例子1</button>

function f3(){  
 let url="http://localhost:8000/ylb/api/v1/users";  
 axios.post(url,"name=lisi&passwd=123456").then( (resp)=>{  
 ***console***.log("请求成功，服务响应数据="+resp)  
 ***console***.dir(resp)  
 }).catch((err)=>{  
 ***console***.log("请求失败："+err)  
 })  
}

例子3 ： 请求参数是json格式， 后端使用@RequestBody

服务器

public class Person {  
 private String name;  
 private String passwd;  
 // set | get  
}

@PostMapping("/api/v1/person")  
public Person getPerson(@RequestBody Person person){  
 System.*out*.println("person="+person);  
 return person;  
}

前端 : 使用axios的配置项，发送请求

<button onclick="f4()">发起post请求例子2</button>

function f4(){  
 let url="http://localhost:8000/ylb/api/v1/person";  
 axios({  
 url:url,  
 method: 'post',  
 data:{  
 'name':'zhangsan',  
 'passwd':'111222'  
 }  
 }).then( (resp) =>{  
 ***console***.log("resp="+resp)  
 }).catch( (error)=>{  
 ***console***.log("错误="+error)  
 })  
}

完整的配置项：

|  |
| --- |
| {  // `url` 是用于请求的服务器 URL  url: '/user',  // `method` 是创建请求时使用的方法  method: 'get', // 默认是 get  // `baseURL` 将自动加在 `url` 前面，除非 `url` 是一个绝对 URL。  // 它可以通过设置一个 `baseURL` 便于为 axios 实例的方法传递相对 URL  baseURL: 'https://some-domain.com/api/',  // `transformRequest` 允许在向服务器发送前，修改请求数据  // 只能用在 'PUT', 'POST' 和 'PATCH' 这几个请求方法  // 后面数组中的函数必须返回一个字符串，或 ArrayBuffer，或 Stream  transformRequest: [function (data) {  // 对 data 进行任意转换处理  return data;  }],  // `transformResponse` 在传递给 then/catch 前，允许修改响应数据  transformResponse: [function (data) {  // 对 data 进行任意转换处理  return data;  }],  // `headers` 是即将被发送的自定义请求头  headers: {'X-Requested-With': 'XMLHttpRequest'},  // `params` 是即将与请求一起发送的 URL 参数  // 必须是一个无格式对象(plain object)或 URLSearchParams 对象  params: {  ID: 12345  },  // `paramsSerializer` 是一个负责 `params` 序列化的函数  // (e.g. https://www.npmjs.com/package/qs, http://api.jquery.com/jquery.param/)  paramsSerializer: function(params) {  return Qs.stringify(params, {arrayFormat: 'brackets'})  },  // `data` 是作为请求主体被发送的数据  // 只适用于这些请求方法 'PUT', 'POST', 和 'PATCH'  // 在没有设置 `transformRequest` 时，必须是以下类型之一：  // - string, plain object, ArrayBuffer, ArrayBufferView, URLSearchParams  // - 浏览器专属：FormData, File, Blob  // - Node 专属： Stream  data: {  firstName: 'Fred'  },  // `timeout` 指定请求超时的毫秒数(0 表示无超时时间)  // 如果请求话费了超过 `timeout` 的时间，请求将被中断  timeout: 1000,  // `withCredentials` 表示跨域请求时是否需要使用凭证  withCredentials: false, // 默认的  // `adapter` 允许自定义处理请求，以使测试更轻松  // 返回一个 promise 并应用一个有效的响应 (查阅 [response docs](#response-api)).  adapter: function (config) {  /\* ... \*/  },  // `auth` 表示应该使用 HTTP 基础验证，并提供凭据  // 这将设置一个 `Authorization` 头，覆写掉现有的任意使用 `headers` 设置的自定义 `Authorization`头  auth: {  username: 'janedoe',  password: 's00pers3cret'  },  // `responseType` 表示服务器响应的数据类型，可以是 'arraybuffer', 'blob', 'document', 'json', 'text', 'stream'  responseType: 'json', // 默认的  // `xsrfCookieName` 是用作 xsrf token 的值的cookie的名称  xsrfCookieName: 'XSRF-TOKEN', // default  // `xsrfHeaderName` 是承载 xsrf token 的值的 HTTP 头的名称  xsrfHeaderName: 'X-XSRF-TOKEN', // 默认的  // `onUploadProgress` 允许为上传处理进度事件  onUploadProgress: function (progressEvent) {  // 对原生进度事件的处理  },  // `onDownloadProgress` 允许为下载处理进度事件  onDownloadProgress: function (progressEvent) {  // 对原生进度事件的处理  },  // `maxContentLength` 定义允许的响应内容的最大尺寸  maxContentLength: 2000,  // `validateStatus` 定义对于给定的HTTP 响应状态码是 resolve 或 reject promise 。如果 `validateStatus` 返回 `true` (或者设置为 `null` 或 `undefined`)，promise 将被 resolve; 否则，promise 将被 rejecte  validateStatus: function (status) {  return status >= 200 && status < 300; // 默认的  },  // `maxRedirects` 定义在 node.js 中 follow 的最大重定向数目  // 如果设置为0，将不会 follow 任何重定向  maxRedirects: 5, // 默认的  // `httpAgent` 和 `httpsAgent` 分别在 node.js 中用于定义在执行 http 和 https 时使用的自定义代理。允许像这样配置选项：  // `keepAlive` 默认没有启用  httpAgent: new http.Agent({ keepAlive: true }),  httpsAgent: new https.Agent({ keepAlive: true }),  // 'proxy' 定义代理服务器的主机名称和端口  // `auth` 表示 HTTP 基础验证应当用于连接代理，并提供凭据  // 这将会设置一个 `Proxy-Authorization` 头，覆写掉已有的通过使用 `header` 设置的自定义 `Proxy-Authorization` 头。  proxy: {  host: '127.0.0.1',  port: 9000,  auth: : {  username: 'mikeymike',  password: 'rapunz3l'  }  },  // `cancelToken` 指定用于取消请求的 cancel token  // （查看后面的 Cancellation 这节了解更多）  cancelToken: new CancelToken(function (cancel) {  })} |

