第三章 UNIAPP与VUE（四）

纲要

1. 模板语法
2. 响应式基础
3. 计算属性
4. 类与样式的绑定
5. 事件处理
6. 表单输入绑定
7. 生命周期钩子
8. 监听器
9. 模板引用

Go,Vue

Vue官网>>文档>>深度指南 的继续学习

- https://cn.vuejs.org/guide/introduction.html

一、模板语法

- 文本插值
- 属性绑定
- 指令

1.文本插值

1. 文本插值：最基本的数据绑定形式，使用的是“Mustache”语法 (即双大括号)

```
<text>Message: {{ msg }}</ text >
```

2.属性绑定

- 响应式地绑定一个 attribute： v-bind 指令 或简写为 :属性

< text v-bind:id="dynamicId"></ text > < text :id="dynamicId"></

<button :disabled="isButtonDisabled">Button</button>

<!--属性值为布尔值： 当 isButtonDisabled 为真值或一个空字符串 (即 <button disabled=""">) 时，元素会包含这个 disabled attribute。而当其为其他假值时 attribute 将被忽略。 -->

- 动态绑定多个属性值（不带参数）
  - （示例如右）

```
data() {
  return {
    objectOfAttrs: {
      id: 'container',
      class: 'wrapper'}}}

<div v-bind="objectOfAttrs"></div>
```

绑定JavaScript 表达式

- Vue 在所有的数据绑定中都支持完整的JavaScript 表达式，

  模板内JavaScript 表达式可以被使用于：

  - 在文本插值中 (双大括号)
  - 在任何 Vue 指令 (以 v- 开头的特殊属性) 的值中

```
<time :title="toTitleDate(date)" :datetime="date">
  {{ formatDate(date) }}
</time>
```

- 每个绑定仅支持**单一表达式**，即**一段能够被求值的 JavaScript 代码**。
- 函数调用：在绑定的表达式中使用一个“组件暴露的方法”。（如右上）
  - 可以使用常用的内置全局对象，比如 Math 和 Date。

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
<div :id="`list-${id}`"></div>
```

以下例子不是“单一表达式”

```
{{ var a = 1 }} <!-- 这是一个语句，而非表达式 -->
<!-- 条件控制也不支持，除非三元表达式 -->
{{ if (ok) { return message } }}
```

3.指令 Directives

- 指令是**带有 v- 前缀的特殊 attribute**。Vue 提供了许多内置指令
- 指令 attribute 的期望值一般是一个 JavaScript 表达式 。(v-for、v-on 和 v-slot几个例外)。一个指令的任务是**在其表达式的值变化时响应式地更新 DOM**。

  eg.　　Now you see me

- 某些指令会需要一个“**参数**”，在指令名后通过一个冒号隔开做标识

<a v-bind:href=“url”> ... 　：href 是一个参数，它告诉 v-bind 指令将表达式 url 的值绑定到元素的 href 属性上 。

- **v-on 指令**将监听 DOM 事件，其参数是要监听的事件名称（下例是click事件）。v-on 缩写是 @ 字符

<a v-on:click="doSomething"> ...

<a @click="doSomething"> ... 　<!-- 简写 -->

指令的动态参数与修饰符

- 指令的动态参数，即使用JavaScript 表达式，需要包含在一对方括号内：

