## Vue简介：

* JavaScript框架
* 简化Dom操作
* 响应式数据驱动

## 第一个vue程序

* 导入开发版本的Vue.js
* 创建Vue实例对象, 设置el属性和data属性
* 使用简洁的模板语法把数据渲染到页面上

  <div id="app">

        {{message}}

    </div>

    <script>

        var vue=new Vue({

            el:"#app",

            data:{

                message:"你好vue！"

            }

        })

    </script>

Vue实例的作用范围是什么呢？

* Vue会管理el选项命中的元素及其内部的后代元素

是否可以使用其他的选择器？

* 可以使用其他的选择器,但是建议使用ID选择器

是否可以设置其他的dom元素呢？

* 可以使用其他的双标签,不能使用HTML和BODY

## el:挂载点

* el是用来设置Vue实例挂载（管理）的元素
* Vue会管理el选项命中的元素及其内部的后代元素
* 可以使用其他的选择器,但是建议使用ID选择器
* 可以使用其他的双标签,不能使用HTML和BODY

    <div id="app">

        <p>{{message}}</p>

        <p>{{school.name}}</p>

        <p>{{campus[0]}}</p>

    </div>

    <script>

        var vue=new Vue({

            el:"#app",

            data:{

                message:"你好vue！",

                school:{

                    name:"黑马程序员",

                    mobile:"400-000-000"

                },

                campus:[

                    "北京校区","上海校区","广州校区"

                ]

            }

        })

    </script>

## data:数据对象

* Vue中用到的数据定义在data中
* data中可以写复杂类型的数据
* 渲染复杂类型数据时,遵守js的语法即可

## v-text指令

* v-text指令的作用是:设置标签的内容(textContent)
* 默认写法会替换全部内容,使用差值表达式{{}}可以替换指定内容
* 内部支持写表达式

    <div id="app">

        <p v-text="message"></p>

        <p v-text="message">北京</p>

        <p v-text="message+'!'"></p>

        <p> {{message}}

        </p>

        <p> {{message}}北京

        </p>

        <p> {{message+'!'}}北京

        </p>

    </div>

    <script>

        var vue = new Vue({

            el: "#app",

            data: {

                message: "你好vue"

            }

        })

    </script>

## v-html指令

* v-html指令的作用是:设置元素的innerHTML
* 内容中有html结构会被解析为标签
* v-text指令无论内容是什么,只会解析为文本
* 解析文本使用v-text,需要解析html结构使用v-html

  <div id="app">

        <p v-text="message"></p>

        <p v-html="message"></p>

        <p v-text="content"></p>

        <p v-html="content"></p>

    </div>

    <script>

        var vue = new Vue({

            el: "#app",

            data: {

                message: "你好vue",

                content:"<a href='http://www.baidu.com'>百度一下</a>"

            }

        })

    </script>

## v-on指令

* v-on指令的作用是:为元素绑定事件
* 事件名不需要写on
* 指令可以简写为@
* 绑定的方法定义在methods属性中
* 方法内部通过this关键字可以访问定义在data中数据
* 注意：双击事件不是dbclick 是dblclick

      <div id="app">

        <input type="button" value="点击事件" v-on:click="dolt">

        <input type="button" value="双击事件" @dblclick="dolt2">

    </div>

    <script>

        var vue = new Vue({

            el: "#app",

            methods: {

                dolt: function () {

                    console.log('点击事件');

                },

                dolt2: function () {

                    console.log('双击事件');

                }

            }

        })

    </script>

    <div id="app">

        <p @click="change">{{food}}</p>

    </div>

    <script>

        var vue = new Vue({

            el: "#app",

            data: {

                food: "西红柿鸡蛋"

            },

            methods: {

                change: function () {

                    this.food += "好好吃"

                }

            }

        })

    </script>

## 小结

* 创建Vue示例时:el(挂载点),data(数据),methods(方法)
* v-on指令的作用是绑定事件,简写为@
* 方法中通过this,关键字获取data中的数据
* v-text指令的作用是:设置元素的文本值,简写为{{}}
* v-html指令的作用是:设置元素的innerHTML

计时器 +-

    <div id="input\_sum">

        <button @click="sub">-</button>

        <span>{{num}}</span>

        <button @click="add">+</button>

    </div>

    <script>

        var vue = new Vue({

            el: "#input\_sum",

            data: {

                num: 0

            },

            methods: {

                add: function () {

                    this.num += 1

                },

                sub: function () {

                    this.num  -= 1

                }

            }

        })

    </script>

加上数值条件

        var vue = new Vue({

            el: "#input\_sum",

            data: {

                num: 0

            },

            methods: {

                add: function () {

                    if (this.num >= 0 && this.num < 10) {

                        this.num += 1

                    }

                    else console.log("别点了，数据上限为10！！！");

                },

                sub: function () {

                    if (this.num > 0 && this.num <= 10) {

                        this.num -= 1

                    }

                    else console.log("别点了，数据下限为0！！！");

                }

            }

        })

