v-memo 是做什么的?

官网对 `v-memo` 定义是这样的:

> 缓存一个模板的子树。在元素和组件上都可以使用。为了实现缓存，该指令需要传入一个固定长度的依赖值数组进行比较。如果数组里的每个值都与最后一次的渲染相同，那么整个子树的更新将被跳过。举例来说：

看起来有点绕，但实际上，很好理解。`v-memo` 所做的与我们现有的计算属性一样，只不过 `v-memo` 的对象是 DOM。

这个新指令将缓存它所控制的DOM部分，如果一个特定的值发生变化，只需运行更新并重新渲染。这些值是由我们自己手动设置。

事例

```ruby 复制代码
<template>
  <div>
    ..the rest of the component
    <div v-memo="[myValue]">
      <svg >
        <title>{{MyValue}}</title>
        ...
      </svg>
      <vue-custom-element :value="myValue"></vue-custom-element>
    </div>
  </div>
</template>
```

对上面解释一下： `v-memo` 通常是作为组件的一部分来使用的，它只是影响组件 dom 的一个子集。

```ini 复制代码
<div v-memo="[myValue]">
```

接着，我们将 `v-memo` 分配给了一个特定的 `DIV` 和它的所有子元素。当调用 `v-memo` 时，需要传递一个值数组，以控制子树的渲染。

数组接受一个或多个值 `v-memo="[valueOne, valueTwo]"` ，也接受像 `v-memo="myValue === true"` 这样的表达。

另外：用一个空数组调用 `v-memo` 相当于使用 `v-once` ，只会渲染该部分组件一次。

```ruby 复制代码
<svg >
  <title>{{MyValue}}</title>
  ...
</svg>
<vue-custom-element :value="myValue"></vue-custom-element>
```

同在看下子树的内容。在我们的例子中，使用了一个 svg 元素和一个自定义 Vue 组件 `vue-custom-element` 。这样做是为了说明一件事： `v-memo` 包含任何元素。

错误的使用方式

```css 复制代码
<div v-memo="[myValue]">
  <p>Static content, no vue values here</p>
</div>
```

在上面的例子中，包含在 `v-memo` 中的子树不需要被缓存，因为它是静态的，不会改变（它不包括任何Vue变量）。Vue3 会对静态进行一个提升，以便提高性能。

在一个静态的HTML上添加 `v-memo` 是没啥作用，不管这个HTML有多复杂。

管理更新

在有些情况下， `v-memo` 不仅可以用来提高性能，还可以通过控制组件的更新周期，实际改善UX（用户体验）。

```css 复制代码
<div v-memo="[allFieldChanged]">
  <p>{{ field1 }}</p>
  <p>{{ field2 }}</p>
  <p>{{ field3 }}</p>
  <p>{{ field4 }}</p>
</div>
```

在上面的例子中，改变一个单独的字段，例如 `field1` ，并不会导致重新渲染。新的字段将在所有字段都被更新后显示。

最近遇到一个情况，一个子组件会对一个大的JSON数据集进行更新和响应。在这种情况下，使用 `v-memo` 真的很有帮助，当所有的变化都完成后，就可以触发更新。

与 v-for 结合使用

使用 `v-memo` 的一个最常见的用例是在处理使用 `v-for` 渲染的非常大的列表时。

```css 复制代码
<div v-for="item in list" :key="item.id" v-memo="[item.id === selected]">
  <p>ID: {{ item.id }} - selected: {{ item.id === selected }}</p>
  <p>...more child nodes</p>
</div>
```

如果不在上面的代码中使用 `v-memo` ， `selected` 变量的每一次改变都会导致列表的完全重新渲染。新指令提供的缓存，允许只更新表达式 `item.id === selected` 发生变化的行，也就是当某个项被选中或者取消时。

如果我们考虑一个有 1000 条数据的列表。使用上述代码的 `v-memo` ，可以为每一个变化节省998个条重新渲染。

无意中停止了子组件触发的更新

我们知道 `v-memo` 会停止子树渲染更新，但需要注意的是，使用这个指令实际上会停止任何可能被更新触发的代码的执行，如 watch 函数等。

```xml 复制代码
<div v-memo="[points > 1000]">
  <myComponent :points="points" />
</div>

//myComponent
<isLevel1 v-if="points <= 1000">....</isLevel1>
<isLevel2 v-if="points > 1000">...</isLevel2>
<script>
...,
watch: {
  points() {
    logPointChange();
  }
}
```

在上面的代码中，如果我们的 `points` 值是 1000 以内变化，那么 `watch` 函数不会被执行，直到 `points` 的值大于 1000 才会触发 watch 函数的执行。

总结

这个新的指令对于要求性能极高的项目有很在帮助了，一般是在比较大型的项目中使用的，当然小型项目，大家可以根据项目需要进行食用。