day01

前端 上课|开发 规范

1. 命名不包括以下几种字符
    - 中文
    - 空格
    - 数字开头
    - 特殊字符
    - 前端工程名不要出现大写字母
    - 可以使用 - _
2. 所有源码放置在 **src** 目录下
3. 所有的样式放置在 **css** 目录下
4. 所有的脚本放置在 **js** 目录下
5. 所有的静态资源,例如图片等放置在 **assets** 目录下
6. 几种命名方式
    - **小驼峰** etoakDemoProject
    - **大驼峰** EtoakDemoProject
    - **连字符** etoak-demo-project 连字符格式中绝对不会出现大写字母
7. 在 body 标签内开始,子标签一定要在父标签向右一个制表符(tab),两种规范都是对的,一个制表符相当于 **4 个空格**或者 2 个空格

表单提交时 get 和 post 的区别

- **get**
- 默认提交方式,速度快,毫无安全性,通过**浏览器地址栏传递值**,格式为?key=value&key2=value2&keyN=valueN,不支持中文,最多传递 2000 个字符,链接提交肯定是 get
- **post**
- 速度慢,安全性高,通过**请求的消息体中的载荷传递值**,我们可以通过浏览器的 f12 开发人员工具中的 **network 网络选项卡**查看传值情况,支持中文,没有大小限制,上传操作必须使用 **post**,链接只能使用 get 无法使用 post

---

day01

课程内容

HTML

Hyper Text Markup Language

超文本标记语言

> 前端三大基础之一,用来渲染页面的结构和样式(样式已被css取代),不被列入编程语言,更像是一种规范

> **超文本**:在后端中存在一句话叫做 '**万物皆对象**'，那么在前端页面中有句话则是'**万物皆为超文本**'，超文本是指不局限于文本，还包括页面的图像、音频、视频、图表等一系列要素。

> **标记**:英文称之为tag，更多翻译做标签，由尖括号组成，在html规范中规定使用各种标记（标签）可以渲染页面中的所有超文本，浏览器内置标签解析器与脚本解析器，使用标签渲染之后浏览器从上往下使用标签解析器编译执行，如果书写无误，则浏览器根据用户书写的标签编译之后在页面渲染出相应的结构和样式，这些标签就像乐队的指挥一样告诉浏览器整个页面哪里需要抑扬顿挫

> 诞生于1993年，由w3c世界互联网组织发布了第一版的html规范，专门用来渲染页面的结构和样式（样式于1996年被css技术取代），html规范存在根据浏览器引擎的不同存在差异性，目前已经发展到html5版本。

标签的种类

- 开闭合标签
- 被嵌套的内容
- *文本倾斜</e m>*
- ***文本加粗</ b>***
- ***整合标签***
-
-
- ***标签的属性***
- ***<tagName 属性名="属性值" 属性名="属性值" 属性名="属性值">***
- ***标签与它内部嵌套的内容又被称之为 xxx 元素例如***
- ***文本倾斜</e m> 又被称之为 em 元素(element)***
- ***文本加粗</ b> 又被称之为 b 元素***

CSS

Cascading Style Sheet

层叠样式表

> *前端三大基础之二,用来渲染页面的样式*

> *1996年由w3c发布了第一版的css，目前最新的版本为css4，其中更新最大，使用最多的是css3版本，css诞生的目的是将页面的结构和样式解耦，使两者不再耦合在一起，之前使用html会同时渲染结构和样式导致页面混乱*

```
<del>

  <ins>

•    <em>

•      <*font* face="楷体" size="20px" color="coral">

•        <h2>

•          二级标题

•        </h2>

•      </*font*>

•    </em>

  </ins>

</del>

<!-- 如果结构和样式耦合在一起,则页面冗余标签严重,浏览解析缓慢,毫无可读性,同时也没有实现复用 -->
```

```
selector{

  样式名:样式值;

  样式名:样式值;

  样式名:样式值;

}

/* 后写的如果和之前写的出现冲突,则覆盖 */
```

1-html表单.html

```
<!-- 使用 alt + shift + a 可以直接书写注释
    上课要求
        1:独立完成,不要依赖同位
        2:不要跟着我敲注释,只跟着我敲代码
        3:有问题要问
        4:每天记一条 sql 语句
        5:自己弄一个网络的笔记的软件 印象笔记 notion 有道云笔记

    html 文件以 .html 和 .htm 为后缀结尾,全文不区分大小写,具有一定的
    容错性,如果我们书写了严重的错误,浏览器停止解析,没有报错功能
    以下首行为 DTD 信息 全称(Document Type Definition 文档类型定义)
    用来提示浏览器使用何种语法规范来解析,以下首行为 html5 规范
-->
<!DOCTYPE html>
<!-- html:根标签,在 html 和 xml 中,全文只有一个根标签,其它标签
都被嵌套在根标签内部,称之为子标签,孙辈标签等...
lang:表示当前页面的语言类型,默认为 en 英文网页,注意如果用户的地区语言
与此处不符,google 浏览器会弹出窗口询问是否需要翻译 -->
<html lang="en">
<!-- head:头信息,用来设置全文的编码,标题等,引入外部资源 -->
<head>
    <!--
        meta:用来设置全文的编码,主要有以下几种编码
            iso-8859-1
            utf-8
            gbk
    -->
    <meta charset="UTF-8">
    <!-- 此处设置兼容性 ie 和 edge -->
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <!-- 设置页面的宽度缩放比例默认为 100%,也就是不缩放,使用 ctrl + +
    可以放大网页 ctrl + - 缩小网页 ctrl + 0 恢复默认也就是这里设置的
    100% -->
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <!-- 设置网页标题 -->
    <title>1:表单</title>
</head>
<!-- body:表示网页的正文部分,我们可以将 body 理解为浏览器的边缘 -->
<body>
    <!-- h1-h6:共有六级标题,随着序号的增大,字体不断减小,自带换行,文本加粗
    在书写 html 标签时,一定要使用 emmet 插件快速书写标签 -->
    <h1>用户注册</h1>
    <hr>
    <!-- form:表示一个表单,在表单内部可以书写各种表单项,这些表单项
    可以通过各种方式收集用户的信息,最终以键值对的方式提交到 action 设置
    的后端地址
        action:表单最终提交到的目的地,一般为后端,这里由于没有后端
        仅仅书写一个静态页面,书写路径存在以下两种方式
            1:绝对路径:以/开头
            2:相对路径:不以/开头
```

```
                ./:表示从当前目录下寻找
                ../:表示从上一级目录下寻找
        method:表单提交方式,存在 get 和 post 两种,如果不写则默认是 get
        ---------------
        String name = request.getParameter("myname")
        String pass = request.getParameter("mypass")

        User u = new User(null,name,pass)
        使用 ORM(Object Mapping Relation )
        对象映射关系工具将这个对象持久化到数据库
        boolean flag = dao.add(u)
        boolean flag = dao.remove(5)
        dao.login(name,pass)
    -->
    <form action="./success.html" method="get">
        <!-- 1:单行文本输入框
                type="text"
                name:表示键值对的键,可以随意书写
                value:表示键值对的值,一般不写,用户填写什么 value
                就是什么
                required:表示表单项必填,如果不写,则表单无法提交
                autofocus:自动获取焦点,光标闪烁
                placeholder:悬浮文本,书写内容立刻消失
                title:验证提示文本
                pattern:正则表达式
                如果不书写 type,则默认就是单行文本输入框
                autocomplete="off" 关闭自动完成功能
        -->
        用户姓名: <input type="text" name="myname"
        value="" required autofocus placeholder="请输入用户姓名"
        title="请输入姓名" autocomplete="off"> <br>
        <!-- 2:单行文本密码框
                type="password"
        -->
        用户密码: <input type="password" name="mypass"
        required placeholder="请输入用户密码" title="请输入密码"
        autocomplete="off"> <br>
        <!-- 4:单选框
                type="radio"
                checked:表示默认选中
        -->
        性别: <input type="radio" name="gender" value="0"
        checked>女
        <input type="radio" name="gender" value="1">男 <br>
        <!-- 5:复选框
                type="checkbox"
                复选框是唯一一个一个键对着多个值的元素
        -->
        爱好:
        <input type="checkbox" name="hobby" value="soccer">足球
        <input type="checkbox" name="hobby" value="running"
        checked>跑步
        <input type="checkbox" name="hobby" value="game">游戏
        <input type="checkbox" name="hobby" value="shopping">购物
        <br>
```

```
        <!-- 6:下拉列表框
                selected:默认选中
        -->
        归属地:
        <select name="location">
            <option value="1">济南</option>
            <option value="2">青岛</option>
            <option value="3" selected>淄博</option>
            <option value="4">德州</option>
            <option value="5">济宁</option>
        </select> <br>
        <!-- 7:邮箱输入框
                type="email" 注意必须输入一个邮箱格式
                但是无法验证这个邮箱的真伪,只能验证格式正确与否
        -->
        邮箱地址: <input type="email" name="email"
        placeholder="请输入邮箱地址" required> <br>
        <!-- 8:日期验证
                type="date" 这里供用户选择一个日期,格式肯定是
                yyyy-MM-dd
        -->
        出生日期: <input type="date" name="birth" required>
        <br>

        <!-- 3:提交和取消按钮
            注意 由于按钮不传递值,所以仅书写 value 属性
            提交按钮可以提交表单到 action 的目的地
            取消按钮可以重置填写的内容
            disabled:禁用,每个元素都有此属性,只要书写,则
            元素被禁用
        -->
        <input type="submit" value="提交">
        <input type="reset" value="取消" disabled>
    </form>
</body>
</html>
```

2-html链接视图列表.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>2:链接视图列表</title>
</head>
<body>
    <!-- a:链接,其实就是简化版的表单,点击链接,则可以
    提交,肯定是 get 方式,链接自带样式,蓝色字体,自带下划线
        href:就表示链接提交到的目的地
    -->
```

```
    <a href="./success.html">点我试试!</a>
    <a href="./success.html?thisiskey=thisisvalue">链接传值</a>
    <!-- target="_blank" 目标页与当前页共存,如果不写,则
    target="_self" 目标页取代当前页 -->
    <a href="./success.html" target="_blank">再点我试试!</a>
    <!-- img:视图,引入一张图片到页面
            src:source 的简写,表示图片的源
            title:鼠标悬浮时显示的文本
            width:设置宽度,注意宽度属于样式,由于我们还未学习 css 因此
            此处使用 html 设置宽度
    -->
    <img src="./assets/etoak.jpg" title="癸卯年来了!"
    width="300px">
    <!-- 链接中可以嵌套图片 -->
    <a href="./success.html">
        <img src="./assets/etoak.jpg" width="200px">
    </a>

    <!-- ul:无序列表
            li:列表项
    -->
    <ul>
        <li>无序列表1</li>
        <li>无序列表2</li>
        <li>无序列表3</li>
        <li>无序列表4</li>
    </ul>
    <ol>
        <li><a href="#">有序列表1</a></li>
        <li>有序列表2</li>
        <li>有序列表3</li>
        <li>有序列表4</li>
    </ol>
</body>
</html>
```

3-css引入方式.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>3:css引入方式</title>
    <!--
        css独立文件以.css为后缀,严格区分大小写,毫无容错性,
        没有报错功能,浏览器如果解析错误,则无法渲染样式
        1:内嵌式
            直接将 css 代码书写在 head 标签内style 中
            这种方式仅仅初步的将结构和样式解耦,解耦不彻底
            复用性依然较差
```

```
    -->
    <style>
        /* 这是 css 的注释方式,不得使用 html 注释,也不能在这里
        使用标签 */
        h2{
            /* 设置字体 */
            font-family:喜鹊招牌体;
            /* 设置字体大小 */
            font-size: 20px;
            /* 设置颜色,主要有三种方式
                1:直接使用英文单词
                red blue green 也可以使用 潘通(pantone)流行色
                coral:珊瑚橘
                cornflowerblue:矢车菊蓝
                crimson:鲑鱼红
                2:使用色号  #ffffff - #000000
                #ff770f:爱马仕橙
                长春花蓝 #6667AB
                勃艮第红 #81001e
                克莱因蓝 #002ea6
                蒂芙尼蓝 #82d8cf
                3:rgb(x,y,z)
                x|y|z:0-255的整数
            */
            color:coral;
        }
        div{
            background-color: purple;
        }
    </style>
    <!-- 2:外链式
        链接外部独立的 css 文件到本页面
        rel="stylesheet" 表示引入的是一个样式表
        href:引入的独立 css 文件路径
        外链式使用最为频繁,复用性较好,将结构和样式彻底解耦
    -->
    <link rel="stylesheet" href="./css/mystyle.css">
</head>
<body>
    <h2>你感染流感了吗?</h2>
    <!-- #:表示提示浏览器不发送任何请求 -->
    <a href="#">点击购买奥司他韦</a>
    <!-- 3:行内式
    直接将 css 书写在标签内,这种书写方式严重违背结构和样式解耦的初衷,但是由于其优先级极高
    所以使用依然较多-->
    <p style="background-color: pink;color:whitesmoke">请佩戴口罩,抵御甲型和乙型流感</p>

    <!-- div:开始和结尾自带换行,一般用来划定一个区域,和段落类似
    但是不会再空一行 -->
    <!-- 三种引入方式的优先级
        行内式 > 外链式和内嵌式 谁放在后面出现冲突听谁的
    -->
    <div>看看我听谁的</div>
```

```
</body>
</html>
```

---

mystyle.css

```
/* 设置 css 文件编码,默认 utf-8,如果不写就是此编码 */
@charset "utf-8";

/* 设置 a 连接默认的样式 */
a{
    /* 设置连接颜色 */
    color:#82d8cf;
    /* 去掉链接自带的下划线 */
    text-decoration: none;
}
/*
    添加伪类,用来表示某个元素某种状态
    选择器:伪类{

    }
    :hover:表示元素被鼠标滑过时的样式
*/
a:hover{
    /* 鼠标滑过时的颜色 */
    color:#ff770f;
    /* 添加下划线 */
    text-decoration: underline;
}
/* 被访问过的样式 */
a:visited{
    /* 设置连接被访问过后显示蕾贝卡紫 */
    color:#663399;
}

div{
    background-color: aqua;
}
```

day02

4-css基本选择器.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>4:css基本选择器</title>
    <style>
```

```
        /* 1:标记选择器
            tagName{
                直接使用标签名作为选择元素的依据,这种引入方式极易引起
                误操作
            }
        */
        span{
            background-color: yellow;
        }
        /* 2:类别选择器
            .class{
                在元素中添加 class 属性,以.class 作为选择元素的依据
                注意,class 全文不唯一,一个元素可以存在多个 class 属性
            }
        */
        .test2{
            background-color: blueviolet;
        }
        /* 3:id 选择器
            #id{
                每个元素可以添加一个 id 属性,注意这个 id 属性全文唯一
                一个元素也只能添加一个,以 #id 作为选择元素依据
            }
        */
        #test3{
            background-color: blue;
        }
    </style>
</head>
<body>
    <!-- span 与 label 被称之为便签标签,结束没有换行,一般搭配
    js 使用,单独使用几乎没有任何效果 -->
    <span>测试1</span>
    <span class="test2" >测试2</span>
    <span id="test3" class="test2" >测试3</span>
    <!--
        如果三种基本选择器出现冲突 优先级如下
        标记选择器 < 类别选择器 < id 选择器
        如果还存在行内式,则一切以行内式为准
    -->
    <span id="test3" class="test2"
    style="background-color: crimson;">测试4</span>
</body>
</html>
```

5-css复合选择器.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>5:css复合选择器</title>
    <style>
```

```
        /* 1:交集选择器
            tagName.class{}
            tagName#id{}
            由一个标记后面紧跟类别或者 id,必须同时满足两个条件
            才可以成功选取
        */
        span.test2{
            /* 文本倾斜 */
            font-style: italic;
        }
        /* 2:并集选择器
            sel1,sel2,sel3,selN{
                由多个基本或者复合选择器用逗号隔开,只要满足其中任意一个
                就可以成功选取
            }
        */
        label.test2,span.test2,h3.title,h4{
            background-color: tomato;
        }
        /* 3:后代选择器
            根据左祖先右后代的层级关系,选择具有特定层级关系的
            最右侧的子元素
            sel1 sel2 selN{

            }
        */
        div#outter em#inner{
            color:teal;
        }
        /* 4:全选选择器
            *{
                相当于 ctrl+a 选择所有元素
            }
        */
        *{
            font-weight: 800;
        }
    </style>
</head>
<body>
    <span>测试 1</span>
    <label class="test2">测试 2</label>
    <span class="test2">测试 3</span>
    <h3 class="title">三级标题</h3>
    <h4 id="etoak">四级标题</h4>
    <div id="outter">济南的<em id="inner">春</em>天开始了</div>
</body>
</html>
```

6-css元素类型.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>6:Css元素类型</title>
    <style>
        span{
            background-color: pink;
            /* 设置宽度 */
            width:300px;
            /* 设置高度 */
            height:400px;
            /* 设置内部文本对齐方式,默认居左,此处为居中对齐 */
            text-align: center;
        }
        div{
            background-color: lightblue;
            /* 设置宽度 */
            width:300px;
            /* 设置高度 */
            height:400px;
            /* 设置内部文本对齐方式,默认居左,此处为居中对齐 */
            text-align: center;
        }
    </style>
</head>
<body>
    <!--
        Css元素类型
            在 css 中,将元素分为很多种类型,其中主要差异较大的有以下
            两种
            1:块元素(block 元素)
                eg: div p h1-h6 ul ol li table 布局元素
                块元素是页面中的主流元素,基本上页面都是由块元素主要组成
                块元素结尾都自带换行,一行只能书写一个,从上到下排列,设置
                盒子模型有效,可以设置行高,设置对齐方式
                块元素在不主动设置宽度的前提下和父元素同宽,如果没有父元素
                则与浏览器同宽
            2:内联元素(inline 元素,行内元素)
                eg: span label a img input 等等
                内联元素在页面中主要用来进行信息的提示灯,结尾没有换行,
                从左往右排列,一行可以排列多个,设置盒子模型无效,设置对齐方式
                无效
                img input:又被称之为 内联块元素(inline-block)
                虽然是内联元素,但是设置盒子模型有效

    -->
    <!-- style="display: block;" 以块元素显示元素 -->
    <span style="display: block;">
        我是 span我是 span我是 span我是 span我是 span我是 span
        我是 span我是 span我是 span我是 span我是 span我是 span
```

```
        我是 span我是 span我是 span我是 span我是 span我是 span
    </span>
    <hr>
    <!-- style="display: inline;" 以内联元素显示元素 -->
    <div style="display: inline;">
        我是 div我是 div我是 div我是 div我是 div我是 div
        我是 div我是 div我是 div我是 div我是 div我是 div
        我是 div我是 div我是 div我是 div我是 div我是 div
    </div>
    <!-- display:none;隐藏元素 -->
    <div style="display: none;">
        我是 div我是 div我是 div我是 div我是 div我是 div
        我是 div我是 div我是 div我是 div我是 div我是 div
        我是 div我是 div我是 div我是 div我是 div我是 div
    </div>
</body>
</html>
```

7-css盒子模型.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>7:Css盒子模型</title>
    <style>
        /*
            标准文本流(文档流 document normal flow)
                其实就是指一种默认的状态,页面中主要由块元素组成,这些元素
            结尾自带换行,因此默认情况下在没有任何 css 渲染的前提下,这些元素
            从上往下排列,这种状态类似水流一样,当我们挪走其中的某个元素,后面
            的元素会递补,继续保持水流从上往下,在书写页面时应该首先书写 html 结构
            当所有的结构都书写完毕,呈现标准文本流之后,才开始书写 css,最终
            添加 js,切忌边写 html 边写 css
            盒子模型
                在css 中将各种元素看做一个一个矩形的盒子,这些盒子具有四个
            边框,从上往下摆放在页面中,通过设置 border 属性可以渲染盒子的边框
            通过设置 margin 属性可以设置盒子边框以外的空间位置,通过设置
            padding 属性可以设置盒子边框以内的空间位置,元素的宽度通过 width
            来设置 高度则有 height 来进行设置,最终共有 14 个参数,这 14 个参数
            被称之为盒子模型,其中大部分元素只有块元素才可以设置
        */
        *{
            /* 去掉默认留白部分 */
            margin:0;
            padding:0;
        }

        /* 谷歌和火狐存在差异 双方一个认为 html 是正文
        一个认为 body 是正文,因此这里两边使用并集 */
        html,body{
            /* 设置高度参照物 */
```

```
            height:100%;
        }

        .container{
            /* 渲染边框:边框类型 边框粗细 边框颜色
                边框类型 solid double dotted
            */
            border:solid 2px red;
            /*
                1:网页整体居中,使用具体像素
                margin:0 auto;
                width:xxxpx;
                height:xxxpx;
                2:网页整体居中,使用百分比,注意这个百分比
                永远是与父元素进行比较,如果没有父元素,则与
                浏览器进行比较,注意如果设置百分比,则必须设置
                宽度和高度的参照物,其中宽度的参照物在 head 标签
                内已经存在
                <meta name="viewport" content="width=device-width,
initial-scale=1.0">
                所以宽度默认浏览器就是百分之百,但是注意高度并没有
                参照物,因此如果我们不设置,高度使用百分比无效
                设置高度参照物
                    html,body{
                        height:100%;
                    }
                3:网页整体居中,使用视口单位
                vw:viewport width
                vh:viewport height
                表示占据当前设备视口的宽度和高度,不需要设置参照物
                注意也不是与父元素比较,而是与当前设备比较
            */
            margin:0 auto;
            /* width:700px;
            height:800px; */
            /* width:80%;
            height:150%; */
            width:80vw;
            height:100vh;
        }
        h2{
            border:solid 2px blue;
            /* 设置外边距 */
            /* margin-top: 50px; */
            /* 设置左外边距 */
            /* margin-left: 20px;
            margin-bottom:30px;
            margin-right:100px; */
            /* 简化写法
                margin:上 右 下 左;
                margin:上 (右左) 下;
                margin:(上下) (右左);
                margin:(上右下左);
            */
            margin:50px 100px 30px 20px;
```

```
            padding-top: 20px;
            padding-left: 50px;
            padding-bottom: 10px;
            /* padding 依然可以使用缩略写法 */
            border-top: double 10px hotpink;
            border-right: dotted 15px yellowgreen;
            border-bottom: dotted 8px black;
        }
        ul{
            border:solid 2px aqua;
        }
        li{
            border:solid 2px purple;
        }
        p{
            border:solid 2px yellow;
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>二级标题</h2>
        <ul>
            <li>列表1</li>
            <li>列表2</li>
            <li>列表3</li>
        </ul>
        <p>我是段落</p>
    </div>
</body>
</html>
```

8-css浮动.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>8:浮动</title>
    <style>
        /*
            浮动:
            元素在设置浮动之后,开始朝着浮动方向不断缩小,当缩小到无法再次
            缩小时,漂浮在页面上,之后的元素为了维持标准文本流递补到之前元素
            的位置,如果多个元素都设置浮动则同时漂浮在页面上,从左往右或者从右往左
            排列,这也是最简单的一种打破页面标准文本流的方式
            同时元素也可以设置 clear 属性来屏蔽因为受到其他元素浮动影响造成的影响
        */
        html,body{
            margin:0;
            padding:0;
        }

```

```
        .container{
            margin:0 auto;
            width:80vw;
            height:100vh;
            background-color: yellow;
            border:solid 1px yellow;
        }

        .box1{
            margin-top: 10px;
            height: 100px;
            background-color: pink;
            /* 设置左浮动 */
            float: left;
        }
        .box2{
            margin-top: 10px;
            height:120px;
            background-color: hotpink;
            float: left;
        }
        .box3{
            margin-top: 10px;
            height:140px;
            background-color: deeppink;
            float: left;
        }
        p{
            background-color: lightblue;
            /* 设置元素不受其他元素浮动影响
                clear:left|right|both;
            */
            clear: left;
        }
        ul{
            /* 设置去掉列表徽记 */
            list-style-type: none;
            margin-top: 100px;
        }
        ul li{
            /* 设置列表项浮动 */
            float: left;
            margin-right: 50px;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="box1">盒子1</div>
        <div class="box2">盒子2</div>
        <div class="box3">盒子3</div>
        <p>
            我是段落我是段落我是段落我是段落我是段落我是段落
            我是段落我是段落我是段落我是段落我是段落我是段落
        </p>
```

```
        <ul>
            <li>列表1</li>
            <li>列表2</li>
            <li>列表3</li>
            <li>列表4</li>
            <li>列表5</li>
        </ul>
    </div>
</body>
</html>
```

9-css定位.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>9:定位</title>
    <style>
        .container{
            border:solid 1px black;
            position: relative;
            margin:0 auto;
            width:80vw;
            height:100vh;
        }
        .sub1{
            border:solid 2px crimson;
            /*
                1:相对定位
                元素根据原先所在位置的左上角进行定位,偏移量
                top bottom left 和 right ,定位之后元素不改变原先的
                类型,原来的位置依然被占用
            */
            position: relative;
            /* 向下偏移 200 像素 */
            top:200px;
            /* 向右偏移 300 像素 */
            left:300px;
        }
        .sub2{
            border:solid 2px coral;
            /*
                2:绝对定位
                    元素根据其距离最近的定位过的祖先元素的左上角进行定位
                    定位之后元素不再保持原先的类型,原来的位置被其他元素占用
                    偏移量与相对定位一直,top bottom left right
                    如果元素的所有祖先元素都没有定位,则根据 body 也就是浏览器
                    左上角进行定位,如果在设置全局居中时,则有可能出现问题
            */
            position:absolute;
            top:400px;
            left:100px;
```

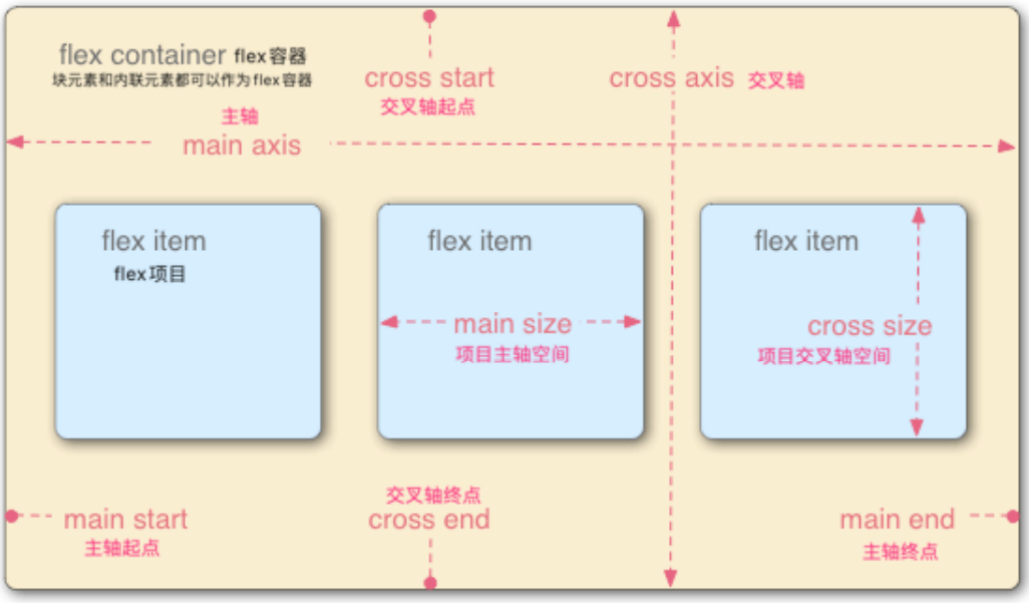
```
        }
        .sub3{
            border:solid 2px yellow;
        }
        .sub4{
            border:solid 2px yellowgreen;
        }
        .sub5{
            border:solid 2px aqua;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="sub1">层1</div>
        <div class="sub2">层2</div>
        <div class="sub3">层3</div>
        <div class="sub4">层4</div>
        <div class="sub5">层5</div>
    </div>
</body>
</html>
```

---

day03

---

flex弹性盒子的名词解释是重点其中



- ***flex-container***

  采用 Flex 布局的元素，称为 Flex 容器（flex container），简称"容器"。

- *item*

  它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称"项目"。

- *主轴*

  容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边 框的交叉点）叫做 main start，结束位置叫做 main end；

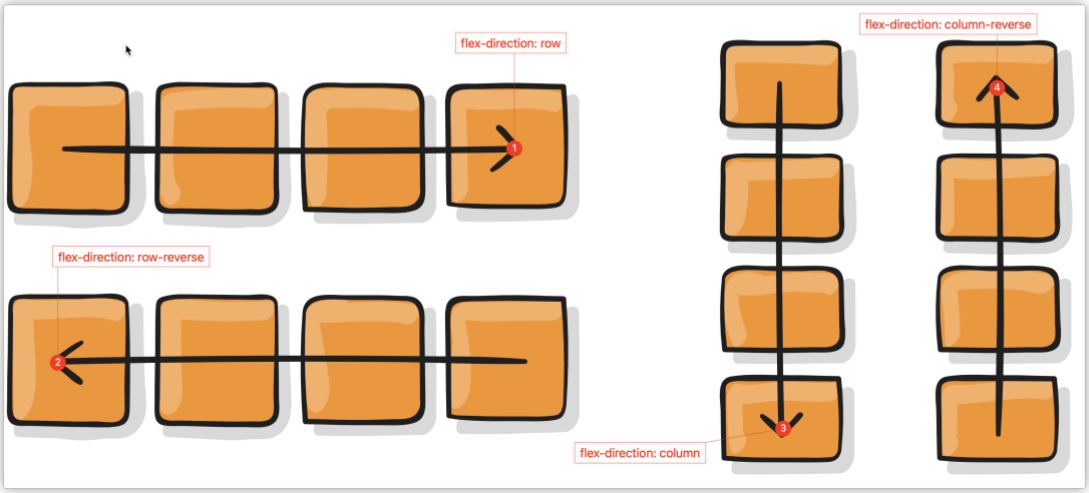
- *交叉轴*

  交叉轴的开始位置叫做 cross start，结束位置叫做 cross end。 项目默认沿主轴排列。单个项目占据的主轴空间叫做 main size，占据的交叉轴空间叫做 cross size

flex-direction:重点

flex-direction 属性决定主轴的方向（即项目的排列方向）。如果未显式设置 flex-direction 属性，Flex 容器则会采用其默认值 row 。

```
.box {
    flex-direction: row | row-reverse | column | column-reverse;
}
```

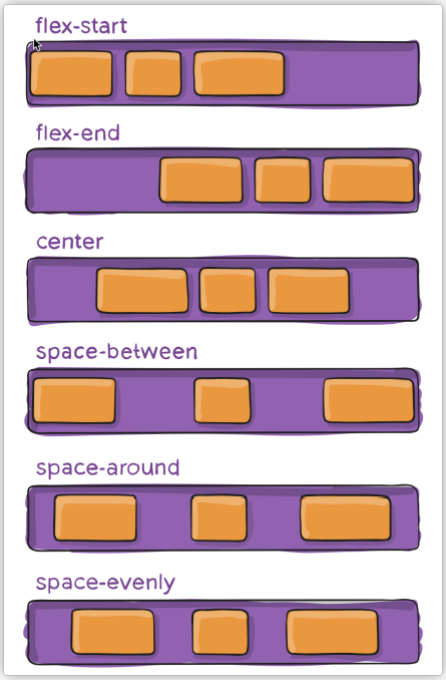


justify-content:重点

justify-content 属性定义了项目在主轴上的对齐方式。

```

.box {
    justify-content: flex-start | flex-end | center | space-between | space-
around;
}
```

在Flex容器中使用 justify-content 来控制Flex项目在Flex容器主轴方向的对齐方式，也可以用来分配Flex容器中主轴方向的剩余空间。使用 justify-content 分配Flex容器剩余空间，主要是将剩余空间按不同的对齐方式，将剩余空间分配给Flex项目的两侧，即控制Flex项目与Flex项目之间的间距。



它可能取5个值，具体对齐方式与轴的方向有关。下面假设主轴为从左到右。

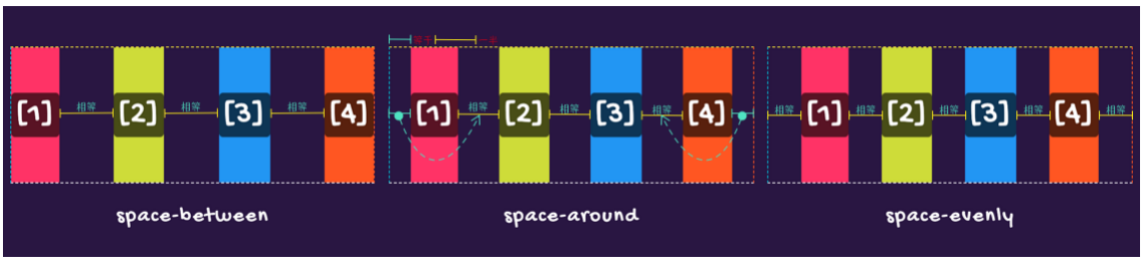
flex-start （默认值）：左对齐

flex-end：右对齐 center： 居中

space-between：两端对齐，项目之间的间隔都相等。

space-around：每个项目两侧的间隔相等。

所以，项目之间的间隔比项目与边框的间隔大一 倍。 注意后三种排列的区别



- *space-between 会让第一个Flex项目的盒子起始边缘与Flex容器主轴起点相稳合，最后一个Flex项目的盒子结束边缘与Flex容器主轴终点相稳合，其它相邻Flex项目之间间距相等。当Flex容器中只 有一个Flex项目时，其表现行为和 flex-start 等同*
- *space-around 会让第一个Flex项目的盒子起始边缘与Flex容器主轴起点间距和最后一个Flex项目的盒子结束边缘与Flex容器主轴终点间距相等，并且等于其他相邻两个Flex项目之间间距的一半。当Flex容器中只有一个Flex项目时，其表现行为和 center 等同*
- *space-evenly 会让第一个Flex项目的盒子起始边缘与Flex容器主轴起点间距和最后一个Flex项目的盒子结束边缘与Flex容器主轴终点间距相等，并且等于其他相邻两个Flex项目之间间距。当Flex容 器中只有一个Flex项目时，其表现行为和 center 等同*

如果Flex容器没有额外的剩余空间，或者说剩余空间为负值时， justify-content 的值表现形式：

flex-start 会让Flex项目在Flex容器主轴结束点处溢出

flex-end 会让Flex项目在Flex容器主轴起点处溢出

center 会让Flex项目在Flex容器两端溢出

space-between 和 flex-start 相同

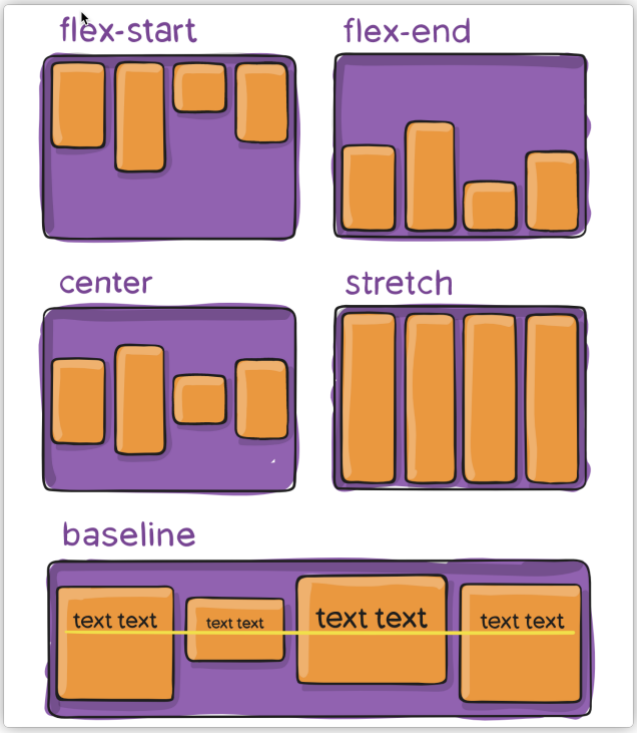
space-around 和 center 相同

space-evenly 和 center 相同

align-items:重点

align-items 属性定义项目在交叉轴上如何对齐。

```
.box {
    align-items: flex-start | flex-end | center | baseline | stretch;
}
```



它可能取5个值。具体的对齐方式与交叉轴的方向有关，下面假设交叉轴从上到下。

- *flex-start ： 交叉轴的起点对齐。*
- *flex-end ： 交叉轴的终点对齐。*
- *center ： 交叉轴的中点对齐。*
- *baseline : 项目的第一行文字的基线对齐。*
- *.box { align-items: flex-start | flex-end | center | baseline | stretch; } 1 2 3*
- *stretch （默认值）： 如果项目未设置高度或设为auto，将占满整个容器的高度。*

> *align-items 的默认值是 stretch ， 但只有Flex项目显式设置 height (或 width)值，Flex项 目才会被拉伸填满整个Flex容器。 如果Flex容器没有剩余空间或剩余空间为负值是:*

- *flex-start 会让Flex项目在Flex容器侧轴终点处溢出*
- *flex-end 会让Flex项目在Flex容器侧轴起点处溢出*
- *center 会让Flex项目在Flex容器侧轴两侧溢出*
- *baseline 会让Flex项目在Flex容器侧轴终点溢出，有点类似于 flex-start*

flex:1;重点

`flex` 属性是 `flex-grow`，`flex-shrink` 和 `flex-basis` 的简写，默认值为 `0 1 auto` 。后两个属性可选。

```
.item {
    flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
}
```

该属性有两个快捷值： `auto` （ `1 1 auto` ）和 none (`0 0 auto`)。

建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值。

Flex项目中使用 `flex` 属性可以根据Flex容器的可用空间对自身做伸缩计算，其包含三个子属性： `flex-basis` 、 `flex-shrink` 和 `flex-grow` 。这几个属性都有其初始值：

- `flex-grow` 的初始值为 `0`
- `flex-shrink` 的初始值为 `1`
- `flex-basis` 的初始值为 `auto`

即 `flex` 的三个子属性： `flex-grow` （扩展比率）、 `flex-shrink` （收缩比率）和 `flex-basis` （伸缩基准）。这三个属性可以控制Flex项目，具体的表现如下：

- `flex-grow` ：设置Flex项目的扩展比率，让Flex项目得到（扩展）多少Flex容器剩余空间（Positive Free Space），即Flex项目可能会变大
- `flex-shrink` ：设置Flex项目收缩比率，让Flex项目减去Flex容器不足的空间（Negative Free Space），即Flex项目可能会变小
- `flex-basis` ：Flex项目未扩展或收缩之前，它的大小，即指定了Flex项目在主轴方向的初始大小

