Vue基本用法(二)

软通大学
ISOFTSTONE UNIVERSITY

本节目标

- 掌握vue过滤器的使用
- 掌握常用键盘修饰符的使用
- 掌握自定义vue指令
- 认识Vue生命周期，掌握其主要钩子函数的运行时机
- Vue-Resource发送网络请求
- 掌握过渡类名动画实现
- 掌握animate.css第三方类库实现动画
- 掌握列表动画实现
- 了解半场动画实现方式

目录 CONTENTS

01 vue过滤器

vue过滤器

Vue是允许开发者自定义过滤器的，通常它被用作一些常见的文本格式化。比如，以上案例中日期的文本格式化，数据的日期拿到之后，我们直接显示的效果可能并不是我们需要的，那么在显示之前，我们需要自定义处理一道，处理完之后的结果再被用作显示，所以叫做”过滤”。 过滤器可以用在两个地方：插值表达式或者v-bind表达式中，过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符指示, 理解为语法:

{{xxx | 过滤器}} 或者 **<p: v-bind=”title | 过滤器”></p>**

过滤器分2种，vue允许定义私有过滤器和全局过滤器。

- 私有过滤器：私有过滤器只能在绑定的vue实例所控制的区域中使用
- 全局过滤器：所有vm实例共享

vue过滤器

私有过滤器

定义vue私有过滤器，如同vm实例上的data、methods属性一样，定义filters属性，表示vm实例私有过滤器。

```
....
filters: {
    // data：要过滤的数据，即管道符 | 前面的数据, 后面的是参数列表
    dateFormat: function (data, pattern = "") {
        ....
        return “xxx”
    }
}
```

全局过滤器:

```
Vue.filter('dateFormat', function (data, pattern = "") {
    var dt = new Date(data);
    ...
    if (pattern.toLowerCase() === 'yyyy-mm-dd') {
        return `${y}-${m}-${d}`
    } else {
        return `${y}-${m}-${d} ${hh}:${mm}:${ss}`
    }
})
```

调用过滤器

```
<td>{{item.ctime | dateFormat("yyyy-mm-dd hh:mm:ss")}}</td>
```

*：当有局部和全局两个名称相同的过滤器时候，会以就近原则进行调用，即：局部过滤器优先于全局过滤器被调用！

02 vue键盘修饰符

vue键盘修饰符

在网页中，有的时候会有这种需求，监听键盘按键抬起响应事件，比如：上述例子中，当用户输入了id和name之后，不通过添加按钮添加一个品牌，而是name输入框按下enter键完成添加;

Vue.js官网中：https://cn.vuejs.org/v2/guide/events.html#按键修饰符， 对于按键修饰符有了说明

```
<label>
      Name:
      <!-- keyup监听键盘抬起事件:enter表示监听enter键抬起 -->
       <input type="text" class="form-control" v-model="name" @keyup.enter="add()">
</label>
```

*:Vue为键盘的常用按键设置了别名, 在keyup事件中，不需要记住这个键的keycode，直接使用别名就可以

.enter . tab
.delete .esc .space
.up .down .left . right

vue键盘修饰符

自定义键盘:如果默认没有我们想要监听的按键，也可以使用它对应的keycode或者为它自定义一个别名

- <!-- 113对应键盘F2 -->
  <input type="text" class="form-control" v-model="name" @keyup.113="add()">

- //自定义键盘修饰符
  Vue.config.keyCodes.f2=113
  //<!-- 113对应键盘F2 -->
  <input type="text" class="form-control" v-model="name" @keyup.f2="add()">

*: 注意的是，有些按键可能是会被系统响应的，比如浏览器的F1, Tab键，那么当定义这些按键的时候，事件会被捕获，也就是先响应系统的，即使是vue内置的键也是一样,所以当真有自定义键盘按键的时候，对于按键的选择就需要有一些甄别了

键盘key Code对照表: 自行百度即可

03 自定义指令

vue自定义指令

有的情况下，vue提供的指令不能实现需求，比如：当页面加载时，该元素自动获得焦点 ，Vue允许我们自定义指令，同样分为全局和私有

全局自定义指令： Vue.directive('指令名称',{}) , 比如: v-focus 指令: Vue.directive("focus",{...})