```
<a :[attributeName]="url"> ... </a>
```

- 还可以将一个函数绑定到动态的事件名称上：

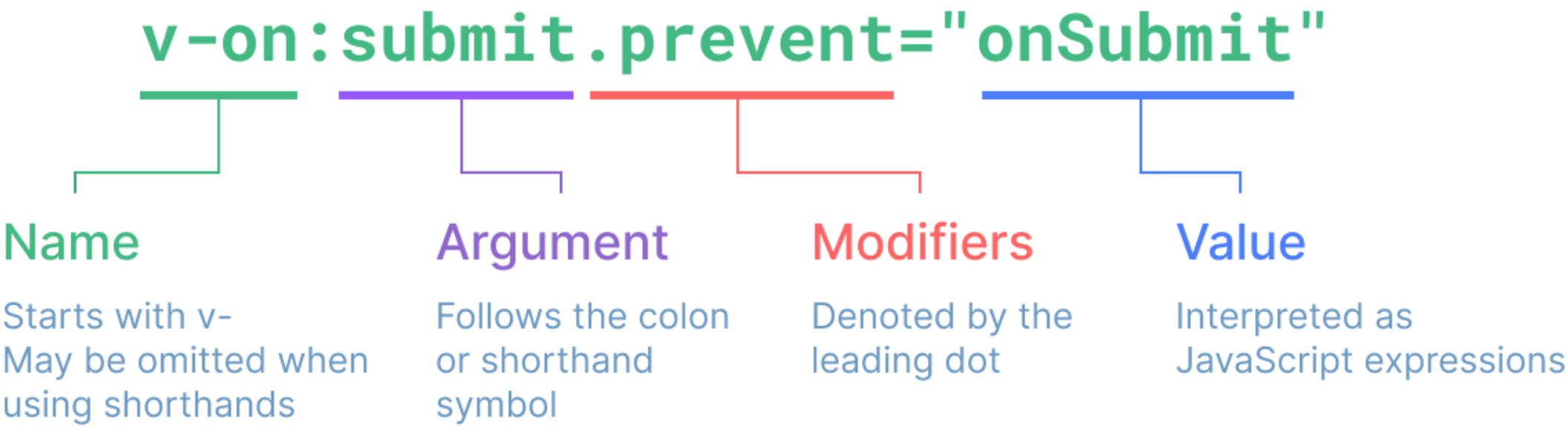
```
<a @[eventName]="doSomething">
```

- 指令的修饰符：以点开头的特殊后缀，表明指令需要以一些特殊的方式被绑定。
  - eg.　prevent 修饰符会告知 v-on 指令对触发的事件调用 event.preventDefault()：

```
<form @submit.prevent="onSubmit">...  </form>
```

完整的指令格式

- 以点开头的特殊后缀，表明指令需要以一些特殊的方式被绑定。



二、响应式基础

- 声明 响应式状态
- 声明 方法

1.声明响应式状态

- 选用选项式 API 时，会用 data 选项来声明组件的响应式状态。
- Data是“返回一个对象”的函数。 例如： chatList:[], myinput:””
- Vue 将在创建新组件实例的时候调用此函数，并将函数返回的对象用响应式系统进行包装。此对象的所有顶层属性都会被代理到组件实例 (即方法和生命周期钩子中的 this) 上。
- 实例上的属性仅在实例首次创建时被添加，若所需的值还未准备好，在必要时也可以使用 null、undefined 或者其他一些值占位。虽然也可以不在 data 上定义，直接向组件实例添加新属性，但这个属性将无法触发响应式更新。
- 确保始终通过 this 来访问响应式状态。
- Vue 在组件实例上暴露的内置 API 使用 $ 作为前缀，内部属性保留 _ 前缀。因此，应该避免在顶层 data 上使用任何以这些字符作前缀的属性。

声明“方法”

- 为组件添加方法，需要用到 methods 选,项它是一个包含所有方法的对象
- Vue 自动为 methods 中的方法绑定了永远指向组件实例的 this，以确保方法在作为事件监听器或回调函数时始终保持正确的 this。定义 methods 时不能使用箭头函数，因为箭头函数没有自己的 this 上下文。
- 调用方法
  - 在其他方法或是生命周期中调用方法
  - 可以在模板上被访问（此时常常被用作事件监听器）

```
export default {
  data() {
      return {     count: 0    }
  },
  methods: {
      increment() {     this.count++     }
  },
  mounted() { this.increment()   }
}
...
<button @click="increment"> {{ count }}
</button>
```

```
data() {
    return {
      obj: {
        nested: { count: 0 },
        arr: ['foo', 'bar']
...
methods: {
    mutateDeeply() {
     this.obj.nested.count++
     this.obj.arr.push('baz')
...
```

三、计算属性

- 计算属性用来描述依赖响应式状态的复杂逻辑，以减轻模板压力并**提高复用性**

- 定义为一个方法method还是计算属性computed，两种方式在结果上是完全相同的。不同之处：

  - 计算属性值会**基于其响应式依赖被缓存**：一个计算属性仅会在其响应式依赖更新时才重新计算。（右例中，只要 author.books 不改变，无论多少次访问 publishedBooksMessage 都会立即返回先前的计算结果，而不用重复执行 getter 函数。）
  - 方法调用总是会在重渲染发生时再次执行函数。

之前是直接写在模板里：

```
<span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
```

用计算属性后：

```
export default {
  data() {…},
  computed: {
    publishedBooksMessage() {// 一个计算属性的 getter
    return this.author.books.length > 0 ? 'Yes' : 'No'   }
  }
  …
```

