vue第2天

今天目标：

- 案例-品牌管理
  - 实现 显示、隐藏 功能
  - 品牌的 删除、筛选 功能
- 熟练使用**computed**计算属性
- 能够创建和使用**过滤器**
- 熟练使用**按键修饰符**
- 了解**自定义指令**的创建和应用
- 掌握Vue生命周期用法

品牌管理实现

vue指令-v-if&v-show

目标：

掌握v-if/v-show的用法，知道实现原理

在vue中，v-if 和 v-show 会根据接收 **true/false** 信息使得页面上元素达到显示或隐藏的效果

语法：

```
<标签 v-if="true/false"></标签>
<标签 v-show="true/false"></标签>
<!--true:显示   false:隐藏-->
```

原理：

v-if：通过 **创建、销毁** 方式达到显示、隐藏效果的(销毁后有一个占位符)

v-show：其是通过css控制达成显示、隐藏效果的

> display:none; 隐藏
>
> display:block; 显示

特点：

v-if 有更高的**切换消耗** 、v-show有更高的**渲染消耗**

如果需要**频繁**切换 则v-show 较合适，如果运行条件不大可能改变 则v-if 较合适。

注意：

v-if使得元素被**隐藏**后，这个元素的物理位置有一个名称为""的占位符，其与html的注释信息**没有关系**

简单案例：

通过按钮控制，使得元素内容在 显示 和 隐藏 之间切换

```
  <style>
    #one{width: 300px;height: 40px;background-color:orange;}
    #two{width: 300px;height: 40px;background-color:lightgreen;}
  </style>
</head>
<body>
  <div id="app">
    <h2>v-if和v-show</h2>
    <button @click="flag=!flag">切换</button>
    <!--xxxxxxxxxxx-->
    <p id="one" v-if="flag">学习vue第二天---v-if</p>
    <p id="two" v-show="flag">学习vue第二天---v-show
      <img src="" alt="">
      <img src="" alt="">
      <img src="" alt="">
      <img src="" alt="">
      <img src="" alt="">
      <img src="" alt="">
      <img src="" alt="">
      <img src="" alt="">
      <img src="" alt="">
    </p>
  </div>
  <script src="./vue.js"></script>
  <script>
    var vm = new Vue({
      el:'#app',
      data:{
        flag:false // 控制标签是否显示true/false
      },
      methods:{
      }
     });
  </script>
```

注意：

事件驱动不仅可以是**methods方法**，也可以是简单的**js语句**

vscode设置代码片段(自学)

vscode编辑器：设置--->User snippets---->html.json，配置如下内容

```
    "Print to Vue&html base code": {
        "prefix": "vh",
        "body": [
            "<!DOCTYPE html>",
            "<html lang=\"en\">",
            "<head>",
            "  <meta charset=\"UTF-8\">",
            "  <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">",
            "  <meta http-equiv=\"X-UA-Compatible\" content=\"ie=edge\">",
            "  <title>Document</title>",

            "</head>",
            "<body>",
            "  <div id=\"app\">",
            "  </div>",
            "  <script src=\"./vue.js\"></script>",
            "  <script>",
            "    var vm = new Vue({",
            "      el:'#app',",
            "      data:{",
            "      },",
            "      methods:{",
            "      }",
            "     });",
            "  </script>",
            "</body>",
            "</html>",

            "",
        ],
        "description": "Vue&html base code"
    }
```

vue指令-v-if&v-else

目标：

判断品牌列表有无，并显示不同内容

在Vue中，v-if 、v-else-if 和 v-else 三个指令结合可以实现多路分支结构

- v-if可以**单独**使用，形成单路分支结构
- v-if 和 v-else 也可以合作使用，实现**双路**分支结构
- v-if 、v-else-if 和 v-else 也可以合作使用，实现**多路**分支结构

语法：

```
<标签 v-if="true/false"></标签>
<标签 v-else-if="true/false"></标签>
<标签 v-else-if="true/false"></标签>
<标签 v-else></标签>
```

> 以上4个标签只分支结构，最终只会走一个，第一个为true的那个标签会执行 或 执行v-else

案例应用：

判断品牌信息是否存在，并显示对应内容

```
<table v-if="brandList.length>0">
  ……
</table>
<table v-else>
  <tr><td>没有任何记录！</td></tr>
</table>
```

注意：

v-if和v-else一并使用，页面只没有占位符了

品牌管理-删除

目标：

实现每个品牌删除的功能

步骤：

1. 制作删除按钮和事件 @click="del(序号)"，把单元序号作为参数进行传递
2. 制作methods方法 del()，通过splice()方法对目标数组元素进行删除操作

代码：

设置单击删除事件

```
<tr v-for="(item,k) in brandsList" :key="item.id">
  <td><input type="checkbox"></td>
  <td>{{ item.id }}</td>
  <td>{{ item.name }}</td>
  <td>{{ item.ctime }}</td>
  <td><button @click="del(k)">删除</button></td>
</tr>
```