## 请求方法的别名

##### axios.get(url[, config])

##### axios.delete(url[, config])

##### axios.post(url[, data[, config]])

##### axios.put(url[, data[, config]])

## axios返回数据

响应结构，是一个json

|  |
| --- |
| {  // `data` 由服务器提供的响应  data: {},  // `status` 来自服务器响应的 HTTP 状态码  status: 200,  // `statusText` 来自服务器响应的 HTTP 状态信息  statusText: 'OK',  // `headers` 服务器响应的头  headers: {},  // `config` 是为请求提供的配置信息  config: {}  } |

<button onclick="f6()">应答数据</button>

function f6(){  
 let ins = axios.create({  
 baseURL:'http://localhost:8000/ylb',  
 timeout:10000  
 })  
 ins.get("/api/v1/user/query?uid=1001")  
 .then( (resp)=>{  
 ***console***.log("resp="+resp.data)  
 ***console***.log("resp="+resp.status)  
 ***console***.log("resp="+resp.statusText)  
 ***console***.log("resp="+resp.data.id)  
 ***console***.log("resp="+resp.data.name)  
 ***console***.log("resp="+resp.data.phone)  
 }).catch((error)=>{  
 ***console***.log("错误="+error)  
 ***console***.dir(error);//error是一个错误对象  
 ***console***.log(error.response.data);  
 ***console***.log(error.response.data.error);  
 ***console***.log(error.response.status);  
 })

}

拦截器

分成请求拦截器和响应拦截器

请求拦截器：在发起请求之前执行，可以对请求内容做修改，比如增加参数，设置请求头等等。

应答拦截器：服务器返回结果，axios的then之前先执行。可以对应答内容对预先的处理。

### 拦截器使用例子

<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <script type="text/javascript" src="js/axios.min.js"></script>  
</head>

<body>  
 <button onclick="f1()">请求拦截器</button>  
 <script>  
 //请求拦截器  
 axios.interceptors.request.use(function(config){  
 //请求之前做些什么：比如改变请求参数，设置timeout等等  
 ***console***.log("======请求之前=====")  
 config.url = config.url+"&name=lisi";  
 return config;  
 },function (error) {  
 // 对请求错误做些什么  
 return ***Promise***.reject(error);  
 })  
  
 //应答拦截器  
 axios.interceptors.response.use(function(response){  
 //对响应做什么  
 ***console***.log("=====响应数据了=======")  
 return response;  
  
 },function(error){  
 // 对响应错误做点什么  
 return ***Promise***.reject(error);  
 })  
  
  
 function f1(){  
 let url="http://localhost:8000/ylb/api/v1/user/query?uid=1021";  
 axios.get(url).then( (resp)=>{  
 ***console***.log("请求成功，服务响应数据="+resp)  
 }).catch((err)=>{  
 ***console***.log("请求失败："+err)  
 })  
 }  
  
  
 </script>  
</body>

## 全局的 axios 默认值

axios.defaults.baseURL = 'https://api.example.com';

axios.defaults.timeout=50000;

axios.defaults.headers.common['Authorization'] = AUTH\_TOKEN;

axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';

