

互联网应用开发技术

Web Application Development

第5课

WEB前端-VUE构件详解

Episode Five

**Vue Components
In-Depth**

陈昊鹏

chen-hp@sjtu.edu.cn

Web Application
Development

- Download
 - <https://github.com/facebook/react/>
- From
 - Vue Guide
 - <https://vuejs.org/v2/guide/>
 - Vue教程
 - <https://cn.vuejs.org/v2/guide/>
 - 基于Idea从零搭建一个最简单的vue项目
 - <https://www.jianshu.com/p/9c1d4f8ed068>

- Component Names

```
Vue.component('my-component-name', { /* ... */ })
```

- Name Casing:

```
Vue.component('my-component-name', { /* ... */ }) ★
```

```
Vue.component('MyComponentName', { /* ... */ })
```

- Global Registration

- These components are globally registered. That means they can be used in the template of any root Vue instance (new Vue) created after registration.

```
Vue.component('my-component-name', {  
  // ... options ...  
})
```

```
Vue.component('component-a', { /* ... */ })  
Vue.component('component-b', { /* ... */ })  
Vue.component('component-c', { /* ... */ })
```

```
new Vue({ el: '#app' })
```

```
<div id="app">  
  <component-a></component-a>  
  <component-b></component-b>  
  <component-c></component-c>  
</div>
```

- This even applies to all subcomponents, meaning all three of these components will also be available inside each other.

- Local Registration

- You can define your components as plain JavaScript objects. Then define the components you'd like to use in a components option.

```
var ComponentA = { /* ... */ }  
var ComponentB = { /* ... */ }
```

```
new Vue({  
  el: '#app',  
  components: {  
    'component-a': ComponentA,  
    'component-b': ComponentB  
  }  
})
```

- Note that locally registered components are not also available in subcomponents.

✓

```
var ComponentA = { /* ... */ }
```

```
var ComponentB = {  
  components: {  
    'component-a': ComponentA  
  },  
  // ...  
}
```

✓

```
import ComponentA from './ComponentA.vue'
```

```
export default {  
  components: {  
    ComponentA  
  },  
  // ...  
}
```

ComponentA:
ComponentA

- Local Registration in a Module System
 - Create a components directory, with each component in its own file.
Import components before local registration.

```
import ComponentA from './ComponentA'  
import ComponentC from './ComponentC'
```

```
export default {  
  components: {  
    ComponentA,  
    ComponentC  
  },  
  // ...  
}
```


- Automatic Global Registration of Base Components
 - Use **require.context** to globally register only these very common base components.

```
const requireComponent = require.context(  
  // The relative path of the components folder  
  './components',  
  // Whether or not to look in subfolders  
  false,  
  // The regular expression used to match base component filenames  
  /Base[A-Z]\w+\.(vue|js)$/   
)
```

```
requireComponent.keys().forEach(fileName => {  
  // Get component config  
  const componentConfig = requireComponent(fileName)  
  
  // Get PascalCase name of component  
  const componentName = upperFirst(  
    camelCase(  
      // Gets the file name regardless of folder depth  
      fileName  
        .split('/')  
        .pop()  
        .replace(/\.\\w+$/, '')  
    )  
  )  
  
  // Register component globally  
  Vue.component(  
    componentName,  
    // Look for the component options on `default`, which will  
    // exist if the component was exported with `export default`,  
    // otherwise fall back to module's root.  
    componentConfig.default || componentConfig  
  )  
})
```


- Prop Casing (camelCase vs kebab-case)
 - HTML attribute names are case-insensitive. If you're using string templates, this limitation does not apply.

```
Vue.component('blog-post', {  
  // camelCase in JavaScript  
  props: ['postTitle'],  
  template: '<h3>{{ postTitle }}</h3>'  
})
```



```
<!-- kebab-case in HTML -->  
<blog-post post-title="hello!"></blog-post>
```

- Prop Types
 - You can list props as an object, where the properties' names and values contain the prop names and types, respectively.

```
props: {  
  title: String,  
  likes: Number,  
  isPublished: Boolean,  
  commentIds: Array,  
  author: Object,  
  callback: Function,  
  contactsPromise: Promise // or any other constructor  
}
```

- Passing Static Props

```
<blog-post title="My journey with Vue"></blog-post>
```

- Passing Dynamic Props

```
<!-- Dynamically assign the value of a variable -->  
<blog-post v-bind:title="post.title"></blog-post>
```

```
<!-- Dynamically assign the value of a complex expression -->  
<blog-post  
  v-bind:title="post.title + ' by ' + post.author.name"  
></blog-post>
```