Vue的methods方法：

```
        // 删除品牌
        // index:被删除品牌的下标信息
        del(index){
          if(!window.confirm('确定要删除该元素么? ')){return false}
          // 数组.splice(下标,长度) 删除数组的元素
          // 该方法会使得目标数组直接发生变化
          // 该方法会把被删除的元素给返回
          this.brandList.splice(index,1)
        },
```

效果：

注意：

1. 在真实项目中删除信息**不会传递**序号信息作为删除条件，临时方案而已
2. splice(下标，长度)：会删除数组中指定的元素，目标数组信息会直接发生变化，splice本身会以数组方式返回被删除的元素

品牌管理-筛选

目标：

根据关键字给品牌列表做筛选功能

v-for做遍历，目标可以是data成员，也可以是methods方法

```
<标签 v-for="(item,k) in data成员" :key="item.id">
<标签 v-for="(item,k) in 复杂表达式" :key="item.id">
```

> 模板容器可以通过“复杂表达式”体现要使用的数据，不仅v-for可以使用

步骤：

1. 给关键字输入框设置v-model="keywords"
2. 在data中声明keywords成员
3. 在methods方法中声明shai()方法，实现数据筛选
4. 模板中 v-for、v-if 对shai()**方法体**代码进行操作

模板代码：

```
<tr>
  <td>
    <input type="text" v-model="newbrand" />
    <button @click="add">添加</button>
  </td>
  <td><input type="text" placeholder="请输入关键字" v-model="keywords"></td>
</tr>
<table v-if="brandList.filter(item=>{ return item.name.includes(this.keywords)
}).length>0">
  ……
  <tr v-for="(item,k) in brandList.filter(item=>{ return item.name.includes(this.keywords)
})"
      :key="item.id">
    ……
  </tr>
</table>
```

> 复杂表达式 只是暂时解决方案

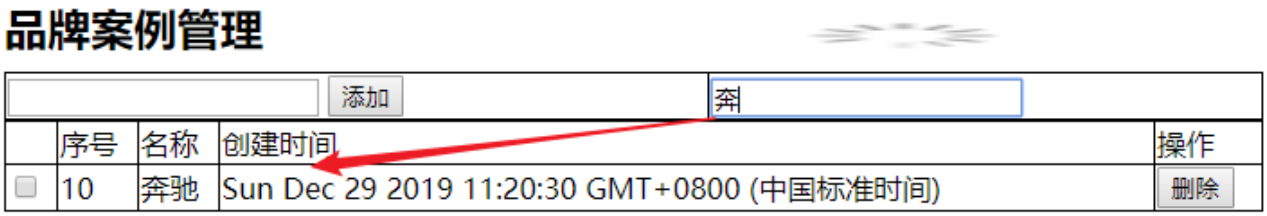
Vue实例的methods方法：

```
// 筛选过滤品牌
shai(){
  // 对brandList遍历，同时判断每个项目是否包括关键字
  // 数组.filter() 对数组做遍历过滤的
  // 1. 有回调函数参数
  // 2. item:分别代表遍历出来的各个元素单元
  // 3. 回调函数变为箭头函数样子，使得内部this与外部保持一致的指引，都是Vue实例
  // 4. 回调函数内部给各个单元做判断，并return返回布尔值
  //    true: 收集当前的项目
  //    false: 抛弃当前的项目
  //    return是filter方法固定语法
  // 5. filter调用完毕，会把过滤的结果进行返回，可以接收使用
```

```
    // this.brandList.filter(function(item){
    var rst = this.brandList.filter(item=>{
      // console.log(this)
      // 判断item.name是否包含keywords
      // 大串.includes(小串)  判断大串是否包含小串的，返回boolean值
      // 任何字符串判断包含空字符串，都返回true
      return item.name.includes(this.keywords)
    })
    console.log(rst)
  },
```

> 注意：
>
> 1. filter(箭头函数)里边设置箭头函数参数，会使得内部this与外部保持一样的指引(new Vue())
> 2. filter回调函数内部必须设置return，返回判断结果
> 3. 该shai的方法没有做具体应用

效果：



注意：

1. vue是“**响应式**”框架，数据发生变化，用到的地方会重新编译执行，包括Vue实例内部和模板页面
2. 由于响应式缘故，搜索框中输入关键字，页面上会立即模糊查找显示相关品牌
3. 数组.filter()：对数组元素做遍历过滤，返回满足条件的新数组
4. 大串.includes(小串)： 判断大串中是否包含小串，返回boolean值，任何字符串判断是否包含 ''空字符串 ，结果都为真

computed计算属性应用**

目标：

知道什么是计算属性及什么时候应用

`computed计算属性`：

Vue本身支持模板中使用**复杂表达式**表现业务数据，但是这会使得模板内容过于杂乱，如果确有需求，可以通过computed计算属性实现，该computed可以对其他data做复杂合成处理的

语法：

```
new Vue({
  el:xx,
  data:xx,
  computed:{
    // 属性名称:function(){
    属性名称(){
      // 业务表达式实现，可以通过this操作data成员
      return  返回结果
    }
  }
})
```

> 计算属性普通函数赋值或简易成员函数 赋值 都可以，不要使用箭头函数

使用：

形式上，如何应用data成员，就如何应用计算属性

```
{{ computed计算属性名称 }}      <!--模板中-->
this.xxx                 <!-- Vue实例内部-->
```