`flex` 属性可以指定 **`1` 个值（单值语法）** 、 **`2` 个值（双值语法）** 或 **`3` 个值（三值语法）** 。

单值语法：值必须为以下其中之一：

- 一个无单位的数（ `<number>` ），比如 `flex: 1` ，这个时候它会被当作 `<flex-grow>` 的值
- 一个有效的宽度（ `width` ）值，比如 `flex: 30vw` ，这个时候它会被当作 `<flex-basis>` 的值
- 关键词 `none` 、 `auto` 或 `initial` （即初始值）

双值语法：第一个值必须为一个无单位数值，并且它会被当作 `<flex-grow>` 的值；第二个值必须为以下之一：

- 一个无单位的数（ `<number>` ），它会被当作 `<flex-shrink>` 的值
- 一个有效的宽度（ `width` ）值，它会被当作 `<flex-basis>` 的值

三值语法：

- 第一个值必须是一个无单位数（ `<number>` ），并且它会被当作 `<flex-grow>` 的值

- 第一个值必须是一个无单位数（ `<number>` ），并且它会被当作 `<flex-grow>` 的值
- 第二个值必须是一个无单位数（ `<number>` ），并且它会被当作 `<flex-shrink>` 的值
- 第三个值必须为一个有效的宽度（ `width` ）值，并且它会被当作 `<flex-basis>` 的值

`flex` 属性的取值可以是：

- `auto` ：Flex项目会根据自身的 `width` 和 `height` 来确定尺寸，但Flex项目根据Flex容器剩余空间进行伸缩。其相当于 `flex: 1 1 auto`
- `initial` ：Flex项目会根据自身的 `width` 和 `height` 来设置尺寸。它会缩短自身以适应Flex容器，但不会伸长并吸收Flex容器中的额外剩余空间来适应Flex容器。其相当于 `flex: 0 1 auto`
- `none` ：Flex项目会根据自身的 `width` 和 `height` 来设置尺寸。它是完全非弹性的（既不会缩短，也不会伸长来适应Flex容器）。其相当于 `flex: 0 0 auto`

- `<flex-grow>` ：定义Flex项目的 `flex-grow` 属性，取值为 `<number>`
- `<flex-shrink>` ：定义Flex项目的 `flex-shrink` 属性，取值为 `<number>`
- `<flex-basis>` ：定义Flex项目的 `flex-basis` 属性。**若值为 `0` ，则必须加上单位，以免被视作伸缩性**

flex-grow计算公式

---

day03

10-用户登录.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>10:用户登录</title>
    <!-- 引入全局 css -->
    <link rel="stylesheet" href="./css/global.css">
    <!-- 引入登录的 css -->
    <link rel="stylesheet" href="./css/login.css">
    <!-- 引入图标 -->
    <link rel="icon" href="./assets/favicon.ico">
</head>
<body>
    <div class="login-container">
```

```
        <form action="./12-圣杯布局.html" class="login-form">

            <h1>用户登录</h1>

            <input type="text" name="username"
            placeholder="请输入用户姓名" required autocomplete="off"
            autofocus>

            <input type="password" name="password"
            placeholder="请输入用户密码" required autocomplete="off">

            <div>
                <input type="submit" value="登录">

                <input type="reset" value="取消">

                <input type="button" value="注册">
            </div>
        </form>
    </div>
    <script>
        document.querySelector('input[type=button]')
        .onclick=()=>{
            location.href = './12-圣杯布局.html'
        }
    </script>
</body>
</html>
```

login.css

```
@charset "utf-8";
/* 设置登录的样式 */
/* 设置最外侧大容器 */
.login-container{
    /* 设置容器绝对定位 */
    position: absolute;
    /* 设置容器宽度 */
    width:100vw;
    /* 设置容器高度 */
    height:100vh;
    /* 设置容器背景图片 no-repeat:表示如果图片大小不合适
    则不会重叠摆放 */
    background: url("../assets/bg1.jpeg") no-repeat;
    /* 设置背景图尺寸 宽度 高度 */
    background-size: 100vw 100vh;
    /* 设置背景图不会随着缩放变动 */
    background-attachment: fixed;
    /* 容器开启弹性盒子 */
    display: flex;
    /* 设置项目,也就是表单 主轴 交叉轴居中 此时默认 水平是主轴
    纵向是交叉轴 */
    justify-content: center;
    align-items: center;
}
```

```
25
26  /* 设置表单 */
27  .login-form{
28      /* 设置表单宽度 */
29      width:350px;
30      /* 设置表单背景色 和 透明度
31      rgba(红色,绿色,蓝色,透明度)
32      红绿蓝为 0-255 的整数 如果都是 255 则是白色
33      透明度 0 完全透明 1 完全不透明 */
34      background-color: rgba(255, 255, 255, 0.6);
35      /* 设置边框为圆角 数字越大越圆,如果设置为 50%则变为圆球
36      表格不能设置 */
37      border-radius: 30px;
38      /* 给表单开启弹性盒子 */
39      display: flex;
40      /* 设置表单内的项目从上往下排列,默认是 row 从左往右 */
41      flex-direction:column;
42      /* 设置交叉轴水平居中摆放,注意这里从左往右是交叉轴了,从上往下
43      才是主轴 因为排列顺序是从上往下 */
44      align-items: center;
45  }
46
47  /*
48      设置单行文本输入框和单行文本密码框
49      [属性名=属性值]{}
50      拿取一个元素,其中属性名等于属性值,注意必须精确匹配
51      这叫属性选择器
52  */
53  input[type=text],
54  input[type=password]{
55      /* 去掉输入框边框 */
56      border:none;
57      /* 去掉获取焦点后边框 */
58      outline: none;
59      /* 设置输入框为圆角 */
60      border-radius: 10px;
61      /* 设置输入框外边距 */
62      margin:5px;
63      /* 设置输入框内边距 */
64      padding:5px;
65      /* 添加阴影 */
66      box-shadow: 5px 5px 5px silver;
67  }
68
69  /*
70      设置三个按钮
71      [属性名^=属性值开头]{}
72      [属性名*=包含属性值]{}
73      [属性名$=属性值结尾]{}
74  */
75  input[type^=sub],
76  input[type*=ese],
77  input[type$=tton]{
78      /* 设置按钮外边距 */
79      margin:10px;
```

```
    /* 去掉按钮边框 */
    border:none;
    /* 更改为圆角 */
    border-radius: 10px;
    /* 设置按钮宽度 */
    width:50px;
    /* 添加按钮背景色 */
    background-color: cornflowerblue;
    /* 添加按钮字体颜色 */
    color:whitesmoke;
    /* 添加阴影 前三个参数为阴影偏移量 第四个参数为阴影颜色 */
    box-shadow: 5px 5px 5px gray;
}
```

global.css(全局布置)

```
/* 此处为全局 css 设置的位置 */
@charset "utf-8";

html,body{
    /* 去掉块元素留白部分 */
    margin:0;
    padding:0;
    /* 设置全文使用的字体 */
    font-family: 喜鹊招牌体;
    /* 设置百分比参照物,否则全文无法使用高度百分比单位 */
    height:100%;
}
```

11-用户注册.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>11:用户注册</title>
    <link rel="stylesheet" href="./css/global.css">
    <link rel="stylesheet" href="./css/register.css">
    <link rel="icon" href="./assets/favicon.ico">
</head>
<body>
    <div class="reg-container">
        <form action="./10-用户登录.html" class="reg-form">

            <h1>用户注册</h1>

            <input type="text" name="username"
            placeholder="请输入用户姓名" autofocus
            autocomplete="off" required>

            <input type="password" name="password"
            placeholder="请输入用户密码"
```

```
            autocomplete="off" required>

            <input type="text" name="realname"
            placeholder="请输入真实姓名"
            autocomplete="off" required>

            <input type="text" name="email"
            placeholder="请输入邮箱地址" required>

            <input type="tel" name="phone"
            placeholder="请输入电话号码" required>

            <div>
                性别: <input type="radio" name="gender" value="0"
                checked>男
                <input type="radio" name="gender" value="1">女
            </div>

            <div>
                权限: <input type="radio" name="role" value="0"
                checked>用户
                <input type="radio" name="role" value="1">管理员
            </div>

            <div>
                <input type="submit" value="注册">
                <input type="reset" value="取消">
            </div>
        </form>
    </div>
</body>
</html>
```

register.css

```
@charset "utf-8";

/* 设置最外侧大容器 */
.reg-container {
    /* 设置容器绝对定位 */
    position: absolute;
    /* 设置容器宽度 */
    width: 100vw;
    /* 设置容器高度 */
    height: 100vh;
    /* 设置容器背景图片 no-repeat:表示如果图片大小不合适
    则不会重叠摆放 */
    background: url("../assets/bg5.jpeg") no-repeat;
    /* 设置背景图尺寸 宽度 高度 */
    background-size: 100vw 100vh;
    /* 设置背景图不会随着缩放变动 */
    background-attachment: fixed;
    /* 容器开启弹性盒子 */
    display: flex;
    /* 设置项目,也就是表单 主轴 交叉轴居中 此时默认 水平是主轴
```

```
    纵向是交叉轴 */
    justify-content: center;
    align-items: center;
}

/* 设置表单 */
.reg-form {
    /* 设置表单宽度 */
    width: 350px;
    /* 设置表单背景色 和 透明度
    rgba(红色,绿色,蓝色,透明度)
    红绿蓝为 0-255 的整数 如果都是 255 则是白色
    透明度 0 完全透明 1 完全不透明 */
    background-color: rgba(255, 255, 255, 0.6);
    /* 设置边框为圆角 数字越大越圆,如果设置为 50%则变为圆球
    表格不能设置 */
    border-radius: 30px;
    /* 给表单开启弹性盒子 */
    display: flex;
    /* 设置表单内的项目从上往下排列,默认是 row 从左往右 */
    flex-direction: column;
    /* 设置交叉轴水平居中摆放,注意这里从左往右是交叉轴了,从上往下
    才是主轴 因为排列顺序是从上往下 */
    align-items: center;
}

input[type=text],
input[type=password],
input[type=email],
input[type=tel] {
    /* 去掉输入框边框 */
    border: none;
    /* 去掉获取焦点后边框 */
    outline: none;
    /* 设置输入框为圆角 */
    border-radius: 10px;
    /* 设置输入框外边距 */
    margin: 5px;
    /* 设置输入框内边距 */
    padding: 5px;
    /* 添加阴影 */
    box-shadow: 5px 5px 5px silver;
}

input[type^=sub],
input[type*=ese] {
    /* 设置按钮外边距 */
    margin: 10px;
    /* 去掉按钮边框 */
    border: none;
    /* 更改为圆角 */
    border-radius: 10px;
    /* 设置按钮宽度 */
    width: 50px;
    /* 添加按钮背景色 */
```

```
    background-color: cornflowerblue;
    /* 添加按钮字体颜色 */
    color: whitesmoke;
    /* 添加阴影 前三个参数为阴影偏移量 第四个参数为阴影颜色 */
    box-shadow: 5px 5px 5px gray;
}
```

12-圣杯布局.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>13:表格</title>
    <link rel="stylesheet" href="./css/global.css">
    <link rel="stylesheet" href="./css/table.css">
    <link rel="icon" href="./assets/favicon.ico">
</head>
<body>
    <div class="table-container">
        <!--
            table:表示渲染一个表格
                tr:表示表格中的行
                td:表示表格中的列
                横行竖列,注意表格最简化必须存在 table tr td
                嵌套关系不能变
        -->
        <table class="tb">
            <!-- caption:表格的标题,不是必须 -->
            <caption>表格标题</caption>
            <!-- thead:表头,一般嵌套表格的首行,多搭配 css 使用 -->
            <thead>
                <tr>
                    <!-- th:表示列,效果同 td,但是使用在表头中
                    内部文字居中 加粗,多使用在表头中 -->
                    <th>列1</th>
                    <th>列2</th>
                    <th>列3</th>
                    <th>列4</th>
                    <th>列5</th>
                </tr>
            </thead>
            <!-- tbody:一般嵌套表格主体,多搭配 css 使用
            注意如果我们不书写,则自动添加 tbody -->
            <tbody>
                <tr>
                    <td>列1</td>
                    <td>列2</td>
                    <td>列3</td>
                    <td>列4</td>
                    <td>列5</td>
                </tr>
                <tr>
```

```
                    <td>列1</td>
                    <td>列2</td>
                    <td>列3</td>
                    <td>列4</td>
                    <td>列5</td>
                </tr>
                <tr>
                    <td>列1</td>
                    <td>列2</td>
                    <td>列3</td>
                    <td>列4</td>
                    <td>列5</td>
                </tr>
            </tbody>
        </table>
    </div>
</body>
</html>
```

table.css

```
@charset "utf-8";

/* 设置外侧容器 */
.table-container{
    margin:0 auto;
    width:80vw;
    display: flex;
    justify-content: center;
    /* 从交叉轴起始位置排列 */
    align-items: flex-start;
}

/* 设置表格 */
.tb{
    /* 设置表格宽度 */
    width:60vw;
    /* 向下偏移 100 像素 */
    margin-top: 100px;
    /* 存在多条边框时合并为一根 */
    border-collapse: collapse;
    /* 设置表格内部文本居中 */
    text-align: center;
}

/* 设置表格边框 */
.tb,tr,td,th{
    /* 设置表格的边框类型 */
    border:solid 1px #ddd;
}

/* 设置表头 */
.tb thead{
    background-color: coral;
    color: navy;
```

```css
}
/* 设置表格主体 */
.tb tbody{
    background-color: azure;
    color:gray;
}
```

13-表格.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>13:表格</title>
    <link rel="stylesheet" href="./css/global.css">
    <link rel="stylesheet" href="./css/table.css">
    <link rel="icon" href="./assets/favicon.ico">
</head>
<body>
    <div class="table-container">
        <!--
            table:表示渲染一个表格
                tr:表示表格中的行
                td:表示表格中的列
                横行竖列,注意表格最简化必须存在 table tr td
                嵌套关系不能变
        -->
        <table class="tb">
            <!-- caption:表格的标题,不是必须 -->
            <caption>表格标题</caption>
            <!-- thead:表头,一般嵌套表格的首行,多搭配 css 使用 -->
            <thead>
                <tr>
                    <!-- th:表示列,效果同 td,但是使用在表头中
                    内部文字居中 加粗,多使用在表头中 -->
                    <th>列1</th>
                    <th>列2</th>
                    <th>列3</th>
                    <th>列4</th>
                    <th>列5</th>
                </tr>
            </thead>
            <!-- tbody:一般嵌套表格主体,多搭配 css 使用
            注意如果我们不书写,则自动添加 tbody -->
            <tbody>
                <tr>
                    <td>列1</td>
                    <td>列2</td>
                    <td>列3</td>
                    <td>列4</td>
                    <td>列5</td>
                </tr>
                <tr>
```

```
                    <td>列1</td>
                    <td>列2</td>
                    <td>列3</td>
                    <td>列4</td>
                    <td>列5</td>
                </tr>
                <tr>
                    <td>列1</td>
                    <td>列2</td>
                    <td>列3</td>
                    <td>列4</td>
                    <td>列5</td>
                </tr>
            </tbody>
        </table>
    </div>
</body>
</html>
```

teble.css

```
@charset "utf-8";

/* 设置外侧容器 */
.table-container{
    margin:0 auto;
    width:80vw;
    display: flex;
    justify-content: center;
    /* 从交叉轴起始位置排列 */
    align-items: flex-start;
}

/* 设置表格 */
.tb{
    /* 设置表格宽度 */
    width:60vw;
    /* 向下偏移 100 像素 */
    margin-top: 100px;
    /* 存在多条边框时合并为一根 */
    border-collapse: collapse;
    /* 设置表格内部文本居中 */
    text-align: center;
}

/* 设置表格边框 */
.tb,tr,td,th{
    /* 设置表格的边框类型 */
    border:solid 1px #ddd;
}

/* 设置表头 */
.tb thead{
    background-color: coral;
    color: navy;
```

```
}
/* 设置表格主体 */
.tb tbody{
    background-color: azure;
    color:gray;
}
```

Javascript基本语法

ES规范

> 完全遵循 ECMA 的 ES 规范,ES5 和 ES6 差异较大,没有兼容性问题,每个浏览器都支持

> Java 基本数据类型:byte short char int float long double boolean

基本数据类型(值数据类型)

- string:字符串

```
//ES5
var str = "etoak";
var count = 100;
//ES6 使用 let 和 const 取代 var,作为标识符
//let:用来设置变量
//const:用来设置常量
//ES6语句具有 ASI 特性,语句结尾自动添加分号,我们不需要书写,但是要注意,语句必须是最终的结尾
//在 ES6 推荐使用单引号取代双引号
let str2 = 'etoak'
const count = 100
```

- number:表示数字,相当于 java 中的 int+float,范围是 正负2^53-1
  - NaN:not a number 的简写,表示不是一个数字
  - Infinity:正无穷
  - -Infinity:负无穷
  - number() parseInt() -0 +
- boolean:true 和 false,在布尔中存在真假值一说
  - 0 null NaN undefined false '' 这些被称之为假值
  - 除去以上的数据,被称之为真值
  - 在流程控制时,真假值就可以当成 true 和 false 使用

```
let str = ''
if(!str){
    //执行这里
}
const arr = [1,2,3,4]
//只要数组长度不为 0 就执行分支内
if(arr.length){
    //执行这里
}
```

- *null*
  - *type of 数据类型为 'Object'*
- *undefined:表示未定义,或者没有数据类型*

```
//java
String str = "etoak";
int count;
//js
let str  //undefined
```

- *bigint(ES8):范围超过了 2^53-1 用来弥补number对于超大数据的局限*

```
let count = 100 //number
let count2 = 100n //bigint
```

- *symbol(ES6):表示一个独一无二的数据,底层开辟内存地址*

复杂数据类型(地址数据类型,引用数据类型)

- *Object:对象*
  - *Array:数组*
  - *Function:函数*
    - *函数不管在任何情况下都有返回值,要么就是我们显式书写的 return ,要么返回 undefined*
  - *RegExp:正则表达式*
  - *Math:内置对象,用来进行数学运算*
  - *Date:内置对象,用来进行日期运算*

BOM(Browser Object Model 浏览器对象模型)

> *提供了一组内置的对象,用来与浏览器进行交互,存在一定的兼容性问题*

- *window:表示整个浏览器窗口,顶层对象,全局变量*
  - *document:表示页面的正文部分,通过 document 可以借助 DOM 来操纵页面的结构和样式*
  - *location:表示浏览器的地址栏,一般可以获取地址信息,控制跳转等*
  - *history:表示浏览器的历史记录,缓存等信息*
  - *navigator:表示当前浏览器的信息,例如版本,内核,品牌等等*
  - *screen:捕捉当前页面屏幕信息,例如解析度,分辨率,色彩,鼠标指针*
  - *frame:页面的一个框架,是 iframe 的前身,已经不推荐使用*

DOM(Document Object Model 文档对象模型)

> *提供了一个模型,js 可以借助 DOM 对页面的结构和样式进行操作,这个模型由以下这些节点组成*
>
> *DOM存在严重的兼容性问题*

- *文档节点:就是指页面的根元素,在 html 中就是指 html 节点,一个 html 只有一个文档节点*
- *元素节点:就是指页面中的各种元素,js 存在多种选择器,通过这些选择器可以使用 js 获取元素节点*
  - *ES5*
    - *Node document.getElementById('id')*
    - *NodeList document.getElementsByTagName('tagName')*
    - *NodeList document.getElementsByName('name')*
    - *NodeList document.getElementsByClassName('class')*
  - *ES6*
    - *Node document.querySelector('sel')*
    - *NodeList document.querySelectorAll('sel')*
- *属性节点:一般存在于元素节点上 <tagName 属性名="属性值" />*
  - *元素节点.setAttribute('属性名','属性值')*
  - *元素节点.getAttribute('属性名')*
  - *元素节点.removeAttribute('属性名')*
- *文本节点:就是指页面中的文本*
- *注释节点:就是指页面中的注释*

---

day04

js能够直接修改页面的结构和样式吗?如果不能为什么?

js不能直接修改页面的结构和样式,当浏览器从上往下解析,如果解析无误,全部解析结束之后,会生成一个DOM(Document Object Model文档对象模型)这个模型结构与样式与用户书写的完全一致,由各种节点组成,它是根在上方的树,大致如下

```
                document
                   |
                 html
                  |
              -----------
              |         |
             head      body
              |         |
          -------      --------
          |     |      |      |
        title meta   button  table
                               |
                               tr
                               |
                               td
```

模型生成之后与当前页面完全一致,当使用 js 要修改页面时,一般是通过 document 对象来修改DOM模型,我们书写的任意的脚本都是对模型进行 crud 操作,当修改结束之后,模型与页面不再保持一致,则浏览器重新渲染页面使页面与模型保持一致,所以说并不是 js 直接去修改了 html 和 css,而是 js 修改了 DOM 模型,浏览器为了与 DOM 保持一致,重新渲染页面

第一个页面的例子,如果我们注释掉 window.onload 则会因为 模型根本没有创建完毕,就去模型中根据 id 属性 btn 来获取元素节点失败,从而绑定单击事件失败

---

var 为什么被 let 和 const 取代了

> *以下是 var 存在的一些问题,以下例子为恶例*

无视块级作用域

```
    function demo(){
        var count = 100
    }
    console.log(count)  //100
    demo()
```

可以重复赋值

```
    var count = 100
    ....
    ....
    var count = true
    ....
    ....
    var count = 'etoak'
```

可以先调用,后声明

```
    function demo(){
        count++
        count--
        count += '!!!'
        count += 'etoak'
        var count = 10
    }
    demo()
```

1-初始js.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
    <title>1:初识 js</title>
    <!-- 类似 css 内嵌式,将 js 脚本直接书写在 script 标签 -->
    <script>
        /* js 注释与 java 完全一致,在此处不能书写 html 与 css

            window:BOM七对象之首,表示当前整个浏览器窗口
            onload:激发事件,所有的激发事件都以 on 开头,此处表示
            窗口载入无误,此处表示 窗口载入没有任何问题,之后激发一个
            函数 etoak
        */
        window.onload = etoak

        /*
            函数的格式

            function 函数名(实参){
                函数体
            }
        */
        function etoak() {
            console.log('这是无所不能的控制台打印!!!')
            //console.info('这是无所不能的控制台打印2!!!')
            //console.error('自己吓唬自己!!')
            /* 弹出一个对话框 */
            //alert('济南降温了!!!')
            /* 根据 id 从正文中获取一个元素节点,这个元素节点数据类型为
            对象 */
            let nodeBtn = document.getElementById('btn')
            /* 给元素节点绑定的一个事件,事件激发执行一个匿名函数 */
            nodeBtn.onclick = function () {
                //alert('测试!!!!')
                /* 获取表格的元素节点 */
                let nodeTb = document.getElementById('tb')
                //debugger
                /*
                    创建一个 tr 元素节点
                    <tr></tr>
                */
                let nodeTr = document.createElement('tr')
                /*
                    修改 tr 元素节点的结构
                    innerHTML:向元素中插入超文本(支持标签)
                    修改前: <tr></tr>
                    修改后: <tr><td>添加的一行</td></tr>
                */
                nodeTr.innerHTML = '<td>添加的一行</td>'
                /* 向现有元素中插入元素节点,后面的元素节点作为子元素
                如果现有元素中已经存在子元素,则插入的子元素,在现有子元素
                之后
                    <table id="tb" border="1px">
                        <tr>
                            <td>默认一行</td>
                        </tr>
                        <tr><td>添加的一行</td></tr>
                        <tr><td>添加的一行</td></tr>
```

```
                    </table>
                */
                nodeTb.append(nodeTr)
                /* 生成 0-255的随机整数 */
                let r = Math.floor(Math.random() * 256)
                let g = Math.floor(Math.random() * 256)
                let b = Math.floor(Math.random() * 256)
                /*
                    使用 js 渲染元素节点的样式
                    元素节点.style.样式名 = 样式值
                    样式名必须使用小驼峰格式,如果原先为连字符,则自己转换为
                    小驼峰
                    background-color => backgroundColor
                    font-size => fontSize
                */
                //nodeTb.style.backgroundColor = 'rgb('+r+','+g+','+b+')'
                //ES6新特性 模板字符串 `${要输出的数据}` 可以避免字符串繁琐的拼接
                nodeTb.style.backgroundColor = `rgb(${r},${g},${b})`
            }
        }

    </script>
</head>

<body>
    <!--
        js如何调错
        1:独立完成,不要依赖同位
        2:出错第一时间查看浏览器 f12 开发人员工具的控制台
        总结自己出过的错误
        3:多使用 console.log() 控制台打印,这个打印功能异常强大
        没有打印不了的数据
        4:添加断点 debugger
    -->
    <button id="btn">点我试试!</button>

    <table id="tb" border="1px">
        <tr>
            <td>默认一行</td>
        </tr>
    </table>
    <script>
        /*
            将 js 脚本书写在文末有以下好处
            1:优先加载 html 和 css
            2:保证模型创建完毕
        */
    </script>
</body>

</html>
```

2-引入js的方式和函数的执行.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>2:js引入方式和函数得执行</title>
</head>
<!-- 1:类似 css 行内式,不推荐使用 -->
<!-- <body onload="javascript:alert('测试!')"> -->
<!-- <body onload="alert('测试!')"> -->
<!-- 这种书写方式可以使用,这里函数 etoak 被调用 传递了
三个实参 -->

<body onload="etoak(100,true,'测试')">

    <script>
        /* 此处函数中三个为形参 可以随意书写 */
        function etoak(arg1, arg2, arg3) {
            console.log(arg1, arg2, arg3)
        }

        /* 函数要么绑定事件,要直接调用 */
        function test() {
            console.log('测试---------')
            let SEASON = 3
            if (SEASON >= 1 && SEASON <= 3) {
                console.log('春天来了')
            } else if (SEASON >= 4 && SEASON <= 6) {
                console.log('夏天来了')
            } else if (SEASON >= 7 && SEASON <= 9) {
                console.log('秋天来了')
            } else if (SEASON >= 10 && SEASON <= 12) {
                console.log('冬天来了')
            }

            /* 弹出对话框,内部可填写内容,如果不写就是后面的
            默认值 */
            let val = prompt('现在是几月?为什么这么冷?', 3)
            console.log(typeof +val, typeof (val - 0))
            /*
                =:赋值
                ==:比较两个数据是否一致,如果不是同一种类型,则
                转换为同一种类型,再进行比较
                ===:比较两个数据是否一致,如果数据类型不一致,立刻
                返回 false,如果一致,再进行比较
            */
            if (+val === 3) {
                console.log('春天就要来了,再等一周就热了!!')
            }

            /* 直接从页面输出,支持标签 */
```

```
            let i = 100
            let j = 'etoak'
            let x = true
            let y = 1.1
            let z = null

            document.write(`${i}+${j}=${i + j}<br>`)
            document.write(`${i}+${y}=${i + y}<br>`)
            document.write(`${i}+${z}=${i + z}<br>`)
            document.write(`${x}+${j}=${x + j}<br>`)
            document.write(`${y}+${j}=${y + j}<br>`)
            document.write(`${x}+${y}=${x + y}<br>`)
            document.write(`${y}+${z}=${y + z}<br>`)

            /* 使用 document.write 从页面输出 99 乘法表 */
            let str = ''
            for(let a = 1;a<=9;a++){
                for(let b = 1;b<=a;b++){
                    str += `${b}*${a}=${b*a}\t`
                }
                str += '<br>'
            }
            document.write(str)
        }

        test()

        /* 使用函数表达式来书写函数 */
        /*
            ES6新特性 箭头函数
            如果出现匿名函数,则可以省略 function 单词
            在参数之后添加 =>,如果只有一个参数,则小括号
            可以省略,如果没有或者一个以上参数,则小括号不能省略
            如果函数体只有一句,或者直接书写的 return 语句,则
            大括号 return 都可以省略
        */
        let test2 = () => console.log('thisistest2---------')

        //console.log(test2)
        test2()
    </script>
</body>

</html>
```

3-字符串函数.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>3:字符串函数</title>
</head>
```

```
<body>
    <script>
        let str = 'http://www.etoak.com'

        /*
            Java:
                String          length()
                Array           length
                List            size()
            Js:
                length
        */
        console.log(`字符串长度是---->${str.length}`)
        console.log(`将 e 替换为 a---->${str.replace('e','a')}`)
        console.log(`从左往右第一个 w 的索引是---->${str.indexOf('w')}`)
        console.log(`从左往右最后一个 w 的索引是---->${str.lastIndexOf('w')}`)
        console.log(`获取索引是 8 的字符---->${str.charAt(8)}`)
        /*
            substring(x,y)
                x:表示从索引值 x 开始截取(包含)
                y:截取到索引值 y(不包含)
            substr(x,length)
                x:表示从索引值 x 开始截取(包含)
                length:表示截取的长度
            以上两个函数参数可以只写一个,用法相同,不能是负值
            slice(x,y):与 substring 完全一致,但是 x,y可以设置
            负值,从右往左算
        */
        console.log(`截取字符串---->${str.substring(3,5)}`)
        console.log(`截取字符串---->${str.substr(3,5)}`)
        /* split():分割字符串,在括号内设置的分隔符处开始分割 */
        let knife = str.split('.')
        for(let i = 0;i<knife.length;i++){
            console.log(knife[i])
        }
    </script>
</body>
</html>
```

4-数组.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>4:数组</title>
</head>
<body>
    <script>
        /* 1:直接通过复杂类型提供的构造器创建 */
        const arr = new Array()
        arr[0] = 100
```

```
        arr[1] = true
        arr[2] = 'etoak'
        arr[3] = null
        /* ES5 for in 循环 */
        for(let index in arr){
            console.log(arr[index])
        }
        /* 2:直接创建数组 */
        const arr1 = [1,2,3,4,5]
        console.log(arr1)
        /* push():从数组结尾追加数据 */
        arr1.push(99)
        console.log(arr1)
        /* unshift():从数组头部添加数据 */
        arr1.unshift(0)
        console.log(arr1)
        /* shift():从数组头部删除数据 */
        arr1.shift()
        console.log(arr1)
        /* pop():从数组结尾删除数据 */
        arr1.pop()
        console.log(arr1)
        /* reverse():逆序排列 */
        arr1.reverse()
        console.log(arr1)
        /* sort():正序排列 */
        arr1.sort()
        /* splice():删除
            splice(起始索引,删除个数,替换的值)
        */
        arr1.splice(0,1,777)
        //[777, 2, 3, 4, 5]
        console.log(arr1)
        /*
            push() unshift() shift() pop() sort()
            reverse() splice()
            以上七个函数可以对原数组进行修改
            --------------------------------
            以下三个函数对原数组没有任何影响,会生成新的数组

            filter():过滤函数
            数组.filter(function(alias){
                return 过滤条件
            })
            只要符合过滤条件,则返回,最终生成一个新的数组,
            原数组不受影响
        */
        const newArr = arr1.filter( a => a>=4 )
        console.log(newArr)
        /*
            find():与 filter()几乎完全一致,仅仅返回
            第一个符合条件的
        */
        let val = arr1.find( a => a>=4 )
        console.log(val)
```

```
        /*
            map():多使用在对象中,如果使用在数组中,
            如果书写表达式,则仅仅返回一个数组,符合表达式
            则返回 true 不符合 返回 false
        */
        const newArr2 = arr1.map( a => a>=4 )
        console.log(newArr2)

        /* indexOf():查询字符的索引
        如果查询不到则返回-1 */
        let value = arr1.indexOf(8888)
        console.log(value)

        /*
            对于基本数据类型来说,let 赋值则是变量
            const 赋值为常量,数据不能存在任何变动,但是
            对于复杂类型来说,如果使用 let 则每次变动都要
            重新进行寻址,内存要开辟地址,而如果使用 const 则
            事先划定好了地址,不会进行频繁的寻址,非常节省资源
        */

        const demo = [1,2,3,4,5]
        /* ES6 for of 迭代 */
        for(let value of demo){
            console.log(value)
        }

        /* ES6 forEach */
        demo.forEach((a,index) =>
        console.log(`第${index}个元素是${a}`))
    </script>
</body>
</html>
```

常用的几个函数push() unshift() shift() pop() sort() reverse() splice() ||filter() find() map()

let const

对于基本数据类型来说,let 赋值则是变量

const 赋值为常量,数据不能存在任何变动,但是

对于复杂类型来说,如果使用 let 则每次变动都要

重新进行寻址,内存要开辟地址,而如果使用 const 则

事先划定好了地址,不会进行频繁的寻址,非常节省资源

for遍历的几种方式

```
/* ES5 for in 循环 */
for(let index in arr){
    console.log(arr[index])
}
        /* ES6 for of 迭代 */
        for(let value of demo){
            console.log(value)
        }

/* ES6 forEach */
demo.forEach((a,index) =>
    console.log(`第${index}个元素是${a}`))
```

5-对象.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>5:对象</title>
</head>
<body>
    <script>
        /* 1:使用对象字面量创建对象
            let|const 对象名 = {
                属性名:属性值,
                属性名:属性值,
                属性名:属性值,
            }
            属性名:没有引号,就表示一个属性,浏览器在解析时
            会将其解析为字符串,但是我们书写时,不要书写引号
            属性值:根据数据类型决定
        */
        let name = '胡桃'

        const person = {
            /*
                ES6新特性
                如果属性名与属性值恰好重名,则仅仅书写属性名
            */
            name,
            age:17,
            hobby:['逛街','恶作剧'],
            info:{
                address:'璃月',
            }
        }
        console.log(person)
        /*
            使用对象自带的两个迭代
```

```
            迭代所有的属性名
        */
        console.log(Object.keys(person))
        /* 迭代所有的属性值 */
        console.log(Object.values(person))
        /*
            获取某个属性值
            属性值 = 对象.属性名
            属性值 = 对象['属性名']
        */
        console.log(person.name,person['name']
        ,person.hobby[0],person.info.address)

        /* 删除属性和属性值 delete 对象.属性名
        如果删除成功则返回 true 失败 false */
        console.log(delete person.age)

        /*
            添加属性
            对象.属性名 = 属性值
        */
        person.myAge = 17

        /* 2:直接使用构造器构造 */
        const obj = new Object()
        obj.name = '张三'
        obj.age = 30
        console.log(obj)
    </script>
</body>
</html>
```

对象自带的两个迭代

```
let person = {
    name,
    age,
    hobby,
}
        /*
            使用对象自带的两个迭代
            迭代所有的属性名
        */
        console.log(Object.keys(person))

        /* 迭代所有的属性值 */
        console.log(Object.values(person))
```

添加对象属性

```
        /*
            添加属性
            对象.属性名 = 属性值
        */
        person.myAge = 17

        /* 2:直接使用构造器构造 */
        const obj = new Object()
        obj.name = '张三'
        obj.age = 30
        console.log(obj)
```

6-动态表格.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>6:动态表格</title>
    <link rel="stylesheet" href="./css/dynamicTable.css">
</head>
<body>
    <div class="container">
        <header class="header">
            <input type="text" placeholder="请输入姓名"
            autofocus autocomplete="off">
            性别: <input type="radio" name="gender" value="0"
            checked>女
            <input type="radio" name="gender" value="1">男
            归属地:
            <select name="location" id="location">
                <option value="济南">济南</option>
                <option value="青岛">青岛</option>
                <option value="淄博">淄博</option>
                <option value="济宁">济宁</option>
            </select>
            <input type="button" value="添加"
            onclick="add()">
        </header>
        <section class="main"></section>
    </div>
    <!-- 引入外部独立的 js 文件,注意存在先后顺序,从上往下先引
    外部资源 -->
    <script src="./js/datas.js"></script>
    <script>
        /* 函数 A:此函数用来组装动态表格 */
        function query(){
            let table =
            `<table class="tb">
                <thead>
                    <tr>
```

```
                        <th>序号</th>
                        <th>姓名</th>
                        <th>性别</th>
                        <th>居住地</th>
                        <th>操作</th>
                    </tr>
                </thead>
                <tbody>`

            empList.forEach((emp,index)=>{
                table +=
                `<tr>
                    <td>${index+1}</td>
                    <td>${emp.name}</td>
                    <td>${emp.gender===0?'女':'男'}</td>
                    <td>${emp['location']}</td>
                    <td><span style="cursor:pointer"
                    onclick="remove(${index})">删除</span></td>
                </tr>`
            })

            table += '</tbody></table>'

            document.querySelector('.main').innerHTML = table
        }

        query()

        /* 函数 B:此函数用来进行删除操作 */
        function remove(index){
            /* 弹出一个可选择式对话框 */
            if(confirm('您确定删除本条记录吗?')){
                /* 删除 */
                empList.splice(index,1)
                /* 回过头来重新查询,这种操作被称之为:回显
                这里复用函数 A */
                query()
            }
        }

        /* 函数 C:此函数用来进行添加操作 */
        function add(){
            /*
                1:获取用户姓名
                trim():去掉字符串两侧空格
            */
            let name =
            document.querySelector('input[type=text]').value.trim()

            if(!name){
                alert('请输入有效内容...')
                return
            }

            /* 2:获取性别 */
```

```
            /* 设置性别默认值 */
            let gender = 0

            let nodeRadios =
            document.getElementsByName('gender')

            nodeRadios.forEach(nodeRadio=>{
                /* 如果某一个被选中了 */
                if(nodeRadio.checked){
                    gender = (nodeRadio.value-0)
                }
            })

            /* 3:获取归属地 */
            let location =
            document.getElementById('location').value

            /* 添加进数组 */
            empList.push({
                /* 注意此处由于没有真正的主键,因此直接使用数组
                长度+1 这是非常不标准的书写方式 */
                id:empList.length+1,
                name,
                gender,
                location,
            })

            /* 回显 */
            query()
        }

    </script>
</body>
</html>
```

>

---

substring-substr-slice区别

---

day05

7-分离运算符.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>7:分离运算符</title>
</head>
<body>
    <script>
        /*
            ES6 新特性 分离运算符 可以使用在 字符串 数组 对象中
            ...:用三个点表示分离运算符
            就是将...后面的字符串 数组 或者对象 分离
            又被称之为 spread syntax 分离语法

            1:字符串
                将字符串分为一个一个字符
        */
        let str = 'etoak'
        console.log(...str,[...str])
        /*
            2:数组
                将数组中的数据从中括号中取出来
        */
        const arr = [1,2,3,4,5]
        console.log(...arr)
        const arr2 = [...arr,99,true]
        console.log(arr,arr2)

        /*
            const test = function(a,b,c,d,e){
                return a+b+c+d+e
            }
        */
        const test = (a,b,c,d,e) => a+b+c+d+e
        console.log(test(...arr))

        /* 3:对象
            ...对象:就是将对象中的属性名 属性值从对象中取出
        */
        const obj = {
            name:'张三',
            age:20,
        }

        const obj2 = {...obj,address:'济南',}
        console.log(obj,obj2)


    </script>
</body>
</html>
```

8-深拷贝和浅拷贝.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>8:浅拷贝和深拷贝</title>
</head>
<body>
    <script>
        /*
            深拷贝:拷贝之后,修改其中任意一个,另一个不受影响,即所谓,你变我不变
            浅拷贝:拷贝之后,修改其中任意一个,则另一个也会被修改,即所谓,你变我也变

            1:等号
                基本数据类型,则是深拷贝
                复杂数据类型,则是浅拷贝
        */
        let str1 = 'etoak'
        let str2 = str1
        console.log(str1,str2)
        str2 += '!!!!'
        console.log(str1, str2)

        const arr1 = [1,2,3,4,5]
        const arr2 = arr1
        console.log(arr1,arr2)
        arr2.push(99)
        console.log(arr1, arr2)
        /*
            2:分离运算符 Array.from() slice() Object.assign()
                对于基本数据类型,分离运算符肯定是深拷贝
                对于复杂类型,则要具体分析
                    如果修改的表层数据为基本类型,则深拷贝
                    如果修改的表层数据为复杂类型,则浅拷贝
                注意在打印时,对于复杂类型,点开箭头要查看复杂类型
                内部结构时,浏览器会重新寻址进行一次查询,保证数据是最新
                的,因此这里头两次打印 也会变为最新的 '学习'
        */
        const obj1 = {
            name:'胡桃',
            age:17,
            hobby:['逛街','恶作剧'],
            info:{
                address:'济南',
            },
        }

        const obj2 = {...obj1}
        console.log(obj1,obj2)
        obj2.name = '甘雨'
        obj2.age = 26
```

```
        obj2.hobby[0] = '学习'
        console.log(obj1, obj2)

        /*
            3:  JSON.parse(JSON.stringify())
            肯定是深拷贝,又名万能转换器
        */

    </script>
</body>
</html>
```

---

jQuery（NODE | NPM）

Write Less Do More

> ***由美国人John Resig 独立发布Javascript前端类库，jQuery可以使用更简短的代码通过对js的封装实现更多功能，完全借鉴了css选择器机制，可以通过许多手段更快速更精确的拿取元素，实现了代码链机制，基本解决了浏览器差异性问题***

使用jQuery类库的步骤

1. ***安装Node***
2. ***安装Node之后NPM平台就安装成功了***
3. ***进入终端 输入 npm -v 可以查看npm平台版本号***
4. ***使用npm初始化本工程，相当于用npm接管我们的前端工程***
5. ***npm init -y***
6. ***更改国内节点 npm set registry https://registry.npm.taobao.org/***
7. ***下载Query依赖 npm i jquery***
8. ***在 node_modules中找到jquery/dist/jquery.js 引入页面即可使用 dist:distribution的简写，表示发布版***

1-Jquery选择器.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>1:jquery选择器</title>
</head>
<body>
    <div class="container">
        <span>段落之前的 span</span>
        <p id="pra">
            <span>段落内部的 span</span>
        </p>
        <span>段落之后紧邻的 span</span>
        <span>段落之后不紧邻的 span</span>
    </div>
```

```
    <!--
        引入 jQuery 依赖,注意一定要先引入,再写我们自己的
        dist:distribution的简写,表示发布版
    -->
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script>
        /*
            # jQuery 元素与 js 的元素节点是同一种元素吗?如果不是为什么?两者
            如何进行转换?