Number

Boolean

Array

Array

Object

- Passing Dynamic Props
 - Passing the Properties of an Object

```
post: {  
  id: 1,  
  title: 'My Journey with Vue'  
}
```

✓ `<blog-post v-bind="post"></blog-post>`

✓ `<blog-post
 v-bind:id="post.id"
 v-bind:title="post.title"
></blog-post>`

- One-Way Data Flow

- The prop is used to pass in an initial value; the child component wants to use it as a local data property afterwards.
- The prop is passed in as a raw value that needs to be transformed.

```
props: ['initialCounter'],
data: function () {
  return {
    counter:
this.initialCounter
  }
}
```

```
props: ['size'],
computed: {
  normalizedSize: function () {
    return
this.size.trim().toLowerCase()
  }
}
```

- Note that objects and arrays in JavaScript are passed by reference, so if the prop is an array or object, mutating the object or array itself inside the child component will affect parent state.

- Prop Validation

- Components can specify requirements for their props, such as the types you've already seen. If a requirement isn't met, Vue will warn you in the browser's JavaScript console.

```
Vue.component('my-component', {
  props: {
    // Basic type check (`null`
    and `undefined` values will pass
    any type validation)
    propA: Number,
    // Multiple possible types
    propB: [String, Number],
    // Required string
    propC: {
      type: String,
      required: true
    },
    // Number with a default
    value
    propD: {
      type: Number,
      default: 100
    },
  },
});
```

```
// Object with a default value
propE: {
  type: Object,
  // Object or array defaults
  must be returned from
  // a factory function
  default: function () {
    return { message: 'hello' }
  },
  // Custom validator function
  propF: {
    validator: function (value) {
      // The value must match one
      of these strings
      return ['success', 'warning',
        'danger'].indexOf(value) !== -1
    }
  }
},
});
```

- Non-Prop Attributes

- A non-prop attribute is an attribute that is passed to a component, but does not have a corresponding prop defined.
- Replacing/Merging with Existing Attributes

```
<input type="date" class="form-control">
```

```
<bootstrap-date-input data-date-picker="activated" class="date-picker-theme-dark">  
></bootstrap-date-input>
```

form-control date-picker-theme-dark

- Disabling Attribute Inheritance

```
Vue.component('my-component', {  
  inheritAttrs: false,  
  // ...  
})
```

- Event Names
 - Unlike components and props, event names don't provide any automatic case transformation. Recommend you always use kebab-case for event names.
- Customizing Component v-model
 - the value of lovingVue will be passed to the checked prop. The lovingVue property will then be updated when <base-checkbox> emits a change event with a new value.

```
Vue.component('base-checkbox', {  
  model: {  
    prop: 'checked',  
    event: 'change'  
  },  
  props: {  
    checked: Boolean  
  },  
  template: `  
    <input  
      type="checkbox"  
      v-bind:checked="checked"  
      v-on:change="$emit('change', $event.target.checked)"  
    >  
  `,  
})  
  
<base-checkbox v-model="lovingVue"></base-checkbox>
```

- Binding Native Events to Components
 - Listen directly to a native event on the root element of a component.

```
Vue.component('base-input', {
  inheritAttrs: false,
  props: ['label', 'value'],
  computed: {
    inputListeners: function () {
      var vm = this
      // `Object.assign` merges objects together to form a new object
      return Object.assign({},
        // We add all the listeners from the parent
        this.$listeners,
        // Then we can add custom listeners or override the
        // behavior of some listeners.
        {
          // This ensures that the component works with v-model
          input: function (event) {
            vm.$emit('input', event.target.value)
          }
        }
      )
    }
  }
})
```

```
template: `
  <label>
    {{ label }}
    <input
      v-bind="$attrs"
      v-bind:value="value"
      v-on="inputListeners"
    >
  </label>
`
})
```


- **.sync Modifier**
 - The “two-way binding” for a prop

```
this.$emit('update:title', newTitle)
```

```
<text-document  
  v-bind:title="doc.title"  
  v-on:update:title="doc.title = $event"  
></text-document>
```

```
<text-document v-bind:title.sync="doc.title"></text-  
document>
```

- Slot Content

- Vue implements a content distribution API inspired by the Web Components spec draft, using the `<slot>` element to serve as distribution outlets for content.

```
<navigation-link url="/profile">  
  Your Profile  
</navigation-link>
```

```
<a  
  v-bind:href="url"  
  class="nav-link"  
>  
  <slot></slot>  
</a>
```

```
<navigation-link url="/profile">  
  <!-- Add a Font Awesome icon -->  
  <span class="fa fa-user"></span>  
  Your Profile  
</navigation-link>
```

```
<navigation-link url="/profile">  
  <!-- Use a component to add an icon -->  
  <font-awesome-icon  
    name="user"></font-awesome-icon>  
  Your Profile  
</navigation-link>
```