参数1:指令的名称，在定义的时候，指令的名称前面，不需要加 v- 前缀，在调用的时候，必须 在指令名称前 加上 v- 前缀来进行调用
参数2:是一个对象，这个对象身上，有一些指令相关的钩子函数，这些函数在特定的阶段执行, 要记住的是，每一个钩子函数的第一个参数表示绑定了这个指令的dom元素，**是一个原生js对象**

```
Vue.directive("focus", {
    bind:function(el){...},
    inserted:function(el){...},
    update: function(el){...},
    componentUpdated:function(el){...},
    unbind:function(el){...}
})
```

- **bind**:每当指令绑定到元素上的时候，会立即执行这个 bind 函数，只执行一次，一般用来做一些准备初始化工作
- **inserted**:表示元素 插入到DOM完成之后，会立即调用，执行一次
- **update**:当VNode更新的时候，会执行 update， 可能会触发多次，也就是绑定了Vue指令的dom元素发生改变时,这个改变包括很多，比如：样式、值、位置等等，只要这个元素和vue进行了数据绑定
- **componentUpdated**: 同update一样，当VNode更新时, 但在它的回调里面，el是改变后的dom
- **unbind**：元素从DOM删除时触发.(仅仅是删除)

vue自定义指令

bind与inserted钩子函数说明:

bind函数里调用focus是无效的，这个钩子函数表示指令被帮到元素上，但是还没有被插入到dom树中去，任何元素，之后再插入到dom之后，才能获取焦点.在实际运用中，通常我们用到最多的钩子函数只有bind和inserted, 一个是样式操作bind,一个是js操作inserted

结论:不需要操作dom，不需要等待dom完成插入，所以和样式相关的操作放在bind内, 要操作dom元素，必须dom完成插入后，所以和js相关的操作放在inserted内

案例: 自定义指令v-color,完成指定元素内容颜色的功能?

```
Vue.directive("color", {
   bind: function (el, binding) {
      el.style.color = binding.value
   }
})
// 这里要注意指令值的，因为style.color=’red’
<input type="text" class="form-control" v-model="keywords" id="search" v-focus  v-color="'green'">
```

vue自定义指令

私有自定义指令:

私有的就是在vm实例上添加directives属性即可。比如: 如下完成自定义指令指定元素字体粗细和字体大小?

```
directives: { // 自定义私有指令
    'fontweight': { // 设置字体粗细的
      bind: function (el, binding) {
        el.style.fontWeight = binding.value
      }
    },
    'fontsize': function (el, binding) { // 注意： 这个 function 等同于 把 代码写到了 bind 和 update 中去
      el.style.fontSize = parseInt(binding.value) + 'px'
    }
  }

  <p v-fontweight="800" v-fontsize="28" v-color="'yellow'">这是自定义的私有指令，改变字体大小和粗细</p>
```