            <span id="etoak">测试</span>

            js:
                let nodeSp = document.getElementById('etoak')
                <span id="etoak">测试</span>

            jQuery:
                let $sp = $('#etoak')
                [<span id="etoak">测试</span>]

            jQuery元素与 js 元素节点不是同一种元素,它是对 js 节点的一个
            轻度的封装,jQuery元素只能使用自己的 函数 方法 属性 同样 js 节点
            也只能使用自己的函数 方法 属性,两者不能通用

            nodeSp.innerHTML = XXXX
            $sp.html(XXXX)

            两者如何转换
            jQuery => js节点
            $sp.get(0)
            $sp[0]

            js节点 => jQuery
            $(nodeSp)
            $(document)
            $(location)
            $(history)
            ------------------------------------------
            jQuery选择器
            1:  $('tagName')
            2:  $('.class')
            3:  $('#id')
            4:  $('tagName.class')
                $('tagName#id')
            5:  $('sel1,sel2,selN')
            6:  $('sel1 sel2 selN')
            7:  $('*')
            8:  $('[属性名=属性值]')
            9:  $('[属性名^=属性值开头]')
            10: $('[属性名$=属性值结尾]')
            11: $('[属性名*=包含属性值]')

            12: $('sel1 > sel2')
            获取特定元素的子元素,注意仅仅只能获取子元素
```

```
            使用 jquery 渲染样式
            sel.css('样式名','样式值')
            sel.css({
                样式名:样式值,
                样式名:样式值,
                样式名:样式值,
            })
            注意没有引号的写法中,样式名必须使用小驼峰格式
        */
        $('.container > span').css({
            backgroundColor:'coral',
            color:'whitesmoke',
        })
        /*
            13: $('sel1 + sel2')
                以下三个条件必须全部满足
                a:向下选取
                b:必须紧邻
                c:互为兄弟
        */
        $('p#pra + span').css('border','solid 2px purple')
        /*
            14: $('sel1 ~ sel2')
                以下两个条件必须全部满足
                a:向下选取
                b:互为兄弟
        */
        $('p#pra ~ span').css('font-style','italic')
    </script>
</body>
</html>
```

jQuery 元素与 js 的元素节点是同一种元素吗?如果不是为什么?两者如何进行转换?

jQuery元素与 js 元素节点不是同一种元素,它是对 js 节点的一个

轻度的封装,jQuery元素只能使用自己的 函数 方法 属性 同样 js 节点

也只能使用自己的函数 方法 属性,两者不能通用

2-事件动作.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>2:事件动作</title>
    <style>
        .red{
            background-color: crimson;
        }
```

```
        .blue{
            background-color: cornflowerblue;
        }
    </style>
</head>
<body>
    <button>你点我试试!</button>

    <input type="button" value="添加一行">

    <table border="1px" class="tb">
        <tr>
            <td>默认的一行</td>
        </tr>
    </table>

    <div style="width:100px;height:100px;border:solid 1px #ddd">
        测试域
    </div>

    <ul class="myul">
        <li>列表1</li>
        <li>列表2</li>
        <li>列表3</li>
        <li>列表4</li>
        <li>列表5</li>
    </ul>

    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script>
        /* 表示全文加载无误之后执行 ready()内部的回调函数
        ready()非常类似 js 中得 window.onload,不同的是
        ready()可以书写多个,从上往下执行,window.onload 全文只能书写一次
        ready()还对内部的脚本起到的保护的作用,效果等同于 js 中的
        自调用函数 */
        $(document).ready(function(){
            /* 1:给特定 jQuery 元素绑定多个事件
                sel.on('事件1 事件2 事件N',function(){

                })

                $(':contains(text)'):获取包含特定文本的
                指定元素
                jQuery的事件就是 js 的事件去掉 on 前缀
                mouseout:鼠标滑出
                mouseover:鼠标滑过
            */
            $('button:contains(点我)').on('click mouseout',
            function(){
                /*
                    $(this):表示已经获取的元素列表
                    attr(属性名,属性值):把元素的属性名更改为属性值
                */
                $(this).attr('disabled',true)
            })
```

```
67
68              /*
69                  2:给元素绑定特定事件
70                  sel.事件(function(){
71
72                  })
73              */
74              $(document).dblclick(function(){
75                  console.log('谁双击我了!!!')
76              })
77
78              $('input[type=button]').click(function(){
79                  /* append():向元素中追加子元素 */
80                  $('.tb').append('<tr><td>添加的一行</td></tr>')
81              })
82
83              /* 3:鼠标滑过元素
84
85                  sel.hover(
86                      function(){
87                          //鼠标滑过执行此函数
88                      },
89                      function(){
90                          //鼠标滑出执行此函数
91                      }
92                  )
93              */
94              $('div:contains(测试域)').hover(
95                  function(){
96                      /*
97                          addClass():添加 class 属性
98                          removeClass():删除 class 属性
99                          toggleClass():如果存在则删除,如果不存在则添加
100                         一个元素可以添加多个class
101                         注意 id 无上面任何操作
102                     */
103                     $(this).removeClass().addClass('red')
104                 },
105                 function(){
106                     $(this).removeClass().addClass('blue')
107                 }
108             )
109
110             /*
111                 4:迭代
112                 sel.each(function(index,alias){
113
114                 })
115             */
116             $('.myul li').each(function(index){
117                 if(index===3){
118                     $(this).addClass('red')
119                 }
120             })
121         })
```

```
    </script>
</body>
</html>
```

第一周复习

1.请简单作答js中Function是否肯定存在返回值,如果不是为什么?如果存在返回什么?(10分)

肯定存在返回值,要么是return的数据要么是undefined

肯定存在返回值，要么是return的数据 要么是undefined

2.请简述JavaScript中基本数据类型有哪些?复杂数据类型有哪些(仅写出类型即可,单词拼写错误不得分10分)

基本数据类型 string number null undefined boolean symbol bigint

复杂数据类型 Object Array Function RegExp Math Date

基本数据类型 string number null undefined boolean symbol bigint
复杂数据类型 Object Array Function RegExp Math Date

3.请说出flex弹性布局中,什么是弹性容器?什么是项目?什么是主轴?什么是交叉轴? (请画图表示10分)

见课件pdf图片

4.请画图表示什么是盒子模型(共14个参数,单词拼写错误 少些 漏写均不得分10分)

见课件图片

5.请详细简述CssFlex弹性盒子中 flex:1 是什么意思? (注意单词拼写错误不得分10分)

注意此参数仅仅针对项目,完全版书写如下,flex:flex-grow flex-shrink flex-basis;

默认值 flex:0 1 auto;

flex-grow:默认是0,如果弹性容器具有剩余空间,则本项目也不放大,如果更改为1,则如果弹性容器具有剩余空间,则本项目放大装满整个容器

flex-shrink:默认是1,如果弹性容器缩小,剩余空间减少,则项目会随之减小,如果设置为0,则项目不会变小,溢出容器

flex-basis:默认是auto,就是设置本项目的盒子模型参数,但是优先级较本身的设置低,所以这里一般不设置盒子模型,使用默认值auto,那么根据以上三个参数最终书写为

flex:0 1 auto;默认值可以省略,所以更改为 flex:0;

我们现在需要项目的特性是如果容器存在剩余空间,则项目放大,如果没有剩余空间,则项目缩小,所以这里要将0改为1

最终 flex:1;

6.请说出三种Css可以使用页面整体居中的单位书写方式(10分)

px 像素

%百分比 注意必须设置 html,body{height:100%}

vw,vh

px:像素,固定单位

%:百分比,与自己的父元素进行比较

vw:当前视口的宽度 vh:当前视口的高度,

7.现有数组const arr = [3,16,22,78,55,0],请写出脚本获取一个新数组,要求数据必须大于等于40,原数组不能受到任何影响,请写出脚本;要求返回第一个大于等于70的数字,请写出脚本(两问各5分,仅写出关键脚本即可,不需要书写函数调用等,单词拼写错误不得分)

const newArr = arr.filter(a => a >=40)

let val = arr.find(a => a>=70)

8.根据以下要求书写脚本: 使用对象字面量创建Js对象,对象名为 stu ,属性名分别为 username realname gender email,属性值分别为 'tom' '张三' 0 'et@et.com',方法 run(),调用方法在控制台输出 'tom的真实姓名是张三,邮箱地址为et@et.com ,请写出关键脚本(单词拼写错误不得分10分)

const stu = {

username:'tom',

realname:'张三',

gender:0,

email:'et@et.com',

run(){

console.log(`${this.name}的真实姓名是${this.realname},邮箱地址是${this.email}` *)*

}

9.请说出Js数组中七个修改数组的函数的作用(仅写出函数名,简述其作用即可,单词拼写错误不得分10分)

push():数组末尾增 unshift():数组头部增 shift():数组头部删 pop():数组尾部删 sort():正序排列 reverse():逆序排列splice():根据传入的索引 删除或者替换

10.请说出几种你所知道的ES6规范新特性,不得少于5个(10分)

for of

forEach

let const

箭头函数

模板字符串

对象属性简略写法

symbol

方法简略写法

document.querySelector()

document.querySelectorAll()

NPM使用指南

> *NPM使用指南.pdf 仔细查阅,要求能够非常熟练的使用 npm 初始化工程,下载依赖,能够分清什么是 生产依赖 什么是 开发依赖*

```
##初始化项目
命令: npm init
##全部由系统决定，不再自己填写相关信息
命令: npm init -y
##查看全局安装的目录
命令: npm root -g
##修改全局安装的目录地址
命令: npm config set prefix "c:\XXX"
##查看全局已经安装的模块
命令: npm list -g
##生产环境安装 || 生产环境安装将依赖信息添加进package.json文件的dependencies中（生产环境依赖）
命令:  npm i modelName@版本号 --save
      npm i modelName@版本号 -S
##开发环境安装 || 开发环境安装将依赖信息添加进package.json文件的devDependencies中（开发环境依赖）
命令:  npm i modelName@版本号 --save-dev
      npm i modelName@版本号 -D
##
##其他常用指令
##
命令: npm install
##如果在现成的工程中只有package.json文件，而没有node_Moduels文件夹，这是我们需要通过此命令下
##载这些js库，输入此命令后，会自动根据package.json中的依赖来下载js类库和其它依赖

命令: npm list
##查看本地安装的所有模块

命令: npm view modelName version
##查看远程模块最新版本
##eg: npm view jquery version

命令: npm uninstall modelName
##卸载局部模块

命令: npm uninstall -g modelName
```

```
##卸载全局模块

命令: npm get registry
##查看当前使用的镜像

命令: npm config set registry https://registry.npm.taobao.org
##更换淘宝镜像，网络状况较为稳定
```

---

day06

理解 json 如何进行转换

1.JSON字符串转换为对象或数组：使用JSON.parse()方法将JSON字符串转换为JavaScript对象或数组

2.对象或数组转换为JSON字符串：使用JSON.stringify()方法将JavaScript对象或数组转换为JSON字符串

3.JSON数据转换为XML：使用第三方库入xml2js或xmlbuilder等将JSON数据转换为XML格式

4.XML数据转换为JSON：同样使用第三方库将XML数据转换为JSON格式

5.JSON数据转换为其他格式CSV、YAML等：同样适用第三方库将JSON数据转换为其他格式

需要根实际需求选择不同的转换方式

```
4.理解什么是异步什么是同步
```

同步和异步是指程序执行的方式

1.同步执行指程序按照顺序执行每一条语句，每一条语句执行完成后再执行下一条语句。在执行一条语句的过程中，如果遇到阻塞，整个程序会暂停等待，直到这条语句执行完成才会继续执行下一条语句

2.异步执行指程序不会按照顺序执行每一条语句，而是在执行某些语句时不会阻塞程序的执行，而是继续执行下一条语句。当异步操作完成后，程序会通过回调函数等方式通知程序异步操作已完成，程序再执行回调函数中的代码。

3.异步执行的好处是可以提高程序的并发性和响应性，避免程序在等待某些操作完成是被阻塞。常见的异步操作包括网络请求、文件读写、定时器等

什么是JSON

> *JavaScript Object Notation*
>
> *一种轻量级的数据交换格式,是目前首选的链接前后端的纽带*
>
> *JSON 其实就是一种特殊格式的字符串,在后端(Java)可以被 System.out.print直接打印,在前端(JS)可以被 console.log()或者 alert()直接打印*

JSON 的格式

- ***map格式***

```
{"key":value,"key":value,"key":value,"key":value}
"key":表示键,必须使用双引号,不得使用单引号,反引号,无引号,本质上肯定是个字符串
value:表示值,根据数据类型决定,任意类型都可以封装,包括json 在内,只有封装后端时,不能是 java.sql.Date()
```

- ***数组格式***

```
[value,value,value,value,value,value]
value:表示值,根据数据类型决定,任意类型都可以封装,包括json 在内,只有封装后端时,不能是 java.sql.Date()
```

前端如何处理后端发送过来的 json

后端发送 json 给前端之后,前端获取 json 可以通过 console.log()或者 alert()直接打印,但是仅此而已,无法很方便的获取内部的数据

```
//以下为接受过来的 json 数据
{
    "id":1,
    "name":"胡桃",
}
//在 js 中提供了转换器,可以将 json 字符串转换为前端使用的数据类型
const obj = JSON.parse(json数据)
/*
    转换规则:
        1:如果是 map 格式的 json,则会被转换为 js 对象
        其中键被转换为对象的属性名,值被转换为对象的属性值
        2:如果是 数组 格式的 json,则会被转换为 js 数组
        内部数据全部转换为 js 类型
*/
//转换结束之后
const obj = {
    id:1,
    name:'胡桃',
}

let name = obj.name
```

前端如何发送 json 给后端

```
const obj = {
    id:1,
    name:'胡桃',
}
//前端如果要发送数据给后端,则必须要转换为 json 使用内置的转换器即可,转换之后对象变为 map 格式的 json,数组变为数组格式的 json
let text = JSON.stringify(obj)
/*
    {
        "id":1,
        "name":"胡桃"
    }
*/
```

注意!

- ***JSON.parse(JSON.stringify())如果嵌套在一切又被称之为万能转换器,肯定是深拷贝,但是对一些数据有限制,谨慎使用***
- ***注意 当后端发送json 给前端时,很多前端框架,会自动使用JSON.parse()进行转换,因此我们不需要自己转换了,jQuery Mock Vue,如果我们不清楚数据是否被转换了,则直接打印即可***

什么是异步

> ***在前端中,异步技术和 Node技术得诞生,正式标志的 web2.0 革命的到来,前端终于可以和后端分庭抗礼,从而诞生了著名的前后端分离思想***

- ***同步***
  - ***用户发出请求之后,必须等待响应的返回,如果响应迟迟没有返回,则用户只能等待在那里,什么也不能干,当响应返回,整个页面会被完全刷新***
  - ***链接提交,表单提交都属于同步请求***
- ***异步***
  - ***用户发出请求,不需要等待响应的返回,用户可以继续自己的操作,当响应返回,整个页面不会完全刷新,而仅仅部分刷新***
  - ***本质上 js 帮我们发送请求,js 帮我们接受响应,用户不必自己发送请求,接受响应了***
  - ***异步技术的组成***
    - ***JavaScript 核心语法***
    - ***xml 用来传输 封装数据,但是目前已经被 json 替代***
    - ***json 已经替代了 xml,作为前后端数据交互的首选***
    - ***html 渲染结构***
    - ***css 渲染样式***
- ***注意***
  - ***如果存在一个场合,同步和异步同时出现,则异步根本不执行***

3-异步注册.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>3:异步注册</title>
    <link rel="stylesheet" href="./css/global.css">
    <link rel="stylesheet" href="./css/register.css">
    <link rel="icon" href="./assets/favicon.ico">
</head>
<body>
    <div class="reg-container">
        <form class="reg-form">

            <h1>用户注册</h1>

            <input type="text" name="username"
            placeholder="请输入用户姓名" autofocus
            autocomplete="off" required
            pattern="[a-zA-Z0-9_]{6,10}"
            title="姓名在 6 到 10 位之间只能是英文数字下划线组成">
            <span class="username-msg"></span>

            <input type="password" name="password"
            placeholder="请输入用户密码"
            autocomplete="off" required
            pattern="[a-zA-Z0-9_]{6,32}"
            title="密码在 6 到 32 位之间只能是英文数字下划线组成">
            <span class="password-msg"></span>

            <input type="text" name="realname"
            placeholder="请输入真实姓名"
            autocomplete="off" required>
            <span class="realname-msg"></span>

            <input type="text" name="email"
            placeholder="请输入邮箱地址" required
            pattern="[A-Za-z0-9\u4e00-\u9fa5]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+"
            title="请输入标准邮箱格式">

            <input type="tel" name="phone"
            placeholder="请输入电话号码" required
            pattern="(?:(?:\+|00)86)?1(?:(?:3[\d])|(?:4[5-79])|(?:5[0-35-9])|(?:6[5-7])|(?:7[0-8])|(?:8[\d])|(?:9[189]))\d{8}"
            title="请输入标准中国大陆标准手机号码">

            <div>
                性别: <input type="radio" name="gender" value="0"
                checked>男
                <input type="radio" name="gender" value="1">女
```

```
            </div>

            <div>
                权限: <input type="radio" name="role" value="0"
                checked>用户
                <input type="radio" name="role" value="1">管理员
            </div>

            <div>
                <input type="submit" value="注册">
                <input type="reset" value="取消">
            </div>
        </form>
    </div>
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script>
        /*
            $(document).ready(function(){
                以下为 此方法的 语法糖
            })
        */
        $(function(){
            /* 1:获取用户姓名 */
            let username = ''

            /* 此函数用来进行验证 */
            function checkUsername(){
                /*
                    $.trim():去掉字符串两端空格
                    val():获取 jQuery 元素的 value 属性,一般使用在
                    表单中
                */
                username = $.trim($('input[name=username]').val())
                let $sp = $('span.username-msg')
                if(username.length<6||username.length>10){
                    $sp.html('<img src="./assets/wrong.png">用户姓名必须在 6
到 10 位之间')
                    .css('color','red')
                    return false
                }
                $sp.html('<img src="./assets/right.png">用户姓名符合要求')
                .css('color', 'green')
                return true
            }

            /* 失去焦点激发验证 */
            $('input[name=username]').blur(function(){
                checkUsername()
            })

            /* -------------------------- */
            /* 2:获取用户密码 */
            let password = ''

            function checkPassword(){
```

```
                password = $.trim($('input[name=password]').val())
                let $sp = $('span.password-msg')

                if(password.length<6||password.length>32){
                    $sp.html('<img src="./assets/wrong.png">用户密码必须在 6 到 32 位之间')
                    .css('color','red')
                    return false
                }
                $sp.html('<img src="./assets/right.png">用户密码符合要求')
                .css('color', 'green')
                return true
            }

            $('input[name=password]').blur(function(){
                checkPassword()
            })
            /* ------------------------------ */
            /* 3:获取真实姓名 */
            let realname = ''

            /* 此函数用来进行验证 */
            function checkRealname() {
                realname = $.trim($('input[name=realname]').val())
                let $sp = $('span.realname-msg')
                if (realname.length < 6 || realname.length > 10) {
                    $sp.html('<img src="./assets/wrong.png">真实姓名必须在 6 到 10 位之间')
                        .css('color', 'red')
                    return false
                }
                $sp.html('<img src="./assets/right.png">真实姓名符合要求')
                    .css('color', 'green')
                return true
            }

            /* 失去焦点激发验证 */
            $('input[name=realname]').blur(function () {
                checkRealname()
            })
            /* ------------------------------ */
            /* 4:获取邮箱 */
            let email = ''

            $('input[name=email]').blur(function(){
                email = $.trim($(this).val())
            })

            /* ------------------------------ */
            /* 5:获取电话 */
            let phone = ''

            $('input[name=phone]').blur(function(){
                phone = $.trim($(this).val())
            })
```

```
158
159             /* ------------------------------ */
160             /* 6:获取性别 */
161             let gender = 0  //默认是男 0
162             $('input[name=gender]').click(function(){
163                 gender = $(this).val()
164             })
165             /* ------------------------------ */
166             /* 7:获取权限 */
167             let role = 0 //默认是用户 0
168             $('input[name=role]').click(function(){
169                 role = $(this).val()
170             })
171
172             /* 提交表单,注意这里默认是同步请求
173                 submit:事件表单提交时,在 js 里是 onsubmit
174             */
175             $('.reg-form').submit(function(){
176                 if(checkUsername()&&checkPassword()&&checkRealname()){
177                     /* 使用 jQuery 内置的异步模块来进行异步注册 */
178                     $.ajax({
179                         /* 1:发送异步的目的地 */
180                         url:'http://db.etoak.com:9527/testUser/add',
181                         /* 2:发送异步的类型 */
182                         type:'post',
183                         /* 3:是否使用异步,默认使用,可以不写,注意如果使用异步
184                         则代码永远不会跳出回调函数 */
185                         async:true,
186                         /* 4:设置要发送给后端的数据
187                         注意这里是 MIME 类型,供浏览器进行解析
188                             text/html       html
189                             text/xml        xml
190                             text/css        css
191                             image/jpeg      jpeg
192                             application/json    json
193                             application/plain   字符串
194                         */
195                         contentType:'application/json',
196                         /* 5:要给后端发送的数据 */
197                         data:JSON.stringify({
198                             username,
199                             password,
200                             realname,
201                             email,
202                             phone,
203                             gender,
204                             role,
205                         }),
206                         /* 6:返回的类型
207                         在 ajax 中支持五种类型的返回值处理
208                         text html xml script json
209                         注意如果是 json,则自动转换为 js 对象或者数组,
210                         不需要我们自己 JSON.parse() */
211                         dataType:'json',
212                         /* 7:如果一切正常的回调函数
```

```
                        response:形参 表示后端返回的数据 */
                        success(response){
                            console.log(response)
                            if(response.flag){
                                /* 主动跳转到登录页,准备开始进行登录 */
                                $(location).attr('href','./4-异步登录.html')
                            }
                        },
                    })
                }

                /* 屏蔽激发事件,这里会导致同步提交被阻止,因为我们要发送异步,如果不阻止,则异步和同步出现在同一个场合,则异步根本不执行 */
                return false
            })

        })
    </script>
</body>
</html>
```

4-异步登录.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>4:异步登录</title>
    <!-- 引入全局 css -->
    <link rel="stylesheet" href="./css/global.css">
    <!-- 引入登录的 css -->
    <link rel="stylesheet" href="./css/login.css">
    <!-- 引入图标 -->
    <link rel="icon" href="./assets/favicon.ico">
</head>
<body>
    <div class="login-container">

        <form class="login-form">
            <h1>用户登录</h1>

            <input type="text" name="username"
            placeholder="请输入用户姓名" required autocomplete="off"
            autofocus>
            <span class="username-msg"></span>

            <input type="password" name="password"
            placeholder="请输入用户密码" required autocomplete="off">
            <span class="password-msg"></span>

```

```
            <div>
                <input type="submit" value="登录">

                <input type="reset" value="取消">

                <input type="button" value="注册">
            </div>
        </form>
    </div>
    <script src="../node_modules/jquery/dist/jquery.js"></script>
    <script>
        /* document.querySelector('input[type=button]')
        .onclick=()=>{
            location.href = './12-圣杯布局.html'
        } */
        $(function(){
            /* 此函数用来控制注册跳转 */
            goReg()
            /* ------------------------ */
            /* 用户姓名和用户密码的验证 */
            let username = ''

            function checkUsername(){
                /* $(':text'):获取单行文本输入框 */
                username = $.trim($(':text').val())
                let $sp = $('span.username-msg')
                if(username.length<6||username.length>10){
                    $sp.html('<img src="./assets/wrong.png">用户姓名必须在 6 到 10 位之间')
                    .css('color','red')
                    return false
                }
                $sp.html('<img src="./assets/right.png">用户姓名符合要求')
                    .css('color', 'green')
                return true
            }

            $(':text').blur(function(){
                checkUsername()
            })

            let password = ''

            function checkPassword(){
                /* $(':password'):获取单行文本密码框 */
                password = $.trim($(':password').val())
                let $sp = $('span.password-msg')
                if(password.length<6||password.length>32){
                    $sp.html('<img src="./assets/wrong.png">用户密码必须在 6 到 32 位之间')
                    .css('color','red')
                    return false
                }
                $sp.html('<img src="./assets/right.png">用户密码符合要求')
                .css('color', 'green')
```

```
                return true
            }

            $(':password').blur(function(){
                checkPassword()
            })

            $('.login-form').submit(function(){
                if(checkUsername()&&checkPassword()){
                    $.ajax({
                        url:`http://db.etoak.com:9527/testUser/login?username=${username}&password=${password}`,
                        type:'get',
                        dataType:'json',
                        success(response){
                            console.log(response)
                            if(response.data){
                                $(location).attr('href','./5-圣杯布局.html')
                            }
                        }
                    })
                }
                return false
            })

        })

        function goReg(){
            $('input[type=button]').on('click',function(){
                $(location).attr('href','./3-异步注册.html')
            })
        }
    </script>
</body>
</html>
```

day07

Vue的工作原理简化版(不涉及生命周期 数据劫持 数据代理)

浏览器从上往下解析模板,插值语法与指令语法,浏览器根本

无法解析,因此这些指令全部出现在标签上,插值语法也直接

显示在页面上,此时浏览器解析结束之后,这种状态称之为虚拟

DOM

之后 Vue 实例开始根据 el 解析相应的模板,将数据插入插值语法

激活指令绑定功能,最终解析结束之后,页面中直接显示初始化的数据

此时称之为真实 DOM,我们最终见到的就是覆盖虚拟 DOM 之后的

真实 DOM

Vue 渐进式框架

Vue发展历史

> *2013年底，时任Google原型设计的中国无锡人尤雨溪在Angular的启发下开始创建了一个十分轻巧的库，VUE.js2014年抱着"我花了这么多时间，不能只有我一个人用，我应该和别人分享，他们也会感觉到 Vue 的好处，他们也会喜欢上 Vue 的。"的简单想法，尤雨溪将这个自己研发的库放到了GitHub上之后开始在Patreon平台众筹全职开发 Vue 的资金。2015年10月1.0版本正式发布，一举成功。成为s中最为著名的渐进式框架Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。在所有的框架中，Vue 可能与 React 最像，但从更广泛的意义上说，在所有框架中，尤雨溪创造了一个概念叫"渐进的框架"。因为 Vue 的核心组成只是数据绑定和组件，和 React 差不多。它只是解决了一小部分很重要的痛点。与 React 相比，Vue 可能更简单易用，只知道一些 HTML，JavaScript 和 CSS 知识的人都可以很快入门 Vue。在VUE的官方文档中是这样介绍它的简单小巧的核心 渐进式技术栈，VUE的大小仅有17k，所谓渐进式Progressive，是指你可以一步步，有阶段的来使用Vue.js，不必一开始就使用所有东西VUE在设计上使用了MVVM（Model-View-View-Model）模式，MVVM模式是由经典的MVC衍生来的，当View视图层发生变化时，会自动更新的ViewModel（视图模型），反之亦然，View和ViewModel之间通过双向绑定（data-binding）建立联系*

- *Vue流行的几个重要原因*
  - *Vue.js体积极小，编码简洁优雅，运行效率高*
  - *没有任何dom操作（对比jQuery），大大提高了网站的应用程序的开发效率*
  - *在进行SPA（Single Page Application单页面应用程序）具有巨大的优势*
- *其它流行的s框架*
  - *jQuery 诞生于2005年，曾经的王者，现在属于白垩纪时代的产物，Js各种类库的入门，但是由于要操作Dom，所以显得非常繁琐，目前使用依然较多*
  - *Angular 2009年诞生，个人开发使用，后来被Google收购了，目前体验越来越差，正在走下坡路*
  - *React 2013年五月开源的，起源于Facebook的内部项目，多使用在大型企业，Vue就是借鉴了React的基础上而诞生*
- *BATE级别的公司*
  - *React > Angular = Vue >> jQuery*
- *中小型公司*
  - *Vue >> React > Angular = jQuery*

`Vue不支持IE8及以下版本，由于Vue使用了ES5的特性，IE8不支持ES5新特性`

`Vue3.0于2021年4月5日正式宣布不再支持IE11`

Vue 前端八股文

1. *什么是单向绑定?什么是双向绑定?*

- *单向绑定*
  - *修改 Vue 实例中data初始化的数据,则页面模板中的数据随之发生更改,两者绑定在一起,则称之为单向绑定,也叫数据实现了可响应式,插值语法和指令语法都自带单向绑定功能*
- *双向绑定*
  - *修改页面模板中的数据,则管理模板的 Vue 实例中 data 中的数据随之发生更改,这称之为双向绑定,默认情况下只有 v-model 支持双向绑定,其它情况可以使用计算属性实现双向绑定*

1-初始Vue.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>1:初识Vue</title>
</head>
<body>
    <!-- 设置一个元素(可随意设置),这个元素及其内部的元素被称之为
    模板(template) -->
    <div id="app">
        <!-- 在 Vue 中存在两种基本语法
                1:插值语法
                    <tagName>{{ 初始化的值 }}</tagName>{{ 初始化的值 }}
                    又被称之为大胡子表达式,专门使用在标签外
                    不能显示 null 和 undefined,无法进行复杂的业务逻辑,
                    只能简单的显示初始化的数据,自带单向绑定功能
        -->
        {{ msg }}
        {{ count/10 }}
        {{ flag?'IOS':'Android' }}
        {{ text.split('.').reverse().join('@') }}
        {{ info.address }}
        {{ val1 }}
        {{ val2 }}
        <!--
                2:指令语法
                    <tagName v-指令名="绑定的值"  />
                指令语法中的各种指令是 Vue 基本语法中最大的一个模块
                我们要熟练使用各种 vue 指令,这些指令必须书写在标签上
                不能使用在标签外,指令名之前肯定存在 v-前缀,可以不绑定值

            v-html:向元素中插入超文本,相当于 js 中的 innerHTML
            v-text:向元素中插入文本,相当于 js 中的 innerText
        -->
        <p v-html="value"></p>
        <p v-text="value"></p>
        <!--
            v-once:一次性插值,之后不再存在绑定效果
        -->
        <p v-once>{{ count }}</p>
        <!-- v-model:专门使用在表单项中,除了单向绑定之后,还自带
        双向绑定功能,这也是唯一自带双向绑定功能的指令元素 -->
        <input type="text" v-model="msg">

        <!-- v-model:不能取反,如果使用在复选框中,则
        如果v-model 后面为真值,则勾选,假值不勾选
            如果主动勾选,则 v-model 后面绑定的值就是 true
            不勾选,则是 false
        -->
```

```
        <input type="checkbox" v-model="flag">
        <!--
            v-bind:绑定元素的属性,使之对应初始化的值
            <tagName v-bind:属性名="初始化的值" />
            语法糖:
            <tagName :属性名="初始化的值" />
        -->
        <img :src="mySrc" :title="myTitle"
        :style="myStyle">

        <!--
            v-on:绑定元素的动作
                <tagName v-on:事件="函数" />
            语法糖:
                <tagName @事件="函数" />
            事件写法就是 js 去掉 on 前缀,注意函数如果没有参数
            则不书写括号
        -->
        <button @click="touch">点我试试!</button>
        <input type="text" v-model="count"
        :style="myCss">
    </div>
    <!-- 如果书写在模板以外,则根本不会被 Vue 实例解析 -->
    {{ msg }}
    <!-- 引入 Vue 依赖 -->
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        /* 引入 Vue 依赖之后,可以通过 Vue 依赖提供的构造器,构造
        一个 Vue 实例 */
        const vm = new Vue({
            /* el:element的简写,表示要管理的模板,这里使用到了 css 选择器
            这也是为数不多使用 css 选择器的脚本 */
            el:'#app',
            /* data:表示模板中初始化的数据 */
            data:{
                msg:'这里是初始化的值',
                count:100,
                flag:true,
                text:'123.456.789',
                info:{
                    address:'济南',
                },
                val1:null,
                val2:undefined,
                value:'<em style="color:red">剑心爱吃爆米花</em>',
                mySrc:'./assets/logo.png',
                myTitle:'渐进式框架',
                myStyle:'width:100px',
                myCss:'background-color:rgb(255,0,255)',
            },
            /* beforeMount() {
                debugger
            }, */
            /* 设置函数(方法) */
            methods:{
```

```
                touch(){
                    /* this:如果书写在 Vue 实例中,则表示
                    本实例,这里就是 vm,通过 this,可以获取
                    整个 Vue 实例中所有的数据 */
                    this.count++
                    let r = Math.floor(Math.random() * 256)
                    let g = Math.floor(Math.random() * 256)
                    let b = Math.floor(Math.random() * 256)
                    this.myCss =
                    `background-color:rgb(${r},${g},${b})`
                },
            },
        })

        /* $el:表示 Vue 实例管理的模板地址 */
        console.log(vm,vm.$el,vm.count,vm.msg,vm.info['address'])
        /*
            Vue的工作原理简化版(不涉及生命周期 数据劫持 数据代理)
                浏览器从上往下解析模板,插值语法与指令语法,浏览器根本
                无法解析,因此这些指令全部出现在标签上,插值语法也直接
                显示在页面上,此时浏览器解析结束之后,这种状态称之为虚拟
                DOM
                之后 Vue 实例开始根据 el 解析相应的模板,将数据插入插值语法
                激活指令绑定功能,最终解析结束之后,页面中直接显示初始化的数据
                此时称之为真实 DOM,我们最终见到的就是覆盖虚拟 DOM 之后的
                真实 DOM
        */

    </script>
</body>
</html>
```

day08

Vue 前端八股文

1.什么是单向绑定?什么是双向绑定?

- *单向绑定*
  - *修改 Vue 实例中data初始化的数据,则页面模板中的数据随之发生更改,两者绑定在一起,则称之为单向绑定,也叫数据实现了可响应式,插值语法和指令语法都自带单向绑定功能*
- *双向绑定*
  - *修改页面模板中的数据,则管理模板的 Vue 实例中 data 中的数据随之发生更改,这称之为双向绑定,默认情况下只有 v-model 支持双向绑定,其它情况可以使用计算属性实现双向绑定*

2.Vue2如何进行样式渲染

- *绑定 class*
  - *初始化的值对应 class 属性*
  - *如果初始化的值为真,则类名存在,如果为假,则类名不存在*
  - *这里就是一个元素多个类名,没有初始化的数据,注意不要漏加引号,因为这是数组*

- 绑定 style
  - 初始化的值就对应行内式的样式值
  - 样式名必须使用小驼峰格式,没有引号,初始化的值对应样式值

3.说明 函数 计算属性 侦听器的不同以及使用场合

函数

- 一般绑定事件,当事件激发被调用,也可以直接被调用,函数仅仅支持单向绑定功能,没有缓存机制,只要被调用立即执行,函数体内部可以书写异步代码,可以有选择的书写 return 语句
- 函数多绑定事件,支持异步功能,没有双向绑定功能

计算属性

- 计算属性就是一个属性,由它依赖的初始化的值通过计算得来,只要它依赖的数据发生更改,则计算属性重新执行,计算属性没有括号,也不会传值,仅仅是个属性,自带缓存机制,不管被强制调用多少次,都仅仅执行一次,除非依赖的属性发生更改,计算属性在书写 get 和 set 之后支持单双向绑定,由于 get 中必须书写 return 语句,因此无法书写异步代码

侦听器

- 一般不去考虑单双向绑定问题,就是设置一个值,侦听器去侦听这个数据,只要这个数据发生更改,则侦听器执行,如果设置 immediate 属性,则立即执行侦听器一次,侦听器默认只能侦听基本类型,无法侦听复杂类型,如果要侦听复杂类型,则必须设置 deep:true,开启深度侦听
- 计算属性能做的事,侦听器都能做到,但是侦听器能做的事,计算属性不一定能实现,例如异步功能

4.Vue2条件渲染的方式

- v-if
  - 如果后面是真值,则元素显示,如果后面是假值,则元素不显示,底层根本不渲染,由于切换消耗较大,因此,适用于切换不频繁的场合
- v-show
  - 如果后面是真值,则元素显示,如果后面是假值,则元素不显示,底层依然渲染,只不过添加了一个 display:none 的行内式,初始载入消耗较大,但是之后切换消耗较小,因此适用于切换频繁的场合
- v-else-if v-else
  - 一般搭配 v-if 使用,不能搭配 v-show,必须紧邻,用来组成简单的流程控制

5.如何使用事件原型获取元素节点

