

- [Vue2.5入门](#)
 - [课堂目标](#)
 - [知识要点](#)
 - [资源](#)
 - [检查点](#)
 - [起步](#)
 - [声明式渲染](#)
 - [纯浏览端体验](#)
 - [单文件组件](#)
 - [条件与循环](#)
 - [处理用户输入](#)
 - [组件化](#)
 - [购物车](#)
 - [组件化](#)
 - [组件按需引入](#)
 - [父子组件通过属性做数据传递](#)
 - [样式和class绑定](#)
 - [计算属性](#)
 - [mock数据](#)
 - [数据持久化](#)
 - [生命周期](#)
 - [回顾](#)

课堂目标

1. 学习Vuejs基础语法
2. 熟悉官方Vue-cli3脚手架
3. 使用.vue单文件开发组件
4. ES6 模块化机制
5. 深入了解Vue的组件化机制
6. 学习使用Vuejs内部通讯机制
7. 使用axios获取网络数据
8. ES7 处理异步中级方案 async+await
9. 本地mock数据
10. 实现一个购物车项目

知识要点

1. vue
2. 组件化

资源

- 1. [vuejs](#)
- 2. [vue-cli脚手架](#)

检查点

起步

- 1. 安装vue-cli3
 - 1. `npm install -g @vue/cli`
 - 2. `npm install -g @vue/cli-service-global`
- 2. 新建项目 `vue create vue-buy`
- 3. 选择配置
 - 1. Babel 代码转义
 - 2. Linter 代码风格
 - 3. 测试
- 4. `npm run serve`
- 5. 项目结构

.

├─ README.md

├─ babel.config.js

├─ package-lock.json

├─ package.json

├─ public

| ├─ favicon.ico

| └─ index.html

└─ src

├─ App.vue

├─ assets

| └─ logo.png

├─ components

| └─ HelloWorld.vue

└─ main.js

文档

babel配置

静态资源

源码

声明式渲染

纯浏览端体验

```
<div id="app">
  {{ message }}
</div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
var app = new Vue({
```

```
    el: '#app',
    data: {
      message: 'Hello Vue!'
    }
  })
</script>
```

单文件组件

`.vue` 是vue单文件组件，一个文件就是一个组件，由template， script和style三个标签构成，分别是html， js和css的内容部分,修改App.vue文件内容

```
<template>
  <div id="app">
    {{name}}
  </div>
</template>

<script>

export default {
  name: 'app',
  data(){
    return {
      name: '开课吧'
    }
  }
}
</script>

<style>
#app{
  color:red;
}
</style>
```

条件与循环

控制切换一个元素是否显示也相当简单：

```
<div id="app">
  <p v-if="showName">{{name}}</p>
</div>
```

```
export default {
  name: 'app',
  data(){
    return {
      name: '开课吧',
      showName:true
    }
  },
  created(){
    // 组件创建后 回自动执行created这个生命周期
    setTimeout(()=>{
      this.showName = false
    },2000)
  }
}
```

还有其它很多指令，每个都有特殊的功能。例如，`v-for` 指令可以绑定数组的数据来渲染一个项目列表：

```
``` html
<div id="app">

 <li v-for='good in goods'>
 {{good.text}}

</div>
```

```
export default {
 name: 'app',
 data(){
 return {
 name: '开课吧',
 goods:[
 {text:'百万年薪架构师'},
 {text:'web全栈架构师'},
 {text:'Python爬虫'},
 {text:'Java架构师'}

]
 }
 },
 created(){
 // 组件创建后 回自动执行created这个生命周期
 setTimeout(()=>{
 this.goods.push({
 text:'UED课程'
 })
 },2000)
 }
}
```