04 vue生命周期

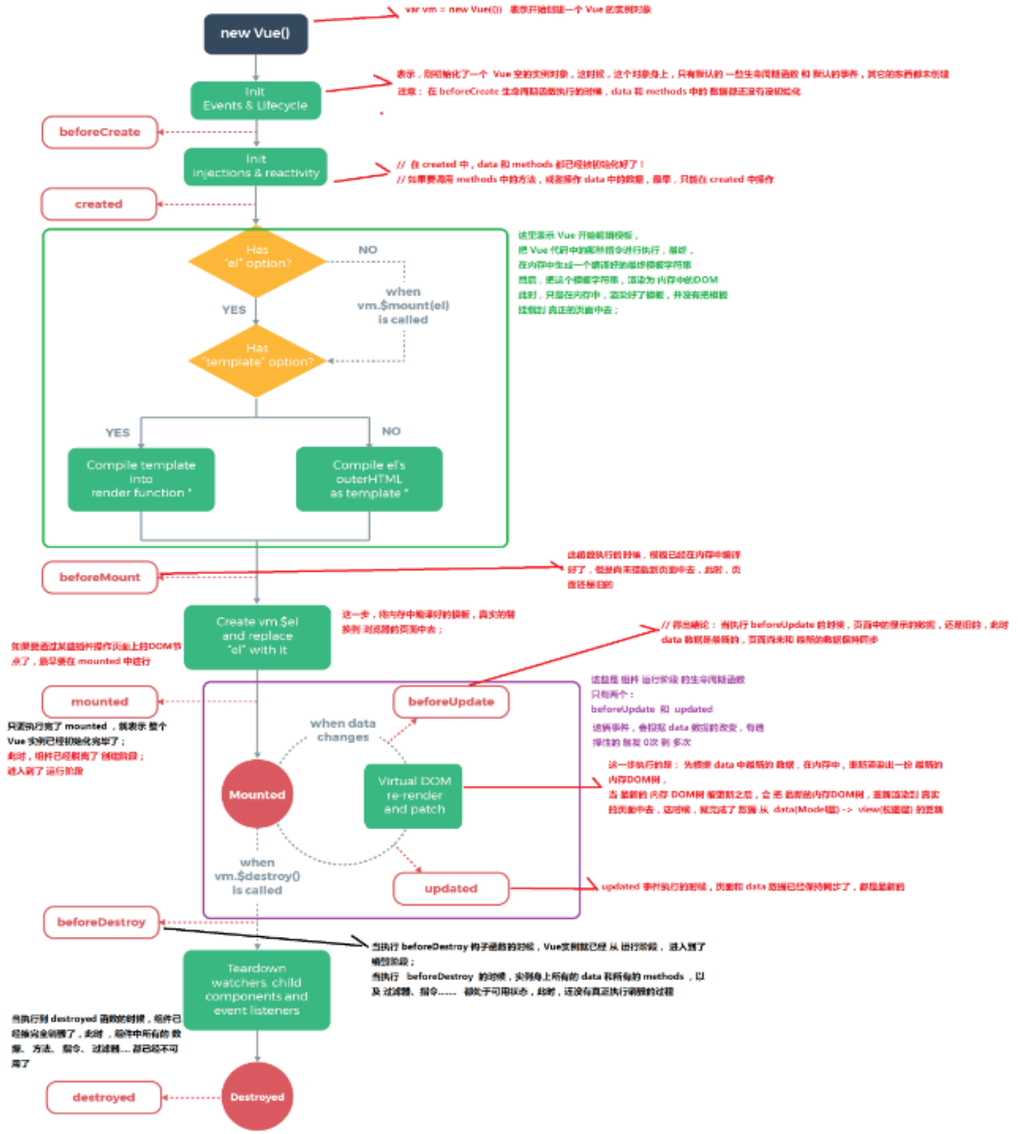
vue生命周期

生命周期:

每一个Vue实例从创建、运行、到销毁的一次完整过程，称之为生命周期，在这个过程中间，伴随着各种各样的事件，这些事件，称之为生命周期事件（生命周期函数、生命周期钩子），我们了解这些函数的时机，可以帮助我们在合适的位置去编写功能代码，比如：要获取数据的时候，应该在哪里发送请求?

Vue生命周期，最直观的方式就是官网的一张图：https://cn.vuejs.org/v2/guide/instance.html#生命周期图示

vue生命周期



beforeCreate: Vue实例创建前，此时data 和 methods 中的 数据都还没有没初始化

created：Vue实例已创建，data 和 methods 都已经被初始化好了

beforeMount: html模板已准备内存中,但未渲染到页面，此时无法进行dom操作

mounted: html模板已挂载到了页面中，此时可以进行dom操作

beforeUpdate:当data数据改变的时候执行的钩子函数,此时，页面中的显示的数据，还是旧的，此时 data 数据是最新的，页面尚未和 最新的数据保持同步

updated:updated 事件执行的时候，页面和 data 数据已经保持同步了，都是最新的

beforeDestory: vm实例被销毁前，此时data和methods等还可用

destoryed：vm实例已销毁, data和methods等所有属性都已经不可用。

vue生命周期

案例: 编写入口js文件，创建Vue实例，并定义生命周期图内的钩子函数，测试钩子函数的各个运行时机?

```
<body>
  <div id="app">
    <input type="button" value="修改msg" @click="msg='No'">
    <h3 id="h3">{{ msg }}</h3>
  </div>
</body>
```

main.js: 完成vue实例的各个生命周期函数测试?