```
<!-- 事件原型

  如果函数中没有任何参数,则默认传递一个事件原型

-->

<button @click="touch1">测试 1

<!-- 如果函数中存在实参,则默认不再传递事件原型,如果想传递,则必须

手动书写 $event 就是实参的事件原型 -->

<button @click="touch2('hello',$event)">测试 2
```

6.简述你使用过的事件修饰符

事件修饰符

对事件的一个补充,存在多种书写方式

@事件.事件修饰符="函数"

@事件.stop:屏蔽事件冒泡

@事件.once:事件仅仅激发一次

@事件.prevent:屏蔽某些事件固有的功能,例如

表单提交 链接提交 keyup:事件 键盘键位抬起

keydown:事件 键盘键位落下

@keyup.键位名:可以监听某个键位的操作

@keydown.tab:注意 tab 键必须搭配 keydown

@keyup.shift.键位

@keyup.ctrl.键位

@keyup.alt.键位

7.Vue如何进行列表渲染(迭代数组和迭代对象)

v-for:迭代数组

v-for="(alias,index) in 数组"

alias:别名

index:索引 可以不写,如果不写,小括号省略

in:可以替换 of

数组:必须是 Vue 实例初始化的数组

此指令书写在哪个元素中,则这个元素开始迭代

此处 v-for 存在一个底层就地复用算法导致的 bug

当使用 unshift() 从数组头部插入数据,并且用户可操作时

可能会出现问题,更改用户已经选取的内容,为了杜绝这个问题

一般使用 可选组件 key 来绑定 主键

如果使用 ElementUI 等组件时出现没有主键,但是要强制

使用 key,则可以绑定索引

8.使用过滤器应该注意什么

使用过滤器必须存在数据字典

过滤器两个使用的注意点

1:过滤器中不能使用 this,因为 this 在过滤器中是 undefined

2:过滤器不能与 v-model 连用

2-样式渲染.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>2:样式渲染</title>
    <style>
        .box1{
            background-color: purple;
        }
        .box2{
            background-color: teal;
        }
        .box3{
            background-color: hotpink;
        }
        .box4{
            background-color: deeppink;
        }
        .box5{
            color:whitesmoke;
        }
    </style>
</head>
<body>
    <div id="app">
        <!-- 样式渲染
                就是使用 Vue 来控制页面的css样式
                共有两种思路
                1:绑定 class
                2:绑定 style

            1.1 <tagName :class="初始化的值" />
            初始化的值对应 class 属性
        -->
        <p :class="test1">1.1:绑定 class,后面是字符串</p>
        <!-- 1.2(使用最多!!)
                <tagName :class="{类名:初始化的值,类名:初始化的值}" />
                如果初始化的值为真,则类名存在,如果为假,则类名不存在
        -->
```

```
        <p :class="{box2:val2,box3:val3}">1.2:绑定 class,后面是对象</p>
        <!-- 1.3
                <tagName :class="['类名','类名']" />
                这里就是一个元素多个类名,没有初始化的数据,注意
                不要漏加引号,因为这是数组
        -->
        <p :class="['box4','box5']">1.3:绑定 class,后面是数组</p>
        <!-- ------------------------------------------ -->
        <!--
            2.1:
            <tagName :style="初始化的值" />
            初始化的值就对应行内式的样式值
            2.2:
            <tagName :style="{样式名:初始化的值,样式名:初始化的值,}">
            样式名必须使用小驼峰格式,没有引号,初始化的值对应样式值
        -->
        <p :style="{backgroundColor:value1,color:value2}">2.2:绑定 style,后面是个对象</p>

    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        /* 删除控制台提示 */
        Vue.config.productionTip = false

        new Vue({
            el:'#app',
            data:{
                /* 1.1这里的值才是类名 */
                test1:'box1',
                val2:0,
                val3:true,
                value1:'yellow',
                value2:'green',
            },
        })
    </script>
</body>
</html>
```

3-函数计算属性侦听器.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>3:函数 计算属性 侦听器</title>
</head>
<body>
    <div id="app">
        <!-- v-model.number:在双向绑定时,自动转换为 number 类型,注意
```

```
        如果无法转换,则不转换 -->
        前端成绩: <input type="text" v-model.number="frontendScore"> <br>
        后端成绩: <input type="text" v-model.number="backendScore"> <br>
        <hr>
        <!-- 注意这里是调用函数,并没有绑定事件,因此没有参数也书写了括号 -->
        总成绩(函数-单向绑定):
        <input type="text" v-model="sumScore()"> <br>
        总成绩(函数-单向绑定):
        {{ sumScore() }}
        <hr>
        <!-- 注意计算属性没有括号,也不会传值 -->
        总成绩(计算属性-单向绑定):
        <input type="text" v-model.number="totalScore"> <br>
        总成绩(计算属性-单向绑定):{{ totalScore }}
        <hr>
        <!-- v-model.lazy:开启懒加载,点击回车才能激活双向绑定 -->
        总成绩(计算属性-双向绑定):
        <input type="text" v-model.number.lazy="totalScores">
        <hr>
        总成绩(侦听器):
        <input type="text" v-model="totalWatch">
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        const vm = new Vue({
            /* 以下属性 统称为 配置项 options
            vue2采用的是 Options API
            vue3采用的是 Composition API 组合式 api */
            el:'#app',
            data:{
                frontendScore:90,
                backendScore:80,
                totalWatch:0,
            },
            /* 设置函数 */
            methods: {
                sumScore(){
                    /* return (this.backendScore-0) +
(this.frontendScore-0)
                    return +this.backendScore + +this.frontendScore */
                    console.log('函数 sumScore 执行了---------')
                    return this.frontendScore + this.backendScore
                },
            },
            /* 设置计算属性 */
            computed:{
                /* 仅支持单向绑定,计算属性的语法糖写法 */
                totalScore(){
                    console.log('计算属性 totalScore 执行了--------')
                    /* 注意单向绑定 必须书写 return */
                    return this.frontendScore + this.backendScore
                },
                /* 支持单双向绑定,这是计算属性的完全版写法 */
                totalScores:{
                    /* 单向绑定 */
```

```
                    get(){
                        return this.frontendScore + this.backendScore
                    },
                    /* 双向绑定,这里主动输入的总成绩就是
                    val */
                    set(val){
                        let avgScore = val/2
                        this.backendScore = avgScore
                        this.frontendScore = avgScore
                    },
                },
            },
            /* 设置侦听器 */
            watch:{
                /* 设置要被侦听的数据,这里侦听的是前端成绩 */
                frontendScore:{
                    /* 表示立即执行一次侦听器,注意不是必须 */
                    immediate:true,
                    /* 只要前端成绩变动,则执行
                    newVal:形参 表示变动后的数据
                    oldVal:形参 表示变动前的数据 */
                    handler(newVal,oldVal){
                        /* 总成绩 = 后端成绩 + 新的前端成绩 */
                        this.totalWatch =
                        this.backendScore + newVal
                    },
                },
            },
        })

        /* 侦听器 也可以书写在外部 这里是侦听 后端成绩的变化 */
        vm.$watch('backendScore',{
            immediate:true,
            /* 开启深度侦听 */
            deep:true,
            handler(newVal,oldVal){
                this.totalWatch = newVal + this.frontendScore
            },
        })

    </script>
</body>
</html>
```

4-条件渲染.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>4:条件渲染</title>
    <style>
```

```
        .box{
            width:100px;
            height:100px;
            background-color: palevioletred;
        }
    </style>
</head>
<body>
    <div id="app">
        <!--
            条件渲染存在两种方式
            v-if 和 v-show
        -->
        隐藏元素:
        <input type="checkbox" v-model="flag1">
        <!--
            v-if:如果后面是真值,则元素显示,如果后面是假值,则
            元素不显示,底层根本不渲染,由于切换消耗较大,因此,适用于
            切换不频繁的场合
        -->
        <div :class="{box:val}" v-if="!flag1">测试域</div>
        <hr>
        隐藏元素:
        <input type="checkbox" v-model="flag2">
        <!--
            v-show:如果后面是真值,则元素显示,如果后面是假值,则
            元素不显示,底层依然渲染,只不过添加了一个 display:none
            的行内式,初始载入消耗较大,但是之后切换消耗较小,因此
            适用于切换频繁的场合
        -->
        <div :class="{box:val}" v-show="!flag2">测试域</div>
        <hr>
        <!-- 如果函数体业务逻辑较简单,可以直接在标签中书写业务逻辑 -->
        <button @click="count++">点我试试!</button>
        <!-- v-else-if v-else Vue2.4新特性
        一般搭配 v-if 使用,不能搭配 v-show,必须紧邻,用来组成简单的
        流程控制 此处推荐使用 template 标签,最终并不会在页面真正
        渲染一个 tempalte 元素 -->
        <template v-if="count===1">React 世界第一</template>
        <template v-else-if="count===2">Vue世界第二</template>
        <template v-else>jQuery濒临淘汰</template>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        new Vue({
            el:'#app',
            data:{
                flag1:false,
                flag2:0,
                val:1,
                count:0,
            },
        })
    </script>
</body>
```

```
</html>
```

5-事件原型和事件修饰符

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>5:事件原型与事件修饰符</title>
</head>
<body>
    <div id="app">
        <!-- 事件原型
            如果函数中没有任何参数,则默认传递一个事件原型
        -->
        <button @click="touch1">测试 1</button>
        <!-- 如果函数中存在实参,则默认不再传递事件原型,如果想传递,则必须
        手动书写 $event 就是实参的事件原型 -->
        <button @click="touch2('hello',$event)">测试 2</button>
        <!--
            事件修饰符
                对事件的一个补充,存在多种书写方式
                @事件.事件修饰符="函数"

                @事件.stop:屏蔽事件冒泡
        -->
        <div @click="todo">
            <button @click.stop="dothis">屏蔽事件冒泡(事件传播)</button>
        </div>
        <!-- @事件.once:事件仅仅激发一次 -->
        <button @click.once="count++">{{ count }}</button>

        <!-- @事件.prevent:屏蔽某些事件固有的功能,例如
        表单提交 链接提交 -->
        <form action="./1-初识 Vue.html"
        @submit.prevent="test">
            <input type="submit" value="表单提交">
        </form>
        <a href="./1-初识 Vue.html"
        @click.prevent="test">链接提交</a>
        <!--
            keyup:事件 键盘键位抬起
            keydown:事件 键盘键位落下
            @keyup.键位名:可以监听某个键位的操作
        -->
        <input type="text" placeholder="监听 w 激发"
        @keyup.w="w">
        <input type="text" placeholder="监听回车激发"
        @keyup.enter="enter">
        <input type="text" placeholder="监听空格激发"
        @keyup.space="space">
        <!--
```

```
            @keydown.tab:注意 tab 键必须搭配 keydown
            @keyup.shift.键位
            @keyup.ctrl.键位
            @keyup.alt.键位
        -->
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        new Vue({
            el:'#app',
            data:{
                content:'你好!',
                count:0,
            },
            methods: {
                touch1(event){
                    /*
                        event:事件原型
                        event.target:这里就是事件的目标也就是
                        button元素节点,之前这些节点都是用选择器获取的
                    */
                    event.target.innerText = this.content
                },
                touch2(val,event){
                    event.target.innerText = val
                },
                dothis(){
                    alert('dothis------')
                },
                todo(){
                    alert('todo------')
                },
                test(){
                    alert('同步请求被屏蔽,可以开始玩异步了!!!')
                },
                w(){
                    alert('w激发了---------')
                },
                enter() {
                    alert('回车激发了---------')
                },
                space() {
                    alert('空格激发了---------')
                },
            },
        })
    </script>
</body>
</html>
```

6-列表渲染.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>6:列表渲染</title>
</head>

<body>
    <div id="app">
        <table border="1px">
            <tr>
                <th>序号</th>
                <th>姓名</th>
                <th>性别</th>
                <th>住址</th>
            </tr>
            <!--
                v-for:迭代数组
                v-for="(alias,index) in 数组"
                alias:别名
                index:索引 可以不写,如果不写,小括号省略
                in:可以替换 of
                数组:必须是 Vue 实例初始化的数组
                此指令书写在哪个元素中,则这个元素开始迭代

                此处 v-for 存在一个底层就地复用算法导致的 bug
                当使用 unshift() 从数组头部插入数据,并且用户可操作时
                可能会出现问题,更改用户已经选取的内容,为了杜绝这个问题
                一般使用 可选组件 key 来绑定 主键
                如果使用 ElementUI 等组件时出现没有主键,但是要强制
                使用 key,则可以绑定索引
            -->
            <tr v-for="(emp,index) in empList" :key="emp.id">
                <td>{{ index+1 }}</td>
                <td>{{ emp.name }}</td>
                <td>{{ emp.gender===0?'女':'男' }}</td>
                <td>{{ emp['location'] }}</td>
            </tr>
        </table>
        <ul>
            <!--
                v-for:迭代对象
                v-for="(value,name,index) in 对象"
                value:属性值
                name:属性名
                index:索引
                in:同上
                对象:必须初始化在 Vue 实例中

                v-for 中 in 后面如果不是数组或者对象 还可以是 一个数字
                表示迭代几次
            -->
            <li v-for="(value,name,index) in student"
```

```
            :key="index">
                第{{ index+1 }}个属性名是{{ name }},属性值是{{ value }}
            </li>
        </ul>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        const empList = [
            /*
                id:模拟主键,注意一定要唯一
                gender:0表示女 1表示男
            */
            { id: 1, name: 'elena', gender: 0, location: '济南', },
            { id: 2, name: 'damon', gender: 1, location: '青岛', },
            { id: 3, name: 'stefan', gender: 1, location: '淄博', },
            { id: 4, name: 'aleric', gender: 1, location: '济宁', },
            { id: 5, name: 'penny', gender: 0, location: '济南', },
            { id: 6, name: 'nancy', gender: 0, location: '青岛', },
            { id: 7, name: 'jack', gender: 1, location: '青岛', },
            { id: 8, name: 'sheldon', gender: 0, location: '济南', },
            { id: 9, name: 'raj', gender: 0, location: '淄博', }
        ]

        const student = {
            name:'胡桃',
            age:17,
            location:'璃月',
        }

        new Vue({
            el: '#app',
            data: {
                /*
                    将数组初始化进 data
                    empList:empList,
                */
                empList,
                student,
            },
        })
    </script>
</body>

</html>
```

7-表单双向绑定.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>7:表单双向绑定</title>
</head>

<body>
    <div id="app">
        <fieldset>
            <legend>表单双向绑定</legend>
            <form @submit.prevent="demo">
                <!-- for 对应 id -->
                <label for="name">
                    <!-- v-model.trim:双向绑定数据同时去掉字符串
                    两侧空格 -->
                    用户姓名: <input type="text" v-model.trim="myForm.vname"
id="name"> <br>
                </label>
                <label for="pass">
                    用户密码: <input type="password" v-
model.trim="myForm.vpass" id="pass">
                </label> <br>
                <!-- 注意 v-model 与 value 值只要对应,则
                勾选 -->
                性别: <input type="radio" name="gender" value="00000" v-
model="myForm.vgender">女

                <input type="radio" name="gender" value="11111" v-
model="myForm.vgender">男
                <br>
                <!-- 注意 复选框 是个数组,同样只要绑定的数据对应 value 值
                则勾选 -->
                爱好: <input type="checkbox" name="hobby" v-
model="myForm.vhobby" value="soccer">足球

                <input type="checkbox" name="hobby" v-
model="myForm.vhobby" value="running">跑步

                <input type="checkbox" name="hobby" v-
model="myForm.vhobby" value="shopping">购物

                <input type="checkbox" name="hobby" v-
model="myForm.vhobby" value="game">游戏 <br>

                <!-- v-model:这里双向绑定最终用户选择的住址 -->
                住址:
                <select name="location" v-model="myForm.vlocation">
                    <option :value="city.id"
                    v-for="city in cities" :key="city.id">
                        {{ city.name }}
                    </option>
                </select>
                <br>
                <!-- cols:列数 rows:行数 -->
                个人介绍:
                <textarea name="weibo" cols="20" rows="3"
                v-model.trim="myForm.vinfo"></textarea> <br>
                <input type="submit" value="提交表单">
```

```
            </form>
        </fieldset>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                myForm: {
                    vname: '',
                    vpass: '',
                    vgender: '',
                    vhobby: [],
                    vlocation:'',
                    vinfo:'',
                },
                cities: [
                    { id: 1, name: '济南' },
                    { id: 2, name: '青岛' },
                    { id: 3, name: '淄博' },
                    { id: 4, name: '济宁' },
                    { id: 5, name: '烟台' }
                ]
            },
            methods: {
                demo(){
                    console.log(this.myForm)
                }
            },
        })
    </script>
</body>

</html>
```

8-过滤器.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>8:过滤器</title>
</head>

<body>
    <div id="app">
        <table border="1px">
            <tr>
                <th>ID</th>
                <th>姓名</th>
                <th>支付方式</th>
```

```
            </tr>
            <tr v-for="data in datas" :key="data.id">
                <td>{{ data.id }}</td>
                <td>{{ data.name }}</td>
                <!-- {{ 要被过滤的值|过滤器名 }} -->
                <td>{{ data.payType|payFilter }}</td>
            </tr>
        </table>
        <!-- <input type="text" :value="content|payFilter"> -->
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        /* payType:支付方式 1|2|3|4 共有四种 */
        const datas = [
            { id: 1, name: 'elena', payType: 1, },
            { id: 2, name: 'penny', payType: 2, },
            { id: 3, name: 'damon', payType: 4, },
            { id: 4, name: 'stean', payType: 3, },
            { id: 5, name: 'raj', payType: 2, },
            { id: 6, name: 'matt', payType: 3, }
        ]

        /* 使用过滤器必须存在数据字典 */
        const payOptions = [
            { id: 1, option: '现金支付', },
            { id: 2, option: '支付宝支付', },
            { id: 3, option: '微信支付', },
            { id: 4, option: '银行卡支付', }
        ]

        new Vue({
            el: '#app',
            data: {
                datas,
            },
            /*
                过滤器两个使用的注意点
                1:过滤器中不能使用 this,因为 this 在过滤器中是 undefined
                2:过滤器不能与 v-model 连用
            */
            filters:{
                /*
                    过滤器名
                    val:形参 就是当前被过滤的值,这里就是 1|2|3|4

                    const arr = [1,2,3,4]
                    let value = arr.find(a => a >=3)    //3

                */
                payFilter(val){
                    const payObj =
                    payOptions.find( payOption => payOption.id === val )

                    return payObj?payObj.option:''
                },
```

```
            },
        })
    </script>
</body>

</html>
```

day09

Vue 前端八股文（完整版）

1. 什么是单向绑定?什么是双向绑定?

- 单向绑定
  - 修改 Vue 实例中data初始化的数据,则页面模板中的数据随之发生更改,两者绑定在一起,则称之为单向绑定,也叫数据实现了可响应式,插值语法和指令语法都自带单向绑定功能
- 双向绑定
  - 修改页面模板中的数据,则管理模板的 Vue 实例中 data 中的数据随之发生更改,这称之为双向绑定,默认情况下只有 v-model 支持双向绑定,其它情况可以使用计算属性实现双向绑定

2. Vue2如何进行样式渲染
   - 绑定 class
     - 初始化的值对应 class 属性
     - 如果初始化的值为真,则类名存在,如果为假,则类名不存在
     - 这里就是一个元素多个类名,没有初始化的数据,注意不要漏加引号,因为这是数组
   - 绑定 style
     - 初始化的值就对应行内式的样式值
     - 样式名必须使用小驼峰格式,没有引号,初始化的值对应样式值
3. 说明 函数 计算属性 侦听器的不同以及使用场合

- 函数
  - 一般绑定事件,当事件激发被调用,也可以直接被调用,函数仅仅支持单向绑定功能,没有缓存机制,只要被调用立即执行,函数体内部可以书写异步代码,可以有选择的书写 return 语句
  - 函数多绑定事件,支持异步功能,没有双向绑定功能
- 计算属性
  - 计算属性就是一个属性,由它依赖的初始化的值通过计算得来,只要它依赖的数据发生更改,则计算属性重新执行,计算属性没有括号,也不会传值,仅仅是个属性,自带缓存机制,不管被强制调用多少次,都仅仅执行一次,除非依赖的属性发生更改,计算属性在书写 get 和 set 之后支持单双向绑定,由于 get 中必须书写 return 语句,因此无法书写异步代码
- 侦听器
  - 一般不去考虑单双向绑定问题,就是设置一个值,侦听器去侦听这个数据,只要这个数据发生更改,则侦听器执行,如果设置 immediate 属性,则立即执行侦听器一次,侦听器默认只能侦听基本类型,无法侦听复杂类型,如果要侦听复杂类型,则必须设置 deep:true,开启深度侦听
  - 计算属性能做的事,侦听器都能做到,但是侦听器能做的事,计算属性不一定能实现,例如异步功能

4. ***Vue2条件渲染的方式***

- ***v-if***
  - ***如果后面是真值,则元素显示,如果后面是假值,则元素不显示,底层根本不渲染,由于切换消耗较大,因此,适用于切换不频繁的场合***
- ***v-show***
  - ***如果后面是真值,则元素显示,如果后面是假值,则元素不显示,底层依然渲染,只不过添加了一个 display:none 的行内式,初始载入消耗较大,但是之后切换消耗较小,因此适用于切换频繁的场合***
- ***v-else-if v-else***
  - ***一般搭配 v-if 使用,不能搭配 v-show,必须紧邻,用来组成简单的流程控制***

5. ***如何使用事件原型获取元素节点***
   - ***event.target***
6. ***简述你使用过的事件修饰符***
   - ***.stop:解决冒泡问题***
   - ***.once:激活一次性事件***
   - ***.prevent:屏蔽元素固有的动作,例如表单 连接提交***
   - ***.native:给组件添加原生事件***
   - ***@keyup.键位名:监听键位的激发***
   - ***@keydown.tab:监听 tab 键激发***
7. ***Vue如何进行列表渲染(迭代数组和迭代对象)***
   - ***v-for="(alias,index) in 数组"***
   - ***v-for="(value,name,index) in 对象"***
   - ***注意存在主键则 :key="主键" 没有主键则 :key="index"***
8. ***使用过滤器应该注意什么***
   - ***不能与 v-model 连用***
   - ***不能使用 this***
9. ***简述你所使用过的指令元素***
   - ***v-once:一次性插值绑定,之后失去绑定功能***
   - ***v-html:向元素中插入超文本,注意!为了防止网络 XSS 攻击,禁止插入脚本***
   - ***v-text:向元素中插入文本***
   - ***v-model:使用在表单项中,支持双向绑定***
     - ***v-model.trim:双向绑定数据,去掉字符串两侧空格***
     - ***v-model.number:双线绑定数据,同时转换为 number,注意如果无法转换,则不转换***
     - ***v-model.lazy:点击回车才会激活双向绑定功能***
   - ***v-bind:语法糖:,绑定元素的属性***
   - ***v-on:语法糖@,绑定事件***
     - ***@事件.once***
     - ***@事件.stop***
     - ***@事件.prevent***
     - ***@事件.native***
     - ***@keyup.键位名***
     - ***@keydown.tab***
   - ***v-for:迭代数据***
   - ***v-if:后面如果为真,则元素显示,为假,元素不显示,底层不渲染***

- *v-else-if:搭配 v-if 使用必须紧邻*
- *v-else:搭配 v-if 使用必须紧邻*
- *v-show:后面如果为真,则元素显示,为假,元素不显示,底层依然渲染,只不过添加了.display:none;css行内式*
- *v-pre:提示 Vue 实例不解析*
- *v-cloak:解决闪现问题*
- *v-slot:父子组件传值时传递模板使用,是插槽的另外一种书写方式*

10. *Vue2实例对用户书写的 data 对象做了哪些处理?*

1:数据劫持

Vue实例获取到用户书写的 data 对象之后,不管封装了几层

都会对这个对象添加可响应式功能,对象的每个属性都被添加了

reactiveGetter()和 reactiveSetter() 两个函数

如果 data 中的数据被读取,则执行 reactiveGetter()

如果 data 中的数据被修改,则执行 reactiveSetter(newVal)

这个函数中同时会修改页面模板中对应的数据,这就是可响应式

或者说单向绑定是如何实现的,这些被施加了可响应式功能的数据以及

get 和 set 函数都被封装在 Vue 实例的 _data 对象

这个操作被称之为数据劫持

2:数据代理

一个对象可以对另外一个对象中的属性进行操作,则称之为数据代理

Vue实例通过数据代理将 _data 中的数据代理到 Vue 实例的表层

这样做的好处是,我们在模板中使用指令或者插值语法时,不需要每次都添加

_data 前缀,因为 Vue 实例表层数据都可以直接在模板中使用

9-自定义指令.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>9:自定义指令</title>
</head>

<body>
    <div id="app1">
        <p v-etoak="content"></p>
```

```
    </div>
    <div id="app2">
        <p v-etoak="content"></p>
        <!-- 使用自定义指令实现自动获取焦点功能 -->
        <input type="text" v-focus>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        /*
            自定义指令
                官方提供的指令元素有时无法满足我们的要求,因此我们
                可以自己创建自定义指令,这些指令可以随意起名,但是不能存在
                v-,因为不管我们如何命名 开头都会自动添加 v- 前缀
                自定义指令分为全局自定义和局部自定义
                全局自定义:可以使用在任意一个 Vue 实例中,
                全局自定义必须放置在 Vue 实例之前
                局部自定义:只能使用在某一个 Vue 实例中
        */

        /* 1:全局自定义 */
        Vue.directive('etoak',{
            /* 渲染样式
                el:形参 就表示指令书写的元素节点
                这里就是 p 段落元素节点
            */
            bind(el){
                /* p 元素节点背景色变为珊瑚橘 */
                el.style.backgroundColor = 'coral'
            },
            /* 渲染一次性动作
                el:同上
                binding:就表示自定义指令
                binding.value:就表示自定义指令绑定的值
            */
            inserted(el,binding){
                el.innerText = binding.value.toUpperCase()
            }
        })

        new Vue({
            el: '#app1',
            data: {
                content: 'thisisetoak',
            },
        })

        new Vue({
            data: {
                content: 'loveu3000',
            },
            /* 2:设置局部自定义指令 */
            directives:{
                /* 设置指令 */
                focus:{
                    bind(el){
```

```
                        //此处未使用,可以不写
                    },
                    inserted(el,binding){
                        /* 注意 事件其实都是函数,这里强制调用
                        激发事件 */
                        el.focus()
                    },
                }
            },
        }).$mount('#app2')
        /* $mount():表示挂载的模板,指明 Vue 实例管理哪一块模板 */
    </script>
</body>
</html>
```

10-指令补遗.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>10:指令补遗</title>
    <style>
        /* 获取属性名为 v-cloak 的元素 */
        [v-cloak]{
            display: none;
        }
    </style>
</head>
<body>
    <div id="app">
        <!--
            v-pre:提示 Vue 实例不解析此处模板,一般使用在
            大段没有插值语法也没有指令语法的元素中
        -->
        <h1 v-pre>{{ title }}</h1>
        <!-- v-cloak:解决闪现问题,所谓闪现问题是指
        当真实 DOM 还未覆盖虚拟 DOM 时,存在极短的时间,用户会看到虚拟 DOM
        显示在页面中,造成非常不好的体验
        当虚拟 DOM 生成时,元素上存在 v-cloak 指令,此时与样式联动
        元素被隐藏,用户根本无法看到虚拟 DOM,之后真实 DOM 覆盖,v-cloak
        被解析,元素上不再存在 v-cloak 指令,因此 css 失效,用户可以看到
        真实 DOM
        -->
        <h2 v-cloak>{{ title }}</h2>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        new Vue({
            data:{
                title:'我是标题!',
            }
```

```
        }).$mount('#app')
    </script>
</body>
</html>
```

11-数据劫持和数据代理.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>11:数据劫持与数据代理</title>
</head>
<body>
    <script>
        /*
            Vue2实例对用户书写的 data 对象做了哪些处理?
            1:数据劫持
                Vue实例获取到用户书写的 data 对象之后,不管封装了几层
                都会对这个对象添加可响应式功能,对象的每个属性都被添加了
                reactiveGetter()和 reactiveSetter() 两个函数
                如果 data 中的数据被读取,则执行 reactiveGetter()
                如果 data 中的数据被修改,则执行 reactiveSetter(newVal)
                这个函数中同时会修改页面模板中对应的数据,这就是可响应式
                或者说单向绑定是如何实现的,这些被施加了可响应式功能的数据以及
                get 和 set 函数都被封装在 Vue 实例的 _data 对象
                这个操作被称之为数据劫持
            2:数据代理
                一个对象可以对另外一个对象中的属性进行操作,则称之为数据代理
                Vue实例通过数据代理将 _data 中的数据代理到 Vue 实例的表层
                这样做的好处是,我们在模板中使用指令或者插值语法时,不需要每次都添加
                _data 前缀,因为 Vue 实例表层数据都可以直接在模板中使用
        */
        const person = {
            name:'胡桃',
            age:17,
        }

        /* person.address = '璃月' */

        /* 给 person 添加一个属性 address 属性值为 璃月 */
        Object.defineProperty(person,'address',{
            value:'璃月',
            /* 设置可枚举,默认不可枚举 */
            enumerable:true,
            /* 设置可写,默认不可写 */
            writable:true,
            /* 设置可删,默认不可删 */
            configurable:true,
        })

        person.address = '济南'
```

```
        console.log(delete person.address)

        console.log(person)
        console.log(Object.keys(person))
        console.log(Object.values(person))
    </script>
</body>
</html>
```

12-模拟数据代理.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>12:模拟数据代理</title>
</head>

<body>
    <script>
        /*
            数据代理
            一个对象可以对另外一个对象中的属性进行操作,则称之为数据代理
        */

        /* 此为源对象:此处模拟已经经过数据劫持实现了可响应式的
        _data 对象 */
        const _data = {
            name: '胡桃',
        }

        /* 此为代理对象:此处模拟 Vue 实例 */
        const vm = {
            $demo1: '原生属性',
            $demo2: '原生属性',
            $demo3: '原生属性',
        }

        Object.defineProperty(vm,'name',{
            /* 如果要读取 vm 的 name 属性则执行此函数 */
            get(){
                console.log('vm的name 属性被读取 getter 函数执行啦-----')
                return _data.name
            },
            set(newVal){
                console.log('vm的name 属性被修改 setter 函数执行啦-----')
                _data.name = newVal
            },
        })
    </script>
</body>
```

```
</html>
```

13-Vue2实现对象可响应式.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>13:Vue2 实现对象可响应式</title>
</head>
<body>
    <div id="app">
        <ul>
            <li>ID:{{ person.id }}</li>
            <li>姓名:{{ person.name }}</li>
            <li>年龄:{{ person.age }}岁</li>
            <li v-if="person.address">住址:{{ person.address }}</li>
        </ul>
        <button @click="add">添加属性</button>
        <button @click="remove">删除属性</button>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        const person = {
            id:1,
            name:'胡桃',
            age:17,
        }

        const vm = new Vue({
            data:{
                person,
            },
            methods:{
                add(){
                    /* 这种书写方式无法实现可响应式,没有单向绑定功能,
                    页面模板不会出现任何变化 */
                    //this.person.address = '济南'
                    /* 以下写法 才可以实现可响应式,实现单向绑定 */
                    this.$set(person,'address','济南')
                    // Vue.set(person,'address','济南')
                },
                remove(){
                    /* 这种书写方式无法实现可响应式,没有单向绑定功能,
                    页面模板不会出现任何变化 */
                    //delete this.person.address
                    this.$delete(person,'address')
                    // Vue.delete(person,'address')
                },
            },
        }).$mount('#app')
    </script>
```

```
</body>
</html>
```

14-Vue2实现数组可响应式.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>14:Vue2实现数组可响应式</title>
</head>
<body>
    <div id="app">
        <ul>
            <li v-for="hobby in hobbies">
                {{ hobby }}
            </li>
        </ul>
        <button @click="add">添加数组元素</button>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        const vm = new Vue({
            data:{
                hobbies:['抽烟','喝酒','烫头'],
            },
            methods: {
                add(){
                    /* 以下写法没有实现可响应式,没有单向绑定功能
                    模板不会出现任何变动 */
                    //this.hobbies[3] = '敲代码'
                    /*
                        在 Vue2 中必须使用以下七个函数对数组进行
                        修改,这七个函数是 js 版加强版,实现了可响应式
                        功能,注意不是 js 原始的七个,只不过重名,功能相同
                        实现了可响应式功能
                        push()
                        unshift()
                        shift()
                        pop()
                        sort()
                        reverse()
                        splice()
                    */
                    this.hobbies.push('敲代码')
                }
            },
        }).$mount('#app')
    </script>
</body>
</html>
```

Vue2中必须使用这七个函数对数组进行修改

在 Vue2 中必须使用以下七个函数对数组进行修改,这七个函数是 js 版加强版,实现了可响应式功能,注意不是 js 原始的七个,只不过重名,功能相同实现了可响应式功能

push()

unshift()

shift()

pop()

sort()

reverse()

splice()

15-模拟数据劫持.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue2数据劫持</title>
</head>

<body>
    <script>
        /*
            STEP1:数据劫持
            用户书写的 data 对象仅仅是一个普通的对象,没有任何可响应式功能,Vue实例对我们写的这个data进行了
            一个封装,在这个封装过程中添加了可响应式功能,其实就是数据更改,则页面模板随之发生更改(单向绑定)
            之后我们被封装的添加了可响应式功能的 data变为 _data
        */
        /* 这个data就是我们自己书写的初始化数据的data */
        let data = {
            name: '胡桃',
            address: '璃月',
        }
        /*
            此处仅模拟数据劫持功能,与官方源码存在出入,仅做演示
            我们自己写的 data 是 怎么变成 实现了内部属性可响应式 的 _data呢
            data = _data
            下文创建了一个用来监视数据的实例对象,这个实例对象可以实时监视传入数据中
```

```
            属性的更改
        */
        let obs = new Observer(data)

        /* 模拟Vue实例 */
        let vm = {}

        vm._data = data = obs

        /* Observer的构造函数 */
        function Observer(obj) {
            /* 拿到传入对象的所有属性名 ['name','address'] */
            let keys = Object.keys(obj)

            keys.forEach(k => {
                /* this:就是这个对象 */
                Object.defineProperty(this, k, {
                    get() {
                        return obj[k]
                    },
                    set(newVal) {
                        console.log(`${k}被修改啦!!,可响应式启动,我要去重新渲染页面模板中的数据了之后
                        生成虚拟DOM,最终被真实DOM覆盖`)
                        obj[k] = newVal
                    }
                })
            })
        }
    </script>
</body>
</html>
```

day10

VueTodoMVC app.js

```
/* 自调用函数,为了防止脚本污染,划定了块级作用域 */
(function (window) {
    /* 表示使用没有任何兼容性的 js 语句,不能使用陈旧的具有兼容性问题的语句 */
    'use strict';

    /*
        设置数据源
        id:模拟主键 有且唯一
        content:任务内容
        completed: true表示已完成 false 待办
    */
    const items = [
```

```
        { id: 1, content: '学习 css', completed: true, },
        { id: 2, content: '学习 vue', completed: true, },
        { id: 3, content: '学习 typescript', completed: false, },
        { id: 4, content: '学习 vue-cli', completed: false, },
        { id: 5, content: '学习 vue-router', completed: false, },
        { id: 6, content: '学习 webpack', completed: false, },
        { id: 7, content: '学习 vue3', completed: false, },
        { id: 8, content: '学习 javascript', completed: true, }
    ]

    /*
        STEP9)数据持久化
        在 js 中的 window 中存在两个属性 localStorage 和 sessionStorage
        可以将数据保存进当前的浏览器,数据类型只能是字符串,最大 5m,支持中文
        后端 Cookie Session Token

        localStorage:保存在用户浏览器本地,只要不主动删除,则一直有效
        sessionStorage:保存在用户浏览器本地,只要关闭浏览器,立即失效

        以上两个对象都具有以下功能
            setItem(key,value):保存信息 key:秘钥 必须是字符串 value 保存的
            信息,必须是字符串,一般为 json 格式
            getItem(key):获取信息
            removeItem(key):删除信息
    */

    const KEY = 'et2301todomvc'

    const itemsStorage = {
        /* 此方法用来从浏览器获取数据 */
        fetch(){
            return JSON.parse(localStorage.getItem(KEY)||'[]')
        },
        /* 此方法用来保存数据进浏览器 */
        save(val){
            localStorage.setItem(KEY,JSON.stringify(val))
        },
    }

    const app = new Vue({
        /* 初始化的数据 */
        data:{
            /* 初始化数据源 */
            /* STEP9)数据持久化 */
            items:itemsStorage.fetch(),
            /* STEP7)编辑任务
                currentItem:表示正在被编辑的任务项,
                类型是对象,由于一打开页面没有正在被编辑的任务项,因此
                默认为 null
            */
            currentItem:null,
            /*  STEP8)路由状态切换
                这里表示默认的状态
            */
            status:'all',
```

```
        },
        /* 函数 */
        methods:{
            /* STEP2)添加待办事项 */
            addItem(event){
                /* 1:获取用户输入的内容 */
                let content = event.target.value.trim()
                /* 2:简单验证 */
                if(!content){
                    alert('请输入有效信息...')
                    return
                }
                /* 3:如果填写无误,则添加到数组中 */
                this.items.push({
                    /* 由于这里不是真正的主键,因此这里仅做模拟使用
                    存在 bug,仅做演示 */
                    id:this.items.length+1,
                    content,
                    completed:false,
                })
                /* 4:清空输入框 */
                event.target.value = ''
            },
            /* STEP5)删除任务 */
            removeItem(item){
                /* 这里第一个参数都是删除前计算的索引值
                如果直接传递索引,由于存在添加 和 编辑操作,因此
                可能会出现问题 */
                this.items.splice(this.items.indexOf(item),1)
            },
            /* STEP6)批量删除剩余待办事项 */
            removeCompleted(){
                /* 自己过滤自己,因此完成的都被过滤掉了 */
                this.items =
                this.items.filter( item => item.completed === false )
            },
            /* STEP7)编辑任务 */
            /* 双击开始编辑 item 就是被双击的任务项对象 */
            toEdit(item){
                /* 给正在被编辑的任务赋值因此 从 null 变为 某一个任务 */
                this.currentItem = item
            },
            /* 失去焦点或者点击回车完成编辑 */
            finishEdit(item, event){
                /* 1:获取用户输入的新的信息 */
                let newContent = event.target.value.trim()
                /* 2:校检是否合法 */
                if(!newContent){
                    /* 3:输入都是空格删除任务项,复用 STEP5) */
                    this.removeItem(item)
                    return
                }
                /* 4:替换现有内容 */
                item.content = newContent
                /* 5:恢复当前正在被编辑的任务为 null
```