```
}
```

两秒后 会多出一条 UED课程

## 处理用户输入

为了让用户和你的应用进行交互，我们可以用 `@` 添加一个事件监听器，通过它调用在 Vue 实例中定义的方法：

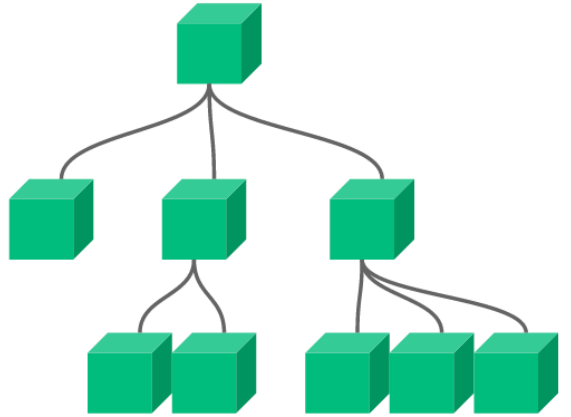
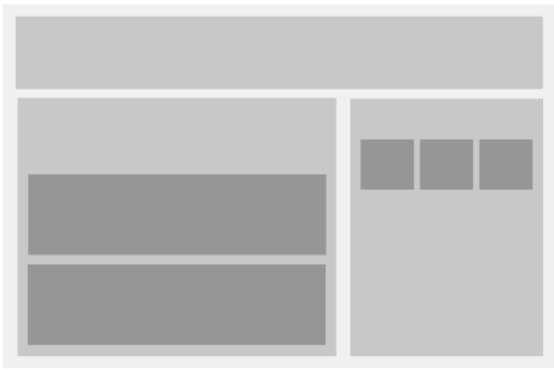
```
<div id="app">
 <input type="text" v-model='text'>
 <button @click="addGood">添加</button>

 <li v-for='good in goods'>
 {{good.text}}

</div>
```

```
export default {
 name: 'app',
 data(){
 return {
 name: '开课吧',
 text: '',
 goods:[
 {text:'百万年薪架构师'},
 {text:'web全栈架构师'},
 {text:'Python爬虫'},
 {text:'Java架构师'}
]
 }
 },
 methods:{
 addGood(){
 if(this.text){
 this.goods.push({
 text:this.text
 })
 this.text = ''
 }
 }
 }
}
```

## 组件化



## 购物车

删除输入框和按钮，添加价格数据

```
<li v-for='(good, index) in goods' @click='toggle(index)'>

 {{good.text}}

 ¥ {{good.price}}

 <button @click="addCart(index)">添加购物车</button>


```

```
goods: [
 { text: '百万年薪架构师',price:100},
 { text: 'web全栈架构师',price:80},
 { text: 'Python爬虫',price:60}
]
```

## 组件化

新建Cart.vue 购物车组件

### 组件按需引入

```
import Cart from './components/Cart'
export default {
 name: 'app',
 components:{Cart},
```

```
}
```

## 父子组件通过属性做数据传递

```
<Cart :name="name" :cart="cart"></Cart>
```

```
// 子组件
```

```
<div>
 {{name}}购物车
</div>
```

```
export default {
 name: 'Cart',
 props:['name',"cart"],
 data () {
 return {

 }
 },
 methods: {

 }
}
```

```
// 属性校验
```

```
props:{
 name:{
 type:String,
 required:true
 },
 cart:{
 type:Array
 }
},
```

```
购物车数据：{
text:'名字',
price:'单价',
active:是否选中,
count:购买数量
}
```

```
addCart(i){
 const good = this.goods[i]
 const ret = this.cart.find(v=>v.text==good.text)
 console.log(good,ret)
```

```
if(ret){
 ret.count+=1
}else{
 this.cart.push({...good, active:true, count:1})
}
}
```

Cart组件

```
<div>
 <div>
 {{name}}购物车

 </div>

 <table border='1'>
 <tr>
 <th>课程名</th>
 <th>单价</th>
 <th>数量</th>
 <th>价格</th>
 </tr>
 <tr v-for="c in cart" :key="c.name">
 <td>
 {{c.text}}
 </td>
 <td>
 ¥{{c.price}}

 </td>
 <td>
 {{c.count}}
 </td>
 <td>
 {{c.count*c.price}}
 </td>
 </tr>
 </table>

</div>
```

```
export default {
 name: 'Cart',
 props:{
 name:{
 type:String,
 required:true
 },
 cart:{
```