05 vue-resource发送请求

vue-resource导入

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

Vue-Resource是提供了使用XMLHttpRequest或JSONP发送web请求和处理响应服务的一个Vue.js插件

引入Vue-Resource实现ajax请求开发步骤:

1. 安装vue-resource依赖:
   cnpm i vue-resource –S

2. 导入vue-reources,并调用Vue.use(vue-reources)方法使用vue-resource插件

```
import Vue from "vue"
//导入vue-reource
import  VueResource  from 'vue-resource'

//vue使用vue-reource插件
Vue.use(VueResource);
```

vue-resource发送请求

vue-resource发送网络请求，获取服务器数据：分别发送get、post、jsonp请求

```
<body>
  <div id="app">
    <input type="button" value="get请求" @click="getInfo">
    <input type="button" value="post请求" @click="postInfo">
    <input type="button" value="jsonp请求" @click="jsonpInfo">

    <!-- 展示get请求后的数据 -->
    <li v-for="(item,index) in items" :key="item.t">
      <p>{{item.name}}</p>
      <p><img :src="item.image0"><img :src="item.image1"><img :src="item.image2"></p>
      <p>{{item.text}}</p>
    </li>
  </div>
</body>
```

1-请求的服务器来自于网络免费api接口测试：https://www.jianshu.com/p/e6f072839282

2-html界面只展示get请求后的数据作为示范

get请求

this.$http:获取vue-resource的http请求对象

发送get请求:示例接口:随机推荐热门段子（包含文字、图片、GIF、视频）

```
this.$http.get('https://www.apiopen.top/satinApi?type=1&page=1').then(function (result) {
    // 通过 result.body 拿到服务器返回的成功的数据
    console.log(result.body)
    //分析数据结构
    this.items = this.items.concat(result.body.data);
})
```

post请求

this.$http:获取vue-resource的http请求对象

发送post请求:示例接口:小说搜索接口

```
this.$http.post('https://www.apiopen.top/novelSearchApi', {
        name: "鬼吹灯"
}, {
        emulateJSON: true
}).then(result => {
        console.log(result.body)
})
```

1: 发送post请求，默认没有表单格式的，要指定enctype: application/x-wwww-form-urlencoded, post方法的第三个参数：{**emulateJSON: true**}就是设置提交表单类型。

2：post方法的第二个参数，设置提交参数，eg: name:”鬼吹灯”表示搜索名字是鬼吹灯的小说

jsonp请求

jsonp请求: 跨域请求

由于浏览器的安全性限制，不允许AJAX访问 协议不同、域名不同、端口号不同的 数据接口，浏览器认为这种访问不安全，实现跨域请求的原理是通过动态创建script标签的形式，把script标签的src属性，指向数据接口的地址，因为script标签不存在跨域限制，这种数据获取方式，称作JSONP

vue-resource支持直接发送jsonp请求：示例接口:360搜索

```
this.$http.jsonp('https://sug.so.360.cn/suggest',{
    //根据接口规则传递参数
    params:{word:'a',encodeout:'utf-8'}
}).then(result => {
    console.log(result.body)
})
```

在同一个html中如果用get请求发送360搜索，是行不通的，因为访问了一个不同的服务器，浏览器是不支持的，所以直接发送jsonp请求

Vue-resource设置全局http参数

在真实项目中，我们所请求的接口地址都是有接口文档的，是有一定规范，比如: 接口的请求根路径都是相同，默认ajax传输格式也都是json

Vue使用vue-resource后，配置全局http参数, 可以减少发送请求的一些重复操作

```
//设置全局http参数
Vue.http.options.root="https://www.apiopen.top/"
Vue.http.options.emulateJSON = true
```

发送请求时，就可以省略主机路径部分，发送post请求时，也不用单独设置参数了:

```
//url地址一定不能带/
this.$http.get('satinApi?type=1&page=1').then(function (result) {
    console.log(result.body)
    this.items = this.items.concat(result.body.data);
})
this.$http.post('novelSearchApi', {name: "鬼吹灯"}).then(result => {
        console.log(result.body)
})
```