- Slot Content

- Vue implements a content distribution API inspired by the Web Components spec draft, using the `<slot>` element to serve as distribution outlets for content.

```
<navigation-link url="/profile">
  Your Profile
</navigation-link>
```

```
<a
  v-bind:href="url"
  class="nav-link"
>
  <slot></slot>
</a>
```

When the component renders:

```
<navigation-link url="/profile">
  <!-- Add a Font Awesome icon -->
  <span class="fa fa-user"></span>
  Your Profile
</navigation-link>
```

Or even other components:

```
<navigation-link url="/profile">
  <!-- Use a component to add an icon
-->
  <font-awesome-icon
name="user"></font-awesome-icon>
  Your Profile
</navigation-link>
```

- **Compilation Scope**

- Everything in the parent template is compiled in parent scope; everything in the child template is compiled in the child scope.

```
<navigation-link url="/profile">
  Clicking here will send you to: {{ url }}
<!--
  The `url` will be undefined, because this content is passed
  _to_ <navigation-link>, rather than defined _inside_ the
  <navigation-link> component.
-->
</navigation-link>
```

- **Fallback Content**

- Everything in the parent template is compiled in parent scope; everything in the child template is compiled in the child scope.

```
<button type="submit">
  <slot>Submit</slot>
</button>
```

In a parent component:

```
<submit-button></submit-button>
<button type="submit">
  Submit
</button>
```

- Named Slots

- Everything inside the `<template>` elements will be passed to the corresponding slots. Any content not wrapped in a `<template>` using `v-slot` is assumed to be for the default slot.

multiple slots:

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

v-slot:

```
<base-layout>
  <template v-slot:header>
    <h1>Here might be a page
title</h1>
  </template>

  <template v-slot:default>
    <p>A paragraph for the main
content.</p>
    <p>And another one.</p>
  </template>

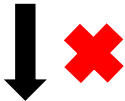
  <template v-slot:footer>
    <p>Here's some contact info</p>
  </template>
</base-layout>
```

- Scoped Slots

- Sometimes, it's useful for slot content to have access to data only available in the child component.

Imagine a `<current-user>` component with the following template:

```
<span>
  <slot>{{ user.lastName }}</slot>
</span>
```



```
<current-user>
  {{ user.firstName }}
</current-user>
```

```
<span>
  <slot v-bind:user="user">
    {{ user.lastName }}
  </slot>
</span>
```



```
<current-user>
  <template v-slot:default="slotProps">
    {{ slotProps.user.firstName }}
  </template>
</current-user>
```

- Scoped Slots-Abbreviated Syntax for Lone Default Slots
 - When only the default slot is provided content, the component's tags can be used as the slot's template.

```
<current-user v-  
slot:default="slotProps">  
  {{ slotProps.user.firstName }}  
</current-user>
```



```
<current-user v-slot="slotProps">  
  {{ slotProps.user.firstName }}  
</current-user>
```

- Note that the abbreviated syntax for default slot cannot be mixed with named slots, as it would lead to scope ambiguity:
- Whenever there are multiple slots, use the full <template> based syntax for all slots:

```
<!-- INVALID, will result in warning -->  
<current-user v-slot="slotProps">  
  {{ slotProps.user.firstName }}  
  <template v-slot:other="otherSlotProps">  
    slotProps is NOT available here  
  </template>  
</current-user>
```



```
<current-user>  
  <template v-slot:default="slotProps">  
    {{ slotProps.user.firstName }}  
  </template>  
  
  <template v-slot:other="otherSlotProps">  
    ...  
  </template>  
</current-user>
```



- **Scoped Slots-Destructuring Slot Props**
 - The value of v-slot can actually accept any valid JavaScript expression that can appear in the argument position of a function definition. So in supported environments (single-file components or modern browsers), you can also use ES2015 destructuring to pull out specific slot props

```
<current-user v-slot="{ user }">
  {{ user.firstName }}
</current-user>
```



```
<current-user v-slot="{ user: person }">
  {{ person.firstName }}
</current-user>
```



```
<current-user v-slot="{ user = { firstName: 'Guest' } }">
  {{ user.firstName }}
</current-user>
```

- **Dynamic Slot Names**
 - Dynamic directive arguments also work on v-slot, allowing the definition of dynamic slot names

```
<base-layout>
  <template v-slot:[dynamicSlotName]>
    ...
  </template>
</base-layout>
```