```
                这就意味着 class="editing" 失效了,也就意味着
                input 隐藏了 div 回来了 */
                this.currentItem = null
            },
            /* 点击 esc 放弃编辑 */
            cancelEdit(){
                /*
                    恢复正在被编辑的任务为 null
                    这就意味着 class="editing" 失效了,也就意味着
                    input 隐藏了 div 回来了
                */
                this.currentItem = null
            },
            /* 修改后的点击回车完成编辑 */
            triggerBlur(event){
                /* 激发失去焦点 */
                event.target.blur()
            },
        },
        /* 计算属性 */
        computed:{
            /* STEP3)统计剩余待办任务数 */
            remaining(){
                const unItems =
                /* this.items.filter( item => !item.completed ) */
                this.items.filter(item => item.completed === false)
                return unItems.length
            },
            /* STEP4)任务状态切换 */
            toggleAll:{
                /*
                    单向绑定
                    如果所有任务都已经完成(剩余待办任务为 0),
                    计算属性应该返回真值则此处勾选
                */
                get(){
                    /* 复用 STEP3  */
                    return this.remaining === 0
                },
                /* 双向绑定
                    主动勾选此处复选框,则 toggleAll 为 true主动不勾
                    选则为 false,那么这里 val 要么是 true 要么是 false
                */
                set(val){
                    /* 所有的任务都是 true 或者 都是 false */
                    this.items.forEach(item => item.completed = val)
                },
            },
            /* STEP8)路由状态切换 */
            filterItems(){
                /* 这里根据当前状态进行过滤 */
                switch(this.status){
                    case 'active':
                        /* 获取待办的 */
```

```
                        return this.items.filter( item => !item.completed
)
                    case 'completed':
                        /* 获取完成的 */
                        return this.items.filter(item => item.completed)
                    default:
                        /* 全部获取 */
                        return this.items
                }
            },
        },
        /* 侦听器 */
        watch:{
            /* 侦听 items 数据的变动 */
            items:{
                /* 由于是复杂类型,因此开启深度侦听 */
                deep:true,
                handler(newVal,oldVal){
                    /* 将数据保存进浏览器 */
                    itemsStorage.save(newVal)
                },
            },
        },
        /* 自定义指令 */
        directives:{
            focus:{
                /* 设置一次性动作
                只要 DOM 发生变动,立即失效 */
                inserted(el,binding){
                    /* 强制激发获取焦点 */
                    el.focus()
                },
                /* 设置持久性动作,无法 DOM 如何更改
                一直生效 */
                update(el,binding){
                    /* binding.value:表示指令后面绑定的值
                    这里是 绑定的值为真 */
                    if(binding.value){
                        /* 则获取焦点 */
                        el.focus()
                    }
                },
            },
        },
    })

    app.$mount('.todoapp')

    /*
        STEP8)路由状态切换
        onhashchange:当浏览器地址栏的哈希值发生更改时执行
    */
    window.onhashchange = ()=>{
        /* 截取当前的哈希
            http://www.etoak.com/#/etoak
```

```
            则 这里 #/etoak 被称之为哈希值,从#开始浏览器根本不会发送任何请求
            location.hash:就是获取当前地址栏的哈希值例如上例 就是 #/etoak
            location.hash.substring(2): etoak
            这里哈希可能是以下三种
                '#/' '#/active' '#/completed'
                则截取之后变为
                'all' 'active' 'completed'

        */
        let hash = location.hash.substring(2)||'all'
        /* 给 vue 实例的状态赋值 all active completed 三种可能 */
        app.status = hash
    }
    /* 强制激发一次哈希值更改的函数,给当前状态赋值 那么当前状态
    为 all|active|completed 三选一 */
    onhashchange()
})(window);

```

VueTodoMVC index.html

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Template • TodoMVC</title>
        <link rel="stylesheet" href="node_modules/todomvc-common/base.css">
        <link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
        <!-- CSS overrides - remove if you don't need it -->
        <link rel="stylesheet" href="css/app.css">
    </head>
    <body>
        <section class="todoapp" v-cloak>
            <header class="header">
                <h1>ET2301 练习</h1>
                <!-- STEP2)添加待办事项
                        @keyup.enter="addItem"
                        点击回车开始添加任务,后面是个函数
                -->
                <input class="new-todo" placeholder="您有什么待办事项?"
                v-focus @keyup.enter="addItem">
            </header>
            <!-- This section should be hidden by default and shown when there are todos -->
            <!-- STEP1)列表渲染
                    此处 v-show="items.length"
                    如果数组没有数组,则 v-show 后面为假,元素隐藏
                    由于切换频繁,因此这里使用 v-show
            -->
            <section class="main" v-show="items.length">
                <!-- STEP4)任务状态切换
```

```
                        此处使用计算属性,单双向绑定都要书写
                            v-model="toggleAll"
                        单向:如果所有任务都已经完成(剩余待办任务为 0),计算属性应该返回真值
                        则此处勾选
                        双向:主动勾选此处复选框,则 toggleAll 为 true
                        主动不勾选则为 false
                -->
                <input id="toggle-all" class="toggle-all"
                type="checkbox" v-model="toggleAll">
                <label for="toggle-all">Mark all as complete</label>
                <ul class="todo-list">
                    <!-- These are here just to show the structure of the list items -->
                    <!-- List items should get the class `editing` when editing and `completed` when marked as completed -->
                    <!--
                        STEP1)列表渲染
                            这里 li 用来显示列表项,根据 class 的不同存在以下三种样式

                            1:class="completed"表示任务已经完成,复选框勾选字体发浅,存在贯穿线
                            2:没有 class 表示任务待办,复选框未勾选,字体发黑无贯穿线

                        STEP7)编辑任务
                            3:class="editing" 任务正处于编辑状态 li 中的 div 被 input 取代,用户开始进行编辑
                    -->
                    <!-- STEP8)路由状态切换
                            v-for="(item,index) in filterItems"
                            此处迭代从直接数组中迭代变为从计算属性中也就是根据当前状态过滤后的数组进行迭代
                    -->
                    <li :class="{completed:item.completed,editing:item === currentItem}"
                    v-for="(item,index) in filterItems"
                    :key="item.id">
                        <div class="view">
                            <input class="toggle" type="checkbox"
                            v-model="item.completed">
                            <!-- STEP7)编辑任务
                                    双击 label 时激活编辑,class 变为 editing 外侧的 div 元素隐藏,取而代之的是 input 元素进入编辑状态
                                    1:双击开始编辑  @dblclick="toEdit(item)"
                            -->
                            <label @dblclick="toEdit(item)">{{ item.content }}</label>
                            <!-- STEP5)删除任务
                                    单击开始删除任务
                            @click="removeItem(item)"
                            -->
```

```
                            <button class="destroy"
@click="removeItem(item)"></button>
                        </div>
                        <!--
                            STEP7)编辑任务
                                2:点击回车完成编辑

 @keyup.enter="finishEdit(item,$event)"
                                    注意点击回车也会激发失去焦点,导致删除两条的
bug
                                    因此点击回车修改为以下
                                    @keyup.enter="triggerBlur"
                                3:失去焦点完成编辑
                                    @blur="finishEdit(item,$event)"
                                4:点击 esc 放弃编辑
                                    @keyup.esc="cancelEdit"
                                5:输入都是空格删除任务项
                                    if(!newContent){
                                        /* 输入都是空格删除任务项,复用 STEP5)
*/
                                        this.removeItem(item)
                                        return
                                    }
                        -->
                        <input class="edit" :value="item.content"
                        @keyup.enter="triggerBlur"
                        @blur="finishEdit(item,$event)"
                        @keyup.esc="cancelEdit"
                        v-focus="item === currentItem">
                    </li>
                </ul>
            </section>
            <!-- This footer should hidden by default and shown when
there are todos -->
            <!-- STEP1)列表渲染
                    此处 v-show="items.length"
                    如果数组没有数组,则 v-show 后面为假,元素隐藏
                    由于切换频繁,因此这里使用 v-show
            -->
            <footer class="footer" v-show="items.length">
                <!-- This should be `0 items left` by default -->
                <!-- STEP3)统计剩余任务数
                        此处没有事件,因此使用计算属性,仅仅单向绑定即可
                -->
                <span class="todo-count"><strong>{{ remaining }}</strong>
item{{ remaining === 1?'':'s' }} left</span>
                <!-- Remove this if you don't implement routing -->
                <ul class="filters">
                    <li>
                        <!-- STEP8)路由状态切换
                            此处当前状态如果与哈希匹配则证明链接被激活,因此添加
                            class="selected" 添加后链接存在一个橙色边框
                        -->
                        <a :class="{selected:status === 'all'}"
href="#/">All</a>
```

```
                    </li>
                    <li>
                        <a :class="{selected:status === 'active'}" href="#/active">Active</a>
                    </li>
                    <li>
                        <a :class="{selected:status === 'completed'}" href="#/completed">Completed</a>
                    </li>
                </ul>
                <!-- Hidden if no completed items are left ↓ -->
                <!-- STEP6)批量删除已完成任务

                        总任务数 === 剩余待办任务 那就说明所有任务都待办
                        等式不成立 因此按钮隐藏
                        总任务数 > 剩余待办任务 那就说明存在完成的
                        等式成立 因此按钮显示
                -->
                <button class="clear-completed"
                v-show="items.length > remaining"
                @click="removeCompleted">Clear completed</button>
            </footer>
        </section>
        <footer class="info">
            <p>双击开始编辑任务项</p>
            <!-- Remove the below line ↓ -->
            <p>模板制作<a href="http://sindresorhus.com">Sindre Sorhus</a></p>
            <!-- Change this out with your name and url ↓ -->
            <p>作者<a href="http://todomvc.com">Joshua</a></p>
            <p>ET2301todoMVC练习</p>
        </footer>
        <!-- Scripts here. Don't remove ↓ -->
        <script src="node_modules/todomvc-common/base.js"></script>
        <!-- 引入 vue 依赖 -->
        <script src="./node_modules/vue/dist/vue.js"></script>
        <!-- 我们自己的脚本书写在此 -->
        <script src="js/app.js"></script>
    </body>
</html>

```

1-局部组件和全局组件.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>1:局部组件和全局组件</title>
</head>
```

```
<body>
    <div id="app1">
        <!-- 引用 reference 全局组件 注意组件名必须使用连字符,禁止使用
        大小驼峰,并且不能与html标签重名 -->
        <component-a></component-a>
    </div>
    <div id="app2">
        <!-- 引用全局组件 reference 全局组件 注意组件名必须使用连字符,禁止使用
        大小驼峰,并且不能与html标签重名 -->
        <component-a></component-a>
        <!-- 引用局部组件 -->
        <component-b></component-b>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        /*
            组件(Components)
                组件是 Vue 中除了绑定以外,最大的特色,也是进阶语法的入门,
                组件就是一种特殊的 Vue 实例,它可以自己封装模板,其它配置项
                与 Vue 实例完全一致,组件天生就是被复用,从而实现特定模板的复用
                组件分为 局部组件和全局组件
                全局组件可以使用在任意一个 Vue 实例中
                局部组件只能使用在某个 Vue 实例中

            1:全局组件
                Vue.component('组件名',{
                    template:`模板`,
                    data(){
                        return {

                        }
                    }
                })
                组件名:注意如果存在引号,则组件名必须使用连字符,如果没有引号
                则必须使用大小驼峰
        */
        Vue.component('component-a',{
            /* 设置模板 */
            template:`<p :style="val">{{ content }}</p>`,
            /* 初始化数据 */
            data(){
                return {
                    val:'background-color:coral',
                    content:'全局组件',
                }
            }
        })


        new Vue({
            data:{

            }
        }).$mount('#app1')

```

```
        new Vue({
            /* 2:局部组件 */
            components:{
                /* 注意这里没有引号,使用大小驼峰均可 */
                componentB:{
                    template:`<p :style="val">{{ content }}</p>`,
                    data(){
                        return {
                            val:'background-color:pink',
                            content:'局部组件',
                        }
                    },
                },
            },
        }).$mount('#app2')
    </script>
</body>
</html>
```

2-组件极致化.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>2:组件极致化</title>
</head>
<body>
    <div id="app"></div>
    <script src="../node_modules/vue/dist/vue.js"></script>

    <!-- 引入页眉组件 -->
    <script src="./components/AppHeader.js"></script>
    <!-- 引入主体组件 -->
    <script src="./components/AppMain.js"></script>
    <!-- 引入页脚组件的子组件,先引晚辈再引长辈 -->
    <script src="./components/child/Info.js"></script>
    <!-- 引入页脚组件 -->
    <script src="./components/AppFooter.js"></script>

    <!-- 引入根组件,注意先后顺序,从上往下,先引晚辈再引长辈 -->
    <!-- 这里引入根组件 -->
    <script src="./App.js"></script>
    <!-- 引入主函数 也叫作入口文件 -->
    <script src="./main.js"></script>
</body>
</html>
```

main.js

```
(function () {
    new Vue({
        /* 如果书写了 template 配置项,则后面的标签
        会取代 el 或者 $mount 管理的模板 这里 <app />
        覆盖了 div#app */
        template: '<app />',
        /* 注册子组件 */
        components: {
            /* 注册根组件 */
            App,
        },
    }).$mount('#app')
})()
```

App.js

```
(function () {
    /* Vue2中 模板必须存在根元素,否则报错
    因此这里添加了 div 根标签 */
    const template =
    `<div>
        <!-- 引用页眉组件 -->
        <app-header></app-header>
        <!-- 引用主体组件 -->
        <app-main></app-main>
        <!-- 引用页脚 -->
        <app-footer></app-footer>
    </div>`

    window.App = {
        template,
        /* 注册子组件 */
        components:{
            /* 注册页眉 */
            AppHeader,
            /* 注册主体 */
            AppMain,
            /* 注册页脚 */
            AppFooter,
        },
    }
})()
```

---

day11

前端八股文（最终版）

1. ***什么是单向绑定？什么是双向绑定？***

- ***单向绑定***
  - ***修改 Vue 实例中data初始化的数据，则页面模板中的数据随之发生更改，两者绑定在一起，则称之为单向绑定***
- ***双向绑定***
  - ***修改页面模板中的数据，则管理模板的 Vue 实例中 data 中的数据随之发生更改，这称之为双向绑定***

2. ***Vue2如何进行样式渲染***
   - ***绑定 class***
     - ***初始化的值对应class属性***
     - ***如果初始化的的值为真，则类名存在，如果为假，则类名不存在***
     - ***这里就是一个元素多个类名，没有初始化的数据，注意不要漏加引号，因为这是数组***
   - ***绑定style***
     - ***初始化的值就对应行内式的样式值***
     - ***样式名必须使用小驼峰格式，没有引号，初始化的值对应样式值***
3. ***说明 函数 计算属性 侦听器的不同以及使用场合***
   - ***函数***
     - ***一般绑定事件，当事件激发被调用，也可以直接被调用，函数仅仅支持单向绑定功能，没有缓存机制，只要被调用立即执行，函数体内部可以书写异步代码，可以有选择的书写return语句***
     - ***函数多绑定事件，支持异步功能，没有双向绑定功能***
   - ***计算属性***
     - ***计算属性就是一个属性，由它依赖的初始化的值通过计算得来，只要它依赖的数据发生更改，则计算属性重新执行，计算属性没有括号，也不会传值，仅仅是个，自带缓存机制，不管被强制调用多少次，都仅仅执行一次，除非依赖的属性发生更改，计算属性在书写get和set之后支持单双向绑定***
     - ***由于get中必须书写return语句，因此无法书写异步代码***
   - ***侦听器***
     - ***一般不去考虑单双向绑定问题，就是设置一个值，侦听器去侦听这个数据，只要这个数据发生更改，则侦听器执行，如果设置 immediate 属性，则立即执行侦听器一次，侦听器默认只能侦听基本数据类型，无法侦听复杂类型，如果要侦听复杂类型，则必须设置 deep:true ， 开启深度侦听***
     - ***计算属性能做的事，侦听器都能做到，但是侦听器能做的事，计算属性不一定能实现，例如异步功能***
4. ***Vue2条件渲染的方式***
   - ***v-if***
     - ***如果后面是真值，则元素显示，如果后面是假值，则元素不显示，底层根本不渲染，由于切换消耗较大，因此，适用于切换不频繁的场合***
     - ***v-show***
       - ***如果后面是真值,则元素显示,如果后面是假值,则元素不显示,底层依然渲染,只不过添加了一个 display:none 的行内式,***
       - ***初始载入消耗较大,但是之后切换消耗较小,因此适用于切换频繁的场合***
     - ***v-else-if v-else***
       - ***一般搭配 v-if 使用,不能搭配 v-show,必须紧邻,用来组成简单的流程控制***
5. ***如何使用事件原型获取元素节点***
   - ***event.target***

6. ***简述你使用过的事件修饰符***
    - ***.stop:解决冒泡问题***
    - ***.once:激活一次性事件***
    - ***.prevent:屏蔽元素固有的动作,例如表单 连接提交***
    - ***.native:给组件添加原生事件***
    - ***@keyup.键位名:监听键位的激发***
    - ***@keydown.tab:监听 tab 键激发***
7. ***Vue如何进行列表渲染(迭代数组和迭代对象)***
    - ***v-for="(alias,index) in 数组"***
    - ***v-for="(value,name,index) in 对象"***
    - ***注意存在主键则 :key="主键" 没有主键则 :key="index"***
8. ***使用过滤器应该注意什么***
    - ***不能与 v-model 连用***
    - ***不能使用 this***
9. ***简述你所使用过的指令元素***
    - ***v-once:一次性插值绑定,之后失去绑定功能***
    - ***v-html:向元素中插入超文本,注意!为了防止网络 XSS 攻击,禁止插入脚本***
    - ***v-text:向元素中插入文本***
    - ***v-model:使用在表单项中,支持双向绑定***
        - ***v-model.trim:双向绑定数据,去掉字符串两侧空格***
        - ***v-model.number:双线绑定数据,同时转换为 number,注意如果无法转换,则不转换***
        - ***v-model.lazy:点击回车才会激活双向绑定功能***
    - ***v-bind:语法糖:,绑定元素的属性***
    - ***v-on:语法糖@,绑定事件***
        - ***@事件.once***
        - ***@事件.stop***
        - ***@事件.prevent***
        - ***@事件.native***
        - ***@keyup.键位名***
        - ***@keydown.tab***
    - ***v-for:迭代数据***
    - ***v-if:后面如果为真,则元素显示,为假,元素不显示,底层不渲染***
    - ***v-else-if:搭配 v-if 使用必须紧邻***
    - ***v-else:搭配 v-if 使用必须紧邻***
    - ***v-show:后面如果为真,则元素显示,为假,元素不显示,底层依然渲染,只不过添加了.display:none;css行内式***
    - ***v-pre:提示 Vue 实例不解析***
    - ***v-cloak:解决闪现问题***
    - ***v-slot:父子组件传值时传递模板使用,是插槽的另外一种书写方式***
10. ***Vue2实例对用户书写的 data 对象做了哪些处理?***

    1:数据劫持

    Vue实例获取到用户书写的 data 对象之后,不管封装了几层
    都会对这个对象添加可响应式功能,对象的每个属性都被添加了
    reactiveGetter()和 reactiveSetter() 两个函数

如果 data 中的数据被读取,则执行 reactiveGetter()
如果 data 中的数据被修改,则执行 reactiveSetter(newVal)
这个函数中同时会修改页面模板中对应的数据,这就是可响应式
或者说单向绑定是如何实现的,这些被施加了可响应式功能的数据以及
get 和 set 函数都被封装在 Vue 实例的 _data 对象
这个操作被称之为数据劫持

2:数据代理

一个对象可以对另外一个对象中的属性进行操作,则称之为数据代理
Vue实例通过数据代理将 _data 中的数据代理到 Vue 实例的表层
这样做的好处是,我们在模板中使用指令或者插值语法时,不需要每次都添加
_data 前缀,因为 Vue 实例表层数据都可以直接在模板中使用

11. *Vue2如何实现对象的可响应式,如何实现数组的可响应式*
    - *对象可响应式*
        - *this.$set(对象,'属性名',属性值)*
        - *this.$delete(对象,'属性名')*
        - *Vue.set(对象,'属性名',属性值)*
        - *Vue.delete(对象,'属性名',属性值)*
    - *数组可响应式*
        - *在 Vue2 中必须使用以下七个函数对数组进行修改,这七个函数是 js 版加强版,实现了可响应式功能,注意不是 js 原始的七个,只不过重名,功能相同实现了可响应式功能*
            - *push()*
            - *unshift()*
            - *shift()*
            - *pop()*
            - *sort()*
            - *reverse()*
            - *splice()*
12. *如何进行组件传值*
    - *props父子传值*
        - *父组件 <子组件 :自定义属性="要传递的数据" />*
        - *props:['自定义属性']*
        - *props:{ 自定义属性:数据类型 }*
        - *props:{ 自定义属性:{ type:数据类型,required:true,default:默认值 } }*
    - *$emit子父传值*
        - *父组件 <子组件 @自定义动作="函数">*
        - *this.$emit('父组件中的自定义事件',要传递的值)*
    - *slot父子插槽分发*
        - *父组件*
        - ```
          <子组件>
              <要传递的模板 slot="插槽名" />
              <template v-slot:插槽名>
                  <要传递的模板 />
              </template>
          </子组件>
          ```
        - *子组件*
          ```

13. ***如何给组件绑定原生事件***
    <组件 @原生事件.native="函数" />

Vue组件通讯规则

- ***不要在子组件中修改父组件传递的数据***
- ***数据初始化，要根据初始化的数据是否用于多个组件中，如果需要被应用在多个组件中，则初始化在父组件中，如果只在一个组件中使用，那就初始化在这个要使用的组件中***
- ***数据初始化在哪个组件，更新数据的函数就应该定义在哪个组件***

props父组件向子组件传递数据（父子数据）

- ***props只用于父组件向子组件传递数据***
- ***所有的标签属性多会成为组件对象的属性，模板页面可以直接引用***
- ***如果需要向非子后代传递数据，必须逐层传递***
- ***兄弟组件不能直接使用props通讯，必须借助父组件***

> ***父组件:***

```
1       <子组件 :绑定的属性="初始化的值"></子组件>
```

> ***格式1:***
> ***子组件***
> ***props:['绑定的属性']***
> ***格式2:***
> ***子组件***
> ***props:{***
> ***绑定的属性:数据类型,***
> ***绑定的属性2:数据类型***
> ***}***
> ***数据类型:Number Array Object Function Boolean String***
> ***注意不能传递标签***
> ***格式3:***
> ***props:{***
> ***绑定的属性:{***
> ***type:数据类型,***
> ***required:是否需要设置数据类型,***
> ***default:基本数据类型,***
> ***default:()=>复杂数据类型,***
> ***}***
> ***}***

$emit自定义事件（子父数据）

- 自定义事件只用于子组件向父组件发送数据
- 不能在隔代组件通讯时使用

> 子组件:
> this.$emit('父组件中的自定义事件',要传递的值)

> 父组件:
> <给父组件传值的子组件 :hobbies="mydatas" @自定义事件="要调用的函数"></ 给父组件传值的子组件>

slot插槽分发内容（父子标签+数据）

- 用于父组件向子组件传递 标签和数据(上面那二位只能传递数据)一般用于某个位置需要经常动态切换显示效果
- 数据必须初始化在父组件中

> 父组件

```
<子组件>
   <tagName slot="插槽名"></tagName>
</子组件>
```

> 子组件

```
<slot name="插槽名"></slot>
```

PubSubJs（任意组件数据）

- 用来实现非父子组件之间的通信，使用PubSubJs进行消息发布与订阅模式，来进行数据传递
- 必须安装
  npm install pubsub-js
- 由于是第三方插件，必须使用箭头函数

> 订阅:

```
PubSub.subscribe('订阅的事件',(event,传递的数据)=>{

})
```

> 发布:

```
PubSub.publish('订阅的事件',传递的数据)
```

---

vue2-component-project8

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
    <meta name="description" content="">
    <meta name="author" content="">
    <title>Dashboard Template for Bootstrap</title>
    <!-- Bootstrap core CSS -->
    <link href="style/bootstrap.min.css" rel="stylesheet">
    <!-- Custom styles for this template -->
    <link href="style/dashboard.css" rel="stylesheet">
</head>

<body>
    <div id="app"></div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <!-- 引入页眉 -->
    <script src="./components/AppHeader.js"></script>
    <!-- 引入侧边栏 -->
    <script src="./components/AppAside.js"></script>
    <!-- 引入主体子组件 面板组件 -->
    <script src="./components/children/Dashboard.js"></script>
    <!-- 引入列表组件的子组件 Item 组件 -->
    <script src="./components/children/Item.js"></script>
    <!-- 引入主体子组件 列表组件 -->
    <script src="./components/children/HomeList.js"></script>
    <!-- 引入主体 -->
    <script src="./components/AppMain.js"></script>
    <!-- 引入根组件 -->
    <script src="./App.js"></script>
    <!-- 引入主函数 入口文件 -->
    <script src="./main.js"></script>
</body>

</html>
```

main.js

```
(function () {
    new Vue({
        /* 注册子组件 */
        components: {
            /* 注册根组件 */
            App,
        },
        /* 覆盖 el 或者 mount 管理的模板,根组件替换 */
        template: '<app />',
    }).$mount('#app')
})()
```

App.js

```
(function () {
    const template =
    `<div>
        <!--1)头部导航区域-->
        <!-- 引用页眉 -->
        <app-header></app-header>

        <!--2)核心区域:分左右两边-->
        <div class="container-fluid">
            <div class="row">

                <!--2.1)左边菜单栏区域-->
                <!-- 引用侧边栏 -->
                <app-aside></app-aside>

                <!--2.2)右边主页面区域: 分上下两个区域-->
                <!--
                    3:父子插槽分发
                        <父组件>
                            <要传递的模板 slot="插槽名"/>
                        </父组件>
                -->
                <app-main>
                    <!-- 以下模板也就是 h1 元素传递到子组件中,插入到
                    插槽名为 etoak1 的 slot 中 -->
                    <h1 class="page-header" slot="etoak1">{{ title1 }}
</h1>
                    <!-- 如果针对大量模板,则必须使用 template 对应 v-slot
                    指令,注意此指令只能使用在 template 和 组件中,不能使用在
                    html 标签中 -->
                    <template v-slot:etoak2>
                        <h2 class="sub-header">{{ title2 }}</h2>
                    </template>
                </app-main>
            </div>
        </div>
    </div>`

    window.App = {
        data(){
```

```
            return {
                title1:'面板',
                title2:'雇员信息',
            }
        },
        template,
        /* 注册子组件 */
        components:{
            /* 注册页眉 */
            AppHeader,
            /* 注册侧边栏 */
            AppAside,
            /* 注册主体 */
            AppMain,
        },
    }
})()
```

AppHeader.js

```
(function () {
    const template =
    `<nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container-fluid">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed"
data-toggle="collapse" data-target="#navbar"
                    aria-expanded="false" aria-controls="navbar">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="#">山东易途</a>
            </div>
            <div id="navbar" class="navbar-collapse collapse">
                <ul class="nav navbar-nav navbar-right">
                    <li><a href="#">Dashboard</a></li>
                    <li><a href="#">Settings</a></li>
                    <li><a href="#">Profile</a></li>
                    <li><a href="#">Help</a></li>
                </ul>
                <form class="navbar-form navbar-right">
                    <input type="text" class="form-control"
placeholder="请填入搜索内容...">
                </form>
            </div>
        </div>
    </nav>`

    window.AppHeader = {
        template,
    }
})()
```

AppAside.js

```
(function () {
    const template =
    `<div class="col-sm-3 col-md-2 sidebar">
        <ul class="nav nav-sidebar">
            <li class="active"><a href="#">Overview <span class="sr-only">
(current)</span></a></li>
            <li><a href="#">Reports</a></li>
            <li><a href="#">Analytics</a></li>
            <li><a href="#">Export</a></li>
        </ul>
        <ul class="nav nav-sidebar">
            <li><a href="">Nav item</a></li>
            <li><a href="">Nav item again</a></li>
            <li><a href="">One more nav</a></li>
            <li><a href="">Another nav item</a></li>
            <li><a href="">More navigation</a></li>
        </ul>
        <ul class="nav nav-sidebar">
            <li><a href="">Nav item again</a></li>
            <li><a href="">One more nav</a></li>
            <li><a href="">Another nav item</a></li>
        </ul>
    </div>`

    window.AppAside = {
        template,
    }
})()
```

AppMain.js

```
(function () {
    const template =
    `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
        <!--2.2.1)右边上半区域-->
        <!--
            <h1 class="page-header">Dashboard</h1>
            3:父子插槽分发
                子组件
                    <slot name="插槽名"></slot>
        -->
        <slot name="etoak1"></slot>

        <!-- 引用面板子组件 -->
        <!--
            1.1:props传值
            父组件
                <子组件 :自定义属性="初始化的数据"></子组件>
            2:$emit子父传值
                <子组件 @自定义事件="函数"></子组件>
        -->
        <dashboard :myHobby="hobbies"
```

```
        @etoak="removeHobby"
        hello="请谨慎"></dashboard>

        <!--2.2.2)右边下半区域-->
        <!--
        <h2 class="sub-header">Section title</h2>
        -->
        <slot name="etoak2"></slot>
        <!-- 引用列表子组件 -->
        <!-- 1.2:props传值
            父组件
                <子组件 :自定义属性="初始化的数据"></子组件>

            注意 在组件中如果要绑定原生事件,则必须添加事件修饰符 native
            <组件 @click.native="函数" />
            否则在组件中无法分辨原生事件和自定义事件
        -->
        <home-list :empList="empList"
        :removeEmp="removeEmp"></home-list>
    </div>`

    window.AppMain = {
        data() {
            return {
                /* 初始化数据 给子组件 Dashboard */
                hobbies: ['抽烟', '喝酒', '烫头', '敲代码'],
                /* 初始化数据 给子组件 HomeList */
                empList: [
                    { id: 1, name: 'elena', gender: 0, salary: 25000, },
                    { id: 2, name: 'penny', gender: 0, salary: 25000, },
                    { id: 3, name: 'nancy', gender: 0, salary: 15000, },
                    { id: 4, name: 'aleric', gender: 1, salary: 15000, },
                    { id: 5, name: 'tommy', gender: 1, salary: 35000, },
                    { id: 6, name: 'matt', gender: 1, salary: 45000, },
                    { id: 7, name: 'jack', gender: 1, salary: 55000, },
                    { id: 8, name: 'stefan', gender: 1, salary: 25000, }
                ],
            }
        },
        methods: {
            /* 删除 雇员信息 */
            removeEmp(index){
                this.empList.splice(index,1)
            },
            /* 删除 爱好 */
            removeHobby(index){
                this.hobbies.splice(index,1)
            },
        },
        template,
        /* 注册子组件 */
        components: {
            /* 注册面板子组件 */
            Dashboard,
            /* 注册列表子组件 */
```

```
            HomeList,
        },
    }
})()
```

Dashboard.js

```
(function () {
    const template =
    `<div class="row placeholders">
        <div class="col-xs-6 col-sm-3 placeholder"
        v-for="(hobby,index) in myHobby" >
            <img
src=""
                width="200" height="200" class="img-responsive"
alt="Generic placeholder thumbnail">
            <h4>{{ hobby }}</h4>
            <span class="text-muted" @click="del(index)"
            style="cursor:pointer">{{ hello }}删除</span>
        </div>
    </div>`

    window.Dashboard = {
        template,
        /*
            1.1:props父子组件
            props:['自定义属性']
            只有父子里面的子才书写 props
            凡是 props 里面书写的内容,都直接可以在模板中使用,甚至我们可以
            将 props 视作 data
        */
        props: ['myHobby','hello'],
        methods: {
            del(index){
                /* 2:$emit子父传值
                    this.$emit('激发父组件中的自定义事件',要传递的值)
                    此处激发了父组件中的事件 etoak,之后传递了 index 索引值
                */
                this.$emit('etoak',index)
            },
        },
    }
})()
```

HomeList.js

```
(function () {
    const template =
    `<div class="table-responsive">
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>序号</th>
```

```
                    <th>姓名</th>
                    <th>性别</th>
                    <th>薪资</th>
                    <th>操作</th>
                </tr>
            </thead>
            <tbody>
                <!--
                <tr v-for="(emp,index) in empList" :key="emp.id">
                    <td>{{ index+1 }}</td>
                    <td>{{ emp.name }}</td>
                    <td>{{ emp.gender===0?'女':'男' }}</td>
                    <td>{{ emp.salary }}</td>
                </tr>
                    引用子组件
                        :index="index" 向子组件传递索引
                        :emp="emp" 向子组件传递对象
                        :removeEmp="removeEmp" 向子组件传递函数
                -->
                <item v-for="(emp,index) in empList" :key="emp.id"
                :index="index" :emp="emp" :removeEmp="removeEmp"></item>
            </tbody>
        </table>
    </div>`

    window.HomeList = {
        template,
        /*
            1.2:props 传值
            props:{
                自定义属性:数据类型,
            }
            数据类型支持以下几种
            String,Number,Boolean,Array,Function,Object
        */
        props:{
            /* 接受父组件传递过来的数组 */
            empList:Array,
            /* 接受父组件传递过来的函数 */
            removeEmp:Function,
        },
        /* 注册子组件 */
        components:{
            Item,
        },
    }
})()
```

Item.js

```
(function () {
    const template =
    `<tr>
        <td>{{ index+1 }}</td>
        <td>{{ emp.name }}</td>
```

```
        <td>{{ emp.gender===0?'女':'男' }}</td>
        <td>{{ emp.salary }}</td>
        <td><span style="cursor:pointer"
        @click="removeEmp(index)">删除</span></td>
    </tr>`

    window.Item = {
        template,
        /*
            1.3:props父子传值

            props:{
                自定义属性:{
                    type:数据类型,
                    required:true,  //表示必须传递这个类型的数据
                    default:默认值,  //如果没有传递,则使用此默认值
                }
            }
        */
        props:{
            /* 接受父组件传递过来的索引 */
            index:{
                type:Number,
                required:true,
            },
            /* 接受父组件传递过来的对象 */
            emp:{
                type:Object,
                required:true,
            },
            /* 接受父组件传递过来的函数 */
            removeEmp:{
                type:Function,
                required:true,
            },
        },
    }
})()
```

插槽slot

```
<!--父组件-->
<app-main>
    /* 插槽slot */
    <h1 class="page-header" slot="etoak1">{{title1}}</h1>
    /* v-slot */
    <template v-slot:etoak2>
        <h2 class="sub-header">{{title2}}</h2>
    </template>
</app-main>

data(){
```

```
    return {
        title1:'面板',
        title2:'雇员信息',
    }
},
```

```
<!--子组件-->
<slot name="etoak1"></slot>

<slot name="etoak2"></slot>
```

---

day12

普通组件与路由组件

> *注意!官方从来没有普通组件和路由组件一说,就是组件.本文仅为便于记忆*

- ***普通组件***
  - ***永远放置在某个位置,不会根据哈希值的变动切换显示与否,或者切换位置***
  - ***被注册父组件中,或者根组件中***
  - ***一般被放置在 components 包中***
  - ***永远被引用在页面中,直接书写组件名的连字符格式***
- ***路由组件***
  - ***会随着哈希值的变动选择显示在哪里,是否显示***
  - ***被注册在路由表中***
  - ***一般被放置在 views 包中***
  - ***永远显示在 路由出口中***

---

vue2-router-project9

1-路由入门.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>1:路由入门</title>
    <style>
        .router-link-active{
            border:solid 1px silver;
            background-color: tomato;
            color:whitesmoke;
        }
    </style>
</head>
<body>
```

```
    <div id="app">
        <ul>
            <li><a href="#/bar">显示 Bar 组件</a></li>
            <li><a href="#/foo">显示 Foo 组件</a></li>
            <li>
                <!-- 设置激活哈希值的标签,默认取代 a 链接
                    这是 vue-router 提供的标签,不是 html 标签
                    to:属性,相当于 a 连接的 href,注意省略
                    #,默认情况下浏览器会将其解析为 a 链接
                -->
                <router-link to="/bar">显示 Bar 组件</router-link>
            </li>
            <li>
                <router-link to="/foo">显示 Foo 组件</router-link>
            </li>
            <!-- tag:如果设置此属性,则浏览器会根据属性值进行解析,
            这里 router-link 被浏览器视作 li 标签,如果不写,则默认是 a -->
            <router-link to="/bar" tag="li">显示 Bar 组件</router-link>
            <router-link to="/foo" tag="li">显示 Foo 组件</router-link>
        </ul>
        <hr>
        <!-- 设置路由出口,注意此标签为 vue-router 插件提供,
        不是 html 标签 -->
        <router-view></router-view>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <!-- 引入路由依赖 -->
    <script src="../node_modules/vue-router/dist/vue-router.js"></script>
    <!-- 引入组件 -->
    <script src="./views/Bar.js"></script>
    <script src="./views/Foo.js"></script>
    <!-- 引入路由表 -->
    <script src="./router/index.js"></script>
    <script>
        /*
            VueRouter
                Vue的第一个官方插件,用来控制页面中的组件,何时显示,
                是否显示,显示在哪里,从而实现页面上组件的切换
                要使用路由,则必须下载依赖
                npm i vue-router@3.5.1 -S
        */
        new Vue({
            /* 加载路由表 router:router, */
            router,
        }).$mount('#app')
    </script>
</body>
</html>
```

index.js(路由表)

```
(function () {
    /* 新建一个路由表 */
    window.router = new VueRouter({
        /* 设置为哈希模式,会根据当前浏览器地址栏的哈希值
        显示组件,注意默认就是哈希模式,可以不写,除此之外还有
        history 历史模式,但是历史模式,需要后端框架支持,否则无法原地刷新 */
        mode:'hash',
        /* 设置路由 注意单词拼写 这里是 routes 不是 router */
        routes: [
            {
                /* path:表示正在被监听的哈希值,注意#省略 */
                path:'/bar',
                /* 出现哈希值之后从路由出口显示的组件 */
                component:Bar,
            },
            {
                path:'/foo',
                component:Foo,
            },
            {
                path:'/',
                /* 如果是#/则重定向哈希为#/bar */
                redirect:'/bar',
            }
        ],
    })
})()
```

Bar.js(子组件)

```
(function () {
    const template =
    `<p :style="val">Bar组件</p>`

    window.Bar = {
        template,
        data(){
            return {
                val:'background-color:purple',
            }
        },
    }
})()
```

