Advanced Component Usage

Passing Content with Slot

Another way of passing content from parent to child components. In the child component, adding a tag in the template. For example, there are a parent component, **App.vue**, and its child component, **Quote.vue**. In the Quote component, adding the **slot** tag.

```
// Quote.vue
<template>
  <div>
    <slot></slot>
  </div>
</template>
```

In the parent, you can pass *HTML content* to the child component by putting the content in between the child's placeholder (aka selector).

```
// App.vue
<template>
...
    <app-quote >
        <!-- passing content using slot -->
        <h2> {{ quoteTitle }}</h2>
        <p>A nice wonderful super duper Quote</p>
    </app-quote>
...
</template>
```

There are two important rules:

1. Child component's style will be applied to the content passed by slot.
2. Parent component still handles all the updates of the content, not the child.

Using Multiple Slots (Named Slot)

We can use **named slot** to distribute the passing content into different location in child's DOM. In the *child's template*, we add the **name** attribute to the **slot** tag in the child component. In the *parent component template*, we add also the **name** attribute to the DOM in the passing data.

```
// Quote.vue - child
<template>
  <div>
    <div class="title">
      <slot name="title"></slot>
    </div>
    <hr>
    <div>
      <slot name="content"></slot>
    </div>
  </div>
</template>

// App.vue - parent
<template>
...
    <app-quote >
        <!-- passing data with slot -->
        <h2 slot="title"> {{ quoteTitle }}</h2>
        <p slot="content">A nice wonderful super
duper Quote</p>
    </app-quote>
...
</template>
```

Default Slots and Slot Defaults

The content *without* **name** attribute will be rendered in the **default slot** (the slot without the **name** attribute).

```
// Quote.vue - child
<template>
  <div>
    <div class="title">
      <slot name="title"></slot>
    </div>
    <hr>
    <div>
      <slot></slot>  <!-- default slot -->
    </div>
  </div>
</template>

// App.vue - parent
<template>
...
    <app-quote >
        <!-- named content -->
        <h2 slot="title"> {{ quoteTitle }}</h2>
        <!-- unnamed content -->
        <p>A nice wonderful super duper Quote</p>
    </app-quote>
...
</template>
```

It is possible to setup a **default content** in the case that there is no data for a specific **named slot**.

```
<template>
  <div>
    <div class="title">
      <slot name="title"></slot>

      <!-- setup default content here !!!-->
      <span style="color: #ccc"><slot
name="subtitle">The subtitle</slot></span>

    </div>
    <hr>
    <div>
      <slot name="content"></slot>
    </div>
  </div>
</template>
```

Dynamic Components

There is a way to implement dynamic components in Vue.js. The **component** tag will be used for this.

```
// App.vue
<template>
...
    <button @click="selectedComponent =
'appQuote'">Quote</button>
    <button @click="selectedComponent =
'appAuthor'">Author</button>
    <button @click="selectedComponent =
'appNew'">New</button>
    <hr>
    <p> {{ selectedComponent }}</p>

    <!-- implement dynamic components -->
```

```
    <component :is="selectedComponent">
        <p>Default Content</p>
    </component>

...
</template>

<script>
    import Quote from './components/Quote.vue';
    import Author from './components/Author.vue';
    import New from './components/New.vue';

    export default {
        data: function() {
            return {
                quoteTitle: '"The Quote"',
                selectedComponent: 'appQuote'
            }
        },
        components: {
            appQuote: Quote,
            appAuthor: Author,
            appNew: New
        }
    }
</script>
```

Keeping Dynamic Component Alive

Are these components get created everytime? or do we use the existing components? By default, each component is destroyed and recreated evertime. However, it is possible to keep them alive by using the **keep-alive** tag.

```
// App.vue
<template>
```

```
...
    <keep-alive>
        <component :is="selectedComponent">
            <p>Default Content</p>
        </component>
    </keep-alive>
...
</template>
```

Dynamic Component Life cycle hook

There are two new life cycle hooks we can use now: **activated()** and **deactivated()**

```
// App.vue

<script>
  export default {
    data: function() {
      return {
        counter: 0
      }
    },
    // with <keep-alive>, destroyed() is now useless
    destroyed() {
      console.log('Destroyed!');
    },
    deactivated() {
      console.log('deactivated');
    },
    activated() {
      console.log('activated');
    }
  }
</script>
```