SVELTE • RUNTIME

Lifecycle hooks

ON THIS PAGE

In Svelte 5, the component lifecycle consists of only two parts: Its creation and its destruction. Everything in-between — when certain state is updated — is not related to the component as a whole; only the parts that need to react to the state change are notified. This is because under the hood the smallest unit of change is actually not a component, it's the (render) effects that the component sets up upon component initialization. Consequently, there's no such thing as a "before update"/"after update" hook.

onMount

The `onMount` function schedules a callback to run as soon as the component has been mounted to the DOM. It must be called during the component's initialisation (but doesn't need to live *inside* the component; it can be called from an external module).

`onMount` does not run inside a component that is rendered on the server.

```
<script>
  import { onMount } from 'svelte';

  onMount(() => {
    console.log('the component has mounted');
  });
</script>
```

If a function is returned from `onMount`, it will be called when the component is unmounted.

```
  import { onMount } from 'svelte';

  onMount(() => {
    const interval = setInterval(() => {
      console.log('beep');
    }, 1000);

    return () => clearInterval(interval);
  });
</script>
```

> This behaviour will only work when the function passed to `onMount` *synchronously* returns a value. `async` functions always return a `Promise`, and as such cannot *synchronously* return a function.

onDestroy

Schedules a callback to run immediately before the component is unmounted.

Out of `onMount`, `beforeUpdate`, `afterUpdate` and `onDestroy`, this is the only one that runs inside a server-side component.

```
function onDestroy(fn: () => any): void;
```

Schedules a callback to run immediately before the component is unmounted.

Out of `onMount`, `beforeUpdate`, `afterUpdate` and `onDestroy`, this is the only one that runs inside a server-side component.

```
<script>
  import { onDestroy } from 'svelte';

  onDestroy(() => {
    console.log('the component is being destroyed');
  });
```

While there's no "after update" hook, you can use `tick` to ensure that the UI is updated before continuing. `tick` returns a promise that resolves once any pending state changes have been applied, or in the next microtask if there are none.

```
<script>
  import { tick } from 'svelte';

  $effect.pre(() => {
    console.log('the component is about to update');
    tick().then(() => {
        console.log('the component just updated');
    });
  });
</script>
```

Deprecated: beforeUpdate / afterUpdate

Svelte 4 contained hooks that ran before and after the component as a whole was updated. For backwards compatibility, these hooks were shimmed in Svelte 5 but not available inside components that use runes.

```
<script>
  import { beforeUpdate, afterUpdate } from 'svelte';

  beforeUpdate(() => {
    console.log('the component is about to update');
  });

  afterUpdate(() => {
    console.log('the component just updated');
  });
</script>
```

Instead of `beforeUpdate` use `$effect.pre` and instead of `afterUpdate` use `$effect` instead - these runes offer more granular control and only react to the changes you're actually

To implement a chat window that autoscrolls to the bottom when new messages appear (but only if you were *already* scrolled to the bottom), we need to measure the DOM before we update it.

In Svelte 4, we do this with `beforeUpdate`, but this is a flawed approach — it fires before *every* update, whether it's relevant or not. In the example below, we need to introduce checks like `updatingMessages` to make sure we don't mess with the scroll position when someone toggles dark mode.

With runes, we can use `$effect.pre`, which behaves the same as `$effect` but runs before the DOM is updated. As long as we explicitly reference `messages` inside the effect body, it will run whenever `messages` changes, but *not* when `theme` changes.

`beforeUpdate`, and its equally troublesome counterpart `afterUpdate`, are therefore deprecated in Svelte 5.

Before

After

```
<script>
  import { beforeUpdate, afterUpdate, tick } from 'svelte';

  let updatingMessages = false;
  let theme = $state('dark');
  let messages = $state([]);

  let viewport;

  beforeUpdate(() => {
  $effect.pre(() => {
    if (!updatingMessages) return;
    messages;
    const autoscroll = viewport && viewport.offsetHeight + viewport.scrollTop > viewport.

    if (autoscroll) {
      tick().then(() => {
        viewport.scrollTo(0, viewport.scrollHeight);
```

Docs

```
  });

  function handleKeydown(event) {
    if (event.key === 'Enter') {
      const text = event.target.value;
      if (!text) return;

      updatingMessages = true;
      messages = [...messages, text];
      event.target.value = '';
    }
  }

  function toggle() {
    toggleValue = !toggleValue;
  }
</script>

<div class:dark={theme === 'dark'}>
  <div bind:this={viewport}>
    {#each messages as message}
      <p>{message}</p>
    {/each}
  </div>

  <input onkeydown={handleKeydown} />

  <button onclick={toggle}> Toggle dark mode </button>
</div>
```

Edit this page on GitHub

PREVIOUS
Context

NEXT
Imperative component API