    </script>

## v-show指令

* v-show指令的作用是:根据真假切换元素的显示状态
* 原理是修改元素的display,实现显示隐藏
* 指令后面的内容,最终都会解析为布尔值
* 值为true元素显示,值为false元素隐藏
* 数据改变之后,对应元素的显示状态会同步更新

    <div id="app">

        <div>

            <p v-show="true">第一段文字</p>

        </div>

        <div>

            <p v-show="isShow">第二段文字</p>

        </div>

        <div>

            <p v-show="age>18">第三段文字</p>

        </div>

        <button @click="changeShow">第二段取反</button>

    </div>

    <script>

        var vue = new Vue({

            el: "#app",

            data: {

                isShow: false,

                age: 19

            },

            methods: {

                changeShow: function () {

                    this.isShow = !this.isShow;

                }

            }

        })

    </script>

## v-if指令

* v-if指令的作用是:根据表达式的真假切换元素的显示状态
* 本质是通过操纵dom元素来切换显示状态
* 表达式的值为true,元素存在于dom树中,为false,从dom树中移除
* 频繁的切换v-show,反之使用v-if,前者的切换消耗小

    <div id="app">

        <p v-if="isShow"> v-if 根据表达值的真假,切换元素的显示和隐藏(操纵dom元素)</p>

        <button @click="changeShow">切换show</button>

    </div>

    <script>

        var app = new Vue({

            el: "#app",

            data: {

                isShow: false

            },

            methods: {

                changeShow: function () {

                    this.isShow = !this.isShow;

                }

            }

        })

    </script>

## v-bind指令

* v-bind指令的作用是:为元素绑定属性 (比如:src,title,class)
* 完整写法是 v-bind:属性名
* 简写的话可以直接省略v-bind,只保留 :属性名
* 需要动态的增删class建议使用对象的方式，class加{} 如**:class="{active:isActive**}"