# Vue环境搭建

## nodejs

下载nodejs: <http://nodejs.cn/download/current/>



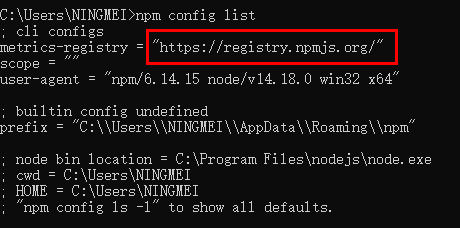
## 测试安装结果

命令行执行 npm -v



可以设置淘宝的仓库镜像

先查看现有仓库地址 npm config list



设置淘宝镜像（和maven使用阿里云镜像地址一样的作用）

npm config set registry https://registry.npm.taobao.org

## 安装webpack

npm i -g webpack webpack-cli

## 安装vue-cli

安装文档地址： <https://cli.vuejs.org/zh/guide/installation.html>

npm i -g @vue/cli

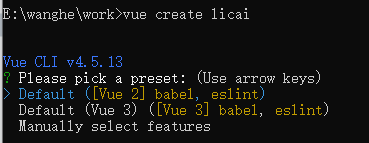
## 查看vue-cli

命令行 执行 vue -V

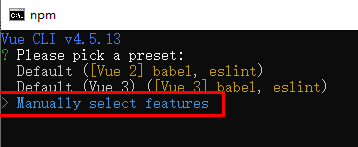


## vue-cli创建项目

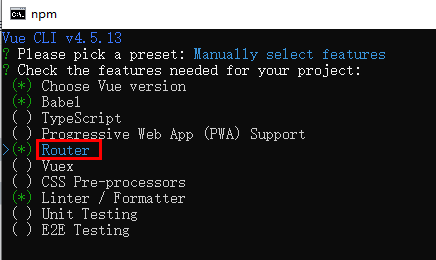
进入某一目录，例如work



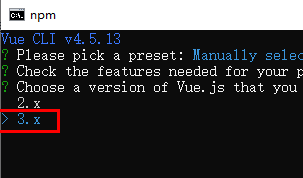
按下箭头键，选择最后一项



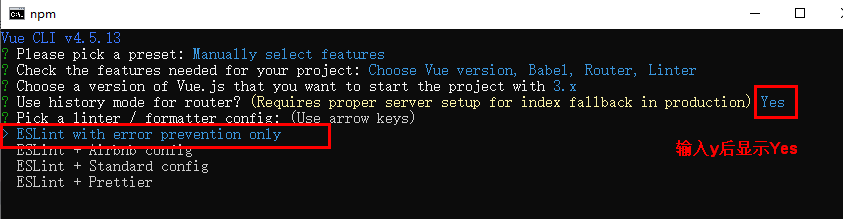
按回车键，新界面选择Router （使用空格键）



回车后，界面选择3.x（使用空格键）



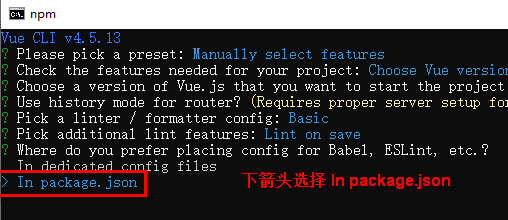
再回车



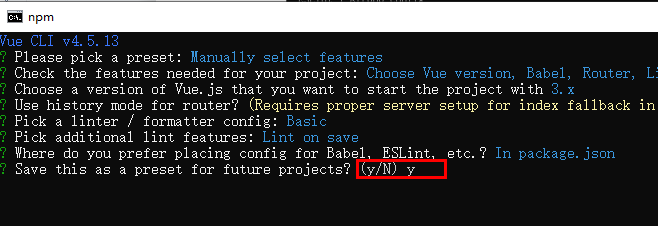
再回车，默认选项就可以



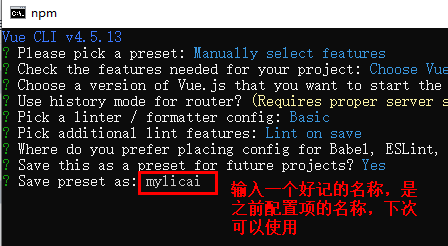
再回车



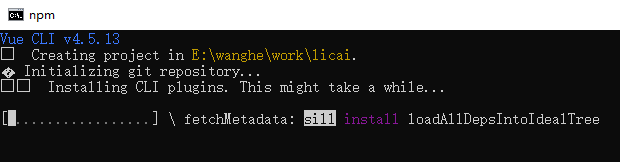
再回车，输入y



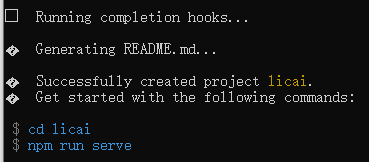
再回车



开始创建项目，等着完成



创建成功



## 在工程中安装vue

进入工程目录 cd /work/licai

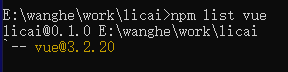
安装vue最新版本

npm i --save vue

## 查看vue版本

npm list vue 或者到项目目录 package.json 依赖位置查看版本

查看vue 版本 3.2.20



## 运行项目

进入到 work/licai 目录

cd E:\wanghe\work\licai

执行命令：npm run serve