什么时候应用：

如果页面需要访问一个数据，这个数据比较复杂，是需要通过**其他data**经过复杂逻辑制作出来的，就可以使用“计算属性”

特点：

1. 计算属性关联的data如果发生变化，会重新编译执行 获得 并 使用 对应新结果，即**响应式**(入口)
2. 计算属性的返回信息有变化，使用的地方也会重新编译执行，还存在出口**响应式**
3. 计算属性内部可以使用this关键字，与Vue对象等效
4. 每个计算属性都需要通过**return**关键字返回处理结果

与methods方法的区别：

1. computed计算属性本身有“**缓存**”，在关联的data没有变化的情况下，后续会使用缓存结果，节省资源

   methods方法没有缓存，每次访问 方法体 都需要加载执行，耗费资源
2. methods应用**逻辑较复杂**，例如内部可以嵌入ajax，或互相调用，而computed比较纯粹，只是操作data的

案例：

通过computed计算属性**获取**并**应用**筛选的品牌数据

步骤：

1. 创建计算属性

   在Vue实例内部创建计算属性(与el、data、methods并列位置处)，名称为 brandFilters

```
// 声明计算属性
computed:{
  // 创建一个名称为result的计算属性
  brandsFilters () {
    // 可以正常使用this关键字
    return this.brandList.filter(item=>{
      return item.name.includes(this.keywords)
    })
  }
},
```

2. 应用计算属性

```
<table v-if="brandsFilters.length>0">
  <tr>
    <td></td>
    <td>序号</td>
    <td>名称</td>
    <td>创建时间</td>
    <td>操作</td>
  </tr>
  <tr v-for="(item,k) in brandsFilters" :key="item.id">
    <td><input type="checkbox"></td>
    <td>{{ item.id }}</td>
    <td>{{ item.name }}</td>
    <td>{{ item.ctime }}</td>
    <td><button @click="del(k)">删除</button></td>
  </tr>
</table>
```

> 之前走 复杂表达式，现在走brandsFilters计算属性了

过滤器

介绍

`目标`：

知道什么是过滤器

`什么是过滤器`：

答: 在项目应用中，同样一份数据信息，表现形式确有千差万别，例如**时间信息**可以是对象、时间戳、格式化等，**字符串**可以是大写的、小写的、首字母大写的等等，如果**提供方**给我们提供的信息是其中一种形式，而我们需要的是**另一种**，在Vue中，可以通过 “**过滤器**” 转换处理。过滤器是Vue中实现数据格式**转换**的一种机制。本质就是函数

如下时间信息通过 **对象**形式 或 **时间戳**方式显示都不合适，但是变为格式化时间就比较友好

```
Thu Mar 21 2019 17:48:17 GMT+0800     =>   2019-03-21 17:48:17

1553161717                                          =>  2019-03-21 17:48:17
```

过滤器关键字：filter、filters

私有方式应用

`目标`：

品牌案例中，利用过滤器实现**对象时间**转换为**格式化时间**

`声明私有过滤器语法`：

Vue实例化过程中，与el、data平行的位置声明 `filters` 成员并在其中制作过滤器，这个过滤器只能被当前Vue实例使用，称为 "私有过滤器"

```
new Vue({
  filters:{
    // 如下方法格式是es6简易设置方式，完整写法：  过滤器名称:function(被处理数据){]}
    过滤器名称(被处理的数据){
      // 对数据进行加工处理
      return 结果
    },
    ...
  }
})
```

`使用`：

```
{{ 时间信息成员 | 过滤器名称 }}
```

> 过滤器被设置到应用数据的尾部，通过 “|竖线” 连接

设置字符串以指定的位数输出，不够就在字符串前边补位

```
// 字符串.padStart(位数,补位信息)
'hello'.padStart(8,0)      // 000hello
```

注意：

过滤器只可以用在**两个**地方： 插值表达式和 :冒号 属性绑定表达式。

> 1) 插值表达式： {{ 数据 | 过滤器 }}
>
> 2) v-bind属性绑定中使用： <标签 :属性="数据 | 过滤器">
>
> v-if="city|xx" 错误

案例应用：

给品牌案例中各个时间信息做格式化处理

```
<td>{{ item.ctime | timeFMT }}</td>
```

过滤器：

```
// 注册过滤器【私有的】
filters:{
  // 语法:
  // 过滤器名称:function(被处理的目标数据){}
  // timeFMT:function(){
  timeFMT(origin){
    // console.log(origin)
    // 根据origin重新实例化一个时间对象
    // var tm = new Date(对象/时间戳)
    var tm = new Date(origin)
    // 分别获得年、月、日、时、分、秒，并做拼接即可
    var yyyy = tm.getFullYear()
    var mm = (tm.getMonth()+1).toString().padStart(2,0)
    var dd = tm.getDate().toString().padStart(2,0)

    var hh = tm.getHours().toString().padStart(2,0)
    var ii = tm.getMinutes().toString().padStart(2,0)
    var ss = tm.getSeconds().toString().padStart(2,0)

    return `${yyyy}-${mm}-${dd} ${hh}:${ii}:${ss}`

    // 把时间数据从 对象 格式转变为 格式化 样子
    // 注意：需要return返回转换好的结果
    // return '2019-12-29 11:30:57'
  }
},
```