```
 type:Array
 },
 data () {
 return {

 }
 },
 methods: {

 }
}
```

## 组件内部数量修改

```
<table border='1'>
 <tr>
 <th>课程名</th>
 <th>单价</th>
 <th>数量</th>
 <th>价格</th>
 </tr>
 <tr v-for="(c,i) in cart" :key="c.name">
 <td>
 {{c.text}}
 </td>
 <td>
 ¥ {{c.price}}
 </td>
 <td>
 <button @click="minus(i)">-</button>
 {{c.count}}
 <button @click="add(i)">+</button>
 </td>
 <td>
 {{c.count*c.price}}
 </td>
 </tr>
</table>
```

```
methods: {
 add(i){
 this.cart[i].count +=1
 },
 minus(i){
 const count = this.cart[i].count
 if(count>1){
 this.cart[i].count -=1
 }
 }
}
```

```
 }else{
 this.remove(i)
 }
 },
 remove(i){
 if(window.confirm(`是否删除商品${this.cart[i].text}?`)){
 this.cart.splice(i,1)
 }
 }
}

}
```

## 样式和class绑定

设置单选框，选中与否 修改购物车文字颜色

内联样式 v-bind:style, 可以简写为 `:style`

```
<tr v-for="(c,i) in cart" :key="c.name" :style="{color:c.active?'red':'black'}">
 <td>
 <input type="checkbox" v-model='c.active'>
 </td>
 ...
</tr>
```

class绑定 使用:class

```
<tr v-for="(c,i) in cart" :key="c.name" :class="{active:c.active}">
```

```
tr.active{
 color:red;
}
```

## 计算属性

使用compouted字段，可以进行复杂逻辑的计算

```
<tr>
 <td></td>
 <td colspan='2'>{{activeCount}}/ {{count}}</td>
 <td colspan='2'>¥{{total}}</td>
</tr>
```

```

computed:{
 total(){
 let num = 0
 this.cart.forEach(v=>{
 if(v.active){
 num+= v.price*v.count
 }
 })
 return num

 //或者

 //return this.cart.reduce((sum,v)=>{
 // if(v.active){
 // return sum + v.price*v.count
 // }
 // },0)
 },
 count(){
 return this.cart.length
 },
 activeCount(){
 return this.cart.filter(v=>v.active).length
 }
},
},
`

```

## 任意两个组件传递

父子组件通过props传递，如果组件没有明显的父子关系 使用总线机制进行传递

vue每个实例都有订阅/发布模式的实现，使用\$on和\$emit来使用

```

````js
Vue.prototype.$bus = new Vue()

```

```

// App.vue
addCart(i){
  const good = this.goods[i]
  this.$bus.$emit('addCart',good)
}

// Cart.vue
created() {
this.$bus.$on("addCart", good => {
  const ret = this.cart.find(v => v.text == good.text);
  if (ret) {
    ret.count += 1;

```

```
    } else {
      this.cart.push({ ...good, active: true, count: 1 });
    }
  });
},
```

更复杂的数据传递，可以使用vuex 后面课程会介绍

mock数据

简单的mock，使用自带的webpack-dev-server即可，新建vue.config.js扩展webpack设置

```
module.exports = {
  configureWebpack: {
    devServer: {
      before(app) {
        app.get('/api/goods', function (req, res) {
          res.json({
            list: [
              { text: '百万年薪架构师', price: 100 },
              { text: 'web全栈架构师', price: 80 },
              { text: 'Python爬虫', price: 60 }
            ]
          });
        });
      }
    }
  }
}
```

访问<http://localhost:8080/api/goods> 看到mock的数据

使用axios获取接口数据 `npm install axios`

```
created(){
  axios.get('/api/goods').then(res=>{
    this.goods = res.data.list
  })
},
```

使用ES7的async+await语法

```
async created(){
  const res = await axios.get('/api/goods')
  this.goods = res.data.list
}
```