Foo.js(子组件)

```
(function () {
    const template =
    `<p :style="val">Foo组件</p>`

    window.Foo = {
        template,
        data() {
            return {
                val: 'background-color:coral',
            }
        },
    }
})()
```

---

vue2-router-project10

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
    <meta name="description" content="">
    <meta name="author" content="">
    <title>Dashboard Template for Bootstrap</title>
    <!-- Bootstrap core CSS -->
    <link href="style/bootstrap.min.css" rel="stylesheet">
    <!-- Custom styles for this template -->
    <link href="style/dashboard.css" rel="stylesheet">
</head>

<body>
    <div id="app"></div>
    <script src="../node_modules/vue/dist/vue.js"></script>

    <!-- 引入路由依赖 -->
    <script src="../node_modules/vue-router/dist/vue-router.js"></script>

    <!-- 引入页眉 -->
    <script src="./components/AppHeader.js"></script>
    <!-- 引入侧边栏 -->
    <script src="./components/AppAside.js"></script>
    <!-- 引入主体子组件 面板组件 -->
    <script src="./components/children/Dashboard.js"></script>
    <!-- 引入列表组件的子组件 Item 组件 -->
    <script src="./components/children/Item.js"></script>
```

```
    <!-- 引入主体子组件 列表组件 -->
    <script src="./components/children/HomeList.js"></script>
    <!-- 引入主体 -->
    <script src="./components/AppMain.js"></script>

    <!-- 引入 News 的子路由 Sport 和 Tech -->
    <script src="./views/children/Sport.js"></script>
    <script src="./views/children/Tech.js"></script>
    <!-- 引入路由组件 新闻和关于我们 -->
    <script src="./views/News.js"></script>
    <script src="./views/About.js"></script>

    <!-- 引入根组件 -->
    <script src="./App.js"></script>

    <!-- 引入路由表 -->
    <script src="./router/index.js"></script>

    <!-- 引入主函数 入口文件 -->
    <script src="./main.js"></script>
</body>

</html>
```

index.js(路由表)

```
(function () {
    window.router = new VueRouter({
        /* 此处表示判定本路由链接中只要被激活(链接对应的哈希与当前浏览器
            哈希相同)则添加后面的属性值,这个值就是 class */
        linkActiveClass:'active',
        routes:[
            {   /* 此处为默认哈希 #/ */
                path:'/',
                component:AppMain,
            },
            {
                path:'/news',
                component:News,
                /* 设置默认值,否则必须点击体育和科技之后才能显示子路由
                这里一点击新闻,立刻显示一个默认的子路由 */
                redirect:'/news/sport',
                /* 开启子路由 */
                children:[
                    {
                        path:'/news/tech',
                        component:Tech,
                    },
                    {
                        path:'/news/sport',
                        component:Sport,
                    }
                ]
            },
            {
```

```
                path:'/about',
                component:About,
            }
        ]
    })
})()
```

New.js(子组件)

```
(function () {
    const template =
    `<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
        <div class="header clearfix">
            <nav>
                <ul class="nav nav-pills">
                    <!--
                    <li class="active"><a href="#">体育</a></li>
                    <li><a href="#">科技</a></li>
                    -->
                    <router-link to="/news/sport" tag="li">
                        <a>体育</a>
                    </router-link>
                    <router-link to="/news/tech" tag="li">
                        <a>科技</a>
                    </router-link>
                </ul>
            </nav>
            <hr>
        </div>
        <!-- 此处要么显示体育 要么显示科技 二选一,因此设置路由出口 -->
        <router-view />
        <!--体育栏目-->

        <!--科技栏目-->

    </div>`

    window.News = {
        template,
    }
})()
```

1-vue生命周期.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
    <title>vue2生命周期</title>
</head>
<body>
    <div id="app">
        <button @click="touch">{{ count }}</button>
        <!-- $destroy():强制 Vue 实例销毁 -->
        <button @click="$destroy()">不活了!</button>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <script>
        const vm = new Vue({
            data:{
                count:0,
            },
            beforeCreate() {
                console.log('beforeCreate----------------')
                /*
                    this._data:获取实现数据劫持后添加了可响应式的数据
                    this.$el:获取 Vue实例管理的模板
                    this.touch():表示调用函数
                    beforeCreate()
                    + 执行此钩子之前 Vue实例还未挂载任何模板,
                    也未初始化任何数据,数据劫持 数据代理都没有发生,
                    函数也没有准备好
                */
                console.log(this._data,this.$el,)
            },
            created() {
                /*
                    created()
                    + 在此步之前,数据代理 和 数据劫持已经完成用户的
                    数据已经被施加可响应式之后代理到Vue实例表层,函数也
                    已经全部准备好,激发事件或者直接可以调用,此时created中
                    是最早获取用户数据的时机,因此此处可以发送异步
                */
                console.log('created----------------')
                console.log(this._data, this.$el,this.touch())
            },
            beforeMount() {
                /*
                    beforeMount()
                    + 在此钩子执行之前,首先查看是否存在el配置项如果没有则
                    调用mount加载模板,之后再查看是否存在template配置项
                    如果存在template则使用render函数进行渲染,模板就是
                    template配置的的模板,如果没有template则使用el或者
                    mount作为模板,此时Vue管理的模板已经准备好,但是Vue实例
                    还未将数据挂载到模板中,页面直接显示插值语法和指令语法,
                    所有的DOM操作都不奏效,此时呈现的是虚拟DOM
                */
                console.log('beforeMount----------------')
                console.log(this._data, this.$el)
                //debugger
            },
            mounted() {
                /*
```

```
                    mounted()
                    + 此时Vue实例已经将数据挂载到模板中,真实DOM将虚拟DOM覆盖
                    ,页面就是最终的状态,此时可以发送异步,操作DOM等
                */
                console.log('mounted----------------')
                console.log(this._data, this.$el)
            },
            beforeUpdate() {
                /*
                    beforeUpdate()
                    + 只要data更改,则此生命周期执行,
                    此时初始化的数据已经更新为新的数据,新的虚拟DOM已经生成
                    但是还未转换为新的真实DOM,页面模板中显示的还是老的数据
                    这也是唯一一个 数据与模板不统一的时机
                */
                console.log('beforeUpdate----------------')
                console.log(this._data, this.$el)
                //debugger
            },
            /*
                beforeDestroy()
                + 在销毁之前执行的最后的生命周期钩子,此时 data methods
                等都可以正常工作,绑定依然有效一般在此进行收尾工作,例如清除缓存
                ,关闭定时器取消订阅等
            */
            beforeDestroy() {
                console.log('beforeDestroy----------------')
                console.log(this._data, this.$el)
            },
            /*
                destroyed()
                + 实例销毁后调用。该钩子被调用后，对应 Vue 实例的所有
                指令都被解绑，所有的事件监听器被移除，所有的子实例也都被销毁。
                注意这里的事件是指自定义事件,而原生事件会一直有效,
                函数依然可以执行,但是没有绑定功能
            */
            destroyed() {
                console.log('destroyed----------------')
            },
            methods: {
                touch(){
                    console.log('我是touch----------------')
                    this.count++
                }
            },
        })
        vm.$mount('#app')
        /*
            以下几个生命周期在 日常开发 和 面试中经常被使用
            1:created() 最早获取数据,发送异步
            2:mounted() 页面已经被真实 DOM 覆盖完毕,处于稳定状态 此时可以操作
DOM,
            也可以发送异步
            3:beforeUpdate() 唯一模板与数据不统一的时机
```

```
            4:beforeDestroy() 即将要销毁,一般用来清除缓存删除已经占用的资源等善后工作
        */
    </script>
</body>
</html>
```

2-axios实现异步功能.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>2:axios 实现异步功能</title>
</head>
<body>
    <div id="app">
        <table border="1px">
            <tr>
                <th>序号</th>
                <th>姓名</th>
                <th>性别</th>
                <th>薪资</th>
            </tr>
            <tr v-for="(emp,index) in empList" :key="emp.id">
                <td>{{ index+1 }}</td>
                <td>{{ emp.name }}</td>
                <td>{{ emp.gender===0?'女':'男' }}</td>
                <td>{{ emp.salary }}</td>
            </tr>
        </table>
    </div>
    <script src="../node_modules/vue/dist/vue.js"></script>
    <!--
        在 Vue 中没有异步模块,因此无法实现使用 Vue 指令等发送异步,
        如果想发送异步,则必须使用其他技术,在 Vue 中整合 axios 是目前
        使用最多的一个搭配,这里 axios 对于 Vue 来说算是一个第三方插件
        因此需要注意以下两个关键问题
            1:下载第三方依赖
                npm i axios -S
            2:注意如果是第三方依赖,则必须使用箭头函数,否则 this 指向 undefined
            而不是 Vue 实例了
        以下引入 axios 依赖
    -->
    <script src="../node_modules/axios/dist/axios.js"></script>
    <script>
        new Vue({
            data:{
                empList:[],
            },
            methods: {
                fetchData(){
```

```
                    /* 使用 axios 向服务器端发送异步请求,
                    接受响应,axios是异步对象,通过此对象可以发送异步请求
                    axios 是一个 Promise 承诺对象 */
                    axios.get('http://127.0.0.1:5500/src/db/empList.json')
                    /* 异步成功的回调 response:形参,表示服务器返回的数据 */
                    .then(response=>{
                        console.log('then执行了......')
                        console.log(response)
                        /* 这里 response.data 就是封装的服务器端返回的数据
                        这就是 json中的数据已经被转换为js数组 */
                        this.empList = response.data
                    })
                    /* 异步失败的回调 */
                    .catch(err=>{

                    })
                    /* 不管成功失败都执行 */
                    .finally(()=>{
                        console.log('finally执行了.......')
                    })
                },
            },
            created() {
                /* 函数早于 created 早已经准备好,在 created 中可以
                直接调用 */
                this.fetchData()
            },
        }).$mount('#app')
    </script>
</body>
</html>
```

day13模块化开发

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>js模块化开发</title>
</head>
<body>
    <!--
        使用 js 原生模块化开发需要注意以下两个问题
            1:必须书写 type="module"
            2:必须假设到服务器端
    -->
    <script src="./js/main.js" type="module"></script>
```

```
</body>
</html>
```

bar.js

```
/*
    模块化开发
    在 ES6 之前 js 就存在各种模块化开发的语法规范,但是由于一直未统一
    所以语法较为混乱,在 ES6 之后,语法已经完全统一,本文全部使用 ES6
    的模块化语法
    每个独立的 js 或者其它资源例如 css 等,都被视作一个模块(Module)
    每个模块都存在以下两个功能
        1:导出功能:一个模块只有导出了数据,其它模块才可以使用导入
            1.1:导出默认成员:一个模块只能导出一个默认成员,不需要
            指定成员名
                export default xxx
            1.2:导出普通成员:一个模块可以导出任意个普通成员,必须
            指定成员名
                export xxx
        2:导入功能
            import 成员名|别名 from 路径
*/
/* A:导出默认成员,成员是一个对象 */
/* export default {
    name:'胡桃',
    age:17,
} */
/* B:导出默认成员,成员是一个函数 */
/* export default function(){
    console.log('我是函数!!!!!!')
} */
/* C:导出默认成员,成员是普通类型 */
export default '吃个桃桃'

/* D:导出多个普通成员,什么类型都有,必须指定成员名 */
export let str = '我是字符串'
export const arr = [1,2,3,4]
export const obj = {
    oid:1,
    oname:'penny',
    oaddress:'济南',
}

export function sum(a,b,c,d){
    return a+b+c+d
}
```

main.js

```
/*
    A:导入默认成员,成员是个对象
    B:导入默认成员,成员是个函数
    C:导入默认成员,成员是普通类型
*/
```

```
import bar from './bar.js'

//A:直接打印对象
//C:直接打印字符串
console.log(bar)

//B:直接调用函数
//bar()

/* D:导入多个普通成员,成员名必须对应 */
import {str,arr,obj,sum} from './bar.js'

console.log(str,arr,obj,sum(...arr))

/*
    E:导入模块的全部成员
    import * as 别名 from 路径
*/
import * as all from './bar.js'

console.log(all,all.default)


/*
    import Vue from '../node_module/vue/dist/vue.js'
*/
```

---

webpack(不常用,参考模板)

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>webpack</title>
</head>
<body>
    <!-- <script src="./src/main.js"></script> -->
    <!-- 引入打包好的文件 -->
    <!-- <script src="./dist/bundle.js"></script> -->
    <div id="app"></div>
</body>
</html>
```

main.js

```
/*
    导入 Vue 依赖
    注意此处自动导入的并不是我们之前常用的完全版的 vue.js,而是一个运行版
    vue.common.js,这个版本缺失以下两个功能
    1:无法编译 .vue 后缀的文件
    2:无法将根组件渲染到页面中替换div#app
    解决方案
        A:使用完全版 不推荐
        import Vue from '../node_modules/vue/dist/vue.js'
        打包体积是运行版的接近五倍
        B:弥补缺失的功能
            1:无法编译 .vue 后缀的文件
                安装 webpack 插件
                "vue-loader": "^15.9.5",
                "vue-template-compiler": "^2.6.12",
            2:无法将根组件渲染到页面中替换div#app
                Vue 实例中 template 配置项已然失效,我们手动配置 render 函数
                之前书写 template 其实就是调用底层的 render 函数,这里我们
                自己调用,将根组件渲染到模板中替换 div#app
*/
import Vue from 'vue'
/* 导入根组件 */
import App from './App.vue'

new Vue({
    /* 注册根组件 */
    /* components:{
        App,
    }, */
    /* 替换 el 或者 mount 管理的模板 */
    //template:'<app/>',
    /* 手动书写 render 将根组件渲染到模板中 */
    //render:function(h){
        /* h是一个函数可以将传递的实参,这里是根组件渲染到页面中替换
        div#app */
        //return h(App)
    //},
    render:h => h(App)
}).$mount('#app')
```

App.vue

```
<template>
    <!-- 书写模板,注意在 Vue2 中必须存在根元素 -->
    <div class="container">
        <!-- 引用页眉 -->
        <app-header></app-header>
        <!-- 引用主体 -->
        <app-main></app-main>
    </div>
</template>

```

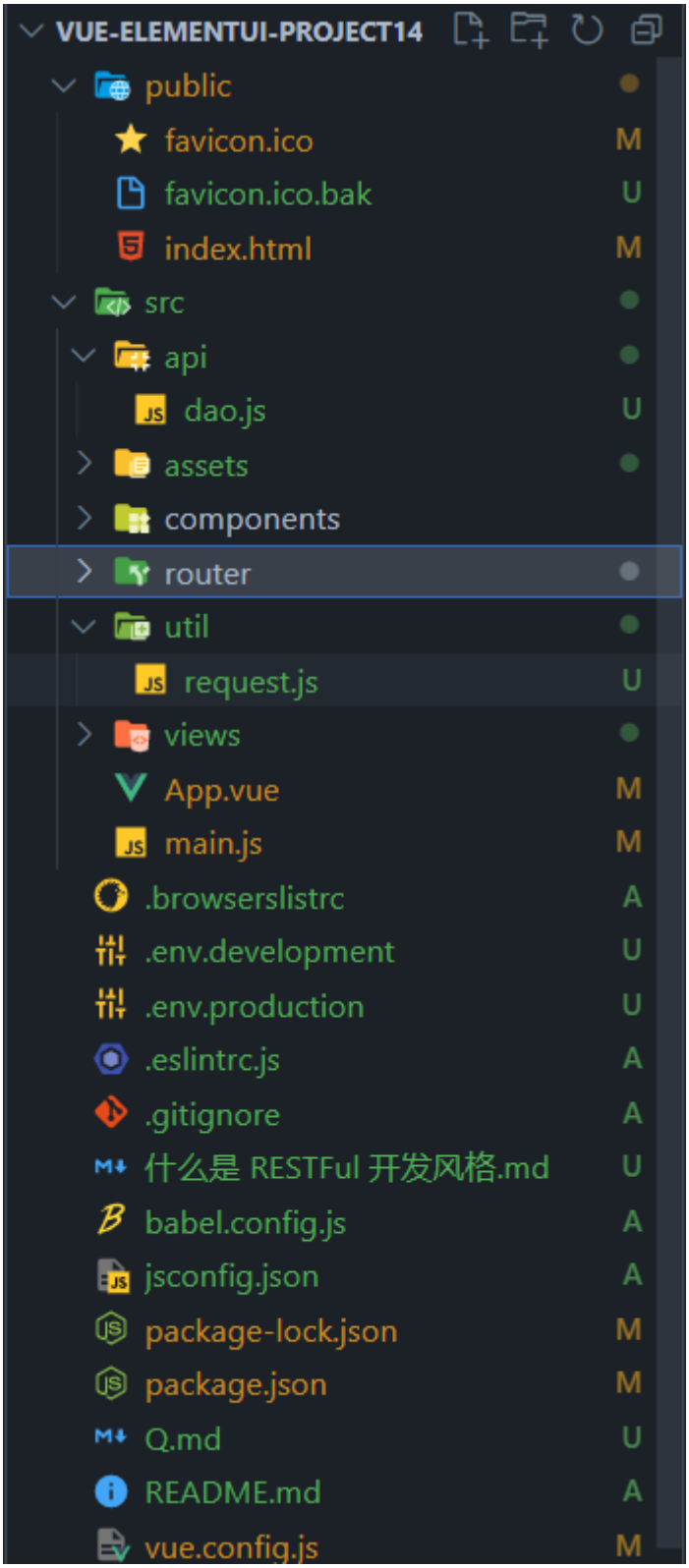
```vue
<script>
    /* 导入页眉和主体组件 */
    import AppHeader from './components/AppHeader.vue'
    import AppMain from './components/AppMain.vue'

    export default {
        /* 注册子组件 */
        components:{
            AppHeader,
            AppMain,
        }
    }
</script>

<style>
    /* 此处书写模板的样式,注意 scoped,如果存在此参数,则书写的样式
    仅对本组件有效,如果没有此参数,则同时影响子组件 */
    html,body{
        margin:0;
        padding:0;
        font-family: 喜鹊招牌体;
        height:100%;
    }
    .container{
        margin:0 auto;
        width:80vw;
        height:100vh;
        display: flex;
        flex-direction: column;
        box-shadow: 10px 10px 10px silver;
    }
</style>
```

day14(ElementUI 注册登录)

什么是 RESTFul 开发风格

> *RESTFul 表述性状态转移 接口名称*
> *根据需求不同(增删查改)选择使用何种方式提交请求*

- *post:一般适用于添加操作,可以发送 json,也可以通过地址栏传递值,没有大小限制,支持中文*
- *get:一般适用于查询操作,不能发送 json,只能通过地址栏传递值,最多 2000 个字符,不支持中文*
- *delete:一般适用于删除操作,不能发送 json,只能通过地址栏传递值,最多 2000 个字符,不支持中文*
- *put:一般适用于修改操作,可以发送 json,也可以通过地址栏传递值,没有大小限制,支持中文*



api dao.js

```
/* 导入自定义 axios 实例 */
import request from '@/util/request'

/* API  application program interface 此处表示应用程序接口 */
export default {
    /* 1:登录
        username:形参 用户姓名
        password:形参 用户密码
```

```
    */
    login(username,password){
        /* 返回一个 */
        return request({
            /* url:此处表示设置进阶地址
            注意最终发送的完整地址是 基本地址+进阶地址 */
            url:`/testUser/login?
username=${username}&password=${password}`,
            method:'get',
        })
    },
    /* 2:注册
        pojo:形参,对象,内部封装了要注册的八个字段
    */
    reg(pojo){
        return request({
            url:'/testUser/add',
            method:'post',
            /* 发送 json 注意这里直接发送 js 对象或者数组即可,不需要
            自己转换 */
            data: pojo,
        })
    },
}
```

.env.development

```
# 只有以 VUE_APP_ 开头的变量会被 webpack 静态嵌入到项目中进行使用
process.env.VUE_APP_xxxxxx
# 在项目任意模块文件中，都可以使用 process.env.VUE_APP_BASE_API 获取值

# 目标服务接口地址，这个地址是按照你自己环境来的， 添加 或者更改配置后，需要重启服务
# VUE_APP_SERVICE_URL = 'https://www.easy-
mock.com/mock/5f40867637dd743fd5db5cf2'
# VUE_APP_SERVICE_URL = 'http://rap2.taobao.org:38080/app/mock/264680'

# 开发环境的前缀
# 以下为设置的基本地址
VUE_APP_BASE_API = 'http://db.etoak.com:9527'


# 本文为开发阶段使用的配置文件,所有重要的参数都放置在本文件中,注意更改此文件
# 必须重启服务器
```

.env.production

```
## 本文件未使用,配置的都是生产阶段的参数
```

vue.config.js

```
module.exports = {
    devServer: {
```

```
        port: 8001, // 端口号，如果端口被占用，会自动提升 1
        open: true, // 启动服务自动打开浏览器
        https: false, // 协议
        host: "localhost", // 主机名，也可以 127.0.0.1 或 做真机测试时候 0.0.0.0
    },
    lintOnSave: false, // 是否关闭eslint语法检查，默认 true，警告仅仅会被输出到命令行，且不会使得编译失败。
    outputDir: "dist", // 默认是 dist ,存放打包文件的目录,
    assetsDir: "assets", // 存放生成的静态资源 (js、css、img、fonts) 的 (相对于 outputDir) 目录
    indexPath: "out/index.html", // 默认 index.html，指定生成的 index.html 的输出路径 (相对于 outputDir)。
    productionSourceMap: false, // 打包时，不生成 .map 文件，加快打包构建
};

```

router index.js

```
/* 引入 Vue 依赖 */
import Vue from "vue";
/* 引入 VueRouter 依赖 */
import VueRouter from "vue-router";
/* Vue加载插件 VueRouter */
Vue.use(VueRouter);

const routes = [
    {
        path:'/',
        /*
            @:忽略当前路径,直接从工程 src 目录下寻找
            如果文件命名为 index.vue 则仅仅导包即可
        */
        component:()=>import('@/views/login'),
    },
    {
        path:'/register',
        component:()=>import('../views/register')
    },
    {
        path:'/layout',
        component:()=>import('@/views/layout')
    }
];

const router = new VueRouter({
    routes,
});

export default router;
```

util request.js

```

```

```
/* 导入 axios 依赖 */
import axios from 'axios'

/* 创建一个自定义 axios 实例,这个自定义 axios 实例可以根据我们自己的需求
实现很多特定功能,注意这个自定义 axios 实例也是一个 Promise 对象,如果发送异步成功
则底层自动调用 resolve(response),则外界可以调用 then(response=>{})
如果内部异步失败,则自动调用 reject(err),则外界可以调用 catch(err=>{})
不管失败与否,外界都可以调用 finally(()=>{}) */
const request = axios.create({
    /* 设置发送异步的基本地址,注意这里仅仅是基本地址,并不是完整地址 */
    baseURL: process.env.VUE_APP_BASE_API,
    /* 设置超时时间 */
    timeout: 5000,
})

/* 导出自定义 axios 实例 */
export default request
```

main.js

```
/* 导入 Vue 依赖注意这里是运行版,因此需要手写 render 渲染根组件到模板 */
import Vue from "vue";
/* 导入 ElementUI 的依赖 */
import ElementUI from 'element-ui';
/* 导入 ElementUI 全局 css */
import 'element-ui/lib/theme-chalk/index.css';
/* 导入根组件 */
import App from "./App.vue";
/* 导入路由表 */
import router from "./router";
/* 关闭控制台部署提示 */
Vue.config.productionTip = false;
/* Vue 实例加载 ElementUI 图形界面库 */
Vue.use(ElementUI);

new Vue({
    /* 加载路由表 */
    router,
    /* 手写 render 将根组件覆盖到模板中 div#app 处 */
    render: (h) => h(App),
}).$mount("#app");

```

App.vue

```
<template>
    <!-- 此处直接显示路由组件 login register layout 三选一
    默认情况显示 loing -->
    <router-view />
</template>

```

```
<script>
    export default {

    }
</script>

<style>
    html,body{
        margin:0;
        padding:0;
        font-family: 喜鹊招牌体;
        height:100%;
    }
</style>
```

views login index.vue(登录)

```
<template>
    <!--
        此处使用了 ElementUI 的 Container 容器组件
        此组件天生自带弹性盒子
    -->
    <el-container class="login-container">
        <!-- 此处使用了 ElementUI 的 Form 表单
            el-form:表单组件
                :model:双向绑定登录的表单对象
                :rules:对应表单项的验证规则
                ref:相当于 js 中的 id,多搭配 Vue 选择器,这是 Vue 自己的语法
                size:表单尺寸 存在 medium small mini 三种尺寸
                status-icon:如果验证成功显示绿色对勾,验证失败红色叉号
                label-width:设置表单项左侧的文本长度,如果设置为 auto,则自动
                调整
        -->
        <el-form :model="loginForm" :rules="loginRules"
        ref="myLoginForm" size="small" status-icon label-width="auto"
        class="login-form">
            <h1>用户登录</h1>
            <!--
                el-form-item:在表单项外侧嵌套
                label:表示表单项左侧(默认)的文本
                prop:对应验证规则的属性名,除了对应验证规则,还对应重置功能,
                如果此处不写,则重置功能失效
            -->
            <el-form-item label="用户姓名" prop="username" >
                <!-- 此处使用了 ElementUI 的 Input 输入框组件
                        v-model:双向绑定封装的表单项,注意使用
                        .trim 无效,自动去掉空格
                -->
                <el-input v-model="loginForm.username"
                placeholder="请输入用户姓名"></el-input>
            </el-form-item>

            <el-form-item label="用户密码" prop="password" >
                <!-- show-password:注意此处如果添加了此属性,则自动转换为
```

```
                密码输入框 -->
                <el-input v-model="loginForm.password"
                show-password placeholder="请输入用户密码"></el-input>
            </el-form-item>

            <el-form-item>
                <!-- 此处使用了 ElementUI 的 Button 组件
                    el-button:表示按钮
                        type: primary warning success danger info  text
                                蓝色     黄色      绿色     红色    灰色   无色
                        round:圆角按钮
                        plain:朴素按钮(空心)
                        size: medium small mini 对应按钮尺寸
                -->
                <el-button type="primary" round size="mini"
                class="btn" @click="submitForm('myLoginForm')">登录</el-
button>

                <!-- @click="resetForm('myLoginForm')" 点击取消激发函数
                注意这里传递的是 ref 值的实参 -->
                <el-button round size="mini"
                @click="resetForm('myLoginForm')">取消</el-button>

                <!-- 在引入 vue-router 路由插件之后,则可以使用路由中提供得
                两个内置对象,$router 和 $route -->
                <el-button type="primary" round plain size="mini"
                @click="$router.push({path:'/register'})">注册</el-
button>
            </el-form-item>
        </el-form>
    </el-container>
</template>

<script>
    /* 导入 dao */
    import dao from '@/api/dao'

    export default {
        data(){
            return {
                /* 封装登录的初始化的表单项,此处对应 :model */
                loginForm:{
                    /* 此处对应 <el-input v-model="对应这里" > */
                    username:'',
                    password:'',
                },
                /* 封装每个表单项的验证规则,此处对应 :rules */
                loginRules:{
                    /* 验证规则属性名:此处对应 <el-form-item prop="对应这里">
*/
                    username:[
                        /*
                            required:true 必填
                            message:报错信息
                            trigger:事件
```

```
                            min:最小长度 max:最大长度
                        */
                        { required:true,message:'请输入用户姓
名',trigger:'blur', },
                        { min:6,max:10,message:'姓名在6到10位之
间',trigger:'blur', }
                    ],
                    password:[
                        { required:true,message:'请输入用户密
码',trigger:'blur', },
                        { min:6,max:32,message:'密码在6到32位之
间',trigger:'blur', }
                    ]
                },
            }
        },
        methods:{
            /* 此函数用来重置表单,这里 formName 是形参,对应表单的 ref 值 */
            resetForm(formName){
                /*
                    this.$refs[formName]
                    相当于
                    this.$refs.formName
                    相当于
                    document.getElementById('formName')
                    这就 Vue 选择器
                    resetFields():内置函数,表示重置特定的表单,注意
                    每个表单项必须存在 prop 属性,否则此功能对表单项
                    无效
                */
                this.$refs[formName].resetFields()
            },
            /* 此函数用来提交表单,这里 formName 是形参,对应 ref 值 */
            submitForm(formName){
                /* validate:在提交表单时,启动表单的验证功能,
                如果全部验证成功,则 valid 为 true,只要存在一个失败的,则
                valid 为 false */
                this.$refs[formName].validate(valid=>{
                    if(valid){

 dao.login(this.loginForm.username,this.loginForm.password)
                        .then(response=>{
                            console.log(response)

                            /* 此处使用了 ElementUI 的 Message 信息提示组件 */
                            this.$message({
                                /*
                                    success/warning/info/error
                                    绿色      黄色      银色    红色
                                */
                                type:
response.data.flag?'success':'error',
                                /* 信息提示的信息 */
                                message:response.data.msg,
                                /* 显示关闭按钮 */
```

```
                                showClose:true,
                            })

                            /* 如果业务逻辑一切正常 */
                            if(response.data.flag){
                                /* 获取权限信息 */
                                const user = response.data.data
                                /* 如果能获取权限 */
                                if(user){

 localStorage.setItem('et2301elementui',
                                    JSON.stringify(user))

                                    setTimeout(() => {
                                        /* 激活哈希完成路由切换 */
                                        this.$router.push({
                                            path:'/layout',
                                        })
                                    }, 1200);
                                }
                            }
                        })
                        return
                    }
                })
            },
        },
    }
</script>

<style scoped>
    /* 设置外侧大容器 */
    .login-container{
        /* 设置容器绝对定位 */
        position: absolute;
        /* 设置容器宽度 */
        width:100vw;
        /* 设置容器高度 */
        height:100vh;
        /* 设置容器背景图片 no-repeat:表示如果图片大小不合适
        则不会重叠摆放 */
        background: url("../../assets/bg1.jpeg") no-repeat;
        /* 设置背景图尺寸 宽度 高度 */
        background-size: 100vw 100vh;
        /* 设置背景图不会随着缩放变动 */
        background-attachment: fixed;
        /* 容器开启弹性盒子 */
        display: flex;
        /* 设置项目,也就是表单 主轴 交叉轴居中 此时默认 水平是主轴
        纵向是交叉轴 */
        justify-content: center;
        align-items: center;
    }

    /* 设置表单 */
```

```
    .login-form{
        /* 设置表单宽度 */
        width:350px;
        /* 设置表单背景色 和 透明度
        rgba(红色,绿色,蓝色,透明度)
        红绿蓝为 0-255 的整数 如果都是 255 则是白色
        透明度 0 完全透明 1 完全不透明 */
        background-color: rgba(255, 255, 255, 0.6);
        /* 设置边框为圆角 数字越大越圆,如果设置为 50%则变为圆球
        表格不能设置 */
        border-radius: 30px;
        /* 给表单开启弹性盒子 */
        display: flex;
        /* 设置表单内的项目从上往下排列,默认是 row 从左往右 */
        flex-direction:column;
        /* 设置交叉轴水平居中摆放,注意这里从左往右是交叉轴了,从上往下
        才是主轴 因为排列顺序是从上往下 */
        align-items: center;
    }

    /* 设置所有输入框长度相同
        凡是组件标签,都自带 class 属性,属性值就是标签名
    */
    .el-input{
        width:140px;
    }

    .btn{
        margin-left: -80px;
    }

</style>
```

views register index.vue(注册)

```
<template>
    <el-container class="reg-container">
        <el-form :model="regForm" :rules="regRules"
        ref="myRegForm" class="reg-form" status-icon size="mini"
        label-width="auto">
            <h1>用户注册</h1>
            <el-form-item label="用户姓名" prop="username">
                <el-input v-model="regForm.username"
                placeholder="请输入用户姓名"></el-input>
            </el-form-item>
            <el-form-item label="用户密码" prop="password">
                <el-input v-model="regForm.password"
                placeholder="请输入用户密码" show-password></el-input>
            </el-form-item>
            <el-form-item label="真实姓名" prop="realname">
                <el-input v-model="regForm.realname"
                placeholder="请输入真实姓名"></el-input>
            </el-form-item>
            <el-form-item label="邮箱地址" prop="email">
```

```
                <el-input v-model="regForm.email"
                placeholder="请输入邮箱地址"></el-input>
            </el-form-item>
            <el-form-item label="电话号码" prop="phone">
                <el-input v-model="regForm.phone"
                placeholder="请输入电话号码"></el-input>
            </el-form-item>
            <el-form-item label="性别" prop="gender">
                <!-- 此处使用了 ElementUI 的 Radio 单选框
                这里其实使用的是单选框组
                    el-radio-group:表示单选框组
                        v-model:就表示用户最终选的某一个的 label 值
                -->
                <el-radio-group v-model="regForm.gender">
                    <!-- :label:相当于之前的 value 属性 -->
                    <el-radio :label="0">男</el-radio>
                    <el-radio :label="1">女</el-radio>
                </el-radio-group>
            </el-form-item>
            <el-form-item label="权限" prop="role">
                <el-radio-group v-model="regForm.role">
                    <el-radio :label="0">用户</el-radio>
                    <el-radio :label="1">管理员</el-radio>
                </el-radio-group>
            </el-form-item>
            <el-form-item label="上传头像" prop="path" ref="myUp">
                <!-- 此处使用了 ElementUI 的 Upload 上传组件
                    注意 此组件上传操作没有使用我们的 axios 自定义实例,一切是全自动
的
                    甚至没有使用 axios,所以在获取 response 时没有 data 属性
                    action:上传的目的地,注意这里就是完整地址
                    list-type:上传的 UI 类型 picture-card|picture|text
                    存在以上三种,前两种可以显示上传成功的数据的缩略图,此处仅使用文字
的 text
                    :limit:最多上传的个数
                    :on-exceed:如果超过上传个数运行的钩子函数
                    :before-upload:表示上传之前最后一个执行的钩子
                    :on-success:表示上传成功执行的钩子
                    :on-remove:手动删除掉已经上传的文件
                    :file-list:封装已经上传的文件列表,本例未使用
                    :show-file-list="true":显示已经上传的文件列表
                -->
                <el-upload
action="http://db.etoak.com:9527/sysFile/fileUpload"
                    list-type="text"
                    :on-remove="handleRemove"
                    :limit="1"
                    :on-success="handleAvatarSuccess"
                    :on-exceed="handleExceed"
                    :file-list="fileList"
                    :show-file-list="true"
                    :before-upload="beforeAvatarUpload"
                    ref="up" class="up">
                        <i class="el-icon-plus"></i>
                </el-upload>
```

```
            </el-form-item>
            <el-form-item>
                <el-button type="primary" round size="small"
                class="btn" @click="submitForm('myRegForm')">注册</el-
button>
                <el-button round size="small"
                @click="resetForm('myRegForm')">取消</el-button>
            </el-form-item>
        </el-form>
    </el-container>
</template>

<script>
    import dao from '@/api/dao'
    export default {
        data(){
            return {
                /* 封装已经上传的批量文件,本例未使用 */
                fileList:[],
                /* 封装注册的表单 */
                regForm:{
                    username:'',
                    password:'',
                    realname:'',
                    email:'',
                    phone:'',
                    /* 默认是男 */
                    gender:0,
                    /* 默认是用户 */
                    role:0,
                    /* 设置上传成功后的文件在服务器端的路径 */
                    path:'',
                },
                /* 封装表单提交的验证规则 */
                regRules:{
                    username:[
                        { required:true,message:'请输入用户姓
名',trigger:'blur', },
                        { min:6,max:10,message:'姓名在6到10位之
间',trigger:'blur', },
                        { pattern:/^[a-zA-Z0-9_]*$/,message:'只能英文数字下
划线组成'
                        ,trigger:'blur' }
                    ],
                    password:[
                        { required:true,message:'请输入用户密
码',trigger:'blur', },
                        { min:6,max:32,message:'密码在6到32位之
间',trigger:'blur', },
                        { pattern:/^[a-zA-Z0-9_]*$/,message:'只能英文数字下
划线组成'
                        ,trigger:'blur' }
                    ],
                    realname:[
```

```
                        { required:true,message:'请输入真实姓名',trigger:'blur' },
                        { min:6,max:10,message:'真实姓名在6到10位之间',trigger:'blur'},
                        { pattern:/^[a-zA-Z0-9_]*$/,message:'只能英文数字下划线组成',trigger:'blur' }
                    ],
                    email:[
                        { required:true,message:'请输入邮箱地址',trigger:'blur' },
                        { pattern:/^[A-Za-z0-9\u4e00-\u9fa5]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+$/
                        ,message:'请输入合法邮箱地址',trigger:'blur' }
                    ],
                    phone:[
                        { required:true,message:'请输入电话号码',trigger:'blur' },
                        { pattern:/^(?:(?:\+|00)86)?1\d{10}$/
                        ,message:'请输入合法电话号码',trigger:'blur' }
                    ],
                    path:[
                        { required:true,message:'请上传头像'}
                    ],
                },
            }
        },
        methods:{
            /* 重置表单内容和验证提示 */
            resetForm(formName){
                /* 如果哪个表单项无效则与 prop 属性有关 */
                this.$refs[formName].resetFields()
            },
            submitForm(formName){
                /* 提交表单进行验证 */
                this.$refs[formName].validate(valid=>{
                    if(valid){
                        dao.reg(this.regForm).then(response=>{
                            this.$message({

 type:response.data.flag?'success':'error',
                                message:response.data.msg,
                                showClose:true,
                            })

                            if(response.data.flag){
                                setTimeout(() => {
                                    this.$router.push({
                                        path:'/',
                                    })
                                }, 1200);
                            }
                        })
                    }
                })
            },
```

```
            /* 传递参数超过个数 */
            handleExceed(file,fileList){
                this.$message.error('最多传递一张图片! ');
            },
            /*
                上传之前的钩子
                此处多用来进行上传时的格式查询等等
            */
            beforeAvatarUpload(file){
                const isJPG = file.type === 'image/jpeg';
                const isLt2M = file.size / 1024 / 1024 < 0.5;

                if (!isJPG) {
                    this.$message.error('上传头像图片只能是 JPG 格式!');
                }

                if (!isLt2M) {
                    this.$message.error('上传头像图片大小不能超过 500k!');
                }
                return isJPG && isLt2M;
            },
            /*
                文件上传成功时的钩子
                response:上传成功返回的响应
                file:每次上传的文件对象
                fileList:目前所有已经上传的文件数组
            */
            handleAvatarSuccess(response, file, fileList){
                /* 一般在上传成功后在此处进行业务逻辑，例如将返回的路径添加进表单双向绑定的data中的对象 */
                /*
                    {
                        "code":null,
                        "flag":true,
                        "msg":"上传成功~",
                        "data":{
                            "fileId":null,
                            "fileName":"1",
                            "fileSize":48558,
                            "path":"http://etoak.vip:9528/4028823a74559fc80174559fc81e0000.jpg"
                        }
                    }
                */
                if(response.flag){
                    /* 将返回的路径添加进封装路径的表单项属性 */
                    //this.ruleForm.paths.push(response.data.path)
                    this.regForm.path = response.data.path
                    /* 清空上传组件显示的报错信息,因为已经上传成功了 */
                    this.$refs['myUp'].clearValidate()
                }
            },
            /*
                文件移除的钩子
                file:点击的要删除的已经上传成功的文件对象
```