效果：

品牌案例管理

添加 | 请输入关键字

	序号	名称	创建时间	操作
☐	10	宝马	2019-12-15 16:34:33	删除
☐	11	奔驰	2019-12-15 16:34:33	删除
☐	12	奥迪	2019-12-15 16:34:33	删除

注意：

过滤器不让使用this关键字，或者this关键字不是“Vue实例”

全局方式应用

目标：

知道什么是全局过滤器

全局过滤器：

在new Vue()前边，直接给Vue调用filter声明的过滤器称为“全局过滤器”

全局的意思是过滤器可以供**所有Vue实例**使用

```
Vue.filter(名称, function(被处理的数据){})
var vm = new Vue()
var vm2 = new Vue()
```

全局和私有取舍：

理论上，多个Vue实例都需要使用的过滤器声明为全局的，只是当前Vue实例使用的过滤器声明为私有的

实际情况：私有的用的更多

注意：

全局的过滤器需要在new Vue()**之前**声明

应用：

声明一个全局过滤器，可以给多个Vue实例应用

```
<body>
```

```
  <div id="app">
    <p>{{ city }}</p>
    <p>{{ city | qian }}</p>
    <p>{{ city | hou }}</p>
    <p>{{ city | qian | hou }}</p>
  </div>
  <hr />
  <div id="app2">
    <p>{{ address }}</p>
    <p>{{ address | qian}}</p>
    <p>{{ address | hou}}</p>
  </div>

  <script src="./vue.js"></script>

  <script>
    // 全局过滤器，所有的Vue实例都可以使用
    // 注意：需要在new Vue()前边设置
    // Vue.filter(名称,function(被处理数据){return xx})
    Vue.filter('hou',function(origin){
      return origin+',很冷'
    })

    var vm = new Vue({
      el:'#app',
      // 私有过滤器，只是当前"自己Vue实例"可以使用
      filters:{
        qian(origin){
          return '我喜欢'+origin
        }
      },
      data:{
        city:'北京'
      },
    });

    var vm2 = new Vue({
      el:'#app2',
      data:{
        address:'上海'
      }
    })


  </script>
</body>
```

注意：

1. 创建多个div容器、多个Vue实例的情形只是技术实现而已，真实项目很少有这样用的
2. 多个过滤器可以被同时使用，{{ 信息 | 过滤器 | 过滤器 | 过滤器 ..... }}，形成一个信息被多次处理效果

上午总结：

- 案例-品牌管理
  - v-if 和 v-show
  - v-if 和 v-else
  - 实现 显示、隐藏 功能
  - 品牌的 删除
    - del(k下标)
    - methods：del() 数组.splice(下标，长度)
  - 品牌筛选 功能
    - v-model="keywords"
    - data:keywords
    - methods: shai() 数组.filter(function(item){ 判断当前项目是否包含关键字 includes() })
    - v-for 和 v-if: 复杂表达式
- 熟练使用**computed**计算属性
  - 数据--->其他data合成--->使用computed
  - computed--->return
  - 使用：与data样子一致
- 能够创建和使用**过滤器**
  - 数据格式转换机制，函数
  - 场合：{{}} 或 属性绑定:xxx="表达式 | 过滤器"
  - 类型：私有(filters:{名称(被处理数据){}})、全局(Vue.filter(名称,function(被处理数据){}))

按键修饰符

介绍

目标：

知道什么是按键修饰符

什么是：

按键修饰符：使得键盘事件只针对某个(或某几个)按键生效

应用中有许多 键盘事件 (onkeyup、onkeydown、onkeypress、oninput等等)，每个事件在执行的时候可以通过**许多键子**达成，有时我们要求只有按到某个键子时，才激活该事件，例如只有触碰 **回车键**或**ESC键** 才有效果，那么可以通过 **按键修饰符** 实现

> oninput：触碰键盘给输入框做输入动作时会触发执行
>
> onkeyup：键盘抬起触发执行
>
> onkeypress：按下任何字母数字键时触发执行，系统按钮（例如，箭头键和功能键）无法得到识别

> onkeydown：按下任何键盘键（包括系统按钮，如箭头键和功能键）时触发执行

常用控制键键码值：

控制键键码值(keyCode)							
按键	键码	按键	键码	按键	键码	按键	键码
BackSpace	8	Esc	27	Right Arrow	39	-_	189
Tab	9	Spacebar	32	Dw Arrow	40	.>	190
Clear	12	Page Up	33	Insert	45	/?	191
Enter	13	Page Down	34	Delete	46	`~	192
Shift	16	End	35	Num Lock	144	[{	219
Control	17	Home	36	;:	186	|	220
Alt	18	Left Arrow	37	=+	187	]}	221
Cape Lock	20	Up Arrow	38	,<	188	'"	222