1: 请求的url的路径相同部分可以省略了.
2: 发送post请求时候，第三个表单参数类型可以省略了。

06 Vue动画

Vue动画

Vue动画能够提高用户的体验，

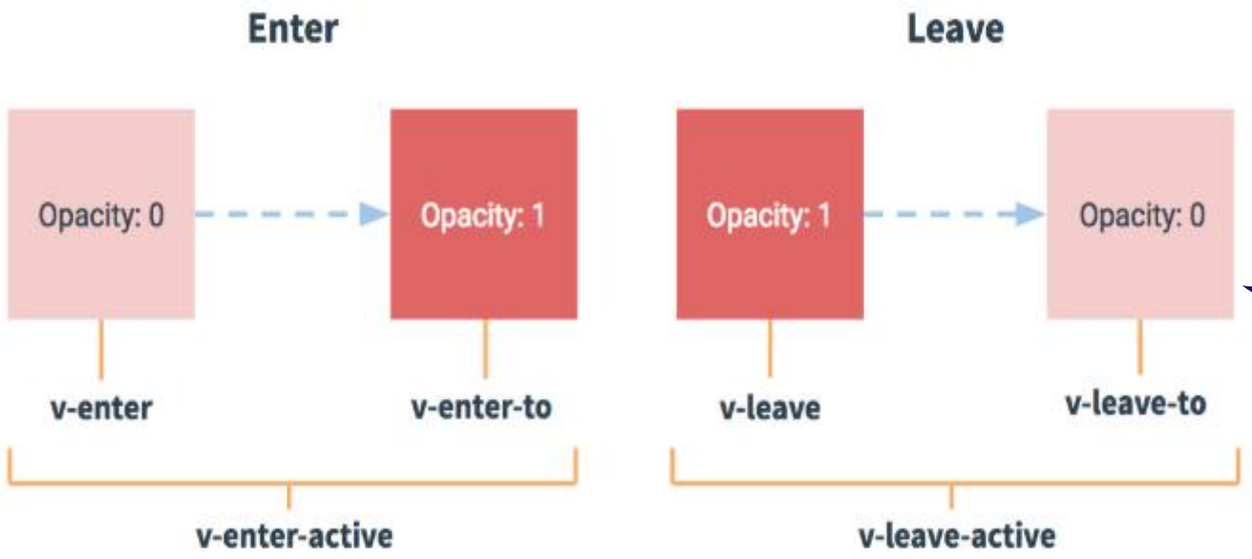
https://cn.vuejs.org/v2/guide/transitions.html 这里提供了官方示例，

结合使用场景和动画实现方式，主要有以下几种动画的实现方式：

- 过渡类名实现动画
- animate.css第三方类实现动画
- 钩子函数实现半场动画
- 列表中的动画

过渡类名实现动画

所谓过渡就是元素出现和消失有一个缓慢的过程，为这个过程设置变化效果，给了用户动画的直观感受。官方有一张图，有了不错的说明



enter代表进场，leave代表离场，如图上所描述，元素进场开始和结束的透明度从0-1； 元素离场开始和结束的透明度从1-0;

过渡分为入场和离场，入场有入场开始和入场结束，离场有离场开始和离场结束，而所谓动画就是这开始和结束中间的过程变化

过渡类名实现动画

过渡类名实现动画：

1：将要实现动画的元素用**transition**标签包裹

2：定义入场和离场的2组6个css

```
<body>
  <div id="app">
    <!-- 点击按钮，让 h3 显示，再点击，让 h3 隐藏 -->
    <input type="button" value="toggle" @click="flag=!flag">
    <!-- transition包裹的元素表示要应用过渡动画的元素 -->
    <transition>
      <h3 v-if="flag">这是一个H3</h3>
    </transition>
  </div>
</body>
```

(transition包裹元素)

(6个css)

```
<style>
  /* 入场 */
  .v-enter{
    opacity: 0;
    /* 一定要带px单位 */
    transform: translateX(150px);
  }
  .v-enter-to{ opacity: 1; transform: translateX(0px);}
  .v-enter-active{
    /* 入场过滤效果 */
    transition: all 0.8s ease;
  }

  /* 离场 */
  .v-leave{ opacity: 1; transform: translateX(0px);}
  .v-leave-to{ opacity: 0; transform: translateX(150px);}
  .v-leave-active{
    /* 离场过渡效果 */
    transition: all 0.8s ease;
  }
</style>
```

animate.css第三方类实现动画

https://daneden.github.io/animate.css/，查看支持的所有动画

1. 安装animate.css

   cnpm i animate.css -S

2. import导入animate.css,并调用Vue.use()全局使用

```
import Vue from "vue"

//引入animate.css
import animated from "animate.css"

//Vue使用动画库
Vue.use(animated);
```

```
<div id="app">
<input type="button" value="toggle"
@click="flag=!flag">
    <!-- enter-active-class:指定入场效果 -->
    <!-- leave-active-class:指定离场效果 -->
    <!-- duration：分别指定入场和离场效果的持续时长,毫秒为单位 -->
    <transition enter-active-class="animated bounceIn" leave-active-class=" animated bounceOut"
      :duration="{ enter: 200, leave: 400 }">
      <h3 v-if="flag">这是一个H3</h3>
    </transition>
</div>
```