稍等一会， 在浏览器运行程序 <http://localhost:8080/>

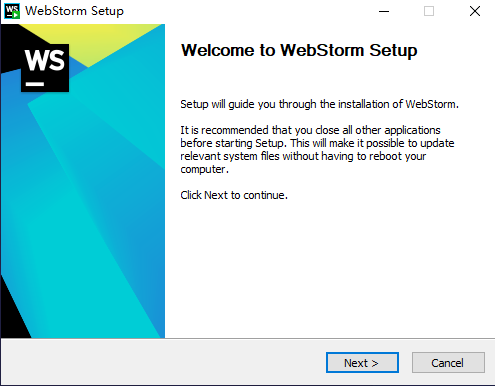
# 安装前端开发工具

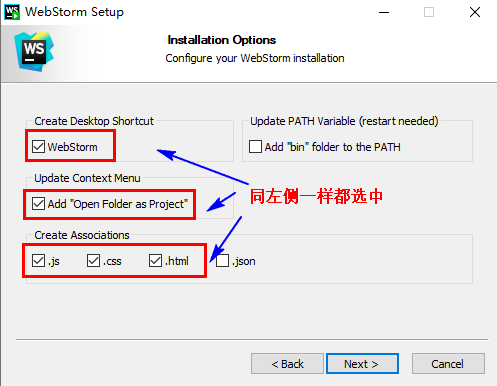
前端工具使用自己的熟悉就可以。 常用的WebStorm ; Visual Studio Code(VSCode) ; HBuilder等. 我这里使用的是WebStorm

## 安装WebStorm

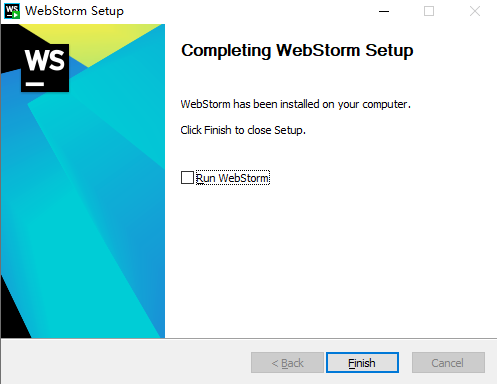
编写文档时WebStorm最新版本是2021.3（其他版本也可以）

WebStorm安装和Idea一样。 下一步，下一步就可以。





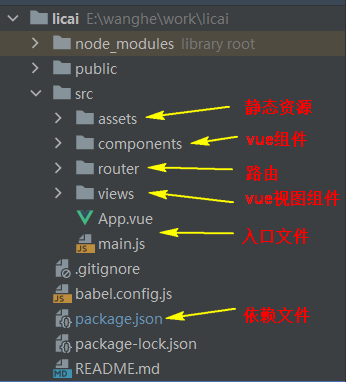
以后点击next 完成安装。



## 使用WebStorm打开项目



## 项目的目录结构



# 开发Vue前端工程

## 整理项目文件

### 删除不需要的代码 App.vue



删除后 App.vue

<template>  
 <div id="nav"></div>  
 <router-view/>  
</template>  
<style>  
</style>

### views/Index.vue

views目录下创建Index.vue ,作为首页

<template>  
 <div>  
 Index  
 </div>  
</template>  
  
<script>  
export default {  
 name: "Index"  
}  
</script>  
  
<style scoped>  
  
</style>

### 修改首页路由

router/index.js 把原来Home的配置修改为 Index

import Index from "../views/Index";  
  
const routes = [  
 {  
 path: '/',  
 name: 'Index',  
 component: Index  
 },

运行程序， 首页看到Index文字

删除 views/Home.vue

## 添加router配置

创建项目没有加入router， 可以使用命令 vue add router 增加router功能

## 创建vue配置文件

项目根目录下创建vue.config.js文件

module.exports = {  
 devServer: {  
 port:8080,  
 proxy: {  
 '/api': {  
 target: 'http://localhost:9000/api',  
 ws: true,  
 changeOrigin: true  
 }  
 }  
 }  
}