键码值：键盘每个按键都对应一个**数字码**，称为 键码值

全部按键键码值： https://blog.csdn.net/qq_39207948/article/details/79882229

语法：

```
<input  @keyup.键码值/别名="处理">
<!--要求只有触碰回车键 才执行该事件-->
<input  @keyup.13="处理">  <!--键码值-->
<input  @keyup.enter="处理">  <!--别名-->

<!--键码值：键盘的每个键子都有一个数字码，就是键码值，event.keyCode 就获取到了-->
```

vue考虑到**键码值**使用多有不便，已经给常用**键码值**(event.keyCode)设置**别名**了

- `.enter`
- `.tab`
- `.delete` (捕获“删除”和“退格”按键)
- `.esc`
- `.space`
- `.up`
- `.down`
- `.left`
- `.right`

也可以自定义其他的**按键别名**

```
// 要求使用 `@keyup.f6`
Vue.config.keyCodes.f6 = 118
<input @keyup.f6="xxx" />  // 只有单击f6键才会触发xxx的回调
```

注意：

如果有的需求比较特殊，需要多个按键一并触发该事件，也可以

`@keyup.ctrl.enter="xxx"` 表示 ctrl和enter一并触碰，才触发事件执行

品牌应用

目标：

给添加品牌的**输入框**设置 按键修饰符

1. **回车键** 被触碰就添加新品牌
2. **ESC键** 被触碰就把已经输入的新品牌给做清除操作，取消添加

```
<input type="text" v-model="newbrand"
       @keyup.enter="add"
       @keyup.esc="newbrand=''" />
```

> 上述中，一个input输入框设置了两个keyup事件，是可以的，它们是通过不同的键触发的

自定义指令

介绍

目标：

了解什么是自定义指令

什么是：

Vue框架给我们提供了许多指令，例如v-if 、v-else、 v-html、v-show、v-text、v-model、v-bind等等，这些指令都是固定的，并不能任意满足我们，有时我们需要一个指令而框架还没有提供，就需要我们自己定义，称为“自定义指令”

关键字：directive directives

声明语法：

```
// 1. 声明全局指令
Vue.directive(指令名称,{ 配置对象成员 })

// 2. 声明私有指令
new Vue({
  directives:{
    指令名称:{ 配置对象成员 }
  }
})

// 配置对象:
inserted(m){
  m: 代表使用该指令的html标签dom对象，可以通过m进行原始dom操作实现业务需求
}
```

注意：

私有指令directives关键字 与el、data等都是并列的

获得焦点-私有

目标：

给品牌案例创建一个**私有自定义指令**，使得在页面加载完毕后添加品牌 ”输入框“ 自动获得焦点

创建指令：

```
      // 注册自定义指令
      directives:{
        // 指令名称:{配置对象成员}
        // 指令名称注册时不要设置"v-",使用时再设置
        'dian':{
          // inserted是固定用法
          // inserted: 时机的事情，代表是div容器被Vue实例编译完毕，并且也渲染好了
          inserted:function(m){
            // m: 代表使用该指令的元素dom对象
            // dom对象可以通过webapi技术操作页面元素
            // m.style.color = 'red'
            m.focus() // 使得input框元素获得焦点
          }
        }
      },
```

应用指令：

```
<input type="text" v-dian v-model="newbrand" @keyup.enter="add" @keyup.esc="newbrand=''" />
```

注意：

无论是全局指令 还是 私有指令，声明的时候都不用设置v-前缀，使用的时候再添加上即可

获得焦点-全局

目标：

给品牌案例创建一个**全局自定义指令**，使得页面加载完毕后添加品牌 ”输入框“ 自动获得焦点

创建指令：

```
Vue.directive('dian2',{
  inserted(m){
    // m: 代表使用指令的html对象(dom对象)
    // console.dir(m)
    // 使得m对象获得焦点
    m.focus()
  }
})
var vm = new Vue()
```

应用：

```
<input type="text" v-dian2 v-model="newbrand" @keyup.enter="add" @keyup.esc="newbrand=''"
/>
```

扩展

template

目标：

了解template用法

在Vue实例内部可以声明template，其内容可以**覆盖掉**原生的div容器的

```
<div id="app">{{ city }}</div>
<script src="./vue.js"></script>
<script>
  var vm = new Vue({
    template: "<span>上海</span>",
    el: "#app",
    data: {
      city: "北京"
    }
  })
</script>
```

> 上述代码执行，页面上会看到“span上海”内容，相反“div北京”已经被覆盖了

$mount

目标：

了解$mount用法

如果Vue没有提供`el`成员帮助找到div容器，那么可以调用$mount()方法

因此Vue实例与容器联系有两种方式：

1) el:'#app'