<div id="app">

        <img v-bind:src="imgSrc">

        <img :src="imgSrc" :title="imgTitle+'!'">

        <img :src="imgSrc" :class="isActive?'active':''">

        <img :src="imgSrc" :class="{active:isActive}">

    </div>

    <script>

        var app = new Vue({

            el: "#app",

            data: {

                imgSrc: "http://img0.bdstatic.com/static/searchresult/img/logo-2X\_32a8193.png",

                imgTitle: "百度图片",

                isActive: true

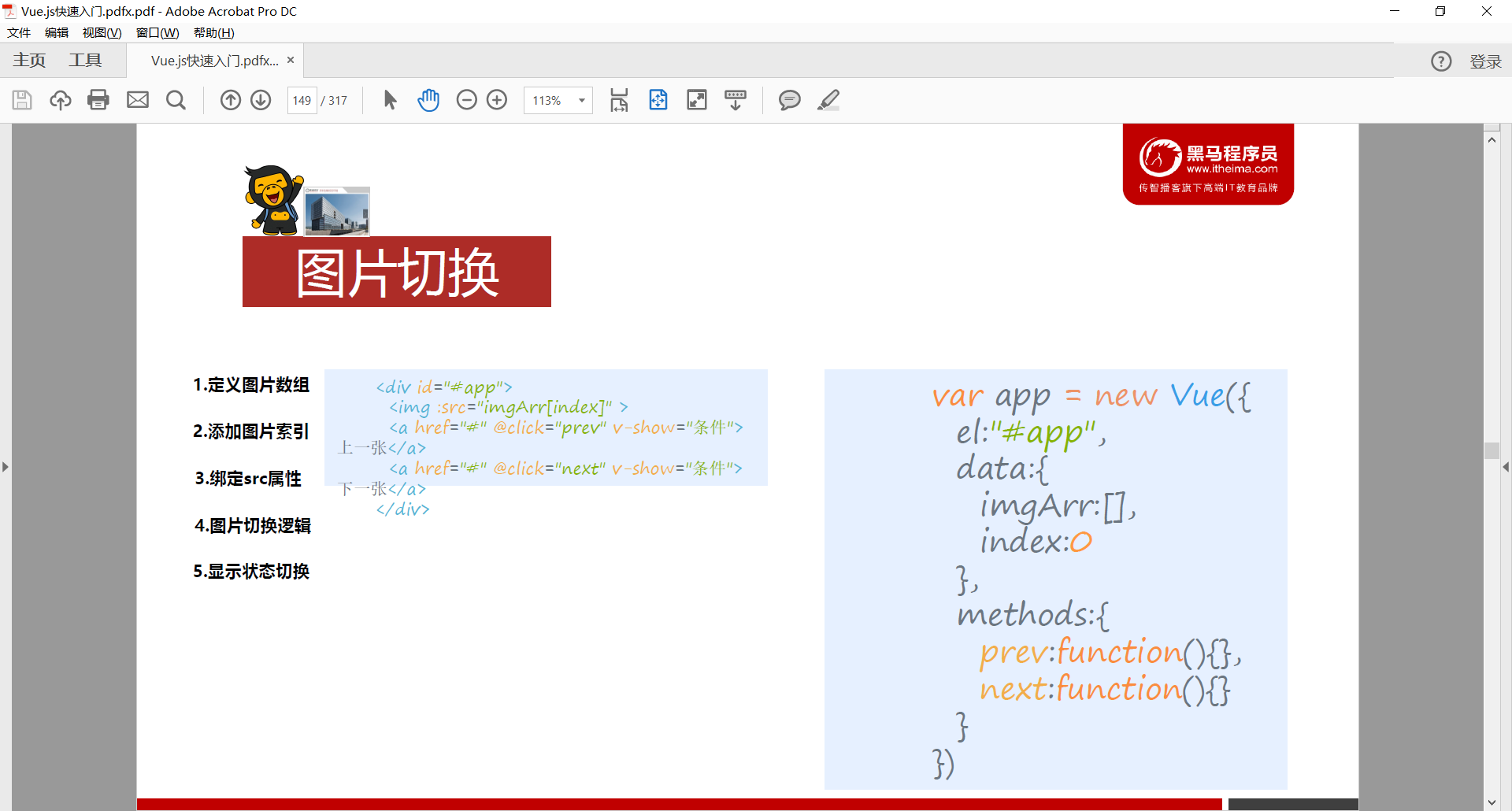
            }

        })

    </script>

Demo 图片切换





* 列表数据使用数组保存
* v-bind指令可以设置元素属性,比如src
* v-show和v-if都可以切换元素的显示状态,频繁切换用v-show

## v-for 根据数据生成列表结构

* v-for指令的作用是:根据数据生成列表结构
* 数组经常和v-for结合使用
* 语法是( item,index ) in 数据
* item 和 index 可以结合其他指令一起使用
* 数组长度的更新会同步到页面上,是响应式的

  <div id="app">

        <input type="button" value="添加数据" @click="add">

        <input type="button" value="移除数据" @click="remove">

        <ul>

            <li v-for="(it,index) in arr">

                {{ index+1 }}黑马程序员校区:{{ it }}

            </li>

        </ul>

        <h2 v-for="item in vegetables" v-bind:title="item.name">

            {{ item.name }}

        </h2>

    </div>

    <script>

        var app = new Vue({

            el: "#app",

            data: {

                arr: ["北京", "上海", "广州", "深圳"],

                vegetables: [

                    { name: "西兰花炒蛋" },

                    { name: "蛋炒西蓝花" }

                ]

            },

            methods: {

                add: function () {

                    this.vegetables.push({ name: "花菜炒蛋" });

                },

                remove: function () {

                    this.vegetables.shift();

                }

            },

        })

    </script>

## v-on传递自定义参数，时间修饰符

* 事件绑定的方法写成函数调用的形式，可以传入自定义参数
* 定义方法时需要定义形参来接收传入的实参
* 事件的后面跟上 .修饰符 可以对事件进行限制
* .enter 可以限制触发的按键为回车
* 事件修饰符有多种

<div id="app">

        <input type="button" value="点我按钮" @click="doit">

        <input type="text" value="点我" @keyup.enter="say">

    </div>

    <script>

        var app=new Vue({

            el:"#app",

            methods:{

                doit:function(){

                    console.log("写代码");

                },

                say:function(){

                    console.log(666);

                }

            }

        })

    </script>

## v-model 获取和设置表单元素的值(双向数据绑定)

* v-model指令的作用是便捷的设置和获取表单元素的值
* 绑定的数据会和表单元素值相关联
* 绑定的数据←→表单元素的值

    <div id="app">

        <input type="text" v-model="message" @keyup.enter="getM">

        <h2>{{message}}</h2>

        <button @click="setM">初始化表单</button>

    </div>

    <script>

        var app=new Vue({

            el:"#app",

            data:{

                message:"你好vue!"

            },

            methods:{

                getM:function(){

                    console.log(this.message);

                },

                setM:function(){

                    this.message="欢迎学习Vue！";

                }

            }

        })

    </script>

Demo 记事本



    <div id="app">

        <!-- 增加功能 -->

        <input type="text" v-model="inputValue" @keyup.enter="addM">

        <h2>{{inputValue}}</h2>

        <ul>

            <li v-for="(item,index) in message">

                <span>{{index+1}}</span>

                {{item}}

                <!-- 删除功能 -->

                <button @click="deleteM(index)">x</button></li>

        </ul>

        <!-- 统计个数 -->

        <!-- 隐藏 -->

        <span v-if="message.length!=0">{{message.length}}

        </span>

        <span>

            <!-- 清空 -->

            <!-- 隐藏 -->

            <button @click="clear" v-if="message.length!=0">情空</button>

        </span>

    </div>

    <script>

        var app = new Vue({

            el: "#app",

            data: {

                message: [

                    "吃饭", "睡觉", "学习"

                ],

                inputValue: "好好学习天天向上",

            },

            methods: {

                addM: function () {

                    this.message.push(this.inputValue);

                },

                deleteM: function (index) {

                    console.log(index);

                    this.message.splice(index, 1);

                },

                clear: function () {

                    this.message = [];

                }

            }

        })

    </script>