### TODO： 项目实战或练习，才会记住。

### 一.环境安装

#### 1．1使用vite创建vue项目

|  |
| --- |
| # npm 6.x  npm init vite@latest <project-name> --template vue  # npm 7+, 需要加上额外的 双短横线  npm init vite@latest <project-name> **--** --template vue  cd <project-name>  npm install # 安装依赖。如果不能翻墙，使用国内淘宝等镜像  npm run dev # 运行 |

#### 1.2 项目结构



src/main.js是入口文件，它会引用src/App.vue

#### 1.3 IDE与插件

安装vs code（ctrl + ` 打开终端）

**安装Volar插件**（语法高亮、代码补全、代码格式化）---已废弃，安装Vue-official插件

安装Chinese中文界面、**Vue 3 Snippets（快速生成基础模板）**

**Vue 3 Snippets具体使用：**

**TODO：**

### 2.安装与使用ElementPlus插件

|  |
| --- |
| **#安装**  **npm install element-plus –save**  **#main.js中引入**    **# vue文件中使用element-plus中的具体组件，比如：button**      <el-button type="primary">Primary</el-button> |

### 3.使用css库

tailwind库

### 4.使用vue-router

4.1 路由

什么是路由？在前端页面上，当点击菜单栏等元素，让浏览器地址发生了变化。



什么是哈希模式？

哈希模式是利用 URL 中的哈希值（**即 URL 中#后面的部分**）来实现页面的导航和状态管理。**哈希值不会被包含在向服务器发送的请求中**，仅用于浏览器端的页面操作和状态记录。比如，https://example.com/page#section2中，#section2就是哈希值。

|  |
| --- |
| #1.安装（vue3使用vue-router 4.0以上版本）  npm install vue-router@4  #2. src下新建router目录，并新建index.js, 内容：  // 导入 创建哈希路由模式 和 创建路由 函数  import {createWebHashHistory, createRouter} from 'vue-router'  // myLoginView名字是本文件里自定义的  import myLoginView from '../page/login.vue' // 创建自己的页面  import NotFound from '../page/404.vue'  // 定义路由表  const routes = [      {          path: "/login", // 因为是哈希模式。所以用host:port/#/login访问          component: myLoginView,      } ,      {          path: '/:pathMatch(.\*)\*',          name: 'NotFound',          component: NotFound      },  ]  // 创建路由  const router = createRouter({      history: createWebHashHistory(), // 函数调用      routes  })  // 导出路由对象/实例  export default router  // 错误写法。下面是导出一个router对象的属性  // export default {  //     router // router:router的简写  // }  # 3. main.js 引用router  import router from './router/index.js'  const app = createApp(App)  app.use(router)  #4. App.vue使用 router-view  <template>   <!-- 路由视图是用来显示 当前的路由，即route中的path对应的component -->      <router-view></router-view>  </template>  #5.浏览器访问：http://localhost:5173/#/login |

4.2对src设置别名

原因：在引文文件等时候，大部分文件是在src下。起别名后，方便编程。

|  |
| --- |
| # 参考vite官方文档：#https://cn.vite.dev/config/sharedoptions.html#resolve-alias  # vite.config.js配置如下：  import path from 'path' // node中的path库  export default defineConfig({    resolve: {      alias: {        //将xxx/xxx/src设置别名为~        "~": path.resolve(\_\_dirname, "src")      }    }  }) |

### 三、登录功能

3.1 Layout布局

el-row: 将屏幕(或者元素的width) 分别24列.

er-col: 通过:span指定占用少列

|  |
| --- |
| <**el-row**>  <**el-col** **:span="12**">  <div class="grid-content ep-bg-purple" />  </el-col>  <**el-col :span="12"**>  <div class="grid-content ep-bg-purple-light" />  </el-col>  </el-row> |

3.2. flex布局

3.3 Element的 input组件中使用图标

参考：

https://element-plus.org/zh-CN/component/input.html#%E5%B8%A6%E5%9B%BE%E6%A0%87%E7%9A%84%E8%BE%93%E5%85%A5%E6%A1%86

|  |
| --- |
| # 安装图标资源  npm install @element-plus/icons-vue  # <srcipt>中导入指定图标  import { Lock, Search } from '@element-plus/icons-vue'  # input组件内使用图标 ( 通过插槽方式 )                          <el-input v-model="form.name">                              <template #prefix>                                  <el-icon class="el-input\_\_icon">                                      <search />                                  </el-icon>                              </template>                          </el-input> |