数据持久化

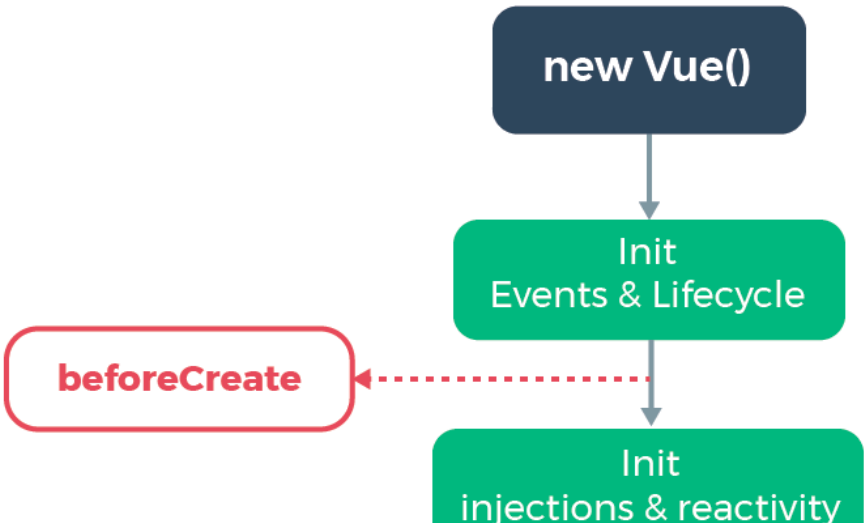
简单的localStorage + vue监听器

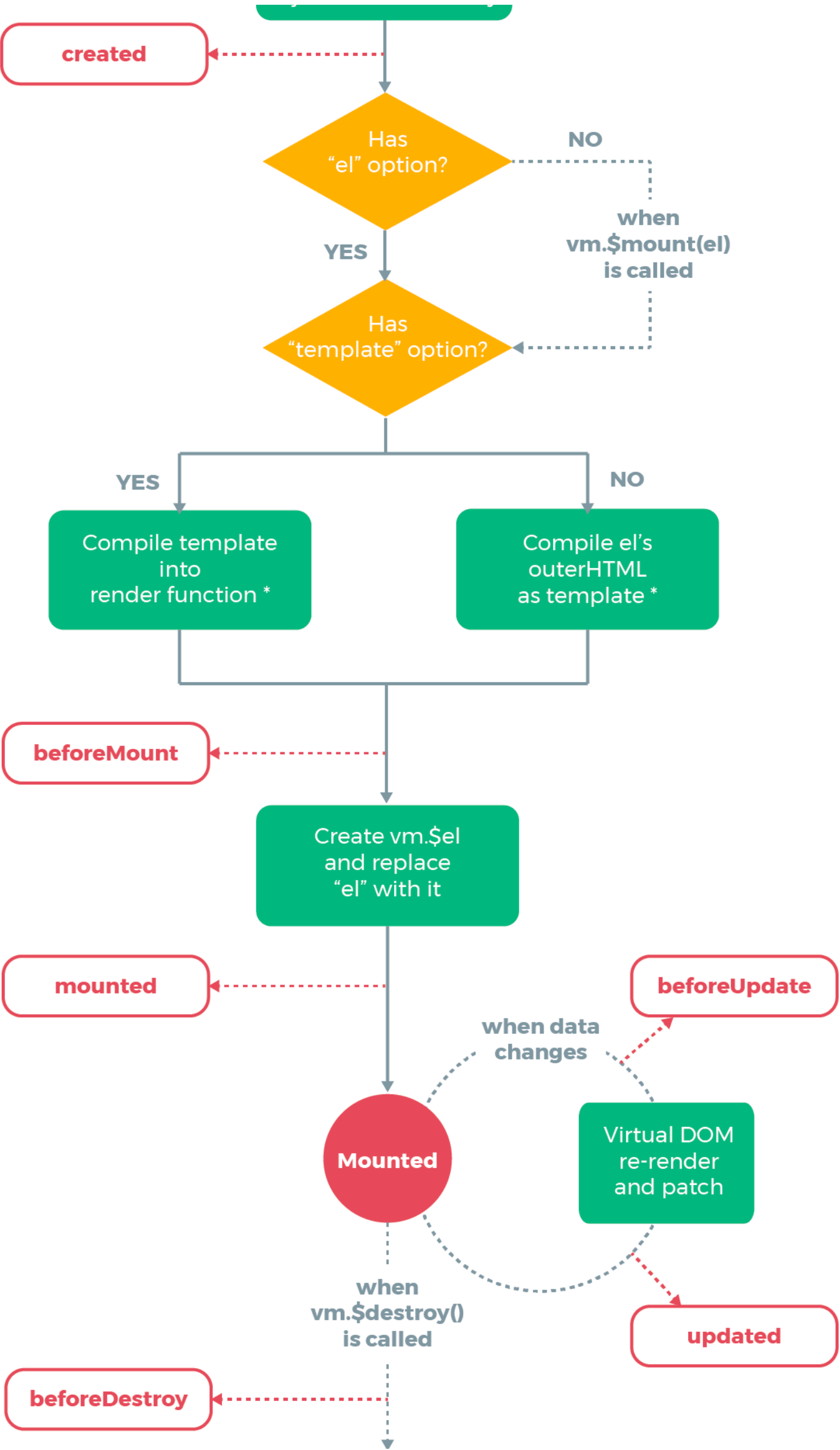
```
watch:{
  // 监听cart变化
  cart:function(){
    this.setLocal()
  }
},
methods:{
  setLocal(){
    window.localStorage.setItem('cart',JSON.stringify(this.cart))
  }
},
created(){
  this.cart = JSON.parse(window.localStorage.getItem('cart'))
}
```