2) $mount方式

```
var vm = new Vue()
vm.$mount('#app')

或

var vm = new Vue().$mount('#app')  // 连贯调用
```

render成员

目标：

了解render成员使用

在Vue中如果定义了render成员，那么其提供的内容会渲染到页面中，并且会**覆盖**原容器，包括template

优先级关系：render >>>>> template >>>>>>默认容器

render最高

```
<div id="app">
  <p>{{ weather }}</p>
</div>

<script src="./vue.js"></script>

<script>
  var vm = new Vue({
    el:'#app', // Vue实例 与 div容器 联系
    data:{
      weather:'cloud'
    },
    methods:{
    },
    template:'<span>sunshine</span>',
    // render: 渲染，去覆盖原生div容器的
    // render:function(create){
    //   // return create(html标签名称，标签内容区域信息)
    //   return create('h2', 'snow') // 创建 <h2>snow</h2> 元素标签了
    // }
    // render:function(h){
    //   // return create(html标签名称，标签内容区域信息)
    //   return h('h2', 'snow') // 创建 <h2>snow</h2> 元素标签了
    // }
    // render: h=>{
    //   // return create(html标签名称，标签内容区域信息)
    //   return h('h2', 'snow') // 创建 <h2>snow</h2> 元素标签了
    // }

    // 箭头函数体内部只有一个语句，并且有return返回，那么{} 和 return 都可以省略
    render: h=>h('h2', 'snow') // 创建 <h2>snow</h2> 元素标签了
  })

</script>
```

> 上述代码会看到 snow 内容，相反 cloud 和 sunshine 都被覆盖了

console使用

目标：

了解console的基本用法

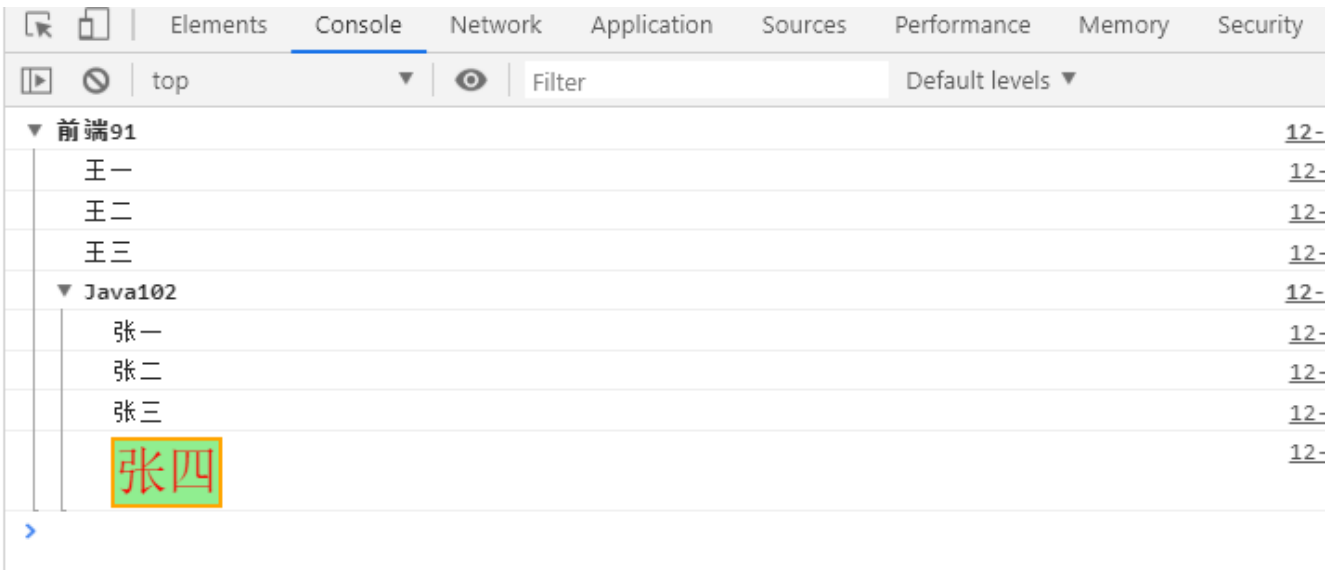
```
console.log()    调试工具普通数据输出
console.dir()    可以把dom对象的各个成员给打印出来

console.group()  对输出的信息做分组处理，更加清楚

console.log('%c%s',css样式设置， 被输出的信息)
c:css样式   与  第2个参数对应
s:string字符串 与   第3个参数对一个
console.log('%c%s','color:red', '你好')
```

应用：

```
console.group('前端91') // 按照分组效果输出调试信息
console.log('王一')
console.log('王二')
console.log('王三')
console.group('Java102') // 按照分组效果输出调试信息
console.log('张一')
console.log('张二')
console.log('张三')
// console.log('%c%s','css样式','输出的信息')
console.log('%c%s','color:red;background-color:lightgreen;font-size:25px;border:2px solid orange;','张四')
```



生命周期

介绍

目标：

知道什么是生命周期

辅助参考：

https://segmentfault.com/a/1190000011381906

什么是：

生命周期是指vue实例(或者组件)从诞生到消亡所经历的各个阶段的总和

生命周期分为3个阶段，分别是创建、运行、销毁

- 创建阶段：由空白期、data/methods初始化、模板挂载、模板渲染等组成
- 运行阶段：分为 更新前 和 更新后 两部分
- 销毁阶段：分为 销毁前 和 销毁后