```
                fileList:目前所有已经上传的文件数组
            */
            handleRemove(file, fileList) {
                /* 必须首先打印此处观察file是什么，fileList是什么 */
                /* 此处进行上传文件被删除之后的逻辑 */

 //this.ruleForm.paths.splice(this.ruleForm.paths.indexOf(file.response.data.path),1)
                this.regForm.path = ''
            },
        },
    }
</script>

<style scoped>
    /* 设置最外侧大容器 */
    .reg-container {
        /* 设置容器绝对定位 */
        position: absolute;
        /* 设置容器宽度 */
        width: 100vw;
        /* 设置容器高度 */
        height: 100vh;
        /* 设置容器背景图片 no-repeat:表示如果图片大小不合适
        则不会重叠摆放 */
        background: url("../../assets/bg5.jpeg") no-repeat;
        /* 设置背景图尺寸 宽度 高度 */
        background-size: 100vw 100vh;
        /* 设置背景图不会随着缩放变动 */
        background-attachment: fixed;
        /* 容器开启弹性盒子 */
        display: flex;
        /* 设置项目,也就是表单 主轴 交叉轴居中 此时默认 水平是主轴
        纵向是交叉轴 */
        justify-content: center;
        align-items: center;
    }

    /* 设置表单 */
    .reg-form {
        /* 设置表单宽度 */
        width: 350px;
        /* 设置表单背景色 和 透明度
        rgba(红色,绿色,蓝色,透明度)
        红绿蓝为 0-255 的整数 如果都是 255 则是白色
        透明度 0 完全透明 1 完全不透明 */
        background-color: rgba(255, 255, 255, 0.6);
        /* 设置边框为圆角 数字越大越圆,如果设置为 50%则变为圆球
        表格不能设置 */
        border-radius: 30px;
        /* 给表单开启弹性盒子 */
        display: flex;
        /* 设置表单内的项目从上往下排列,默认是 row 从左往右 */
        flex-direction: column;
        /* 设置交叉轴水平居中摆放,注意这里从左往右是交叉轴了,从上往下
```

```
        才是主轴 因为排列顺序是从上往下 */
        align-items: center;
    }

    /* 设置输入框和上传组件同宽,否则会被压缩 */
    .el-input,.up{
        width:140px;
    }

    .btn{
        margin-left: -80px;
    }


</style>
```

day15(表格查询 分页查询 条件查询)

api dao.js

```
/* 导入自定义 axios 实例 */
import request from '@/util/request'

/* API  application program interface 此处表示应用程序接口 */
export default {
    /* 1:登录
        username:形参 用户姓名
        password:形参 用户密码
    */
    login(username,password){
        /* 返回一个 */
        return request({
            /* url:此处表示设置进阶地址
            注意最终发送的完整地址是 基本地址+进阶地址 */
            url:`/testUser/login?
username=${username}&password=${password}`,
            method:'get',
        })
    },
    /* 2:注册
        pojo:形参,对象,内部封装了要注册的八个字段
    */
    reg(pojo){
        return request({
            url:'/testUser/add',
            method:'post',
            /* 发送 json 注意这里直接发送 js 对象或者数组即可,不需要
            自己转换 */
            data: pojo,
        })
    },
```

```
    /* 3:分页查询
        page:形参,当前页
        itemsPerPage:形参,每页记录数
        searchMap:形参 对象 内部封装了条件的查询的字段
    */
    query(page,itemsPerPage,searchMap){
        return request({
            url:`/testUser/select?
page=${page}&itemsPerPage=${itemsPerPage}`,
            method:'get',
            /* 由于 get 不能发送 json,这里提供了 params
            来封装对象,根据这个对象中是否存在字段,进行有选择的拼接 */
            params:searchMap,
        })
    },
}
```

views table table.vue

```
<template>
    <el-container class="table" direction="vertical">
        <!-- 此处使用了 ElementUI 的 Form 行内表单
        :inline="true" 开启行内表单 -->
        <el-form :model="searchForm" :inline="true" ref="mySearchForm"
        size="mini" class="top">
            <el-form-item prop="username">
                <el-input v-model="searchForm.username"
                placeholder="请输入用户姓名"></el-input>
            </el-form-item>
            <el-form-item prop="realname">
                <el-input v-model="searchForm.realname"
                placeholder="请输入真实姓名"></el-input>
            </el-form-item>
            <el-form-item prop="gender" label="性别">
                <!-- @input:绑定值变化时触发的事件
                此处只要点击单选框立刻开始查询 -->
                <el-radio-group v-model="searchForm.gender"
                @input="fetchData">
                    <el-radio :label="0">男</el-radio>
                    <el-radio :label="1">女</el-radio>
                </el-radio-group>
            </el-form-item>
            <el-form-item prop="email">
                <el-input v-model="searchForm.email"
                placeholder="请输入邮箱地址"></el-input>
            </el-form-item>
            <el-form-item>
                <el-button type="primary" round size="mini"
                @click="fetchData">
                    查询
                </el-button>
                <el-button round size="mini"
                @click="goReset('mySearchForm')">
                    重置
```

```
                </el-button>
            </el-form-item>
        </el-form>
        <!-- 此处使用了 ElementUI 的 Table 表格组件
            el-table:表示表格
                :data:绑定要迭代的数据
                size:medium small mini
                border:显示边框
                stripe:显示表格斑马纹
                highlight-current-row:鼠标高亮当前行
        -->
        <el-table :data="list" size="small" border stripe
        highlight-current-row class="middle">
            <!--
                el-table-column:表示列
                type="index" 表示本列显示索引,但是注意 从 1 开始
                label:列名
                width:列宽
                align:内部数据对齐方式
            -->
            <el-table-column type="index" label="序号"
            width="50px" align="center"></el-table-column>
            <!-- prop:对应字段名 -->
            <el-table-column label="用户姓名" prop="username"
            align="center"></el-table-column>
            <el-table-column label="真实姓名" prop="realname"
            align="center"></el-table-column>
            <el-table-column label="邮箱地址" prop="email"
            align="center"></el-table-column>
            <el-table-column label="电话号码" prop="phone"
            align="center"></el-table-column>
            <el-table-column label="用户性别" prop="gender"
            align="center">
                <!-- 如果数据就直接展示,则禁用 el-table-column 即可
                如果要对数据进行下一步的加工则需要使用官方提供的插槽 -->
                <template slot-scope="scope">
                    <!--
                        通过 scope.row 可以获取这一行的对象
                        通过 scope.row.属性名 可以获取这一行对象的属性
                    -->
                    {{ scope.row.gender===0?'男':'女' }}
                </template>
            </el-table-column>
            <el-table-column label="用户权限" prop="role"
            align="center">
                <template slot-scope="scope">
                    {{ scope.row.role===0?'用户':'管理员' }}
                </template>
            </el-table-column>
            <el-table-column label="操作"
            align="center" width="220px">
                <el-button type="primary" round
                size="mini">编辑</el-button>

                <el-button type="primary" round
```

```
                size="mini" plain>查看</el-button>

                <el-button type="danger" round
                size="mini">删除</el-button>
            </el-table-column>
        </el-table>
        <!-- 此处使用了 ElementUI 的 Pagination 分页组件
                :current-page:绑定当前页的值
                :page-size:绑定每页记录数
                :total:绑定总记录数
                :page-sizes:每页记录数更改的几种选项
                layout:是 Pagination 组件的六个子组件,哪个功能不需要
                则可以不写,共有 6 个功能
                background:设置背景色,默认蓝色
                @size-change="handleSizeChange":当每页记录数
                更改时执行
                @current-change="handleCurrentChange":当当前页更改
                时执行
        -->
        <el-pagination
            @size-change="handleSizeChange"
            @current-change="handleCurrentChange"
            :current-page="page"
            :page-sizes="[10, 15, 30]"
            :page-size="itemsPerPage"
            layout="total, sizes, prev, pager, next, jumper"
            :total="total" background class="bottom">
        </el-pagination>
    </el-container>
</template>

<script>
    import dao from '@/api/dao'
    export default {
        data(){
            return {
                /* 当前页 */
                page:1,
                /* 每页记录数 */
                itemsPerPage:10,
                /* 总记录数 */
                total:0,
                /* 后端返回的分页的数据 */
                list:[],
                /* 封装条件查询的字段 */
                searchForm:{
                    username:'',
                    realname:'',
                    /* 注意性别没有默认值 因为条件查询可以忽略性别 */
                    gender:'',
                    email:'',
                },
            }
        },
        methods:{
```

```
            /* 分页查询 */
            fetchData(){
                /*
                    this.page:获取当前页
                    this.itemsPerPage:获取每页记录数
                    this.searchForm:获取封装的条件查询的字段
                    以上为三个实参
                */
                dao.query(this.page,this.itemsPerPage
                ,this.searchForm)
                .then(response=>{
                    console.log(response)
                    if(response.data.flag){
                        /* 获取分页数据 */
                        this.list = response.data.data
                        /* 获取总记录数 */
                        this.total = response.data.total
                    }
                })
            },
            /* 当每页记录数更改时执行 val:形参,就表示
            传入的新的每页记录数 */
            handleSizeChange(val) {
                /* 更新每页记录数 */
                this.itemsPerPage = val
                /* 回显重新查询 */
                this.fetchData()
            },
            /* 当当前页更改时执行 val:形参,就表示
            传入的新的当前页的值 */
            handleCurrentChange(val) {
                /* 更新当前页的值 */
                this.page = val
                /* 回显重新查询 */
                this.fetchData()
            },
            /* 条件查询重置功能 */
            goReset(formName){
                /* 重置表单样式和填写的内容 */
                this.$refs[formName].resetFields()
                /* 回显 */
                this.fetchData()
            },
        },
        created(){
            this.fetchData()
        },
    }
</script>

<style scoped>
    .table{
        width:100%;
        height:100%;
        background-color: whitesmoke;
```

```
        /* 如果尺寸溢出,则隐藏不显示滚动条 */
        overflow: hidden;
    }

    /* 设置表格的样式 */
    .middle{
        margin:5px;
        flex:1;
    }
    /* 设置分页的样式 */
    .bottom{
        text-align: center;
        height:5vh;
    }

    /* 设置条件查询表单的样式 */
    .top{
        height:3vh;
        margin:5px;
    }
</style>
```

(圣杯布局)layout index.vue

```
<template>
    <!-- direction:使用在 el-container 中,默认 horizontal 水平排列
    改为 vertical 则为纵向排列,其实就是 flex-direction:row|column -->
    <el-container class="layout-container" direction="vertical">
        <!-- <el-header>页眉</el-header>
        <el-container>
            <el-aside>侧边栏</el-aside>
            <el-main>主页</el-main>
        </el-container>
        <el-footer>页脚</el-footer> -->
        <!-- 引入页眉组件 -->
        <app-header></app-header>
        <!-- 引入主体组件 -->
        <app-section></app-section>
        <!-- 引入页脚组件 -->
        <app-footer></app-footer>
    </el-container>
</template>

<script>
    /* 引入页眉组件 */
    import AppHeader from './components/appheader'
    /* 引入主体组件 */
    import AppSection from './components/appsection'
    /* 引入页脚组件 */
    import AppFooter from './components/appfooter'
    export default {
        components:{
```

```
            AppHeader,
            AppSection,
            AppFooter,
        },
    }
</script>

<style scoped>
    .layout-container{
        height:100vh;
    }
</style>
```

components appsection index.vue

```
<template>
    <el-container class="section">
        <!-- 引入侧边栏 -->
        <app-aside></app-aside>
        <!-- 引入主体 -->
        <app-main></app-main>
    </el-container>
</template>

<script>
    /* 导入侧边栏 */
    import AppAside from '../appaside'
    /* 导入主页 */
    import AppMain from '../appmain'
    export default {
        components:{
            AppAside,
            AppMain,
        }
    }
</script>

<style scoped>
    .section{
        flex: 1;
    }
</style>
```

components appaside index.vue

```
<template>
    <!-- width:设置侧边栏宽度 -->
    <el-aside class="aside" width="200px">
        <!--
            此处使用了 ElementUI 的 NavMenu 导航菜单
                :router:如果设置为 true,则表示开启路由功能,取代 router-link
                ,可以激活哈希默认 false 无法激活哈希
                default-active:当前哪个链接处于激活状态,根据浏览器地址栏
```

```
                当前的哈希来判断 $route.path:就是当前地址栏哈希
                background-color:菜单栏背景色
                text-color:菜单栏文字颜色
                active-text-color:激活的链接的文字颜色
                unique-opened:可折叠菜单栏只能同时展开一个,不能同时展开多个
        -->
        <el-menu :router="true" :default-active="$route.path?$route.path:''"
            background-color="#454c63"
            text-color="whitesmoke"
            active-text-color="coral" unique-opened>
            <!-- 可折叠菜单栏 -->
            <el-submenu index="1">
                <!-- 菜单栏标题 -->
                <template slot="title">
                    <i class="el-icon-location"></i>
                    <span>增删查改</span>
                </template>
                <!-- 菜单列表项 index:类似 router-link 的 to 属性用来激活哈希 -->
                <el-menu-item index="/layout/table"><i class="el-icon-notebook-1"></i>表格测试</el-menu-item>
                <el-menu-item index="/layout/grid"><i class="el-icon-s-grid"></i>栅格测试</el-menu-item>
                <el-menu-item index="/layout/cascade"><i class="el-icon-grape"></i>级联测试</el-menu-item>
            </el-submenu>
            <el-submenu index="2">
                <template slot="title">
                    <i class="el-icon-s-marketing"></i>
                    <span>数据可视化</span>
                </template>
                <el-menu-item index="/layout/echarts"><i class="el-icon-s-data"></i>Echarts测试</el-menu-item>
                <el-menu-item index="/layout/dashboard"><i class="el-icon-s-operation"></i>面板测试</el-menu-item>
            </el-submenu>
            <el-menu-item index="3">
                <i class="el-icon-menu"></i>
                <span slot="title">导航二</span>
            </el-menu-item>
            <el-menu-item index="4">
                <i class="el-icon-document"></i>
                <span slot="title">导航三</span>
            </el-menu-item>
        </el-menu>
    </el-aside>
</template>

<script>
    export default {

    }
</script>

```

```
<style scoped>
    /* 设置侧边栏 */
    .aside{
        /* 设置背景色 */
        background-color: #454c63;
        /* 开启弹性盒子 */
        display: flex;
        /* 主轴交叉轴居中 */
        justify-content: center;
        align-items: center;
    }
    .el-menu {
        text-align: center;
        font-weight:900;
        /* 去掉侧边栏右侧的白边 */
        border-right:none ;
        width:100%;
    }
</style>
```

$router 与 $route的不同

$router

`$router`是一个全局路由对象，是`new Router`的实例，`$router`对象正是`new VueRouter`所创建`router`对象，`this.$router`就等同于`new Router`的实例

```
//常用函数
this.$router.push('/user') //跳转路由
this.$router.replace('/user') //跳转路由,但是不会有记录,不入栈
```

$route

`$route`是一个局部对象，表示当前正在跳转的路由对象，换句话说，当前哪个路由处于活跃状态，`$route`就对应那个路由。`this.$route`代表当前活跃的路由对象

```
$route.path
//字符串，等于当前路由对象的路径，如“/home/news”
$route.params
//对象，包含路由中的动态片段和全匹配片段的键值对
$route.query
//对象，包含路由中query参数的键值对。如“......?name=胡桃&age=16”会得到{“name”：”胡桃”，“age”：16}
$route.name
//当前路径的名字，如果没有使用具名路径，则名字为空
$route.router
//路由规则所述的路由器（以及所属的组件）
$route.matched
//数组，包含当前匹配的路径中所包含的所有片段所对应的配置参数对象
```

总结

`$router`是`new Router`的实例，是全局路由对象，用于进行路由跳转等操作

`$route`是路由信息对象，表示当前活跃的路由对象，用于读取路由参数；

简单来说也就是，操作找`$router`，读参找`$route`

备注

```
vue路由报错：
Uncaught (in promise) Error: Avoided redundant navigation to current location
/*原因：
多次点击跳转同一个路由是不被允许的
解决办法：
在引入vue-router的js文件里加上如下代码即可：*/
//push
const VueRouterPush = VueRouter.prototype.push
VueRouter.prototype.push = function push(to) {
return VueRouterPush.call(this, to).catch(err => err)}
//replace
const VueRouterReplace = VueRouter.prototype.replace
VueRouter.prototype.replace = function replace(to) {
return VueRouterReplace.call(this, to).catch(err => err)}
```

什么是分页

> 后端将数据一部分一部分的给前端的操作称之为分页,分页是查询功能中最重要的一种方式

- 假分页
  - 一次将所有数据取出,用户需要几条,则显示几条,不宜针对大量的数据
- 真分页
  - 用户需要几条数据就从后端获取多少数据,基本上目前都是真分页

分页四要素

> 不管使用何种技术,只要获取了以下四个参数,则分页可解

- 总记录数
  - 后端获取　　select count(id) from 表
- 每页记录数:
  - 前端获取　　自己决定每页显示多少条记录
- 总页数
  - 后端获取　　Java:(总记录数+每页记录数-1)/每页记录数
  - 前端获取　　Js:Math.ceil(总记录数/每页记录数)
- 当前页
  - 前端获取:此值不是一个定值,会随着用户的操作的变化而变化,每次变化之后,都要重新获取,默认值是 1

分页四要素与分页公式

```
    select 字段 from 表 limit x,y;
    x:起始索引(包含)
    y:显示记录数
    select 字段 from 表 limit (当前页-1)*每页记录数,每页记录数;
```

前端分页思想

> 将最新的当前页与每页记录数发送给后端,后端计算出分页数据以及总记录数 返回给前端

---

day16

复习

1 请简述 Vue2 中函数如何激发(2种)?计算属性如何激发(1种)?侦听器如何激发(2种)?

```
函数
    直接调用
    事件激发
计算属性
    依赖的属性值发生更改
侦听器
    immediate:true 打开页面,就会激发一次
    侦听的属性发生改变
```

2 请简述 Vue2 中函数,计算属性,侦听器的不同

```
函数
    函数一般绑定事件,也可以直接调用,支持异步,选择性书写return语句,没有双向绑定功能
    函数没有缓存,调用几次,就执行几次

计算属性
    计算属性,就是一个属性,它依赖的初始化的值发生更改,就会重新执行,就算属性没有小括号,就是个属性,不会传值
    计算属性自带缓存功能,无论强制调用多少次,都只执行一次,除非依赖的初始化的值发生更改
    可以实现双向绑定,通过书写 get和set
    get()中一定有return 所以计算属性不可以发送异步

侦听器
    一般不考虑双向绑定问题,就是设置一个值,侦听器去侦听这个数据,如果设置 immediate 代表立即执行侦听器一次
    侦听器默认只可以侦听基本数据类型,侦听复杂数据类型需要开启深度侦听 deep:true;

综上,计算属性能做到的,侦听器都能做到,但是侦听器能做到的,计算属性不一定能做到,例如 异步

```

3 在vue中 如何获取 DOM ,从而操作DOM

```
this.$refs[formName]
this.$refs.formName
```

4 请写出VueComponents 传值方式,写出关键脚本,描述作用

```
String,Number,Boolean,Array,Object,Function
props 父子传值
    父组件
        <子组件 :自定义属性="属性值"></子组件>
    子组件
        props:['自定义属性1','自定义属性2']
        props:{
            自定义属性1:数据类型,
            自定义属性2:数据类型,
        }
        props:{
            自定义属性:{
                type:数据类型,
                required:true,
                default:默认值
            }
        }

$emit 子父传值(不饿能隔代)
    父组件
        <子组件 @自定义事件1="操作1"></子组件>
    子组件
        <tagName @原生事件="操作2(值)"></tagName>
        操作2(值){
            this.$emit('自定义事件1',值)
        }

slot 父子传模板/数据
    子组件
        <slot name="插槽名"></slot>
    父组件
        <tagName slot="插槽名"></tagName>

        <template v-slot:插槽名>
            <tagName></tagName>
        </template>
```

5 Vue2挑几件渲染几种方式,不同使用场合

```
v-if
    如果为真,元素显示,如果为假,元素不显示,底层不渲染
    切换小号打,适用于切换不频繁的场合
v-show
    如果为真,元素显示,如果为假,元素不显示,添加了display:none;的行内式,底层依然渲染
    初识载入消耗较大,但是之后切换消耗小,适用于切换频繁的场合

v-else-if
v-else
    与v-if配合使用,必须紧邻,用来进行简单流程控制
    不可以与v-show连用
```

6 ES6 中promise对象解决了什么问题,简述它对于某种操作做了什么

```
解决了回调地狱问题

如果发送异步成功,回调函数调用resolve(response),执行 then(response=>{})
如果失败,回调函数调用reject(error),执行catch(error=>{})
如果有finally ,无论成功还是失败,都会执行
```

7 过滤器使用需要注意什么?

```
不可以使用 this,this在过滤器中是undefined
不可以与v-model连用
使用过滤器一定有数据字典
const payOptions = [
    { id: 1, option: '现金支付', },
    { id: 2, option: '支付宝支付', },
    { id: 3, option: '微信支付', },
    { id: 4, option: '银行卡支付', }
]
```

8 Vue2 样式渲染几种方式

```
绑定class
    :class="myClass",myClass初始化的值的内容是类名
    :class="{类名1:val,类名2:val}"
    如果类名后面的初始化的值为true,类名有效,否则无效
    :class="['类名1','类名2']"
    一个元素有多个类名,没有初始化的数据

绑定style
    :style="myStyle"
    初始化的值myStyle就是行内式样式
    :style="{样式名:val}"
    样式名必须小驼峰,val就是样式值,样式名没有引号
```

9 Vue2 如何实现数组/对象可响应式功能

```
对象
    this.$set(对象,'属性名','属性值')
    this.$delete(对象,'属性名')
    Vue.set(对象,'属性名','属性值')
    VUe.delet(对象,'属性名','属性值')
数组
    利用下面7个进阶方法,
    push() pop()
    unshift() shift()
    reverse() sort()
    splice()
    Vue2中必须使用一下七个函数对数组进行修改,是js的加强版,实现了可响应式的功能,与js原始7个只是重名
```

10 Vue2指令元素

```
v-html
v-text
v-once
v-if
v-show
v-model
v-slot
v-else-if
v-else
v-on
v-bind
v-cloak
v-pre
v-for
```

1 什么是单向绑定?什么是双向绑定

```
单向绑定
    修改Vue实例中的数据,页面模板中的数据也会发生更改,两者绑定在一起,称之单向绑定,也称数据实现了可响应式,插值语法和指令语法都自带单向绑定功能

双向绑定
    修改页面模板中的数据,Vue实例data中的数据也会发生更改,惩治双向绑定,默认情况下只有v-model支持双向绑定,
其他情况,可以通过使用计算属性,添加双向绑定
```

2 如何使用事件原型获取元素节点

```
event.target
```

3 使用过的事件修饰符

```
.stop 解决冒泡
.once 一次绑定
.prevent 屏蔽元素固有动作
@keyup.键位 监听键位
@keydown.tab 监听tab
.native 给组件添加原生事件
```

4 如何进行列表渲染

```
v-for="(data,index) in datas" 数组
v-for="(value,name,index) in obj" 对象
:key="主键",没有主键,可以绑定 index
```

5 Vue2实例对用户书写的数据做了哪些操作

```
1. 数据劫持
    Vue获取用户书写的data之后,无论封装基层都会对这个对象添加可响应式的功能嫩,对象的每个属性都被添加了
    reactiveGetter() 和 reactiveSetter(val)函数
    如果数据被读取,执行 reactiveGetter()
    如果数据被修改,执行 reactiveSetter(),同时会修改页面模板中的数据,这就是可响应式/单向绑定的实现
    这些被施加了可响应式功能的数据和函数get()和set() 被封装在 Vue实例的_data对象,这个操作,被称作,数据劫持
2. 数据监听
    一个对象可以对另一个对象的属性进行操作,称之为数据代理,Vue实例通过数据代理将_dat中的数据带力道 Vue实例的表层,在模板中使用插值语法或指令语法,不用再添加_data前缀,Vue实例表层中的数据可以直接在模板中使用
```

6 如何给组件添加原生事件

```
<组件 @原生事件.native="函数"/>
```

7 组件通讯原则

```
1 不要再子组件中修改父组件传递的数据
2 数据初始化,要根据初始化的数据是否用于多个组件之中,如果需要用在多个是组件之中,就初始化再付组件之中,如果在一个组件中使用,就初始化在这个组件之中
3 初始化在那个组件之中,更新数据的函数就写在那个组件
```

8 生命周期

```
created()
    最早获取数据,发送异步
mounted()
    页面已经被真实 DOM 覆盖完毕,处于稳定状态,此时可以操作DOM,也可以发送异步

beforeUpdate()
    唯一模板与数据不统一的时机
```

day16（编辑 查看 删除）

api dao.js

```
/* 导入自定义 axios 实例 */
import request from '@/util/request'

/* API  application program interface 此处表示应用程序接口 */
export default {
    /* 1:登录
        username:形参 用户姓名
        password:形参 用户密码
    */
    login(username,password){
        /* 返回一个 */
        return request({
            /* url:此处表示设置进阶地址
            注意最终发送的完整地址是 基本地址+进阶地址 */
            url:`/testUser/login?username=${username}&password=${password}`,
            method:'get',
        })
    },
    /* 2:注册
        pojo:形参,对象,内部封装了要注册的八个字段
    */
    reg(pojo){
        return request({
            url:'/testUser/add',
            method:'post',
            /* 发送 json 注意这里直接发送 js 对象或者数组即可,不需要
            自己转换 */
            data: pojo,
        })
    },
    /* 3:分页查询
        page:形参,当前页
        itemsPerPage:形参,每页记录数
        searchMap:形参 对象 内部封装了条件的查询的字段
    */
    query(page,itemsPerPage,searchMap){
        return request({
            url:`/testUser/select?page=${page}&itemsPerPage=${itemsPerPage}`,
            method:'get',
            /* 由于 get 不能发送 json,这里提供了 params
            来封装对象,根据这个对象中是否存在字段,进行有选择的拼接 */
            params:searchMap,
        })
    },
    /* 4:根据 id 查询
        id:形参表示要查询的数据 id
    */
```

```
    queryById(id){
        return request({
            /* 如果只有一个参数传递,则可以使用这种简略写法 */
            url:`/testUser/selectById/${id}`,
            method:'get',
        })
    },
    /* 5:修改
        newPojo:对象 形参,封装了要被修改的字段以及被修改的数据的 id
    */
    update(newPojo){
        return request({
            url:'/testUser/update',
            method:'put',
            data:newPojo,
        })
    },
    /* 6:删除 */
    delById(id){
        return request({
            url:`/testUser/delete/${id}`,
            method:'delete',
        })
    }
}
```

table index.vue

```
<template>
    <el-container class="table" direction="vertical">
        <!-- 此处使用了 ElementUI 的 Form 行内表单
        :inline="true" 开启行内表单 -->
        <el-form :model="searchForm" :inline="true" ref="mySearchForm"
        size="mini" class="top">
            <el-form-item prop="username">
                <el-input v-model="searchForm.username"
                placeholder="请输入用户姓名"></el-input>
            </el-form-item>
            <el-form-item prop="realname">
                <el-input v-model="searchForm.realname"
                placeholder="请输入真实姓名"></el-input>
            </el-form-item>
            <el-form-item prop="gender" label="性别">
                <!-- @input:绑定值变化时触发的事件
                此处只要点击单选框立刻开始查询 -->
                <el-radio-group v-model="searchForm.gender"
                @input="fetchData">
                    <el-radio :label="0">男</el-radio>
                    <el-radio :label="1">女</el-radio>
                </el-radio-group>
            </el-form-item>
            <el-form-item prop="email">
                <el-input v-model="searchForm.email"
                placeholder="请输入邮箱地址"></el-input>
            </el-form-item>
```

```
            <el-form-item>
                <el-button type="primary" round size="mini"
                @click="fetchData">
                    查询
                </el-button>
                <el-button round size="mini"
                @click="goReset('mySearchForm')">
                    重置
                </el-button>
            </el-form-item>
        </el-form>
        <!-- 此处使用了 ElementUI 的 Table 表格组件
            el-table:表示表格
                :data:绑定要迭代的数据
                size:medium small mini
                border:显示边框
                stripe:显示表格斑马纹
                highlight-current-row:鼠标高亮当前行
                :row-class-name="tableRowClassName"
                可以根据字段的值来为表格添加不同的 class,从而显示不同的样式
                注意此属性与 stripe 斑马纹冲突
        -->
        <el-table :data="list" size="small" border
        highlight-current-row class="middle"
        :row-class-name="tableRowClassName">
            <!--
                el-table-column:表示列
                type="index" 表示本列显示索引,但是注意 从 1 开始
                label:列名
                width:列宽
                align:内部数据对齐方式
            -->
            <el-table-column type="index" label="序号"
            width="50px" align="center"></el-table-column>
            <!-- prop:对应字段名 -->
            <el-table-column label="用户姓名" prop="username"
            align="center"></el-table-column>
            <el-table-column label="真实姓名" prop="realname"
            align="center"></el-table-column>
            <el-table-column label="邮箱地址" prop="email"
            align="center"></el-table-column>
            <el-table-column label="电话号码" prop="phone"
            align="center"></el-table-column>
            <el-table-column label="用户性别" prop="gender"
            align="center">
                <!-- 如果数据就直接展示,则禁用 el-table-column 即可
                如果要对数据进行下一步的加工则需要使用官方提供的插槽 -->
                <template slot-scope="scope">
                    <!--
                        通过 scope.row 可以获取这一行的对象
                        通过 scope.row.属性名 可以获取这一行对象的属性
                    -->
                    {{ scope.row.gender===0?'男':'女' }}
                </template>
            </el-table-column>
```

```
            <el-table-column label="用户权限" prop="role"
            align="center" width="220px">
                <template slot-scope="scope">
                    <!-- 此处使用了 ElementUI 的 Switch 开关
                        v-model:双向绑定权限的值
                        active-color:激活之后的开关颜色
                        inactive-color:关闭的颜色
                        active-text:开关拨开之后的文本
                        inactive-text:开关关闭之后的文本
                        :active-value:双向绑定开关拨开的值
                        :inactive-value:双向绑定开关关闭的值
                        @change="toggleRole(scope.row.id,scope.row.role)"
                        切换开关时执行函数,第一个实参表示修改哪个,第二个参数表示
                        权限修改成什么值
                    -->
                    <el-switch
                        v-model="scope.row.role"
                        active-color="coral"
                        inactive-color="silver"
                        active-text="管理员"
                        inactive-text="用户"
                        :active-value="1"
                        :inactive-value="0"

 @change="toggleRole(scope.row.id,scope.row.role)">
                    </el-switch>
                </template>
            </el-table-column>
            <el-table-column label="操作"
            align="center" width="220px">
                <template slot-scope="scope">
                    <!-- 编辑分为两个步骤
                            1:预备编辑 @click="handleUpadte(scope.row.id)"
                            2:编辑
                    -->
                    <el-button type="primary" round
                    size="mini"
                    @click="handleUpdate(scope.row.id)">编辑</el-button>

                    <el-button type="primary" round
                    size="mini" plain
                    @click="$router.push({path:`/layout/info?
id=${scope.row.id}`})">查看</el-button>

                    <el-button type="danger" round
                    size="mini" @click="remove(scope.row.id)">删除</el-
button>
                </template>
            </el-table-column>
        </el-table>
        <!-- 此处使用了 ElementUI 的 Pagination 分页组件
                :current-page:绑定当前页的值
                :page-size:绑定每页记录数
                :total:绑定总记录数
                :page-sizes:每页记录数更改的几种选项
```

```
                layout:是 Pagination 组件的六个子组件,哪个功能不需要
                则可以不写,共有 6 个功能
                background:设置背景色,默认蓝色
                @size-change="handleSizeChange":当每页记录数
                更改时执行
                @current-change="handleCurrentChange":当当前页更改
                时执行
        -->
        <el-pagination
            @size-change="handleSizeChange"
            @current-change="handleCurrentChange"
            :current-page="page"
            :page-sizes="[10, 15, 30]"
            :page-size="itemsPerPage"
            layout="total, sizes, prev, pager, next, jumper"
            :total="total" background class="bottom">
        </el-pagination>
        <!-- 此处使用了 ElementUI 的 Dialog 对话框组件
            el-dialog:表示对话框
                title:对话框标题
                :visible.sync:后面对应的值如果是 true,则对话框显示
                false 不显示
                width:设置对话框宽度
        -->
        <el-dialog title="修改信息" :visible.sync="dialogFormVisible"
        width="300px">
            <el-form :model="updateForm" ref="myUpdateForm"
            status-icon :rules="updateRules" size="mini"
            label-width="auto" class="update-form">
                <el-form-item label="真实姓名" prop="realname">
                    <el-input v-model="updateForm.realname"></el-input>
                </el-form-item>
                <el-form-item label="电话号码" prop="phone">
                    <el-input v-model="updateForm.phone"></el-input>
                </el-form-item>
                <el-form-item label="邮箱地址" prop="email">
                    <el-input v-model="updateForm.email"></el-input>
                </el-form-item>
                <el-form-item label="用户性别" prop="gender">
                    <el-radio-group v-model="updateForm.gender">
                        <el-radio :label="0">男</el-radio>
                        <el-radio :label="1">女</el-radio>
                    </el-radio-group>
                </el-form-item>
            </el-form>

            <div slot="footer" class="dialog-footer">
                <el-button @click="dialogFormVisible = false"
                round size="mini">取 消</el-button>
                <el-button type="primary" @click="finishUpdate"
                round size="mini" >确 定</el-button>
            </div>
        </el-dialog>
    </el-container>
</template>
```

```
<script>
    import dao from '@/api/dao'
    export default {
        data(){
            return {
                /* 控制编辑对话框显示与否 */
                dialogFormVisible:false,
                /* 当前页 */
                page:1,
                /* 每页记录数 */
                itemsPerPage:10,
                /* 总记录数 */
                total:0,
                /* 后端返回的分页的数据 */
                list:[],
                /* 封装条件查询的字段 */
                searchForm:{
                    username:'',
                    realname:'',
                    /* 注意性别没有默认值 因为条件查询可以忽略性别 */
                    gender:'',
                    email:'',
                },
                /* 封装被修改的字段 */
                updateForm:{
                    realname:'',
                    phone:'',
                    email:'',
                    gender:'',
                },
                /* 封装修改的验证规则 */
                updateRules:{
                    realname:[
                        { required:true,message:'请输入真实姓名',trigger:'blur' },
                        { min:6,max:10,message:'真实姓名在6到10位之间',trigger:'blur'},
                        { pattern:/^[a-zA-Z0-9_]*$/,message:'只能英文数字下划线组成',trigger:'blur' }
                    ],
                    email:[
                        { required:true,message:'请输入邮箱地址',trigger:'blur' },
                        { pattern:/^[A-Za-z0-9\u4e00-\u9fa5]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+$/
                        ,message:'请输入合法邮箱地址',trigger:'blur' }
                    ],
                    phone:[
                        { required:true,message:'请输入电话号码',trigger:'blur' },
                        { pattern:/^(?:(?:\+|00)86)?1\d{10}$/
                        ,message:'请输入合法电话号码',trigger:'blur' }
                    ],
                },
```

```
                /* 封装修改权限的字段 */
                updateRole:{
                    id:'',
                    role:'',
                },
            }
        },
        methods:{
            /* 分页查询 */
            fetchData(){
                /*
                    this.page:获取当前页
                    this.itemsPerPage:获取每页记录数
                    this.searchForm:获取封装的条件查询的字段
                    以上为三个实参
                */
                dao.query(this.page,this.itemsPerPage
                ,this.searchForm)
                .then(response=>{
                    console.log(response)
                    if(response.data.flag){
                        /* 获取分页数据 */
                        this.list = response.data.data
                        /* 获取总记录数 */
                        this.total = response.data.total
                    }
                })
            },
            /* 当每页记录数更改时执行 val:形参,就表示
            传入的新的每页记录数 */
            handleSizeChange(val) {
                /* 更新每页记录数 */
                this.itemsPerPage = val
                /* 回显重新查询 */
                this.fetchData()
            },
            /* 当当前页更改时执行 val:形参,就表示
            传入的新的当前页的值 */
            handleCurrentChange(val) {
                /* 更新当前页的值 */
                this.page = val
                /* 回显重新查询 */
                this.fetchData()
            },
            /* 条件查询重置功能 */
            goReset(formName){
                /* 重置表单样式和填写的内容 */
                this.$refs[formName].resetFields()
                /* 回显 */
                this.fetchData()
            },
            /* 编辑 步骤 1 预备编辑 */
            handleUpdate(id){
                /* 1:显示对话框 */
                this.dialogFormVisible = true
```

```
                /* 2:对表单内的内容和样式进行重置
                    在 Vue 中如果涉及到操作 DOM 的语句,则执行顺序
                    与我们理解的是有偏差的,因为其实底层是使用的异步

                    1:xxxx
                    2:xxxx
                    3:操作 DOM
                    4:xxxx

                    实际执行顺序 1 2 4 3
                    如果涉及到操作 DOM,则最后执行
                    如果想要顺序执行,则必须添加
                    this.$nextTick(()=>{
                        操作 DOM 的语句
                    })

                    1:xxxx
                    2:xxxx
                    this.$nextTick(()=>{
                        3:操作 DOM
                    })
                    4:xxxx
                    执行顺序 1 2 3 4
                */
                this.$nextTick(()=>{
                    this.$refs['myUpdateForm'].resetFields()
                })
                /* 3:向后台发送信息,获取要被修改的数据,这个数据填充表单 */
                dao.queryById(id).then(response=>{
                    console.log(response)
                    if(response.data.flag){
                        /* 此处使用深浅拷贝均可 */
                        this.updateForm = {...response.data.data}
                    }
                })
            },
            /* 编辑 步骤 2 完成编辑 */
            finishUpdate(){
                /* 表单提交进行验证 */
                this.$refs['myUpdateForm'].validate(valid=>{
                    /* 如果验证成功 */
                    if(valid){
                        dao.update(this.updateForm).then(response=>{
                            this.$message({

type:response.data.flag?'success':'error',
                                message:response.data.msg,
                                showClose:true,
                            })
                            if(response.data.flag){
                                /* 回显 */
                                this.fetchData()
                                /* 关闭对话框 */
                                this.dialogFormVisible = false
                            }
```