- Named Slots Shorthand

- v-slot has a shorthand, replacing everything before the argument (v-slot:) with the special symbol #.

```
<base-layout>
  <template #header>
    <h1>Here might be a page title</h1>
  </template>

  <p>A paragraph for the main content.</p>
  <p>And another one.</p>

  <template #footer>
    <p>Here's some contact info</p>
  </template>
</base-layout>
```

v-slot:header



#header

- the shorthand is only available when an argument is provided. You must always specify the name of the slot if you wish to use the shorthand

```
<!-- This will trigger a warning -->
<current-user #="{ user }">
  {{ user.firstName }}
</current-user>
```

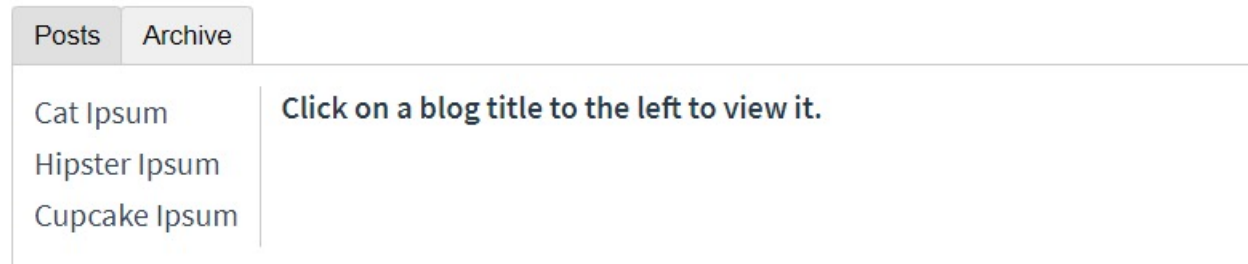


```
<current-user #default="{ user }">
  {{ user.firstName }}
</current-user>
```



- keep-alive with Dynamic Components
 - When switching between these components, maintain their state or avoid re-rendering for performance reasons.

```
<component v-bind:is="currentTabComponent"></component>
```



- Recreating dynamic components `<keep-alive>`

```
<!-- Inactive components will be cached! -->  
<keep-alive>  
  <component v-bind:is="currentTabComponent"></component>  
</keep-alive>
```

- Async Components

- Vue allows you to define your component as a factory function that asynchronously resolves your component definition. Vue will only trigger the factory function when the component needs to be rendered and will cache the result for future re-renders.

```
Vue.component('async-example', function (resolve, reject) {
  setTimeout(function () {
    // Pass the component definition to the resolve callback
    resolve({
      template: '<div>I am async!</div>'
    })
  }, 1000)
})
```

- retrieve the component

```
Vue.component('async-webpack-example', function (resolve) {
  // This special require syntax will instruct Webpack to
  // automatically split your built code into bundles which
  // are loaded over Ajax requests.
  require(['./my-async-component'], resolve)
})
```

- Async Components
 - return a Promise in the factory function

```
Vue.component(  
  'async-webpack-example',  
  // The `import` function returns a Promise.  
  () => import('./my-async-component')  
)
```

- local registration

```
new Vue({  
  // ...  
  components: {  
    'my-component': () => import('./my-async-component')  
  }  
})
```

- Async Components
 - Handling Loading State (New in 2.3.0+)

```
const AsyncComponent = () => ({  
  // The component to load (should be a Promise)  
  component: import('./MyComponent.vue'),  
  // A component to use while the async component is  
loading  
  loading: LoadingComponent,  
  // A component to use if the load fails  
  error: ErrorComponent,  
  // Delay before showing the loading component. Default:  
200ms.  
  delay: 200,  
  // The error component will be displayed if a timeout is  
  // provided and exceeded. Default: Infinity.  
  timeout: 3000  
})
```

- Element & Component Access
 - **Accessing the Root Instance.** In every subcomponent of a new Vue instance, this root instance can be accessed with the `$root` property. All subcomponents will now be able to access this instance and use it as a global store.

```
// The root Vue instance
new Vue({
  data: {
    foo: 1
  },
  computed: {
    bar: function () { /* ... */ }
  },
  methods: {
    baz: function () { /* ... */ }
  }
})
```

```
// Get root data
this.$root.foo
```

```
// Set root data
this.$root.foo = 2
```

```
// Access root computed properties
this.$root.bar
```

```
// Call root methods
this.$root.baz()
```