```
<span>{{ publishedBooksMessage }}</span>
```

修改计算属性

- 尝试修改一个计算属性时，你会收到一个运行时警告。只在某些特殊场景中你可能才需要用到“可写”的属性，你可以通过同时提供 getter 和 setter 来创建
- 每一个计算属性都包含一个 getter 和一个 setter ，默认是利用 getter 来读取。所有 getter 和 setter 的 this 上下文自动地绑定为 Vue 实例。
- 右例中，当运行 this.fullName = 'John Doe' 时，setter 会被调用而 this.firstName 和 this.lastName 会随之更新。

```
export default {
  data() {
    return {
      firstName: 'John',
      lastName: 'Doe'
    }
  },
  computed: {
    fullName: {
      get() // getter
      { return this.firstName + ' ' +
this.lastName },

     set(newValue) {// setter
        [this.firstName, this.lastName] =
newValue.split(' ') // 解构赋值语法
      }}}}
```

- 不要在 getter 中做异步请求或者更改 DOM。一个计算属性的声明中描述的是如何根据其他值派生一个值。因此 getter 的职责应该仅为计算和返回该值。
- 从计算属性返回的值是派生状态。可以把它看作是一个“临时快照”，每当源状态发生变化时，就会创建一个新的快照。更改快照是没有意义的，因此计算属性的返回值应该被视为只读的，并且永远不应该被更改——应该更新它所依赖的源状态以触发新的计算。

四、绑定class与style

- class 和 style 都是 attribute，可以和其他 attribute 一样使用 v-bind 将它们和动态的字符串绑定。

- **绑定class**

  :class (v-bind:class 缩写) 传递一个对象来动态切换 class。:class 指令也可以和一般的 class attribute 共存

```
data() {
  return {
    isActive: true,
    hasError: false } }  ……
<div class="static" :class="{ active: isActive, 'text-danger': hasError }" ></div>
渲染出的结果是:
<div class="static active"></div>
```

- 除了绑定为内联字面量的形式，也可以直接绑定一个对象,或是一个返回对象的计算属性、一个数组

```
data() {return {
    isActive: true,
    error: null  } },
computed: {
  classObject() {
    return {
      active: this.isActive && !this.error,
      'text-danger': this.error && this.error.type ===
'fatal'  } }}
template
<div :class="classObject"></div>  绑定一个计算属性
```

```
data() {
  return {
    classObject: {
      active: true,
      'text-danger': false
    }
  }
}
template //绑定一个对象
<div :class="classObject"></div>
```

```
data() {
  return {
    activeClass: 'active',
    errorClass: 'text-danger'
  }
}
template  //绑定数组
<div :class="[activeClass, errorClass]"></div>
```

绑定Style

```
//绑定内联样式
<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

//绑定数组
<div :style="[baseStyles, overridingStyles]"></div>

//直接绑定一个对象
data() {
  return {
    styleObject: {
      color: 'red',
      fontSize: '13px'
    }
  }
}
Template
<div :style="styleObject"></div>
```

五、事件监听v-on 指令 @

- v-on 指令 (简写为 @) 监听 DOM 事件，并在事件触发时执行对应的 JavaScript。
  - 如：v-on:click="methodName" 或 @click="handler"。
- 事件处理器的值可以是：
  - **内联事件**处理器：事件被触发时执行的内联 JavaScript 语句 (与 onclick 类似)。
  - **方法事件**处理器：指向组件上定义的方法的属性名或是路径。

```
<button @click="count++">Add 1</button>
<p>Count is: {{ count }}</p>
```

```
<button @click="say('hello')">Say hello</button>
```

```
greet(event){
// `event` 是原生 DOM 事件
console.log(event);
uni.showToast({
  title: 'Hello ' + this.name + '!' });
      ......
<button
@click="greet">Greet</button>
```

```
//在内联语句处理器中访问原始的 DOM 事件。
可以用特殊变量 $event 把它传入方法
warn(message, event) {

// 访问原生事件对象
  if (event) {
       //可访问 event.target等原生事件对象，
如 event.target.tagname
                              }
uni.showToast({ title: message});
}