成员方法：

各个阶段在Vue实例内部都有对应的成员方法，可以定义、执行、感知

创建：beforeCreate **created** beforeMount mounted

运行：beforeUpdate updated

销毁: beforeDestroy destroyed

为什么学习：

不同阶段完成不同的任务，开发者可以利用各个阶段的特点完成业务需要的相关功能

创建阶段分析

`目标`：

了解创建阶段各个方法特点，重点记住created

创建阶段一共有4个方法，它们与 el、data都是并列关系的

```
new Vue({
  beforeCreate(){   },
  created(){  },

  beforeMount(){   },
    mounted(){  },
})
```

> beforeCreate：此时Vue对象刚创建好，没有任何成员，data、methods等都没有呢，只有this
>
> **created**：此时vue对象已经长大一点，内部已经完成了data、methods等成员的设置，也是data初始化的最好时机
>
> beforeMount：此时vue实例已经把div容器给获得到了，但是内部的vue指令等信息还没有被编译处理
>
> mounted：此时，vue获取到的div容器内部的原生指令已经被编译处理好了，并且也完成了容器的渲染工作，此时模板中已经看不到vue原始指令了

重点关注方法是 created：

created: 一般用于页面"首屏"数据的获取操作(获取好的数据可以直接赋予给data使用，其可以做到**第1 时间**就把数据赋予给data，进而不影响后续使用)

`注意`：

创建阶段各个函数不设置则以，设置后就会**自动**执行，并且会**顺序**只执行**一次**

运行和销毁阶段分析

`目标`：

了解运行阶段和销毁阶段各个方法特点

`运行阶段`：

```
new Vue({
    beforeUpdate(){ 可以感知到数据变化之前页面上关于该数据的样子 }
    updated(){ 可以感知到数据变化之后页面上该数据的样子 }
})
```

> 运行阶段方法**不会**自动执行，当data成员数据发生变化，就执行了，并且数据变化多次，方法也会**重复**执行多次

销毁阶段：

```
new Vue({
    beforeDestroy(){ 实例销毁之前 }
    destroyed(){ 实例销毁之后 }
})
```

> 当vue实例被销毁后，就要执行以上两个方法，vm.$destroy()

注意：

1. **运行阶段**各个方法与创建阶段不同，本身**不会**自动执行，需要数据变化的条件触发才会执行
2. **销毁阶段**各个方法也不会自动执行，需要Vue实例对象调用$destroy()方法

完整应用示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <!--创建一个div容器，vue对该容器进行控制，设置要显示的内容-->
  <div id="app">
    <h2>{{ msg }}</h2>
  </div>

  <script src="./vue.js"></script>
  <script>
    var vm = new Vue({
      // 1) 生命周期创建阶段(4个函数),会自动执行
      beforeCreate(){
        // Vue实例已经创建完毕，但是相关的成员都没有，el、methods、data等等都没有
        console.group('--------beforeCreate发生调用--------')
        console.log('%c%s','color:red','el现在的样子: '+this.$el)     // undefined
        console.log('%c%s','color:red','data现在的样子: '+this.$data) // undefined
```

```
    console.log('%c%s','color:red','getDate现在的样子: '+this.getDate)  // undefined
  },
  created(){
    // 该函数是非常【重要】的，此时data 和 methods已经准备好了，但是还没有去找div容器
    // 此阶段可以用于页面“首屏”数据获取操作，可以第一时间把数据给到data
    console.group('--------created发生调用--------')
    console.log('%c%s','color:red','el现在的样子: '+this.$el)      // undefined
    console.log('%c%s','color:red','data现在的样子: '+this.$data)  // 实体
    console.log('%c%s','color:red','getDate现在的样子: '+this.getDate)  // 实体
  },
  beforeMount(){
    // 此阶段完成了Vue实例对象 与 div容器联系的过程(本质是div容器已经被Vue实例获取到了)
    // 但是div容器的内容还是没有编译前的原生内容
    console.group('--------beforeMount发生调用--------')
    console.log('%c%s','color:red','el现在的样子: '+this.$el)      // 实体
    console.log(document.getElementsByTagName('h2')[0])  //
  },
  mounted(){
    // 此阶段 Vue实例已经完成了div容器的内容的编译，并且编译好的内容也渲染给div容器了
    console.group('--------mounted发生调用--------')
    console.log('%c%s','color:red','el现在的样子: '+this.$el)      // 实体
    console.log(document.getElementsByTagName('h2')[0])  // 容器编译【后】实体内容
  },

  // 2) 生命周期运行阶段(2个函数),data数据变化后才会执行
  beforeUpdate() {
    console.group('---------beforeUpdate调用--------')
    console.log(
        '%c%s',
        'color:red',
        'h2数据更新【前】的效果: ' + document.querySelector('h2').innerHTML
    )
  },
  updated() {
    console.group('---------updated调用--------')
    console.log(
      '%c%s',
      'color:red',
      'h2数据更新【后】的效果: ' + document.querySelector('h2').innerHTML
    )
  },