## 盈利宝加入静态资源

项目使用的自定义的css，js，html等静态资源加入到项目的assets目录，位置如图

拷贝课件资源文件夹\盈利宝-页面原型\盈利宝pc\public 下所有内容（不需要js目录

）到 asserts 目录

## main.js

//导入全局css  
import '@/assets/css/details.css'  
import '@/assets/css/font-awesome.min.css'  
import '@/assets/css/index.css'  
import '@/assets/css/list.css'  
import '@/assets/css/login.css'  
import '@/assets/css/public-head.css'  
import '@/assets/css/reset.css'  
import '@/assets/css/swiper.css'  
import '@/assets/css/user\_center.css'  
import '@/assets/css/user\_pay.css'

***createApp***(App).use(***router***).mount('#app')

const app = ***createApp***(App)

## md5

npm install js-md5 --save

要使用的文件，import md5 from 'js-md5';

例子：

let m = md5("hello");

console.log("md5==="+m)

计算结果：

md5===5d41402abc4b2a76b9719d911017c592

## UUID

UUID：<https://www.npmjs.com/package/uuid>

npm install uuid

使用：

import { v1 as uuidv1 } from 'uuid';

let a = uuidv1();

console.log("a="+a)

## 配置axios

安装axios , 项目根目录下执行： npm i axios --save

安装 qs , 项目根目录下执行： npm i qs --save

## 安装 layer

layer：弹层（提示框）

https://www.npmjs.com/package/vue-layx

使用文档

<http://chuange.gitee.io/vue-layx/>

安装：

npm i vue-layx -S

配置：

import layx from "vue-layx"

例子1：

普通消息窗口

layx.msg('功能正在完善中');

成功图标的消息窗口

layx.msg('注册成功',{dialogIcon:'success'});

例子2：

带成功图标提示窗口

layx.alert('投资提示','投资成功',null,{dialogIcon:'success'});

警告提示窗口

layx.alert('投资提示','你要投资100元',null,{dialogIcon:'warn'});

错误提示窗口

layx.alert('登录','手机号或密码错误',null,{dialogIcon:'error'});

例子3：

layx.confirm('默认提示','Layx 会做为你下一个项目弹窗的首选吗？',null,{

buttons:[

{

label:'确定',

callback:function(id, button, event){

alert("你点击了确定操作");

layx.destroy(id);

}

},

{

label:'取消',

callback:function(id, button, event){

alert("你点击了取消操作");

layx.destroy(id);

}

}

]

});

## 路由 router

路由功能： component: 加载组件， 组件路径和名称按你的 工程设置

例如：

ProductList.vue ： 分页查询产品页面

path: 访问页面的url地址

import { createRouter, createWebHistory } from 'vue-router'  
import Index from '../views/Index'  
  
const routes = [  
 {  
 path: '/',  
 name: 'Index',  
 component: Index  
 },  
 {  
 path: '/product/list',  
 name: 'ProductList',  
 component: () => import('../views/ProductList.vue')  
 },  
 {  
 path: '/product/detail',  
 name: 'ProductDetail',  
 component: () => import( '../views/ProductDetail.vue')  
 },  
 {  
 path:"/user/register",  
 name:"Register",  
 component: () => import('../views/Register.vue')  
 },{  
 path: '/user/realname',  
 name: 'RealName',  
 component: ()=>import('../views/RealName.vue')  
 },{  
 path:'/user/login',  
 name:"Login",  
 component: ()=> import('../views/Login.vue')  
 },{  
 path:'/user/center',  
 name:"UserCenter",  
 component: ()=> import('../views/UserCenter.vue')  
 },{  
 path:'/user/recharge',  
 name:"UserPay",  
 component: ()=> import('../views/UserPay.vue')  
 }  
  
]  
  
const ***router*** = createRouter({  
 history: createWebHistory(***process***.env.BASE\_URL),  
 routes  
})  
  
export default ***router***

路由使用的主要方法：

1 跳转到其他页面

this.$router.push( { path:'/product/detail'，query:{ 'productId':131699 }

} )

product/detail 对应的页面获取参数

mounted() {  
 let productId = this.$route.query.productId;