效果展示：



3.4 setup语法糖

<script setup> 中的代码会在 每次组件实例被创建的时候执行。

3.5 登录表单验证处理

表单静态验证：

|  |
| --- |
| #1. 表单设置rules属性  <el-form :model="form" ref="formRef" **:rules="rules"** >  #2. 设置rules规则  const rules = {      // key要和表单 const form =reactive(...)定义的一样   // 比如，rules里的name 和 from里定义的name 名字一样      name: [          // trigger: 'blur'，失去焦点          { required: true, message: '用户名不能为空', trigger: 'blur' },          { min: 3, max: 5, message: '用户名长度3-5位', trigger: 'blur' },      ],      pwd: [          { min: 6, max: 10, message: '密码6到10位', trigger: 'blur' },      ]  }  #3. form-item设置 prop属性，要和 rules里定义的key一样    <el-form-item label="密码" **prop="pwd"**>      <!-- 方法二： 使用 : prefix-icon 定义图标 -->      <el-input v-model="form.pwd" :prefix-icon="Lock" />    </el-form-item> |
|  |

总之： prop属性值、rules里定义的key，都要和 定义form表单这个reactive对象里的字段保持一致。

表单提交时验证：

|  |
| --- |
| #1. 设置表单的引用  <el-form :model="form" **ref="formRef"** :rules="rules" >  #2. 在提交中判断  const formRef = ref(null)  const onSubmit = () => {      formRef.value.validate(boolResult => {          if (!boolResult) {              console.log("验证失败")              return          }          console.log("验证成功")      })      console.log('submit!')  } |

3.6使用axios库调用登录接口

|  |
| --- |
| 1. 安装: npm install axios  2. 导入： import axios from ‘axios’  3. 创建实例  const instance = axios({      baseURL: 'http://127.0.0.1:8989',      timeout: 3000  })  4. 发送请求：  instance.post(  “/admin/login”, // url  {username:”123”, pwd:”456”} // data  ) |

注：axios.post是返回一个 promise对象. 用.then(res=>{}) 处理返回值,用.catch(err=>{}) 处理错误信息。

3.7 登录成功使用router.push(“/”)跳转首页

|  |
| --- |
| 1. 导入并初始化：  import { useRouter } from 'vue-router'  const router = useRouter()  2. 跳转  router.push(“/”) |

登录成功、失败后，使用EINotification提示框提示

|  |
| --- |
| 1. import {ElNotifcation} from ‘element-plus’ 2. ElNotification({   title: '错误',  message: '登录失败',  type: 'error',  duration: 2500  }) |

3.8 使用cookie保存token

|  |
| --- |
| 1. 安装： npm install vue3-cookies  2. 导入并初始化：  import {useCookies} from 'vue3-cookies'  const {cookies} = useCookies() 3. 存cookie  cookies.set(“token”, res.data.data.token) |

3.9使用axios拦截器

**请求拦截器**： 在发送请求前做点东西，比如：每个请求都需要在header放token，那么将这个逻辑统一放到请求拦截器中

响应拦截器： 在返回真正的响应前做点什么。比如：vue本身返回的结构体嵌套太深，可以直接返回response.data, 而不是返回response； 响应报错了，统一弹提示框等等。

注： 代码都可在axios官网上找到。

|  |
| --- |
| // 添加请求拦截器  instance.interceptors.request.use(function (config) {      // 在发送请求之前做些什么      // 在发送请求之前，将cooke中的token添加到请求头中      const token = cookies.get('token')      if (token) {          config.headers.Authorization = `Bearer ${token}`      }      return config;    }, function (error) {      // 对请求错误做些什么      return Promise.reject(error);    });  // 添加响应拦截器  instance.interceptors.response.use(function (response) {      // 2xx 范围内的状态码都会触发该函数。      // 对响应数据做点什么      return response.data;  // 返回服务器的响应数据，业务代码不再用response.data.data了    }, function (error) {      // 超出 2xx 范围的状态码都会触发该函数。      // 对响应错误做点什么      // 对响应错误做点什么： 弹出错误信息      ElNotification({          message: error.response.data.msg || '请求失败',          type: 'error',          duration: 3000      })      return Promise.reject(error);    }); |