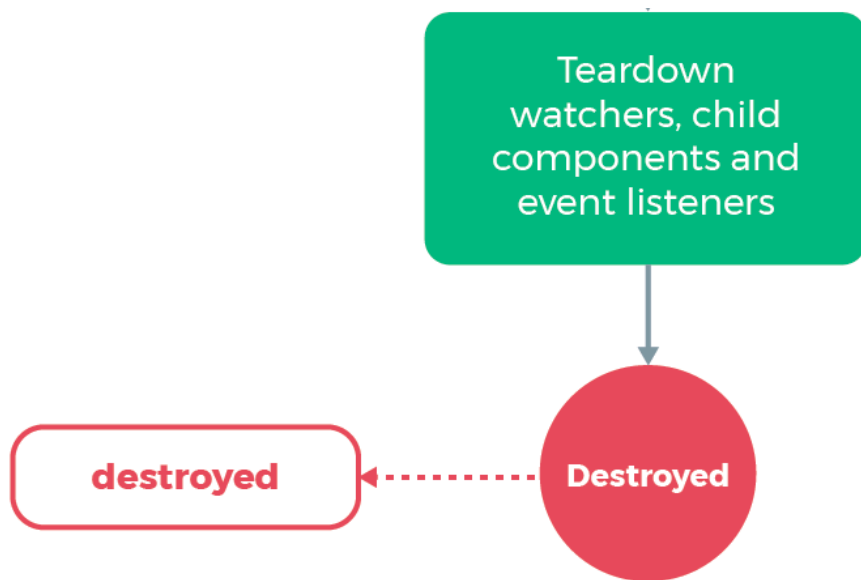
```
// 循环监听

watch:{
  cart:{
    handler(){
      this.setLocal()
    },
    deep:true
  }
}
```

生命周期







* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

回顾

- [Vue2.5入门](#)
 - [课堂目标](#)
 - [知识要点](#)
 - [资源](#)
 - [检查点](#)
 - [起步](#)
 - [声明式渲染](#)
 - [纯浏览端体验](#)
 - [单文件组件](#)
 - [条件与循环](#)
 - [处理用户输入](#)
 - [组件化](#)
 - [购物车](#)
 - [组件化](#)
 - [组件按需引入](#)
 - [父子组件通过属性做数据传递](#)
 - [样式和class绑定](#)
 - [计算属性](#)
 - [mock数据](#)
 - [数据持久化](#)
 - [生命周期](#)
 - [回顾](#)