}

1. 获取当前页面访问url

let currentUrl = this.$route.path

## 首页功能

修改Index.vue文件。

1. 修改图片 public位置替换为 @/assets

<img src="@/assets/image/2-icon5.png" alt="">

1. 创建数据结构

let ***indexDataObject*** = {  
 registerUser:100,  
 avgRate:5.6,  
 sumMoney:90000,  
 xinList:[{  
 id:1001,  
 name:'新手宝',  
 rate:'5.6%',  
 cycle: 1,  
 leftMoney:1000,  
 status:0,  
 type:1,  
 minMoney:0,  
 maxMoney:0  
 }],  
 sanList:[{  
 id:1002,  
 name:'散标',  
 rate:'5.6%',  
 cycle: 1,  
 leftMoney:1000,  
 status:0,  
 type:1,  
 minMoney:0,  
 maxMoney:0  
 }],  
 youList:{  
 id:1003,  
 name:'优选',  
 rate:'5.6%',  
 cycle: 1,  
 leftMoney:1000,  
 status:0,  
 type:1,  
 minMoney:0,  
 maxMoney:0  
 }  
}

1. 完整vue

<template>  
 <div>  
 <Header></Header>  
 <!--banner-->  
 <div class="banner-content">  
 <div class="swiper-container banner-lb">  
 <div class="swiper-wrapper">  
 <div class="swiper-slide">  
 <a href="javascript:;" >  
 <img src="../assets/image/banner.jpg" alt="">  
 </a>  
 </div>  
 <div class="swiper-slide">  
 <a href="javascript:;" >  
 <img src="../assets/image/banner.jpg" alt="">  
 </a>  
 </div>  
 </div>  
 </div>  
 <div class="banner-abs">  
 <div class="banner-abs-box">  
 <div class="banner-abs-title">动力金融网历史年化收益率</div>  
 <b>{{***indexData***.avgRate}}<i>%</i></b>  
 <p>平台用户数</p>  
 <span>{{***indexData***.registerUser}}<i>位</i></span>  
 <p class="banner-abs-border">累计成交金额</p>  
 <span>{{ indexData.sumMoney }}<i>元</i></span>  
 </div>  
 </div>  
 </div>  
 <div class="banner-list">  
 <ul>  
 <li>  
 <img src="../assets/image/banner-tag1.png" alt="">  
 <p>  
 <b>实力雄厚</b>  
 <span>亿级注册资本 ,千万技术投入</span>  
 </p>  
 </li>  
 <li>  
 <img src="../assets/image/banner-tag2.png" alt="">  
 <p>  
 <b>风控严苛</b>  
 <span>30道风控工序，60项信用审核</span>  
 </p>  
 </li>  
 <li>  
 <img src="../assets/image/banner-tag3.png" alt="">  
 <p>  
 <b>投资省心</b>  
 <span>资金安全风控，银行安全托管</span>  
 </p>  
 </li>  
 </ul>  
 </div>  
  
 <!--产品-->  
 <div class="content" v-for="prod in ***indexData***.xinList" :key="prod.id">  
 <h2 class="public-title"><span>{{ prod.name }}</span></h2>  
 <div class="new-user">  
 <div class="new-user-sm">  
 <span>{{ prod.minMoney }}元起投</span>  
 <span>投资最高限额{{ prod.maxMoney }}元</span>  
 <span>当日即投</span>  
 </div>  
 <div class="new-user-number">  
 <ul>  
 <li>  
 <p><b>{{ prod.rate }}</b>%</p>  
 <span>历史年化收益率</span>  
 </li>  
 <li>  
 <p><b>{{prod.cycle}}</b>天</p>  
 <span>投资周期</span>  
 </li>  
 <li>  
 <p><b>{{ prod.leftMoney }}</b>元</p>  
 <span>余利可投资金额</span>  
 </li>  
 </ul>  
 <a href="javascript:;" class="new-user-btn" @click="goLink('/product/detail',{'productId': prod.id})">立即投资</a>  
 </div>  
 <span class="new-tag">新用户专享</span>  
 </div>  
  
 <h2 class="public-title"><span>优选产品</span> <a href="javascript:void(0)" class="public-title-more" @click="goLink('/product/list',{productType:1})">查看更多产品>></a></h2>  
 <ul class="preferred-select clearfix">  
 <li v-for="you in ***indexData***.youList" :key="you.id">  
 <h3 class="preferred-select-title">  
 <span>{{ you.name }}</span>  
 <img src="../assets/image/1-bg1.jpg" alt="">  
 </h3>  
 <div class="preferred-select-number">  
 <p><b>{{you.rate}}</b>%</p>  
 <span>历史年化收益率</span>  
 </div>  
 <div class="preferred-select-date">  
 <div>  
 <span>投资周期</span>  
 <p><b>{{you.cycle}}</b>个月</p>  
 </div>  
 <div>  
 <span>余利可投资金额</span>  
 <p><b>{{you.leftMoney}}</b>元</p>  
 </div>  
 </div>  
 <p class="preferred-select-txt">  
 优选计划项目，投资回报周期{{you.cycle}}个月，起点低，适合短期资金周转、对流动性要求高的投资人。  
 </p>  
 <a href="javascript:;" class="preferred-select-btn" @click="goLink('/product/detail',{'productId': you.id})">立即投资</a>  
 </li>  
 </ul>  
  
