KeepAlive

`<KeepAlive>` is a built-in component that allows us to conditionally cache component instances when dynamically switching between multiple components.

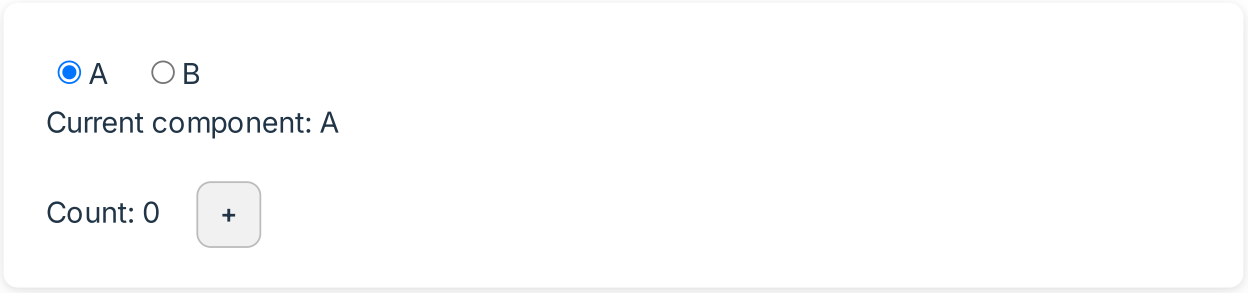
Basic Usage

In the Component Basics chapter, we introduced the syntax for Dynamic Components, using the `<component>` special element:

```template
<component :is="activeComponent" />
```

By default, an active component instance will be unmounted when switching away from it. This will cause any changed state it holds to be lost. When this component is displayed again, a new instance will be created with only the initial state.

In the example below, we have two stateful components - A contains a counter, while B contains a message synced with an input via `v-model`. Try updating the state of one of them, switch away, and then switch back to it:

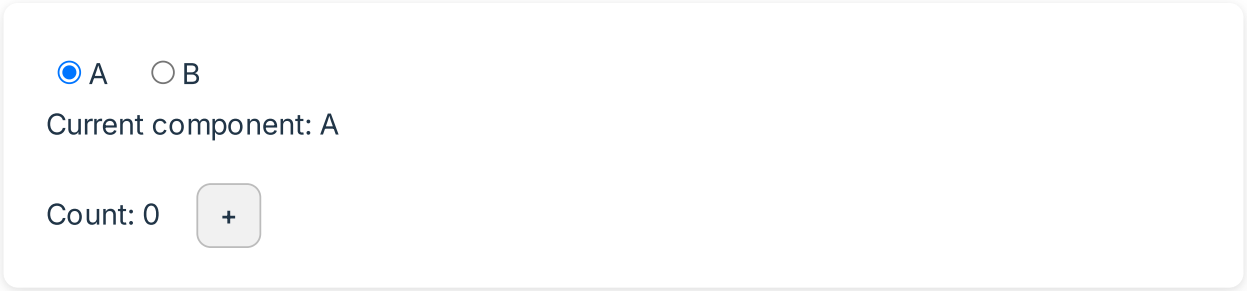


You'll notice that when switched back, the previous changed state would have been reset.

Creating fresh component instance on switch is normally useful behavior, but in this case, we'd really like the two component instances to be preserved even when they are inactive. To solve this problem, we can wrap our dynamic component with the `<KeepAlive>` built-in component:

```
  <component :is="activeComponent" />
</KeepAlive>
```

Now, the state will be persisted across component switches:



Try it in the Playground

> ⓘ TIP
>
> When used in in-DOM templates, it should be referenced as `<keep-alive>`.

Include / Exclude

By default, `<KeepAlive>` will cache any component instance inside. We can customize this behavior via the `include` and `exclude` props. Both props can be a comma-delimited string, a `RegExp`, or an array containing either types:

template

```
<!-- comma-delimited string -->
<KeepAlive include="a,b">
  <component :is="view" />
</KeepAlive>

<!-- regex (use `v-bind`) -->
<KeepAlive :include="/a|b/">
  <component :is="view" />
</KeepAlive>

<!-- Array (use `v-bind`) -->
<KeepAlive :include="['a', 'b']">
  <component :is="view" />
</KeepAlive>
```

The match is checked against the component's `name` option, so components that need to be conditionally cached by `KeepAlive` must explicitly declare a `name` option.

Since version 3.2.34, a single-file component using `<script setup>` will automatically infer its `name` option based on the filename, removing the need to manually declare the name.

Max Cached Instances

We can limit the maximum number of component instances that can be cached via the `max` prop. When `max` is specified, `<KeepAlive>` behaves like an LRU cache: if the number of cached instances is about to exceed the specified max count, the least recently accessed cached instance will be destroyed to make room for the new one.

```template
<KeepAlive :max="10">
  <component :is="activeComponent" />
</KeepAlive>
```

Lifecycle of Cached Instance

When a component instance is removed from the DOM but is part of a component tree cached by `<KeepAlive>`, it goes into a **deactivated** state instead of being unmounted. When a component instance is inserted into the DOM as part of a cached tree, it is **activated**.

A kept-alive component can register lifecycle hooks for these two states using `onActivated()` and `onDeactivated()`:

```vue
<script setup>
import { onActivated, onDeactivated } from 'vue'

onActivated(() => {
  // called on initial mount
  // and every time it is re-inserted from the cache
})

onDeactivated(() => {
  // called when removed from the DOM into the cache
  // and also when unmounted
})
</script>
```

- `onActivated` is also called on mount, and `onDeactivated` on unmount.
- Both hooks work for not only the root component cached by `<KeepAlive>`, but also the descendant components in the cached tree.

Related

- `<KeepAlive>` API reference

Edit this page on GitHub

Previous
TransitionGroup

Next
Teleport