  // 3) 生命周期销毁阶段(2个函数),只有vm调用$destroy()方法后才执行
  beforeDestroy() {
    console.group('---------beforeDestroy调用--------')
    console.log('%c%s', 'color:red', 'el现在的样子: ' + this.$el)
  },
  destroyed() {
    console.group('---------destroyed调用--------')
    console.log('%c%s', 'color:red', 'el现在的样子: ' + this.$el)
  },

  el: '#app',
```

```
      data: {
        msg: '生命周期学习篇'
      },
      methods: {
        getDate(){
          console.log('Sunday')
        }
      }
    })
  </script>
</body>
</html>
```

注意：

生命周期的各个方法与 el、data、methods 等成员都是并列的，它们有固定的执行顺序，与设置顺序没有关系

图示

生命周期参考

VirtualDOM

目标：

了解VirtualDOM是什么

什么是VirtualDOM：

div容器 在 Vue实例中存在的状态，就是 VirtualDOM(虚拟dom内容)，具体是内存信息的体现

在Vue实例运行期间，该VirtualDOM始终存在

VirtualDOM作用：

1. 编译解析div容器，并渲染给浏览器
2. 响应式体现

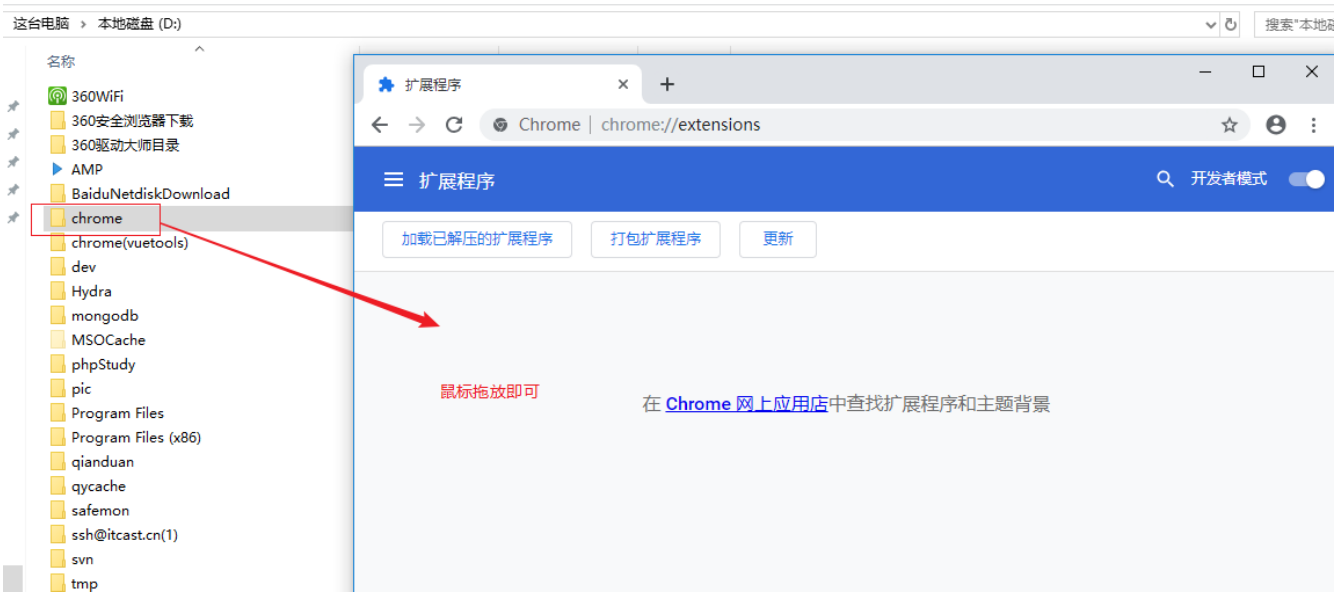
安装devtools工具

目标：

能够给chrome浏览器安装devtools调试工具

devtools是vue在chrome浏览器中的调试工具，方便Vue项目开发

安装devtools有两种方式： 1.通过翻墙软件在chrome浏览器的扩展程序中直接安装 2.在github上下载该工具并自行编译、安装配置



注意：

1. 只有vue开发的项目有调试效果

下午总结：

- 熟练使用**按键修饰符**
  - @keyup.enter/13="xxx"
  - @keyup.enter.ctrl="xxx"
- 了解**自定义指令**的创建和应用
  - v-dian
  - directives:{指令名称:{inserted:function(m){ m.focus }}} 私有
  - Vue.directive(名称, {inserted....})全局
- 掌握Vue生命周期用法
  - 3阶段：创建、运行、销毁
  - 创建：4个函数
    - beforeCreate
    - created：data和methods准备好了，用于首屏数据操作
    - beforeMount
    - mounted

作业：

1. 利用computed给 计算器 实现计算逻辑并输出结果
2. 完成品牌案例管理各个功能
    1. 完成品牌的 删除、筛选 功能
    2. 利用 **computed** 完善**筛选**品牌功能
    3. 利用 **私有过滤器** 完成时间格式化操作
    4. 利用 **按键修饰符** 完成单击**回车键**实现添加品牌功能、单击**esc键**完成清除品牌功能
    5. 利用 **自定义指令** 完成页面加载完成，使得添加品牌输入框**获得焦点**功能