<button @click="warn('Form cannot be
submitted yet.', $event)">   Submit
</button>
```

事件修饰符

- @事件（v-on）提供了事件修饰符:
  - .stop: 各平台均支持， 使用时会阻止事件冒泡，在非 H5 端同时也会阻止事件的默认行为
  - .native: 监听原生事件，各平台均支持
  - .prevent: 仅在 H5 平台支持
  - .self: 仅在 H5 平台支持
  - .capture: 仅在 H5 平台支持
  - .once: 仅在 H5 平台支持
  - .passive: 仅在 H5 平台支持

- 为兼容各端，uniapp规定，事件需使用 @ 的方式绑定，不要使用小程序端的 bind 和 catch 进行事件绑定；也不能在 JS 中使用 event.preventDefault()和event.stopPropagation()方法；
- uni-app 运行在手机端，没有键盘事件，所以不支持Vue中的按键修饰符或鼠标按键修饰符

事件映射表

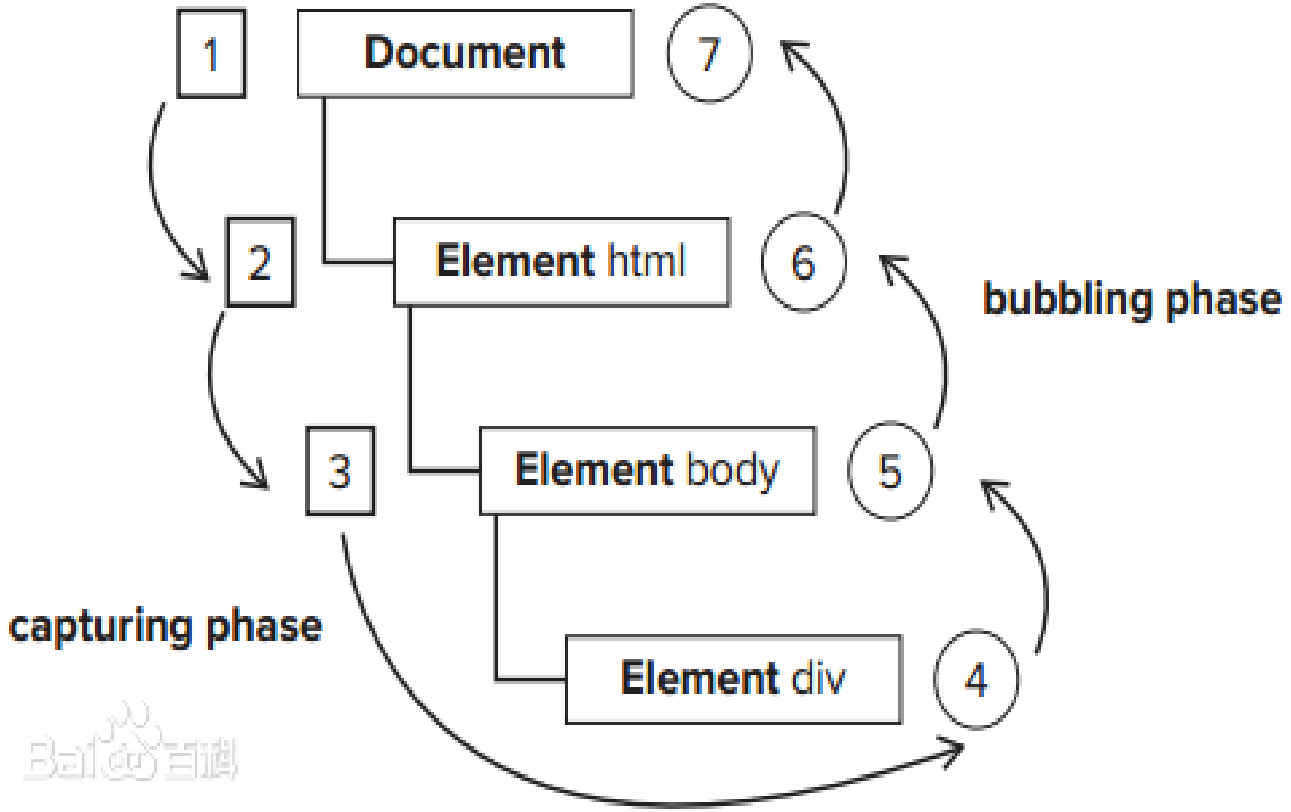
```
//左侧为 WEB 事件，右侧为 ``uni-app`` 对应事件
{
        click: 'tap',
        touchstart: 'touchstart',
        touchmove: 'touchmove',
        touchcancel: 'touchcancel',
        touchend: 'touchend',
        tap: 'tap',
        longtap: 'longtap', //推荐使用longpress代替
        input: 'input',
        change: 'change',
        submit: 'submit',
        blur: 'blur',//元素失去焦点
        focus: 'focus',
        reset: 'reset',
        confirm: 'confirm',
        columnchange: 'columnchange',
        linechange: 'linechange',
        error: 'error',
        scrolltoupper: 'scrolltoupper',
        scrolltolower: 'scrolltolower',
        scroll: 'scroll'  }
```

tap VS click

- tap和click都是点击事件。编译到小程序端，click会被转成tap
- 移动端有太多复杂的功能是click监听不到的，例如，触摸、按住和轻滑。这时候就要用tap
- click是点击放开之后才触发的，时间上会有延迟大概300ms，tap是手指触摸离开时触发
- tap还有一个特点——『事件穿透』，执行完绑定的tap事件之后，如果下面绑定了其他事件或者是本身就存在点子事件的话，也会默认触发。

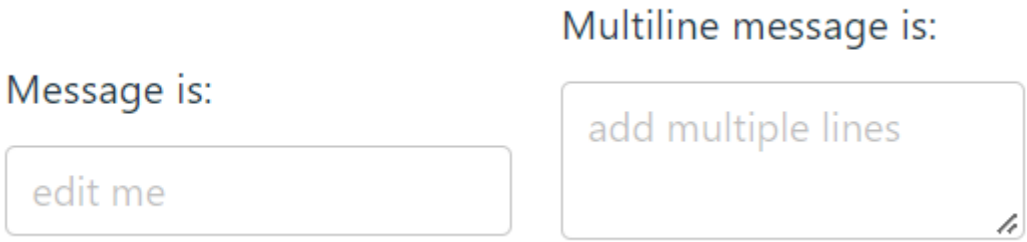
事件穿透与事件冒泡

触发的先后顺序是：touchstart -> touchend -> click click 事件的滞后性设计为事件穿透（点击穿透）埋下了伏笔。
事件穿透是指触发某个目标元素的触摸事件时，会同时触发该目标元素相同位置中其他元素的鼠标点击事件。



六、表单输入绑定

- 在前端处理表单时，常需要将表单输入框的内容同步给 JS 中相应的变量。手动连接值绑定和更改事件监听器可能会很麻烦，v-model 指令简化了这一步骤。
- v-model会根据所使用的元素自动使用对应的 DOM 属性和事件组合
  - 文本类型的 <input> 和 <textarea>： 绑定 value 并侦听 input 事件;
  - <input type="checkbox"> 和 <input type="radio"> 会绑定 checked 并侦听 change 事件;
  - <select> 会绑定 value 并侦听 change 事件
- v-model 会忽略任何表单元素上初始的 value、checked 或 selected attribute。它始终将当前绑定的 JS 状态视为数据的正确来源，需要在data 选项来声明该初始值。