3. 动画元素用transition包括，并分别指定enter-active-class和leave-active-class入场和离场动画

钩子函数实现半场动画

钩子函数一共有8个，分为入场4个和离场4个，只有在定义半场动画的时候，也就是只有入场的时候需要动画，才会使用钩子函数。因为如果都需要，不是半场动画，直接使用过渡函数实现动画即可

```
<transition
 //入场钩子
 v-on:before-enter="beforeEnter"
 v-on:enter="enter"
 v-on:after-enter="afterEnter"
 v-on:enter-cancelled="enterCancelled"

 //离场钩子
 v-on:before-leave="beforeLeave"
 v-on:leave="leave"
 v-on:after-leave="afterLeave"
 v-on:leave-cancelled="leaveCancelled"
>
 <!-- ... -->
</transition>
```

案例：购物车小球半场动画 ？

```
<div id="app">
    <input type="button" value="快去购物车" @click="flag=!flag">
    <transition
      @before-enter="beforeEnter"
      @enter="enter"
      @after-enter="afterEnter">
       <div v-cloak class="ball" v-show="flag"></div>
    </transition>
 </div>
```

*：半场动画只定义了入场的是三个钩子函数，对应vm实例的methods对象的三个函数

钩子函数实现半场动画

```
methods: {
    //el，表示 要执行动画的那个DOM元素，是个原生的 JS DOM对象
    //表示动画入场之前，此时，动画尚未开始，可以 在 beforeEnter 中，设置元素开始动画之前的起始样式
    beforeEnter(el) {
      el.style.transform = "translate(0px, 0px)"
    },
    //表示动画开始之后的样式，这里，可以设置小球完成动画之后的，结束状态
    enter(el, done) {
      //必须el调用一个offset值方法，否则过渡效果不会出现
      el.offsetWidth;
      el.style.transform = "translate(150px, 450px)"
      el.style.transition = 'all 1s ease'
      //enter中必须调用done():表示动画完成，相当于直接调用afterEnter(),否则小球会静止
      done();
    },
    //动画完成之后
    afterEnter() {
      this.flag = !this.flag
    }
  }
```

*:官网中有这么一句话, 当只用 JavaScript 过渡的时候，**在 enter 和 leave 中必须使用 done 进行回调**。否则，它们将被同步调用，过渡会立即完成。

列表动画

使用v-for指令动态渲染列表元素，列表元素在添加或者删除的时候，添加动画效果，实现动态列表过渡，不能使用transition 包裹，需要使用 **transitionGroup**，并同时定义过渡的2组6个css即可添加

```
<!-- tag标签指定transition-group渲染为指定标签元素，默认会渲染成span -->
<!-- appear属性: 列表在第一次场的时候也能应用上过渡动画-->
<transition-group tag="ul" appear>
    <li v-for="(item, i) in list" :key="item.id" @click="del(i)">
        {{item.id}} --- {{item.name}}
    </li>
</transition-group>
```

```
.v-enter {pacity: 0; transform: translateY(80px);}
.v-enter-to {opacity: 1; transform: translateY(0px);}
.v-enter-active {transition: all 0.8s ease}
.v-leave {opacity: 1; transform: translateY(0px);}
.v-leave-to {opacity: 0; transform: translateY(80px);}
.v-leave-active {transition: all 0.8s ease}
```

从下方80px的位置向上浮动上来，但是这样会有一个问题，在删除中间元素的时候，后面的元素整体并没有一个过渡效果?

*:默认删除中间列表元素，是没有整体动画效果的，为列表额外添加2个css

v-move 和 v-leave-active,在删除中间元素的时候，后面的列表元素移动到新的位置上也有过渡效果。

```
.v-move {
    transition: all0.6s ease;
}
.v-leave-active {
    position: absolute;
}
```

本节总结

- Vue过滤器实现日期格式化
- Vue键盘修饰符、自定义键盘修饰符
- Vue自定义指令的定义
- Vue自定义指令的bind和inserted钩子函数
- Vue生命周期
- Vue生命周期函数
- Vue-resource导入
- Vue-resource发送网络请求，获取服务器数据
- Vue动画的4种实现方式

软通大学
ISOFTSTONE UNIVERSITY