3.10 封装通用函数

|  |
| --- |
| 1.将弹窗消息封装到 common/util.js 中    2. 将token操作封装到 common/cookie.js中  import {useCookies} from 'vue3-cookies'  const {cookies} = useCookies()  const TOKEN\_KEY = 'token'  export function setToken(value) {      cookies.set(TOKEN\_KEY, value)  }  export function getToken() {      return cookies.get(TOKEN\_KEY)  }  export function removeCookie() {      cookies.remove()  } |

3.11使用vuex进行状态管理

使用时机： 多个组件共享读取数据时，需要使用vuex进行管理【相当于**全局单例模式**】

|  |
| --- |
| 1. 安装   npm install vuex@next –save   1. 新建store/index.js   import { createStore } from "vuex"  const store = createStore({  // 固定叫state，相当于全局变量  **state**: {  count: 1  },  mutations: {  // 第一个参数固定是state  increment1 (**state**) {  // 变更状态  state.count++  }  // 也可以定义一个传参进来  increment2 (**state**, value) {  // 变更状态  state.count +=  }  }  })  export default store   1. main.js使用store   import store from ‘./store’  app.use(store)   1. vue文件中script修改共享变量   import {useStore} from ‘vuex const store = useStore()  store.commit(“increment2”) store.commit(“increment2”, 20)   1. vue文件中<template>访问共享变量   {{$store.**state**.user}} |

注： 不管访问还是修改，state是枢纽，不要忘记； 这个store的整体流程和router很像

3.12 router的全局前置守卫

用途： 在跳转到某个路由时，先执行下这个beforeEach前置守卫。 那么就可以通过跳转、取消跳转、重定向。

返回值：

false： 取消导航，回到from处

undefined或true： 正常跳转到to处

{name: ‘/login’}: 重定向导航（**from不变, 会再触发执行beforeEach**）

|  |
| --- |
| router.beforeEach((to, from) => {      const token = getToken()      // 1.token不存在 并且 目标不是登录页,则跳转登录页.  一定要检查to.path,防止无限重定向      if (!token && to.path !== "/login") {          return {path: '/login'}      }      // 2.token存在，用户还想到登录页登录，则取消跳转      if (token && to.path === "/login") {          return false      }      // 3.返回undefine或者true,表示正常跳转到to上  }) |

3.13 vuex状态管理优化

状态管理2大特性：

1.响应式：状态的变化（即状态数据，即store.state中的数据）会自动触发Vue组件重新渲染。

2.刷新页面数据丢失： 数据存在内存中，访问速度快，但同时刷新页面时，数据（状态数据，即store.state中的数据）会丢失。

解决刷新时用户信息丢失问题。登录成功后，将用户信息放入到localstorage中（只能放字符串，要json.stringfy下），然后在路由前置守卫中，从 Vuex 状态管理或本地存储中获取当前用户信息后， 再更新Vuex状态管理。

|  |
| --- |
| // 第一种方式报错： Cannot read properties of undefined (reading 'state'). 原因： useStore() 是一个组合式 API 钩子函数，只能在setup中使用， 不能在模块顶层使用(因为此时vuex还没有初始化)  // import {useStore} from 'vuex'  // const store = useStore()  // 第二种方式报错：The requested module '/src/store/index.js' does not provide an export named 'store'。 用{store}这种方式是命名导出。而vue项目是用默认导出 (default export)  // import {store} from '../store'  // 第三种方式不报错  import store from '../store'  router.beforeEach((to, from) => {      // 保存状态管理的数据一直存在(状态管理数据放在内存中,因此访问速度快,同时状态管理数据也是响应式的)      let user = (store.state.user)  || JSON.parse(localStorage.getItem('user'))      if (user && !store.state.user) {          store.commit('setUserInfo', user)  }  // ….  } |

3.14 登录页按下回车就发登录请求

|  |
| --- |
| function handlerKeyDown(e) {      console.log("handlerKeyDown , e:", e)      if (e.key == "Enter") {          onSubmit()      }  }  onMounted(() => {      console.log('组件已经挂载到页面了（即DOM渲染完成了），可以操作DOM或发请求')      document.addEventListener('keydown', handlerKeyDown)  })  onUnmounted(() => {      // 页面销毁时，放移除，防止内存泄漏.      // document表示本vue组件，windows表示整个窗口。只要在登录页面监听用户按回车后，执行登录操作就行      document.removeEventListener('keydown', handlerKeyDown)  }) |

生命周期顺序： setup、 onMounted

因此： setup是在onMounted之前执行，而onMounted表示vue组件加载完成(即dom渲染完成)， 所以在setup内不能操作dom。 setup就是定义ref响应式变量、函数定义等。