```
<span>Multiline message is:</span>

<p style="white-space: pre-line;">{{ message }}</p>

<textarea v-model=“message” placeholder=“add multiple lines”></textarea>
```

注意在：<textarea> 中不支持插值表达式的，需使用 v-model 来替代

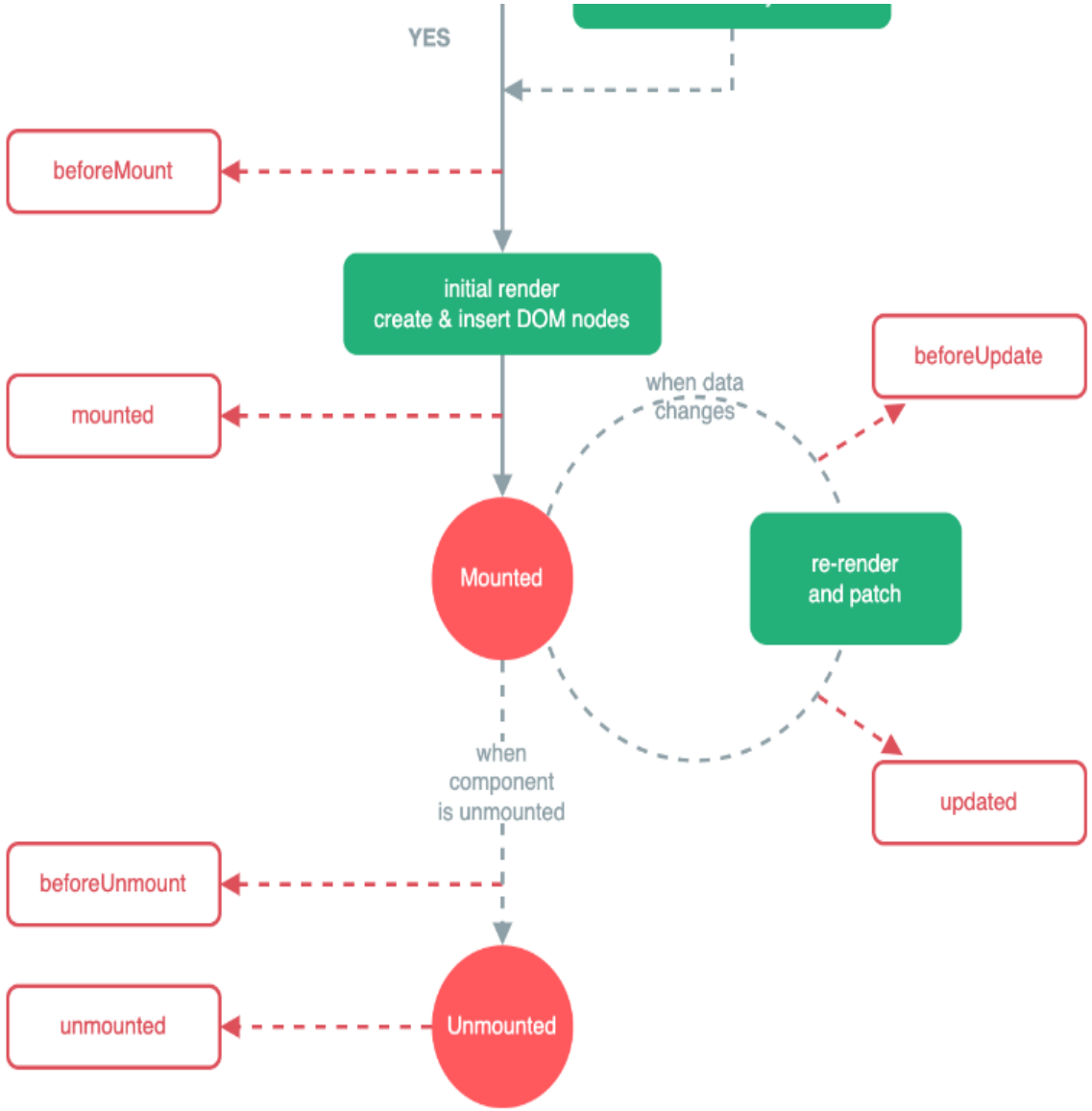
- 值绑定
  - 复选框
  - 单选按钮
  - 选择器选项
- V-model的修饰符 .lazy .number .trim
- https://cn.vuejs.org/guide/essentials/forms.html

七、生命周期钩子

- 每个 Vue 组件实例在创建时都需要经历一系列的初始化步骤，比如设置好数据侦听，编译模板，挂载实例到 DOM，以及在数据改变时更新 DOM。在此过程中，它也会运行被称为生命周期钩子的函数，让开发者有机会在特定阶段运行自己的代码。

  所有生命周期钩子函数的 this 上下文都会自动指向当前调用它的组件实例。

  避免用箭头函数来定义生命周期钩子，箭头函数中无法通过 this 获取组件实例。



生命周期钩子API索引： https://cn.vuejs.org/api/options-lifecycle.html#serverprefetch

uniapp组件生命周期钩子

- uni-app 组件支持的生命周期，与vue标准组件的生命周期相同。这里没有页面级的onLoad等生命周期
- https://uniapp.dcloud.net.cn/tutorial/page.html#componentlifecycle
- beforeCreate （data method不能用）created （data method可用）
- beforeMount mounted： 组件是否渲染到页面上（Dom元素是否可用）
- beforeUpdate updated（仅限H5的虚拟 DOM 重渲染与打补丁前后）
- beforeDestroy destroyed：给引入的组件添加v-if，v-if为false时，组件被销毁。实例销毁后，vue实例所有东西（监听器、子实力等）都会被解绑。

应用生命周期

