



Web Application Development mit Svelte

Ronny Wegener

27. Dezember 2020

Inhalt

Svelte - Cybernetically enhanced web apps

Setup - Toolchain & Projekt

Konzepte

Beispiel App

Fazit

Referenzen

Svelte

Cybernetically enhanced web apps

2013



React

2014



Vue.js

2016 (2010)



ANGULAR

2016



SVELTE



Noch ein UI Komponenten Framework?

- Vollständig zu nativem JavaScript kompiliert
- Keine Abhängigkeiten zur Laufzeit
- Direkte DOM Manipulation (statt virtual DOM)
- Weniger Boilerplate
- Reaktivität als Sprachbestandteil

Setup

Toolchain & Projekt



- Entwicklungsumgebung: NodeJS, NPM
- Empfohlen: VS Code (Svelte Extension)
- Projekt mit Template erstellen:

```
npm init svelte
```

- Abhängigkeiten Installieren:

```
npm install
```

- Ready to Go....:

```
npm run dev
```

→ Live Demonstration

→ Erklärung Dependencies / Projekt Struktur

Konzept

Komponenten

- In sich geschlossene Funktionale Einheit
- Schnittstellen zu anderen Komponenten
- Modulare Komposition zu Anwendung (Baukasten Prinzip)
- Svelte Komponente → Syntax HTML basiert

A screenshot of the Svelte Live Demo web application. The interface is split into two main sections. The left section is a code editor with a light blue background, showing the Svelte component code for 'App.svelte'. The code includes a script block with a variable 'count' set to 0, a style block with a paragraph color set to #888, and a template block with an h1 'Welcome', an hr, and a p 'To Svelte Live Demo ...'. The right section shows the rendered output, with tabs for 'Result', 'JS output', and 'CSS output'. The 'Result' tab is active, displaying the rendered HTML: a large 'Welcome' heading followed by a horizontal line and a paragraph 'To Svelte Live Demo ...'.

```
App.svelte +
1 <script>
2   let count = 0;
3 </script>
4
5 <style>
6   p {
7     color: #888;
8   }
9 </style>
10
11 <h1>
12   Welcome
13 </h1>
14 <hr>
15 <p>
16   To Svelte Live Demo ...
17 </p>
```

Result JS output CSS output

Welcome

To Svelte Live Demo ...

Konzept

Properties & Slots

- Komponente exportiert Properties
- Verwendung der Komponente: Attribute setzen Properties
- Komponenten können Inhalte haben
- Zugriff auf Inhalte über `slot` Platzhalter

App.svelte	Greeting.svelte	+	Result	JS output	CSS output
<pre>1 <script> 2 import Greeting from './Greeting.svelte'; 3 </script> 4 5 <h1> 6 <Greeting language="fr">Alice</Greeting> 7
 8 <Greeting language="en">Bob</Greeting> 9 </h1> 10</pre>			Bonjour Alice Hello Bob		

Konzept

Reactive Statements & Bindings

- Synchronisation von Assignments {value}
- Synchronisation von Abhängigkeiten \$:
- Datenfluss standardmäßig Top Down
- Binding: Two Way Datenfluss



```
App.svelte +
1 <script>
2   let count = 0;
3   let oneway = 'One Way ...';
4   let twoway = 'Two Way ...';
5   $: square = count * count;
6 </script>
7
8 <style>
9   button {
10     min-height: 1.5em;
11     min-width: 2em;
12     cursor: pointer;
13   }
14 </style>
15
16 <button on:click={() => count++}>Clicked: {count}</button> Squared: {square}
17
18 <div>
19   One Way: <input type="text" value={oneway}/>
```