3.14退出登录

|  |
| --- |
| login.vue  import { showModal, showSuccessMessage } from '../common/util';  import { logout } from '../api/api';  import { useRouter } from 'vue-router'  import { useStore } from 'vuex'  const router = useRouter()  const store = useStore()  const logoutBtn = () => {      showModal("确定要退出登录吗？")          .then(res => {              //调用退出登录接口,不管成功失败,都走finally逻辑              logout().finally(() => {                  //1. 清除cookie、清除用户状态                  // store.dispatch本身是同步的，它会触发store中action，action通常是异步的，但是也可以放同步函数.                  store.dispatch("logout")                  // 3. 跳转到首页                  router.push("/login")                  // 4. 提示退出成功                  showSuccessMessage("退出登录成功")              })          })          .catch(() => {              // 点击"取消",啥也不干          })  }  消息提示框：  // 消息提示框  export function showModal(content, title = "") {      // 因为点击"确认"、"取消"会分别执行then、catch回调，所以要将对象return出去      return ElMessageBox.confirm(          content,          title,          {              confirmButtonText: '确认',              cancelButtonText: '取消',          }      )  }  vuex状态:  const store = createStore({      state:{          user: null // 有效数据的格式是对象. 初始化为null，表示没有登录，方便判断（否则要判断为一个非空对象）      },      // mutations是同步操作,使用者调用store.commit函数触发      mutations: {          setUserInfo(state, user) {              state.user = user          }      },      // actions内是放异步操作，使用者通过store.dispatch函数触发. 但是action可以放同步的操作.      // 异步操作 指的是需要等待结果返回的非阻塞操作，例如发送 HTTP 请求      actions: {          logout({commit}) {              // 清除cookie              removeCookie()              // 清除用户状态              commit("setUserInfo", {})          }      }  }) |

3.15 nprogress: 页面顶部进度条 （<https://www.npmjs.com/> ，js各种包的开源库）

|  |
| --- |
| 1. 安装： npm install nprogress  2. 主要功能：  在页面顶部显示一个细细的进度条  支持手动控制进度开始、结束、递增  支持配置样式（颜色、高度等）   1. 纯js基本使用：   import nprogress from 'nprogress';  import 'nprogress/nprogress.css'; // 引入样式  // 开始进度条  NProgress.start();  // 模拟加载，3秒后完成  setTimeout(() => {  nprogress.done();  }, 3000);   1. vue 使用：   import nprogress from 'nprogress';  import 'nprogress/nprogress.css'; // 要导入安装的样式。可放在mian.js  router.beforeEach((to, from,) => {  nprogress.start();  });  router.afterEach(() => {  nprogress.done();  }); |

### 四.后台layout全局

4.1 后台layout主布局

|  |
| --- |
| 1.使用el-container布局容器 |
| 2.修改路由：    1.App.vue是整个应用的入口组件，其中的<router-view>是最顶层的路由视图  2.当访问/login时，不会经过layoutIndex组件，而是直接在App.vue的<router-view>中渲染myLoginView  3.流程  访问/login的渲染层次：  App.vue  └── <router-view> (渲染myLoginView)    访问/的渲染层次：  App.vue  └── <router-view> (渲染layoutIndex)  └── layoutIndex的<router-view> (渲染myIndexView) |

4.2 Header组件布局

|  |
| --- |
| 1.两边居中  display: flex;  justify-content: **space-between**;  2.flex布局下的各个子元素间距  .global-header-left {  display: flex;  align-items: center;  **gap: 10px;**  //gap: 用于设置 Flex 子项之间的间距，避免手动给每个子项加 margin-right 或类似的样式，更方便、语义更清晰。  }  3.fixed固定  position: fixed; // 脱离文档流(不占据原本的位置，对其他元素没有影响，其他元素也不会为它腾地方，所以称为fixed固定。)  top:0; // 盒子模型，盒子上方的top是0  left: 0;  right: 0;  4.鼠标悬停时显示手型，并且更改背景颜色  .g-h-l-icon {  font-size: 20px;  margin-left: 20px;  height: 50px;  width: 50px;  cursor: pointer; // 鼠标悬停时显示手型  &:hover { // 鼠标悬停时，背景颜色变为深蓝色，会将width和height区域的背景颜色改掉，所以设置了height  background-color: #3a8fc0;  }  } |