- 微信小程序：https://developers.weixin.qq.com/miniprogram/dev/reference/api/App.html
  - 微信小程序的应用生命周期App() 必须在 app.js 中调用，必须调用且只能调用一次
  - 微信小程序的应用生命周期函数：
    （1）onLaunch: 初始化小程序时触发，全局只触发一次
    （2）onShow: 小程序初始化完成或用户从后台切换到前台显示时触发
    （3）onHide: 用户从前台切换到后台隐藏时触发
    （4）onError: 小程序发生脚本错误，或者 api 调用失败时，会触发 onError 并带上错误信息
  *** 后台：点击左上角关闭，或者按了设备 Home 键离开微信，并没有直接销毁，而是进入后台
  *** 前台：再次进入微信或再次打开小程序，相当于从后台进入前台

- 与之类似的uniapp应用生命周期 https://uniapp.dcloud.net.cn/collocation/App.html

增加： onUniNViewMessage onUnhandledRejection onPageNotFound onThemeChange

页面生命周期

- Uniapp 页面生命周期https://uniapp.dcloud.net.cn/tutorial/page.html#lifecycle
  - onload （传参参考uni.navigateTo）onShow onReady onUnload
  - 部分平台支持 onResize onPullDownRefresh onReachBottom onTabItemTap onShareAppMessage onPageScroll 。。。

- 微信小程序 页面生命周期函数

  （1）onLoad：首次进入页面加载时触发，可以在 onLoad 的参数中获取打开当前页面路径中的参数。

  （2）onShow：加载完成后、后台切到前台或重新进入页面时触发(可多次)

  （3）onReady：页面首次渲染完成时触发

  （4）onHide：从前台切到后台或进入其他页面触发(与Show对应，可多次)

  （5）onUnload：页面卸载时（与onLoad对应）

八 监听器

- 在选项式 API 中，使用 watch 选项在每次响应式属性发生变化时触发一个函数。监听器一般会在宿主组件卸载时自动停止

