SVELTE • TEMPLATE SYNTAX

transition:

ON THIS PAGE

A *transition* is triggered by an element entering or leaving the DOM as a result of a state change.

When a block (such as an `{#if ...}` block) is transitioning out, all elements inside it, including those that do not have their own transitions, are kept in the DOM until every transition in the block has been completed.

The `transition:` directive indicates a *bidirectional* transition, which means it can be smoothly reversed while the transition is in progress.

```
<script>
  import { fade } from 'svelte/transition';

  let visible = $state(false);
</script>

<button onclick={() => visible = !visible}>toggle</button>

{#if visible}
  <div transition:fade>fades in and out</div>
{/if}
```

Built-in transitions

A selection of built-in transitions can be imported from the `svelte/transition` module.

created or destroyed, *not* when parent blocks are created or destroyed.

```
{#if x}
  {#if y}
    <p transition:fade>fades in and out only when y changes</p>

    <p transition:fade|global>fades in and out when x or y change</p>
  {/if}
{/if}
```

Transition parameters

Transitions can have parameters.

(The double `{{curlies}}` aren't a special syntax; this is an object literal inside an expression tag.)

```
{#if visible}
  <div transition:fade={{ duration: 2000 }}>fades in and out over two seconds</div>
{/if}
```

Custom transition functions

```
transition = (node: HTMLElement, params: any, options: { direction: 'in' | 'out' | 'both'
  delay?: number,
  duration?: number,
  easing?: (t: number) => number,
  css?: (t: number, u: number) => string,
  tick?: (t: number, u: number) => void
}
```

Transitions can use custom functions. If the returned object has a `css` function, Svelte will generate keyframes for a web animation

been applied. *In* transitions run from `0` to `1`, *out* transitions run from `1` to `0` — in other words, `1` is the element's natural state, as though no transition had been applied. The `u` argument is equal to `1 - t`.

The function is called repeatedly *before* the transition begins, with different `t` and `u` arguments.

App.svelte JS TS

```
<script lang="ts">
  import { elasticOut } from 'svelte/easing';

  export let visible: boolean;

  function whoosh(node: HTMLElement, params: { delay?: number, duration?: number, easing?
    const existingTransform = getComputedStyle(node).transform.replace('none', '');

    return {
      delay: params.delay || 0,
      duration: params.duration || 400,
      easing: params.easing || elasticOut,
      css: (t, u) => `transform: ${existingTransform} scale(${t})`
    };
  }
</script>

{#if visible}
  <div in:whoosh>whooshes in</div>
{/if}
```

A custom transition function can also return a `tick` function, which is called *during* the transition with the same `t` and `u` arguments.

> If it's possible to use `css` instead of `tick`, do so — web animations can run off the main thread, preventing jank on slower devices.

App.svelte JS TS

```
    const valid = node.childNodes.length === 1 && node.childNodes[0].nodeType === Node.TE

    if (!valid) {
      throw new Error(`This transition only works on elements with a single text node chi
    }

    const text = node.textContent;
    const duration = text.length / (speed * 0.01);

    return {
      duration,
      tick: (t) => {
        const i = ~~(text.length * t);
        node.textContent = text.slice(0, i);
      }
    };
  }
</script>

{#if visible}
  <p in:typewriter={{ speed: 1 }}>The quick brown fox jumps over the lazy dog</p>
{/if}
```

If a transition returns a function instead of a transition object, the function will be called in the next microtask. This allows multiple transitions to coordinate, making crossfade effects possible.

Transition functions also receive a third argument, `options`, which contains information about the transition.

Available values in the `options` object are:

`direction` - one of `in`, `out`, or `both` depending on the type of transition

Transition events

An element with transitions will dispatch the following events in addition to any standard

introend

outrostart

outroend

```
{#if visible}
  <p
    transition:fly={{ y: 200, duration: 2000 }}
    onintrostart={() => (status = 'intro started')}
    onoutrostart={() => (status = 'outro started')}
    onintroend={() => (status = 'intro ended')}
    onoutroend={() => (status = 'outro ended')}
  >
    Flies in and out
  </p>
{/if}
```

Edit this page on GitHub

PREVIOUS

use:

NEXT

in: and out:

Docs