4.3刷新、全屏按钮功能

|  |
| --- |
| 1.刷新  const handleRefresh = () => {  window.location.reload()  }  2.全屏切换  2.1 安装useVue： npm i @vueuse/core  2.2 导入  import { useFullscreen } from '@vueuse/core'  const { isFullscreen, toggle } = useFullscreen()  2.3 使用  <el-icon class="g-h-r-icon" @click="handleFullscreen">  <!-- 目前是非全屏状态，则显示全屏图标，否则显示已在全屏的图标 -->  <FullScreen v-if="!isFullscreen" />  <Aim v-else />  </el-icon>  const handleFullscreen = () => {  toggle()  } |

1.isFullscreen判断是否全屏。 toggle是自动切换函数，如果处于非全屏，调用toggle就会进入全屏。如果是全屏状态，调用toggle进入非全屏.

2. useVue中有很多插件使用

|  |
| --- |
| 2.4 使用计算属性  <!-- store也是响应式变量,如果退出登录时，将store.state.user设置为null，那这里机会报错. 所以用 计算属性 优雅判断下 -->  <!-- {{ $store.state.user.avatar }} -->  {{ username }}  <script setup>  const username = **computed**(() => {  if (store.state.user) {  return store.state.user.username  }  return ""  })  </script> |

* 1. 修改密码

|  |
| --- |
| 1. 使用Drawer抽屉，展示修改密码的页面   <el-drawer v-model="showDrawer" title="修改密码哦" **:**close-on-click-modal="false">  // 这里放表单  </el-drawer>   1. 展示效果： |

知识点：

1. Drawer 拥有和 Dialog 几乎相同的 API, 在 UI 上不一样。 Drawer是 呼出一个临时的侧边栏.
2. close-on-click-modal是一个布尔属性，为false时表示 阻止点击遮罩层时关闭抽屉
3. : 是 v-bind 指令的缩写，用于将数据绑定到 HTML 属性上。是否需要加 : 取决于你传递的值是静态字符串还是动态表达式。举例：

3.1不需要加 : 的情况：当传递的值是一个固定的字符串时，可以直接写，如 title="修改密码哦"。

3.2需要加 : 的情况：当传递的值是一个 JavaScript 表达式（各种变量、计算结果、函数返回值等）时，必须使用 :，如 :close-on-click-modal="false"

4.5弹框表单组件化(即drawer侧边栏)

|  |
| --- |
| 0.组件代码组织和使用    1.布局 ： 提交按钮在底部，内容占据其它空间，如果内容长度超过一定值，则显示滑动条。  <div class="main">  <div class="content">  <slot></slot>  </div>  <div class="footer">  <el-button type="primary" @click="submit">{{ submitText }}</el-button>  </div>  </div>  // 父元素使用flex布局  .main {  height: 100%;  width: 100%;  // background-color: red;  display: flex;  flex-direction: column;  }  // 子元素1  .content {  // **flex: 1,即自动占据父容器剩余的所有空间。 前提是父容器一定要是flex布局**  // 注： 容器剩余的，也就是父容器下的其他所有子元素空间去掉后，这里就是去掉footer占据的空间  **flex: 1;**  // 元素在y方向超过容器长度，则显示滚动条  **overflow-y: auto;**  }  // 子元素2  .footer {  height: 50px;  }  2.defineProps、 defineExpose、defineEmits  defineProps：暴露属性，参数是对象（则父组件可以设置属性值了）  defineExpose：暴露方法，参数是对象（则父组件可以调用组件的方法了）  defineEmits: 定义事件，参数是list (则父组件可以监听事件并执行对应回调函数)  3.组件使用(设置prop属性，调用expose的方法、监听emit事件执行回调)  <form-drawer ref="formDrawerRef" size="45%" @submitEmit="onUpatePwdSubmit" submitText="确认修改">  <el-form :model="form" ref="formRef" :rules="rules" label-width="auto"  style="max-width: 600px; margin: 1em auto;">  <el-form-item label="旧密码1" prop="oldPwd">  <el-input v-model="form.oldPwd"></el-input>  </el-form-item>  </el-form>  </form-drawer>  const formDrawerRef = ref(null)  const handleModifyPassword = () => {  formDrawerRef.value.open()  }  const onUpatePwdSubmit = () => { //….} |