 <h2 class="public-title"><span>散标产品</span> <a href="javascript:void(0)" @click="goLink('/product/list',{productType:2})" class="public-title-more">查看更多产品>></a></h2>  
 <ul class="preferred-select clearfix">  
 <li v-for="san in ***indexData***.sanList" :key="san.id">  
 <h3 class="preferred-select-title1">{{san.name}}  
 <span>散标</span>  
 </h3>  
 <div class="preferred-select-number">  
 <p><b>{{san.rate}}</b>%</p>  
 <span>历史年化收益率</span>  
 </div>  
 <div class="preferred-select-date">  
 <div>  
 <span>投资周期</span>  
 <p><b>{{san.cycle}}</b>个月</p>  
 </div>  
 <div>  
 <span>余利可投资金额</span>  
 <p><b>{{ san.leftMoney }}</b>元</p>  
 </div>  
 </div>  
 <p class="preferred-select-txt">  
 优选计划项目，投资回报周期{{san.cycle}}个月，起点低，适合短期资金周转、对流动性要求高的投资人。  
 </p>  
 <a href="javascript:void(0);" class="preferred-select-btn" @click="goLink('/product/detail',{'productId': san.id})">立即投资</a>  
 </li>  
  
 </ul>  
  
 </div>  
  
 <!--说明-->  
 <div class="information-box">  
 <ul>  
 <li>  
 <img src="../assets/image/2-icon1.png" alt="">  
 <p>优质借款</p>  
 <span>严苛风控，多重审核</span>  
  
 </li>  
 <li>  
 <img src="../assets/image/2-icon2.png" alt="">  
 <p>次日计息</p>  
 <span>闪电成交，谁比我快</span>  
 </li>  
 <li>  
 <img src="../assets/image/2-icon3.png" alt="">  
 <p>全年无休</p>  
 <span>百万用户，一路同行</span>  
 </li>  
 <li>  
 <img src="../assets/image/2-icon4.png" alt="">  
 <p>知心托付</p>  
 <span>百万用户，一路同行</span>  
 </li>  
 <li>  
 <img src="../assets/image/2-icon5.png" alt="">  
 <p>技术保障</p>  
 <span>千万投入，专注研发</span>  
 </li>  
 </ul>  
 </div>  
  
 <!--公共底部-->  
 <Footer></Footer>  
 </div>  
  
</template>  
  
<script>  
import ***axios*** from 'axios'  
import Header from "@/views/common/Header";  
import Footer from "@/views/common/Footer";  
  
let ***indexDataObject*** = {  
 registerUser:100,  
 avgRate:5.6,  
 sumMoney:90000,  
 xinList:[{  
 id:1001,  
 name:'新手宝',  
 rate:'5.6%',  
 cycle: 1,  
 leftMoney:1000,  
 status:0,  
 type:1,  
 minMoney:0,  
 maxMoney:0  
 }],  
 sanList:[{  
 id:1002,  
 name:'散标',  
 rate:'5.6%',  
 cycle: 1,  
 leftMoney:1000,  
 status:0,  
 type:1,  
 minMoney:0,  
 maxMoney:0  
 }],  
 youList:{  
 id:1003,  
 name:'优选',  
 rate:'5.6%',  
 cycle: 1,  
 leftMoney:1000,  
 status:0,  
 type:1,  
 minMoney:0,  
 maxMoney:0  
 }  
}  
  
export default {  
 name: "Index",  
 components:{  
 "Header":Header,  
 "Footer":Footer  
 },  
 data(){  
 return {  
 indexData:***indexDataObject*** }  
 },  
 mounted() {  
 ***axios***.get('http://localhost:8000/ylb/index?time='+new ***Date***()).then(res=>{  
 this.***indexData*** = res.data.data.object;  
 ***console***.log(res.data.data.object)  
 })  
 },  
 methods:{  
 goLink(linkUrl,param){  
 ***console***.log('goLink',param);  
 this.$router.push( { path:linkUrl, query:param })  
 },  
 userInvest(){  
 this.$router.push( { path:'/product/detail'} )  
 }  
 }  
}  
</script>  
  