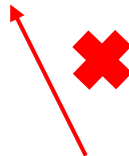
- Element & Component Access

- **Accessing the Parent Component Instance.** The \$parent property can be used to access the parent instance from a child.
- In most cases, reaching into the parent makes your application more difficult to debug and understand, especially if you mutate data in the parent.

```
<google-map>  
  <google-map-markers v-bind:places="iceCreamShops"></google-map-markers>  
</google-map>
```



```
<google-map>  
  <google-map-region v-bind:shape="cityBoundaries">  
    <google-map-markers v-bind:places="iceCreamShops"></google-map-markers>  
  </google-map-region>  
</google-map>
```



```
var map = this.$parent.map || this.$parent.$parent.map
```

- Element & Component Access

- **Accessing Child Component Instances & Child Elements.** Assign a reference ID to the child component using the ref attribute.

```
<base-input ref="usernameInput"></base-input>
```



```
this.$refs.usernameInput
```



```
<input ref="input">
```



```
methods: {  
  // Used to focus the input from the  
  parent  
  focus: function () {  
    this.$refs.input.focus()  
  }  
}
```



```
this.$refs.usernameInput.focus()
```


- Element & Component Access
 - **Dependency Injection.** Two new instance options: provide and inject.

```
<google-map>
  <google-map-region v-bind:shape="cityBoundaries">
    <google-map-markers v-bind:places="iceCreamShops"></google-map-markers>
  </google-map-region>
</google-map>
```

✓

```
provide: function () {
  return {
    getMap: this.getMap
  }
}
```

✓

```
inject: ['getMap']
```

In fact, you can think of dependency injection as sort of “long-range props” , except:

- ancestor components don’ t need to know which descendants use the properties it provides
- descendant components don’ t need to know where injected properties are coming from

- Programmatic Event Listeners
 - Vue instances offer other methods in its events interface.
 - Listen for an event with `$on(eventName, eventHandler)`
 - Listen for an event only once with `$once(eventName, eventHandler)`
 - Stop listening for an event with `$off(eventName, eventHandler)`

```
// Attach the datepicker to an input once
// it's mounted to the DOM.
mounted: function () {
  // Pikaday is a 3rd-party datepicker library
  this.picker = new Pikaday({
    field: this.$refs.input,
    format: 'YYYY-MM-DD'
  })
},
// Right before the component is destroyed,
// also destroy the datepicker.
beforeDestroy: function () {
  this.picker.destroy()
}
```



```
mounted: function () {
  var picker = new Pikaday({
    field: this.$refs.input,
    format: 'YYYY-MM-DD'
  })

  this.$once('hook:beforeDestroy', function () {
    picker.destroy()
  })
}
```



- Circular References

- **Recursive Components.** Components can recursively invoke themselves in their own template. make sure recursive invocation is conditional

```
name: 'unique-name-of-my-component'
```

```
Vue.component('unique-name-of-my-component', {  
  // ...  
})
```

```
name: 'stack-overflow',  
template: '<div><stack-overflow></stack-overflow></div>'
```



max stack size exceeded

- Circular References
 - Circular References Between Components

```
<p>
  <span>{{ folder.name }}</span>
  <tree-folder-
contents :children="folder.children"/>
</p>

<ul>
  <li v-for="child in children">
    <tree-folder v-
if="child.children" :folder="child"/>
    <span v-else>{{ child.name }}</span>
  </li>
</ul>
```

```
beforeCreate: function () { ✓

this.$options.components.TreeFolderContents
= require('./tree-folder-
contents.vue').default
}
```

```
beforeCreate: function () { ✓

this.$options.components.TreeFolderContents
= require('./tree-folder-
contents.vue').default
}
```

If you're requiring/importing components using a **module system**, e.g. via Webpack or Browserify, you'll get an error.



Failed to mount component: template or render function not defined.

- Alternate Template Definitions
 - **Inline Templates.** When the inline-template special attribute is present on a child component, the component will use its inner content as its template, rather than treating it as distributed content.

```
<my-component inline-template>
  <div>
    <p>These are compiled as the component's own template.</p>
    <p>Not parent's transclusion content.</p>
  </div>
</my-component>
```

- **Alternate Template Definitions**
 - **X-Template.** Another way to define templates is inside of a script element with the type text/x-template, then referencing the template by an id.

```
<script type="text/x-template" id="hello-world-template">  
  <p>Hello hello hello</p>  
</script>
```

```
Vue.component('hello-world', {  
  template: '#hello-world-template'  
})
```

- Controlling Updates
 - Forcing an Update **\$forceUpdate**
 - Cheap Static Components with v-once

```
Vue.component('terms-of-service', {  
  template: `  
    <div v-once>  
      <h1>Terms of Service</h1>  
      ... a lot of static content ...  
    </div>  
  `,  
})
```



- *Web*开发技术
- *Web Application Development*

Thank You!