```
data() {
    return {
      question: '',
      answer: 'Questions usually contain a question mark. ;-)'    }    },
  watch: {
    // question 改变时，这个函数就会执行
    question(newQuestion, oldQuestion) {
      if (newQuestion.includes('?')) {
        this.getAnswer()    }    } },
  methods: {
    async getAnswer() {
      this.answer = 'Thinking...'
      try {
        const res = await fetch('https://yesno.wtf/api')
        this.answer = (await res.json()).answer
      } catch (error) {
        this.answer = 'Error! Could not reach the API. ' + error      }    } } }
```

- 更改了响应式状态，可能会同时触发 Vue 组件更新和侦听器回调。默认情况下，用户创建的侦听器回调，会在 Vue 组件更新之前被调用。这意味着侦听器回调中访问的 DOM 将是被 Vue 更新之前的状态。
- 如果想在侦听器回调中能访问被 Vue 更新之后的DOM，需要指明 flush: 'post' 选项。
- watch 默认是仅当数据源变化时才会执行回调。如果希望在创建侦听器时，立即执行一遍回调，如想请求一些初始数据，然后在相关状态更改时重新请求数据。可以用一个对象来声明侦听器，这个对象有 handler 方法和 immediate: true 选项，这样便能强制回调函数立即执行。

```
export default {
  // ...
  watch: {
    question: {
      handler(newQuestion) {// 在组件实例创建时会立即调用 },
      immediate: true,
      flush: 'post'
    }} }
```

- 组件实例的 $watch() 方法来命令式地创建一个侦听器：

```
this.$watch('question', (newQuestion) => {
```

- 当的确需要在组件卸载之前就停止一个侦听器，可以调用 $watch() 返回的函数：

```
const unwatch = this.$watch('foo', callback)

unwatch() // ...当该侦听器不再需要时
```

九 模板引用 $ref

- 特殊的 ref 属性，可以帮助我们直接访问底层 DOM 元素（**在组件挂载后**才能访问模板引用）
- 当在 v-for 中使用模板引用时，相应的引用中包含的值是一个数组

```
<script>
export default {
  mounted() {

this.$refs.input.focus()  }}
</script>
<template>
  <input ref="input" />
</template>
```

```
<script>
export default {
  data() {
    return {
      list: [1, 2, 3]    }  },
  mounted() {
    console.log(this.$refs.items)
  }}</script>
<template>
  <ul>
    <li v-for="item in list" ref="items">
      {{ item }}
    </li>
  </ul>
</template>
```