```
                        })
                    }
                })
            },
            /* 删除 */
            remove(id){
                /* 此处使用了 ElementUI 的 MessageBox弹框  */
                this.$confirm('您确定删除本条记录吗?', '提示', {
                    /* 确定按钮文本 */
                    confirmButtonText: '确定',
                    /* 取消按钮文本 */
                    cancelButtonText: '取消',
                    /* 标题提示 */
                    type: 'warning'
                }).then(() => {
                    /* 点击确定后执行的代码 */
                    dao.delById(id).then(response=>{
                        this.$message({
                            type:response.data.flag?'success':'error',
                            message:response.data.msg,
                            showClose:true,
                        })

                        if(response.data.flag){
                            /* 回显 */
                            this.fetchData()
                        }
                    })
                }).catch(() => {
                    /* 点击取消后执行的代码 */
                    this.$message({
                        type: 'info',
                        message: '已取消删除'
                    });
                });
            },
            /* 修改权限 */
            toggleRole(id,role){
                /* 更新id */
                this.updateRole.id = id
                /* 更新权限修改后的值 */
                this.updateRole.role = role
                dao.update(this.updateRole).then(response=>{
                    this.$message({
                        type:response.data.flag?'success':'error',
                        message:response.data.msg,
                    })

                    if(response.data.flag){
                        /* 回显 */
                        this.fetchData()
                    }
                })
            },
            /* 根据字段值为表格添加 class
```

```
                row:就表示这一行的对象
                rowIndex:就表示索引
            */
            tableRowClassName({row, rowIndex}) {
                /* 如果是男士 */
                if(row.gender===0){
                    /* 则给这一行添加 class="male"
                    返回的就是类名 */
                    return 'male'
                }else if(row.gender===1){
                    /* 则给这一行添加 class="female" */
                    return 'female'
                }
            }
        },
        created(){
            this.fetchData()
        },
    }
</script>

<style scoped>
    .table{
        width:100%;
        height:100%;
        background-color: whitesmoke;
        /* 如果尺寸溢出,则隐藏不显示滚动条 */
        overflow: hidden;
    }

    /* 设置表格的样式 */
    .middle{
        margin:5px;
        flex:1;
    }
    /* 设置分页的样式 */
    .bottom{
        text-align: center;
        height:5vh;
    }

    /* 设置条件查询表单的样式 */
    .top{
        height:3vh;
        margin:5px;
    }

    /* 此处添加的类全部添加到了 el-table 的子组件上
    这里的样式存在 scoped限制,因此我们无法影响子组件,使用
    >>> 穿透选择器可以直接添加到子组件上,忽略 scoped 属性 */
    .el-table >>> .male{
        background-color: lightblue;
    }

    .el-table >>> .female{
```

```
        background-color: pink;
    }
</style>
```

info index.vue

```
<template>
    <el-container class="info" direction="vertical">
        <!-- 此处使用了 ElementUI Descriptions 描述列表
            title:设置描述列表标题
            :column:表示列数
            border:显示边框
        -->
        <el-descriptions title="用户详细信息" :column="1" border
        v-if="user">
            <!-- 此处用来嵌套按钮,添加描述列表的功能 -->
            <template slot="extra">
                <el-button type="primary" size="mini"
                round>操作</el-button>
            </template>
            <!-- v-if="user.path" 如果没有图片路径在不显示本列 -->
            <el-descriptions-item v-if="user.path">
                <template slot="label">
                    <i class="el-icon-user"></i>
                    用户头像
                </template>
                <!-- 此处使用了 ElementUI的 Image 图片
                    :src:表示图片的路径,注意此处路径必须是一个完整的
                    网络地址,不能使用本地路径
                -->
                <el-image :src="user.path"
                style="width:200px;height:80px"></el-image>
            </el-descriptions-item>
            <!-- el-descriptions-item 此处表示列表项 -->
            <el-descriptions-item>
                <template slot="label">
                    <i class="el-icon-user"></i>
                    用户姓名
                </template>
                {{ user.username }}
            </el-descriptions-item>
            <el-descriptions-item>
                <template slot="label">
                    <i class="el-icon-user-solid"></i>
                    真实姓名
                </template>
                {{ user.realname }}
            </el-descriptions-item>
            <el-descriptions-item>
                <template slot="label">
                    <i class="el-icon-platform-eleme"></i>
                    邮箱地址
                </template>
                {{ user.email }}
            </el-descriptions-item>
```

```
            <el-descriptions-item>
                <template slot="label">
                    <i class="el-icon-mobile-phone"></i>
                    手机号码
                </template>
                {{ user.phone }}
            </el-descriptions-item>
            <el-descriptions-item>
                <template slot="label">
                    <i class="el-icon-s-custom"></i>
                    用户性别
                </template>
                <!--{{ user.gender ===0?'男':'女' }}-->
                <!--过滤器-->
                {{ user.gender | genderFilter }}
            </el-descriptions-item>
            <el-descriptions-item>
                <template slot="label">
                    <i class="el-icon-s-opportunity"></i>
                    用户权限
                </template>
                <!-- 此处使用了 ElementUI 的 Tag 标签
                    type:如果是 success 绿色标签 info 灰色标签
                -->
                <el-tag :type="user.role===1?'success':'info'">
                    {{ user.role===1?'管理员':'用户' }}
                </el-tag>
            </el-descriptions-item>
        </el-descriptions>
    </el-container>
</template>

<script>
    const genderOptions = [
        { id:0,type:'男' },
        { id:1,type:'女' }
    ]
    import dao from '@/api/dao'
    export default {
        data(){
            return {
                user:null,
            }
        },
        created(){
            /* 接受传递过来的值
                this.$route.query.键
            */
            let id = this.$route.query.id
            if(id){
                dao.queryById(id).then(response=>{
                    if(response.data.flag){
                        this.user = response.data.data
                    }
                })
```

```
            }

        },
        filters:{
            /* 过滤器不能使用 this,同时不能与 v-model 连用 */
            genderFilter(val){
                const genderObj = genderOptions.find(genderOption =>
                genderOption.id === val)
                return genderObj?genderObj.type:''
            },
        }
    }
</script>

<style scoped>
    .info{
        width:100%;
        height:100%;
        background-color: whitesmoke;
        justify-content: center;
        align-items: center;
        overflow: hidden;
    }
</style>
```

day17(第三周考试题)

```
                                                                    姓名:_____          分数:_____
1.请简单说出路由(激活)跳转的两种方式(仅仅写出脚本语句即可,拼写错误不得分,一个5个共10分)?
    this.$router.push({path:'哈希值'})
    <router-link to="哈希值">

2.请说出Vue2生命周期几个钩子在开发中的作用(至少3个仅写出开发中作用即可10分)
    created():最早获取数据的时机,一般用来发送异步开启定时器等操作
    mounted():真实DOM覆盖完整页面呈现稳定状态,一般用来发送异步,操作DOM等
    beforeDestroy():Vue实例或者组件销毁之前执行,用来释放资源,关闭定时器,取消订阅等收尾工作

3.Promise承诺对象解决了开发中什么问题?请简单说出其做了什么(单词拼写错误不得分)?
    解决回调地狱问题
    如果发送异步成功则底层调用 resolve(response) 则外界可以知道内部异步成功
    调用then(response=>{})
    如果发送异步失败,则底层调用 reject(err) 则外界可以知道内部异步失败
    调用catch(err=>{})
    不管成功失败,外界都可以调用 finally

4:请说出Vue2流程控制的区别及其使用场合?
    v-if:如果后面是真值,则元素显示,如果后面是假值,则元素不显示,底层根本不渲染,由于切换消耗较大,因此,适用于切换不频繁的场合
```

```
    v-show: 如果后面是真值,则元素显示,如果后面是假值,则元素不显示,底层依然渲染,只不过添加了一个 display:none 的行内式,      初始载入消耗较大,但是之后切换消耗较小,因此适用于切换频繁的场合

5.请简述Vue2组件传值的方式? (注意单词拼写错误不得分,写出三种即可10分)
    props:父子传值
    $emit:子父传值
    slot:插槽分发

6.Vue2组件中 scoped属性的作用是?
    仅仅影响本组件模板样式,不影响子组件模板样式

7.组件中data为什么是一个函数?
    组件经常被复用,函数存在块级作用域,保护内部数据不受外界污染,Vue 实例不会被复用,内部数据无需保护

8. Vue2中函数 计算属性 侦听器如何激发? (仅仅写出如何激发即可,拼写错误不得分,总分10分)v-for="province in provinces"
    函数:激发事件 直接被调用
    计算属性:依赖的数据发生更改
    侦听器:侦听的数据发生变动

9.请写出你使用过的Vue2指令(写出指令名,简述其作用即可,单词拼写错误不得分10分)
    + v-once:一次性插值绑定,之后失去绑定功能
    + v-html:向元素中插入超文本,注意!为了防止网络 XSS 攻击,禁止插入脚本
    + v-text:向元素中插入文本
    + v-model:使用在表单项中,支持双向绑定
    + v-bind:语法糖:,绑定元素的属性
    + v-on:语法糖@,绑定事件
    + v-for:迭代数据
    + v-if:后面如果为真,则元素显示,为假,元素不显示,底层不渲染
    + v-else-if:搭配 v-if 使用必须紧邻
    + v-else:搭配 v-if 使用必须紧邻
    + v-show:后面如果为真,则元素显示,为假,元素不显示,底层依然渲染,只不过添加了.display:none;css行内式
    + v-pre:提示 Vue 实例不解析
    + v-cloak:解决闪现问题
    + v-slot:父子组件传值时传递模板使用,是插槽的另外一种书写方式

10.Vue2如何实现对象和数组的可响应式(仅简单写出脚本语句即可10分)
    + 对象
        + this.$set(对象,'属性名',属性值)
        + this.$delete(对象,'属性名')
    + 数组
        + push() unshift() reverse() sort() pop() splice() shift()
```

day18（导航守卫）

permission.js

```
/* 导入路由表 */
import router from './router'

/*
    导航守卫
    是路由提供的一种功能,用来进行轻量级的权限控制,更加复杂了权限控制功能推荐使用
    Vuex,只要哈希值发生更改,则立刻执行 beforeEach() 函数,执行函数发生在跳转之前
    类似一个守卫要查验
    to from next:三个形参 都是函数
    to:负责控制哈希值更改后抵达的目的地
    from:负责控制哈希值更改前的七点
    next:负责跳转过程
*/
router.beforeEach((to,from,next)=>{
    /* 获取权限信息 */
    const user = JSON.parse(localStorage.getItem('et2301elementui'))

    if(!user){
        /* 如果没有权限信息 */
        if(to.path !== '/'){
            /* 且不是去登录页面 */
            if(to.path === '/register'){
                /* 无权限,也不是去登录页,但是去注册页,则依然放行 */
                next()
                return
            }
            /* 无权限信息,也不是去登录页,也不是去注册页
            强制将哈希变为 #/ 从而终端当前的哈希切换,立刻切换到 #/
            从而跳转到登录组件 */
            next({path:'/'})
        }else{
            /* 没有权限信息,且是去登录页面,则放行 */
            next()
        }
    }else{
        /* 如果存在权限信息,此处放行,继续后面的操作 */
        next()
    }

})
```

配置main.js

```
/* 导入 Vue 依赖注意这里是运行版,因此需要手写 render 渲染根组件到模板 */
import Vue from "vue";
/* 导入 ElementUI 的依赖 */
import ElementUI from 'element-ui';
/* 导入 ElementUI 全局 css */
import 'element-ui/lib/theme-chalk/index.css';
/* 导入根组件 */
import App from "./App.vue";
/* 导入路由表 */
```

```
import router from "./router";
/* 关闭控制台部署提示 */
Vue.config.productionTip = false;
/* Vue 实例加载 ElementUI 图形界面库 */
Vue.use(ElementUI);
/* 导入导航守卫 */
import './permission'

new Vue({
    /* 加载路由表 */
    router,
    /* 手写 render 将根组件覆盖到模板中 div#app 处 */
    render: (h) => h(App),
}).$mount("#app");
```

day19(Echarts配置)

dao.js

```
/* 导入自定义 axios 实例 */
import request from '@/util/request'

/* API  application program interface 此处表示应用程序接口 */
export default {
    /* 1:登录
        username:形参 用户姓名
        password:形参 用户密码
    */
    login(username, password) {
        /* 返回一个 */
        return request({
            /* url:此处表示设置进阶地址
            注意最终发送的完整地址是 基本地址+进阶地址 */
            url: `/testUser/login?username=${username}&password=${password}`,
            method: 'get',
        })
    },
    /* 2:注册
        pojo:形参,对象,内部封装了要注册的八个字段
    */
    reg(pojo) {
        return request({
            url: '/testUser/add',
            method: 'post',
            /* 发送 json 注意这里直接发送 js 对象或者数组即可,不需要
            自己转换 */
            data: pojo,
        })
    },
    /* 3:分页查询
        page:形参,当前页
```

```
        itemsPerPage:形参,每页记录数
        searchMap:形参 对象 内部封装了条件的查询的字段
    */
    query(page, itemsPerPage, searchMap) {
        return request({
            url: `/testUser/select?
page=${page}&itemsPerPage=${itemsPerPage}`,
            method: 'get',
            /* 由于 get 不能发送 json,这里提供了 params
            来封装对象,根据这个对象中是否存在字段,进行有选择的拼接 */
            params: searchMap,
        })
    },
    /* 4:根据 id 查询
        id:形参表示要查询的数据 id
    */
    queryById(id) {
        return request({
            /* 如果只有一个参数传递,则可以使用这种简略写法 */
            url: `/testUser/selectById/${id}`,
            method: 'get',
        })
    },
    /* 5:修改
        newPojo:对象 形参,封装了要被修改的字段以及被修改的数据的 id
    */
    update(newPojo) {
        return request({
            url: '/testUser/update',
            method: 'put',
            data: newPojo,
        })
    },
    /* 6:删除 */
    delById(id) {
        return request({
            url: `/testUser/delete/${id}`,
            method: 'delete',
        })
    },
    /* 7:省市区级联查询
        此处发送 0 则返回所有省份
        此处发送具体某个省份 pid 则返回这个省份下面的城市
        此处发送具体某个城市 pid 则返回这个城市下面的区域
    */
    search(pid) {
        return request({
            url: `/district/select?pid=${pid}`,
            method: 'get',
        })
    },
    /* -------------以下为虚拟接口,没有实际发送异步请求----------------- */
    /* 8:获取面板数据 */
    getDatas() {
        return new Promise((resolve, reject) => {
```

```
            resolve({
                code: 200,
                flag: true,
                msg: '查询成功!',
                data: [123, 456, 789],
            })
        })
    },
    /* 9:获取堆叠柱状图数据 */
    getBarChartDatas() {
        return new Promise((resolve, reject) => {
            resolve({
                code: 200,
                flag: true,
                msg: '查询成功!',
                data: {
                    title: '2023济南超市销量',
                    legend: ['统一银座', '橙子便利店', '便利蜂', '711',
'Lowsen'],
                    xAxis: ['10-08', '10-09', '10-10', '10-11', '10-12'],
                    datas: [
                        [320, 302, 301, 334, 390, 330, 320],
                        [320, 332, 301, 334, 490, 330, 310],
                        [220, 182, 191, 234, 290, 330, 310],
                        [150, 212, 201, 154, 190, 330, 410],
                        [420, 532, 501, 234, 290, 330, 320]
                    ],
                },
            })
        })
    },
    /* 10:拿取饼状图数据 */
    getPieDatas() {
        return new Promise((resolve, reject) => {
            resolve({
                code: 200,
                msg: '查询成功',
                data: {
                    datas: [
                        { value: 800, name: 'SonyPs5' },
                        { value: 400, name: 'SonyPs4' },
                        { value: 799, name: '任天堂NS' },
                        { value: 540, name: '微软XSX' },
                        { value: 100, name: '其它' }
                    ],
                    nameList: ['SonyPs5', 'SonyPs4', '任天堂NS', '微软XSX',
                        '其它'],
                }
            })
        })
    },
    /* 11:拿取柱状图数据 */
    getBarDatas() {
        return new Promise((resolve, reject) => {
            resolve({
```

```
                code: 200,
                msg: '查询成功',
                data: {
                    xDatas: ['IPhone13', 'MI11', 'MIMIX4', 'Oppo',
'Samsung'],
                    values: [1200, 900, 550, 400, 150],
                }
            })
        })
    }
}
```

dashboard组件

```
<template>
    <el-container class="dashboard" direction="vertical">
        <!-- 引用面板 v-if="panelDatas.flag" 用来控制子组件的有无,
        如果为 false,则根本没有子组件 -->
        <panel-group v-if="panelDatas.flag"
        :myNumber1="panelDatas.myNumber1"
        :myNumber2="panelDatas.myNumber2"
        :myNumber3="panelDatas.myNumber3"></panel-group>
        <!-- 引用堆叠柱状图 -->
        <mix-bar-chart v-if="chartDatas.flag"
        :myTitle="chartDatas.title"
        :myLegend="chartDatas.legend"
        :myXaxis="chartDatas.xAxis"
        :myDatas="chartDatas.datas" ></mix-bar-chart>
    </el-container>
</template>

<script>
    import dao from '@/api/dao'
    import PanelGroup from './components/panelgroup'
    import MixBarChart from './components/mixbarchart'
    export default {
        components:{
            PanelGroup,
            MixBarChart,
        },
        data(){
            return {
                /* 封装面板数据 */
                panelDatas:{
                    myNumber1:0,
                    myNumber2:0,
                    myNumber3:0,
                    /* 用来控制子组件的显示与否,默认 false 没有子组件 */
                    flag:false,
                },
                /* 封装图表数据 */
                chartDatas:{
                    title:'',
                    legend:[],
                    xAxis:[],
```

```
                    datas:[],
                    /* 用来控制子组件的显示与否,默认 false 没有子组件 */
                    flag:false,
                },
            }
        },
        methods:{
            /* 获取面板数据 */
            async queryPanelDatas(){
                const response = await dao.getDatas()
                if(response.flag){
                    this.panelDatas.myNumber1 = response.data[0]
                    this.panelDatas.myNumber2 = response.data[1]
                    this.panelDatas.myNumber3 = response.data[2]
                    this.panelDatas.flag = true
                }
            },
            /* 获取图表数据 */
            async queryChartDatas(){
                const response = await dao.getBarChartDatas()
                if(response.flag){
                    this.chartDatas.title = response.data.title
                    this.chartDatas.legend = response.data.legend
                    this.chartDatas.xAxis = response.data.xAxis
                    this.chartDatas.datas = response.data.datas
                    this.chartDatas.flag = true
                }
            },
        },
        created(){
            this.queryPanelDatas()
            this.queryChartDatas()
        },
    }
</script>

<style scoped>
    .dashboard{
        width:100%;
        height:100%;
        background-color: whitesmoke;
        overflow: hidden;
    }
</style>
```

panelgroup子组件

```
<template>
    <el-header height="15vh" class="header">
        <el-row :gutter="12">
            <el-col :xs="8" :sm="8" :md="8" :lg="8" :xl="8">
                <el-card :body-style="{height:'10vh'}"
                shadow="hover">
                    <!--
                        :start-val:起始数字
```

```
                        :end-val:结束数字
                        :duration:滚动时间 单位 ms
                    -->
                    今日新增:
                    <count-to :start-val="0" :end-val="myNumber1"
                    :duration="3000" class="panel-num"></count-to>
                </el-card>
            </el-col>
            <el-col :xs="8" :sm="8" :md="8" :lg="8" :xl="8">
                <el-card :body-style="{height:'10vh'}"
                shadow="hover">
                    今日治愈:
                    <count-to :start-val="0" :end-val="myNumber2"
                    :duration="3000" class="panel-num"></count-to>
                </el-card>
            </el-col>
            <el-col :xs="8" :sm="8" :md="8" :lg="8" :xl="8">
                <el-card :body-style="{height:'10vh'}"
                shadow="hover">
                    无症状感染:
                    <count-to :start-val="0" :end-val="myNumber3"
                    :duration="3000" class="panel-num"></count-to>
                </el-card>
            </el-col>
        </el-row>
    </el-header>
</template>

<script>
    /* 导入 vue 小插件 */
    import CountTo from 'vue-count-to'
    export default {
        props:['myNumber1','myNumber2','myNumber3'],
        /* 将插件注册为子组件 */
        components:{
            CountTo,
        }
    }
</script>

<style scoped>
    .el-card{
        margin-top: 10px;
    }

    .panel-num{
        font-size:xx-large;
        color:coral;
    }
</style>
```

maxbarchart子组件

```
<template>
    <el-main class="main">
        <el-row>
            <el-col :xs="24" :sm="24" :md="24" :lg="24" :xl="24">
                <el-card :body-style="{padding:'0px'}">
                    <!-- echart必须被渲染在 html 元素中 -->
                    <div ref="myChart"
                    style="width:100%;height:68vh"></div>
                </el-card>
            </el-col>
        </el-row>
    </el-main>
</template>

<script>
    /* 导入 echarts 依赖 */
    import echarts from 'echarts'
    /* 导入主题文件 不是必须 */
    require('echarts/theme/vintage')
    export default {
        props:{
            myTitle:{
                type:String,
                default:'标题错误',
            },
            myLegend:{
                type:Array,
                /* 注意默认值如果是复杂类型,必须使用函数返回数据的写法
                这里是箭头函数 function(){return ....} */
                default:()=>['图例错误'],
            },
            myXaxis:{
                type:Array,
                default:()=>['x轴数据错误'],
            },
            myDatas:{
                type:Array,
                default:()=>['数据系列错误'],
            }
        },
        data(){
            return {
                /* 此对象用来进行图表的渲染 */
                chart:null,
            }
        },
        methods:{
            /* 此函数用来在指定的 DOM 中渲染堆叠聚合柱状图 */
            initChart(){
                /* 此句表示将图表渲染到哪里,这里表示渲染到 ref 为 myChart 的
                html 元素 */
                this.chart =
echarts.init(this.$refs['myChart'],'vintage')
```

```
                /* 加载配置项 */
                this.chart.setOption({
                    /* 设置标题 */
                    title: {
                        /* 主标题 */
                        text: this.myTitle,
                    },
                    /* 设置提示框 */
                    tooltip: {
                        /* 显示提示信息 item:图形触发 axis:坐标轴触发 none:不触发 */
                        trigger: 'axis',
                        /* 坐标轴指示器 type:属性 line shadow cross none */
                        axisPointer: {
                            type: 'shadow'
                        }
                    },
                    /* 设置图例 */
                    legend: {
                        /* 注意此处必须与数据系列series中的name属性一一对应 */
                        data: this.myLegend,
                    },
                    /* 设置y轴，注意如果是水平的柱状图，则交换x轴 y轴即可 */
                    yAxis: {
                        /* value表示具体的值 */
                        type: 'value'
                    },
                    /* 设置x轴 */
                    xAxis: {
                        /* category表示类别 */
                        type: 'category',
                        /* x轴上显示的数据 */
                        data: this.myXaxis,
                    },
                    /* 数据系列 */
                    series: [
                        {
                            /* 此处必须一一对应图例legend中的值 */
                            name: '统一银座',
                            /* bar表示柱状 如果是 line则表示折线 */
                            type: 'bar',
                            /* 设置此参数开启堆叠，如果无此参数，则无堆叠效果，凡是此参数一样的
                            数据都会堆叠，叫什么无所谓，但是要保持一致 */
                            stack: '总量',
                            /* 设置柱状图上每段柱子是否显示具体数据 */
                            label: {
                                /* 显示数据 */
                                show: true,
                                /* 在柱子内部显示 还可以书写 top right left bottom */
                                position: 'inside'
                            },
                            data: this.myDatas[0]
                        },
```

```
                        {
                            name: '橙子便利店',
                            type: 'bar',
                            stack: '总量',
                            label: {
                                show: true,
                                position: 'inside'
                            },
                            data: this.myDatas[1]
                        },
                        {
                            name: '便利蜂',
                            type: 'bar',
                            stack: '总量',
                            label: {
                                show: true,
                                position: 'inside'
                            },
                            data: this.myDatas[2]
                        },
                        {
                            name: '711',
                            type: 'bar',
                            stack: '总量',
                            label: {
                                show: true,
                                position: 'inside'
                            },
                            data: this.myDatas[3]
                        },
                        {
                            name: 'Lowsen',
                            type: 'bar',
                            stack: '总量',
                            label: {
                                show: true,
                                position: 'inside'
                            },
                            data: this.myDatas[4]
                        }
                    ]
                })

            },
        },
        /* 这里要操作 DOM,因此将画图的操作放置在 mounted 中
        因为此时真实 DOM 已经准备完毕,已经覆盖了虚拟 DOM */
        mounted(){
            /* 为了保险起见推荐操作 DOM 的语句放置在 nextTick 中
            保证立即执行 */
            this.$nextTick(()=>{
                this.initChart()
            })
        },
        /* 当组件销毁前执行 */
```

```
        beforeDestroy(){
            /* 如果已经为 null 则返回 */
            if(!this.chart){
                return
            }
            /* 强制销毁 chart dispose() 此函数是 echarts 提供的函数*/
            this.chart.dispose()
            this.chart = null
        },
    }
</script>

<style scoped>
    .main{
        flex:1;

    }
</style>
```

---

day20（Vue3.2.47）

使用 Vite 构建 Vue3 前端工程

- ***使用 NPM 步骤***
  - ***npm create vite@latest my-vue-app -- --template vue***
  - ***注意这里仅仅使用了 Vue 模板,Vite 还支持各种技术的模板***
    - ***vanilla，vanilla-ts, vue, vue-ts，react，react-ts，react-swc，react-swc-ts，preact，preact-ts，lit，lit-ts，svelte，svelte-ts。***
  - ***安装成功后 使用 npm i 下载依赖,与 VueCli 不同 Vite 事先没有下载任何依赖需要我们自己下载***
  - ***npm run dev 开启服务器***

1-初始Vue3.0.vue

```
<template>
    <h1>初识Vue3.0 setup 的舞台</h1>
    {{ count }}
    {{ person }}{{ person.name }}{{ person.age }}
    <button @click="touch1">点我试试!</button>
</template>

<script>
    export default {
        /* 此处为 Vue3.0 的老写法,在脚本中只存在一个配置项就是此函数
        这个函数返回一个对象 */
        setup(){
            let count = 0

```

```
            const person = {
                name:'胡桃',
                age:17,
            }

            const touch1 = () => count++

            return {
                /* 如果要在模板中直接使用,则需要返回数据 */
                count,
                person,
                /* 必须返回函数,否则无法在模板中使用 */
                touch1,
            }
        }
    }
    /* 注意以上书写方式,没有实现可响应式,没有单向绑定功能 */
</script>

<style scoped>

</style>
```

2-ref和reactive.vue

```
<template>
    <h1>Vue3.2最新书写方式</h1>
    <button @click="touch1">{{ count }}</button>
    <button @click="touch2">修改对象</button>
    <!-- 注意如果使用 ref 封装也不需要从模板中书写.value -->
    {{ person.name }}{{ person.hobby }}
</template>

<script setup>
    /* 在 Vue3.2中开发者不需要再书写那个唯一的配置项 setup
    只需要再 script 标签中书写即可
    不需要导出默认成员,会自动导出 */
    /* 如果要实现数据的可响应式则必须导入 ref 和 reactive 函数 */
    import {ref,reactive} from 'vue'
    /*
        ref:函数,可以实现基本类型和复杂类型的可响应式
        如果要对基本类型添加可响应式,则必须 ref(基本类型)
        注意在脚本中必须通过 基本类型.value 才能获取数据,被封装的基本类型
        是一个 引用对象 RefImpl 内部通过 value 属性才可以获取到基本类型的
        值
        reactive:函数,可以实现复杂类型的可响应式
    */
    let count = ref(0)
    console.log(count)

    const touch1 = () => count.value++

    /*
        const person = ref({
            name:'胡桃',
```

```
            age:17,
        })

        const touch2 = () => person.value.name = '甘雨'
        不推荐使用 ref 封装复杂类型,而推荐使用 reactive 封装复杂类型
        封装之后是一个 proxy 代理对象,底层由反射 reflect 实现
        不需要像 ref 一样通过.value 获取值
    */

    const person = reactive({
        name:'胡桃',
        age:17,
        hobby:['吃饭','逛街'],
    })

    console.log(person)
    const touch2 = () => {
        person.name = '甘雨'
        person.hobby[2] = '恶作剧'
    }

    /* 在正常开发中,推荐即使基本类型也使用 reactive,可以将多个基本类型
    封装到 对象中,这样就可以合法的使用 reactive 封装了 */
</script>

<style scoped>

</style>
```

3-计算属性.vue

```
<template>
    <h1>计算属性</h1>
    姓: <input type="text" v-model.trim="person.firstName">
    <br>
    名: <input type="text" v-model.trim="person.lastName">
    <br>
    全名(单向):{{ person.fullName1 }}
    <br>
    全名(单双向):<input type="text"
    v-model.trim.lazy="person.fullName2">
    <hr>

</template>

<script setup>
    import {reactive,computed} from 'vue'

    const person = reactive({
        firstName:'张',
        lastName:'三',
    })

    /* 使用计算属性,单向绑定
        computed(function(){
```

```
            return xxxx
        })
    */
    person.fullName1 =
    computed(()=>`${person.firstName}-${person.lastName}`)

    person.fullName2 =
    computed({
        /* 单向 */
        get(){
            return `${person.firstName}-${person.lastName}`
        },
        /* 双向 */
        set(newVal){
            const arr = newVal.split('-')
            if(arr.length===1){
                alert('请输入-作为连字符')
                return
            }
            person.firstName = arr[0]
            person.lastName = arr[1]
        },
    })
</script>

<style scoped>

</style>
```

4-侦听器.vue

```
<template>
    <h1>侦听器</h1>
    <button @click="count++">更改count</button>
    <button @click="str+='!'">更改str</button>
    <hr>
    {{ count }}<br>{{ str }}
    <hr>
    <button @click="update">修改对象</button>
    {{ person }}
</template>

<script setup>
    import {ref,reactive,watch} from 'vue'

    let count = ref(0)
    let str = ref('thisisetoak')

    /* 侦听 ref 封装的基本类型 count */
    /* watch(count,(newVal,oldVal)=>{
        console.log(newVal,oldVal)
    },{immediate:true}) */

    /* 同时侦听 count 和 str */
    watch([count,str],(newVal,oldVal)=>{
```

```
        console.log(newVal,oldVal)
    },{immediate:true})

    const person = reactive({
        name:'胡桃',
        age:17,
    })

    const update = () =>{
        person.name += '!'
        person.age++
    }

    /* 侦听被 reactive 封装的复杂类型
    此处存在一个小 bug newVal 和 oldVal 一直是一样 */
    watch(person,(newVal,oldVal)=>{
        //console.log(newVal,oldVal)
    })

    /* 侦听被 reactive 封装的复杂类型中的某个属性 */
    watch(()=>person.name,(newVal,oldVal)=>{
        //console.log(newVal,oldVal)
    })

    /* 侦听被 reactive 封装的复杂类型中的某多个属性 */
    watch([()=>person.name,()=>person.age]
    ,(newVal,oldVal)=>{
        console.log(newVal,oldVal)
    })


</script>

<style scoped>

</style>
```

5-watchEffect侦听器.vue

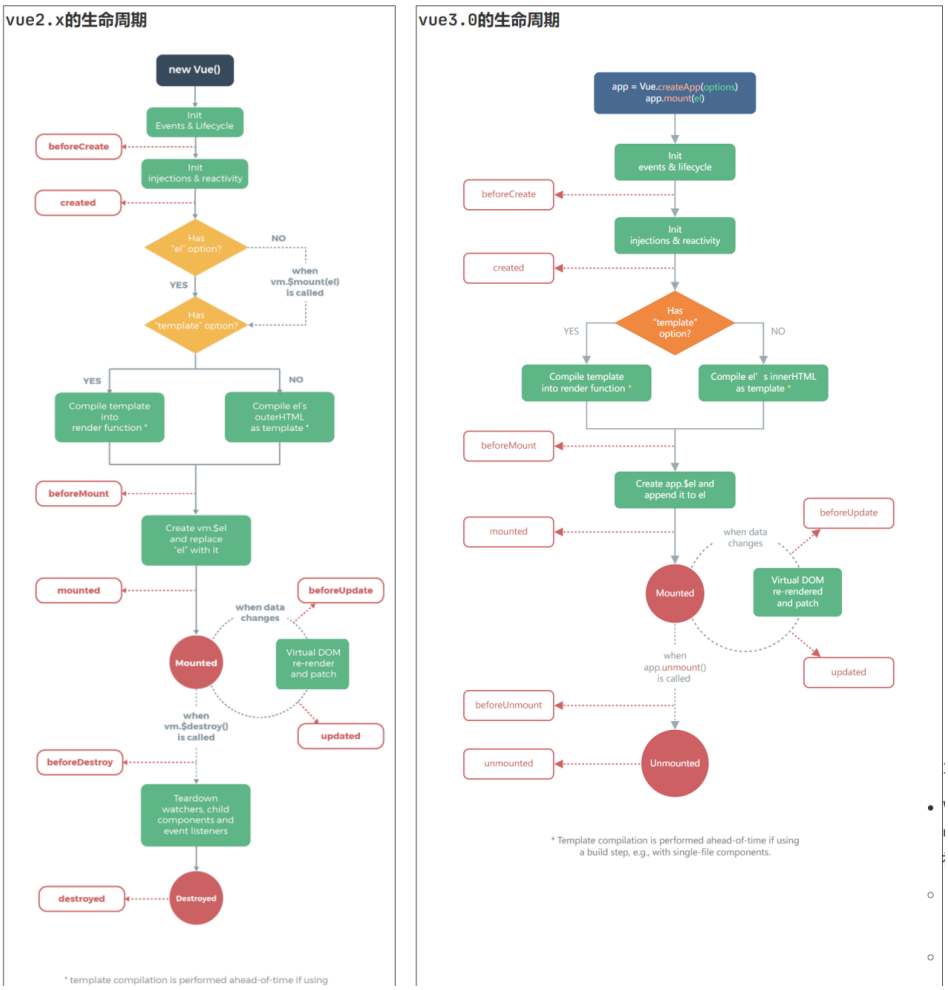
```
<template>
    <h1>watchEffect侦听器</h1>
    <button @click="update">点击测试</button>
</template>

<script setup>
    import {ref,reactive,watchEffect} from 'vue'

    let count = ref(100)

    const person = reactive({
        name:'甘雨',
        age:27,
    })

    const update = () =>{
```

```
        person.name += '!'
        count.value++
    }
    /* 内部涉及到的参数,只要发生改动,则立即执行侦听器
    有点类似计算属性,依赖的数据发生改动,则重新执行 */
    watchEffect(()=>{
        console.log('我是 watchEffect-------------')

        let value1 = count.value
        let value2 = person.name
        console.log(value1,value2)
    })
</script>

<style scoped>

</style>
```

6-Vue3.2生命周期.vue

```
<template>
    <button @click="add">{{ count }}</button>
</template>

<script setup>
    /*
        Vue3与Vue2的生命周期类似,仅仅部分发生了变化,有以下两个命名发生了变化
        beforeDestroy更名为beforeUnmount
        destroyed更名为unmounted
        在书写时必须添加 on 前缀
    */
    import {ref,onBeforeMount,onMounted,onBeforeUpdate,onUpdated
    ,onBeforeUnmount,onUnmounted} from 'vue'

    let count = ref(0)

    const add = () =>{
        count.value++
    }

    onBeforeMount(()=>{ console.log('onBeforeMount---') })

    onMounted(()=>{ console.log('onMounted---') })

    /* 数据更改时才执行 */
    onBeforeUpdate(()=>{ console.log('onBeforeUpdate---') })
    onUpdated(() => {console.log('onUpdated---')})

    /* 组件销毁时执行 */
    onBeforeUnmount(()=>{console.log('onBeforeUnmount---') })
    onUnmounted(()=>{console.log('onUnmounted---') })


</script>

```

```vue
<style scoped>

</style>
```



- 最大的变化Vue3首先加载好模板,才开始进行一切钩子的执行
- Vue3与Vue2的生命周期类似,仅仅部分发生了变化,有以下两个命名发生了变化

beforeDestroy 更名为 beforeUnmount

destroyed 更名为 unmounted

7-defineProps父子传值

父组件

```vue
<template>
    <div class="father">
        <h3>父组件</h3>
        <!-- 引用子组件 -->
        <child :key1="key1" :key2="key2"
        key3="helloworld!!"></child>
    </div>
</template>

<script setup>
    /* 导入子组件 */
    import Child from './components/Child.vue'
```

```
    import {ref,reactive} from 'vue'
    let key1 = ref('你好世界!')
    const key2 = reactive(['抽烟','喝酒','烫头'])
</script>

<style scoped>
    .father{
        width:500px;
        height:500px;
        background-color: lightblue;
        border-radius: 30px;
        display: flex;
        justify-content: center;
        align-items: center;
    }
</style>
```

子组件

```
<template>
    <div class="child">
        <h3>我是孩子</h3>
        <ul>
            <li>{{ key1 }}</li>
            <li>{{ key2 }}</li>
            <li>{{ key3 }}</li>
        </ul>
    </div>
</template>

<script setup>
    /* 使用 defineProps 接受父组件传递过来的值 */
    const props = defineProps({
        /* 书写方式 1 */
        key1:String,
        /* 书写方式 2 */
        key2:{
            type:Array,
            required:true,
        },
        key3:{
            type:String,
            required:true,
            default:'传递失败!',
        },
    })
</script>

<style scoped>
    .child{
        width:300px;
        height:300px;
        background-color: coral;
        border-radius: 30px;
        text-align: center;
```

```
    }
</style>
```

8-defineEmits父子传递函数

父组件

```
<template>
    <div class="father">
        <h3>我是爸爸</h3>
        <!-- 使用自定义事件绑定了一个函数 -->
        <child2 @hello="show"></child2>
    </div>
</template>

<script setup>
    /* 导入子组件 */
    import Child2 from './components/Child2.vue'

    const show = (val1,val2) => alert(val1+val2)
</script>

<style scoped>
    .father{
        width:500px;
        height:500px;
        background-color: lightblue;
        border-radius: 30px;
        display: flex;
        justify-content: center;
        align-items: center;
    }
</style>
```

子组件

```
<template>
    <div class="child">
        <h3>我是孩子</h3>
        <button @click="touch">激发父组件的函数</button>
    </div>
</template>

<script setup>
    /* 使用 Vue3.2提供的 defineEmits 绑定父组件的自定义事件 */
    const emit = defineEmits(['hello'])
    /* 激发自定义事件 emit('激发的父组件自定义事件','传递的值') */
    const touch = ()=>emit('hello','完结','撒花!!!')
</script>

<style scoped>
    .child{
        width:300px;
        height:300px;
```

```
        background-color: coral;
        border-radius: 30px;
        text-align: center;
    }
</style>
```