<style scoped>  
  
</style>

## 创建公共组件

创建页头，页脚Header ，Footer公共组件

在views目录下创建common， 分别创建Footer.vue , Header.vue

Header.vue

<template>  
 <!--头部-->  
 <div class="public-head">  
 <div class="public-head-nav">  
 <div class="public-head-left">  
 <h1 class="public-head-logo"><a href="javascript:;">  
 <img src="../../assets/image/logo.png" alt="">  
 </a></h1>  
 <ul class="public-head-list">  
 <li><a href="javascript:void(0);" @click="goLink('/')">主页</a></li>  
 <li class="public-head-hover">  
 <a href="javascript:void(0);">我要投资</a>  
 <!--二级导航-->  
 <div class="two-title">  
 <a href="javascript:;" @click="goLink('/product/list',{productType:1})">优先类产品</a>  
 <a href="javascript:;" @click="goLink('/product/list',{productType:2})">散标类产品</a>  
 </div>  
 </li>  
 <li><a href="javascript:void(0);alert('系统正在维护')">理财信息</a></li>  
 <li><a href="javascript:;" target="\_blank">信息披露</a></li>  
 <li><a href="javascript:;" target="\_blank">安全计划</a></li>  
 </ul>  
 </div>  
  
 <template v-if="isLogin" class="public-head-list">  
 <div class="public-head-right">  
 <a href="javascript:void(0);" @click="goLink('/user/login',{})">登录</a>  
 <a href="javascript:void(0);" @click="goLink('/user/register',{})">注册</a>  
 </div>  
 </template>  
  
 <template v-else>  
 <div class="public-head-right">  
 <a href="javascript:void(0);" @click="goLink('/user/center',{})">用户中心</a>  
 <a href="javascript:void(0);" @click="goLink('/user/recharge',{})">充值</a>  
 </div>  
 </template>  
 </div>  
 </div>  
 <!--end-->  
</template>  
  
<script>  
export default {  
 name: "Header",  
 data() {  
 return {  
 isLogin: true  
 }  
 },  
 methods: {  
 goLink(linkUrl, param) {  
 ***console***.log('goLink', param);  
 this.$router.push({path: linkUrl, query: param})  
 },  
 userLogin() {  
 let currentUrl = this.$route.path  
 this.$router.push({path: '/user/login', query: {targetUrl: currentUrl}})  
 }  
 }  
}  
</script>  
  
<style scoped>  
  
</style>

Footer.vue

<template>  
 <!--公共底部-->  
 <div class="public-bottom">  
 <div>  
 <p>版权所有 ©2017 北京动力节点教育科技有限公司 京ICP备09027468号 | 京公网安备京公网安备11030808027468</p>  
 <p>一家只教授Java的培训机构<a href="javascript:;">Visit the HomePage</a></p>  
 </div>  
 </div>  
</template>  
  
<script>  
export default {  
 name: "Footer"  
}  
</script>  
  
<style scoped>  
  
</style>

## 产品详情

## 更多产品

## 注册用户

增加样式：

.err {  
 display: inline;  
 color: red;  
}

增加错误提示组件

<div id="phoneErr" class="err">{{ phoneErr }}</div>

<div id="passworErr" class="err">{{ passwordErr }}</div>

<div id="codeErr" class="err">{{ codeErr }}</div>

短信验证码

<div class="form-yzm form-border">  
 <input class="yzm-write" type="text" @click="checkCode" v-model="code" name="" placeholder="输入短信验证码">  
 <input class="yzm-send" type="text" @click="requestCode" v-bind:isSend="isSend" v-bind:value="yzmButtonText" id="yzmBtn" readonly="readonly" >  
</div>

数据定义：

data() {  
 return {  
 phone: "13899999999",  
 phoneErr: "",  
 password: "",  
 passwordErr: "",  
 code: "",  
 codeErr: "",  
 yzmButtonText:"获取验证码",  
 agree:false,  
 isSend:false  
 }  
},

发送短信验证码：

requestCode() { //请求发送验证码  
  
 if(this.isSend){  
 return;  
 }  
 this.isSend = true;  
  
 let $yzmBtn = ***document***.getElementById("yzmBtn")  
 $yzmBtn.style.color='#FF8888'  
 $yzmBtn.style.cursor="not-allowed"  
  
 let second = 10;  
 let intervalIns = setInterval( ()=>{  
 second = second -1;  
 this.yzmButtonText=second+"秒后从新获取"  
 if( second < 1 ){  
 ***window***.clearInterval(intervalIns)  
 this.yzmButtonText="获取验证码"  
 this.isSend = false;  
 $yzmBtn.style.color='#688EF9'  
 $yzmBtn.style.cursor="pointer"  
 }  
 },1000)

## 登录系统

## 登出系统

## 实名认证

## 用户中心

## 投资产品

## 投资排行榜

## 定时任务服务

## 支付充值

## 支付查询