Result JS output CSS output

Clicked: 5 Squared: 25

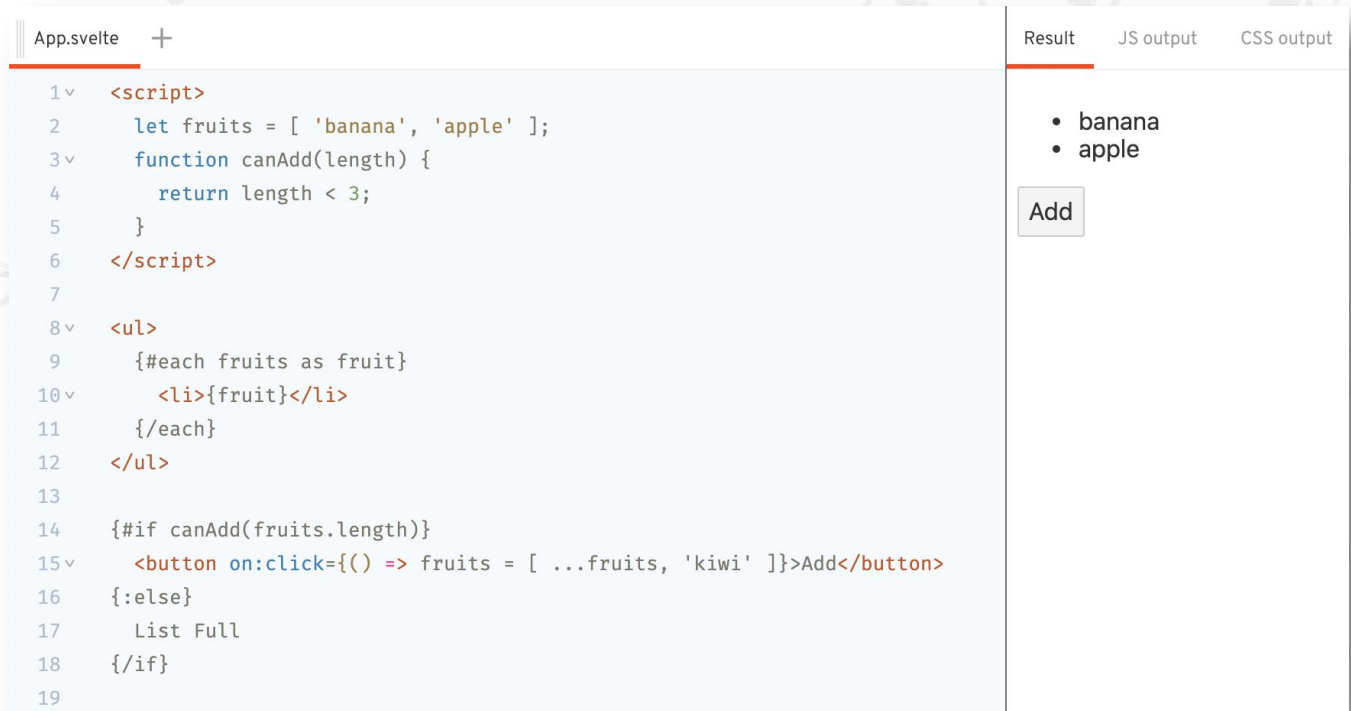
One Way: One Way ... One Way ...

Two Way: Two Way ... Two Way ...

Konzept

Logik Blöcke

- HTML Dynamisch Evaluieren
- Fallunterscheidungen, Listen von Items
- **TIP:** Modifikatoren (e.g. Array.push) sind nicht, Re-Assignment Trick nutzen



The screenshot shows a Svelte REPL interface. The left pane contains Svelte code for an application named 'App.svelte'. The code defines a 'fruits' array with 'banana' and 'apple', a 'canAdd' function that returns true if the array length is less than 3, and a list of fruits. It also includes a conditional button that adds 'kiwi' to the fruits array when clicked, or displays 'List Full' otherwise. The right pane shows the 'Result' tab, which displays the rendered HTML: a bulleted list with 'banana' and 'apple', and an 'Add' button below it.

```
App.svelte +
1 <script>
2   let fruits = [ 'banana', 'apple' ];
3   function canAdd(length) {
4     return length < 3;
5   }
6 </script>
7
8 <ul>
9   {#each fruits as fruit}
10    <li>{fruit}</li>
11  {/each}
12 </ul>
13
14 {#if canAdd(fruits.length)}
15   <button on:click={() => fruits = [ ...fruits, 'kiwi' ]}>Add</button>
16 {/if}
17   List Full
18 {/if}
19
```

Result JS output CSS output

- banana
- apple

Add

Konzept

Stores

- Globaler Zustände / Datenaustausch
- Abo Model mit `subscribe`
- Änderung über `set` oder `update`
- Autosubscriptions mit `$` Prefix



```
App.svelte Random.svelte stores.js +
1 <script>
2   import { onDestroy } from 'svelte';
3   import { random } from './stores.js';
4   import Random from './Random.svelte';
5
6   let num;
7   const unsubscribe = random.subscribe(value => num = value);
8   onDestroy(unsubscribe);
9 </script>
10
11 <div>Subscribed: {num}</div>
12 <div>Referenced: {$random}</div>
13 <Random></Random>
14
```

Result JS output CSS output

Subscribed: 0
Referenced: 0

Change

[illegible]

→ Live Demonstration

Fazit

Pro:

- Syntax Einfach und Verständlich
- Typescript Support

Con:

- 3rd Party Library Import schwierig (Rollup)
- Debugging Probleme (Source Mapping)
- Root Element Styling nicht möglich

Ausblick:

- Svelte-Kit in Entwicklung (Routing, SSR)
- Snowpack Integration (Stateful HMR)
- UI Bibliotheken kommen (Material, Carbon)

Referenzen

- [Klassische Dokumentation](#)
- [Interaktives Tutorial](#)
- [REPL \(Playground\)](#)
- [The Svelte Handbook](#), *Flavio Copes*