这块整体的代码，要记住。很有代表性。

4.5侧边栏菜单

|  |
| --- |
| 1.只有一级菜单(只使用el-sub-menu)  <el-menu class="my-menu" @open="handleOpen" @close="handleClose">      <el-sub-menu index="1">          <template #title>              <span>only一级啦</span>          </template>      </el-sub-menu>  </el-menu>    2.二级菜单  <el-menu class="my-menu" @open="handleOpen" @close="handleClose">      <el-sub-menu index="1">          <template #title>              <span>一级啦</span>          </template>        <el-menu-item index="1-4-1">二级啦</el-menu-item>      </el-sub-menu>  </el-menu>    3.三级菜单  <el-menu class="my-menu" @open="handleOpen" @close="handleClose">      <el-sub-menu index="1">         <template #title><span>一级</span></template>         <el-sub-menu index="1-4">              <template #title>二级</template>              <el-menu-item index="1-4-1">三级</el-menu-item>          </el-sub-menu>      </el-sub-menu>  </el-menu> |

总结：

1. 用el-sub-menu表示子菜单，其尾部有箭头标志； 用el-menu-item表示是最底层的菜单，其尾部没有箭头标志。
2. 如果需要在菜单出增加图标，在<template #title>中添加

|  |
| --- |
| <el-menu-item index="1-4-1">  <**template** #title>  **<el-icon> <location /> </el-icon>**  <span>items One</span>  </template>  </el-menu-item> |

* 1. 使用v-for、v-if构建菜单

|  |
| --- |
| 1. 使用template标签，它是一个 不会被渲染成 DOM 的占位标签，方便操作。 |
| v-if判断是否是带子项的菜单，如果是的话，则用v-for渲染子项菜单(这里默认只有二级菜单), 如果不是的话，那就走到v-else渲染不带子项的菜单 |

4.6动态渲染图标

|  |
| --- |
| <component :is="xxx">， 它是Vue 的动态组件语法  xxx 可以是组件名字符串（如 'ElHomeFilled'）  xxx或者可以是组件对象（如 import { HomeFilled } from '@element-plus/icons-vue'）  所以，<component :is="HomeFilled" />，等价于 <HomeFilled />  1.正确写法：  <el-icon>  <component :is="item.icon"></component>  </el-icon>   1. 错误写法   <el-icon>  {{ item.icon }}  </el-icon>  item.icon 是一个变量，不是一个模板标签，Vue 不会自动解析它为组件 |

4.7点击菜单进行路由跳转

|  |
| --- |
| <el-menu>开启router模式：启用该模式会在激活导航时以 index 作为 path 进行路由跳转, 使用 default-active 来设置加载时的激活项. |

参考文档：

vue-router: <https://router.vuejs.org/zh/>

vuex: https://vuex.vuejs.org/zh/guide/

vue3: https://cn.vuejs.org/guide/introduction

**其他：**

1.如果store目录下有一个index.js文件，那么其他vue文件可以用下面简写的格式：

import {store} from '~/store'

而不必

import {store} from '~/store/index.js'

2. element-plus的所有内容在id=“app”中，它仅次于body下面。





1. 重点关注这3块东西（调试很有用）



第一块： 可以在页面指定看某个元素的信息。在页面选择一个元素后，可以查看它的标签名等等

第二块： styles， 可以看标签的css

第三块： 盒子模型，看宽度、高度、padding、margin等

1. 为什么全局设置box-sizing：border-box？

box-sizing: border-box 是 CSS 中一个非常重要的属性值，它决定了 **元素的宽高是如何计算的**。

一、默认行为（content-box）

box-sizing: content-box; /\* 默认值 \*/

宽高计算方式：

元素的总宽度 = content 宽度 + padding + border

例：

div {

width: 200px;

padding: 20px;

border: 10px solid black;

box-sizing: content-box;

}

实际占据宽度 = 200（内容） + 40（padding 左右） + 20（border 左右） = **260px**

二、推荐用法（border-box）

box-sizing: border-box;

宽高计算方式：

元素的总宽度 = 设置的宽度（包括 padding 和 border）

内容区宽度 = width - padding – border

例：

div {

width: 200px;

padding: 20px;

border: 10px solid black;

box-sizing: border-box;

}

实际占据宽度 = 200px，内容区宽度 = 200 - 40 - 20 = **140px**

**三、使用border-box的原因:**

* **更符合直觉**：设置宽度后，不会因 padding 和 border 撑大盒子。
* **布局更好控**：尤其在响应式布局和栅格系统中，尺寸更容易统一。
* **避免盒子“溢出”容器**的问题。

四、实践：

\* {

box-sizing: border-box;

}