Svelte

1. Introduction
2. Why Svelte
3. Svelte Basic Concept
4. How to Setup Svelte
5. Example of Svelte

1. Introduction

Introduction

Svelte is a free and open-source front end JavaScript framework.

Created by Rich Harris and maintained by Harris and other Svelte core team members.

Building a Svelte application generates code to manipulate the DOM, which may reduce the size of transferred files as well as give better client startup and run-time performance.

Svelte has its own compiler for converting app code into client-side JavaScript at build time which is written in TypeScript.

History

The predecessor of Svelte is Ractive.js.

Version 1 was written in JavaScript and was released in 29 November 2016.

Version 2 was released in 19 April 2018.

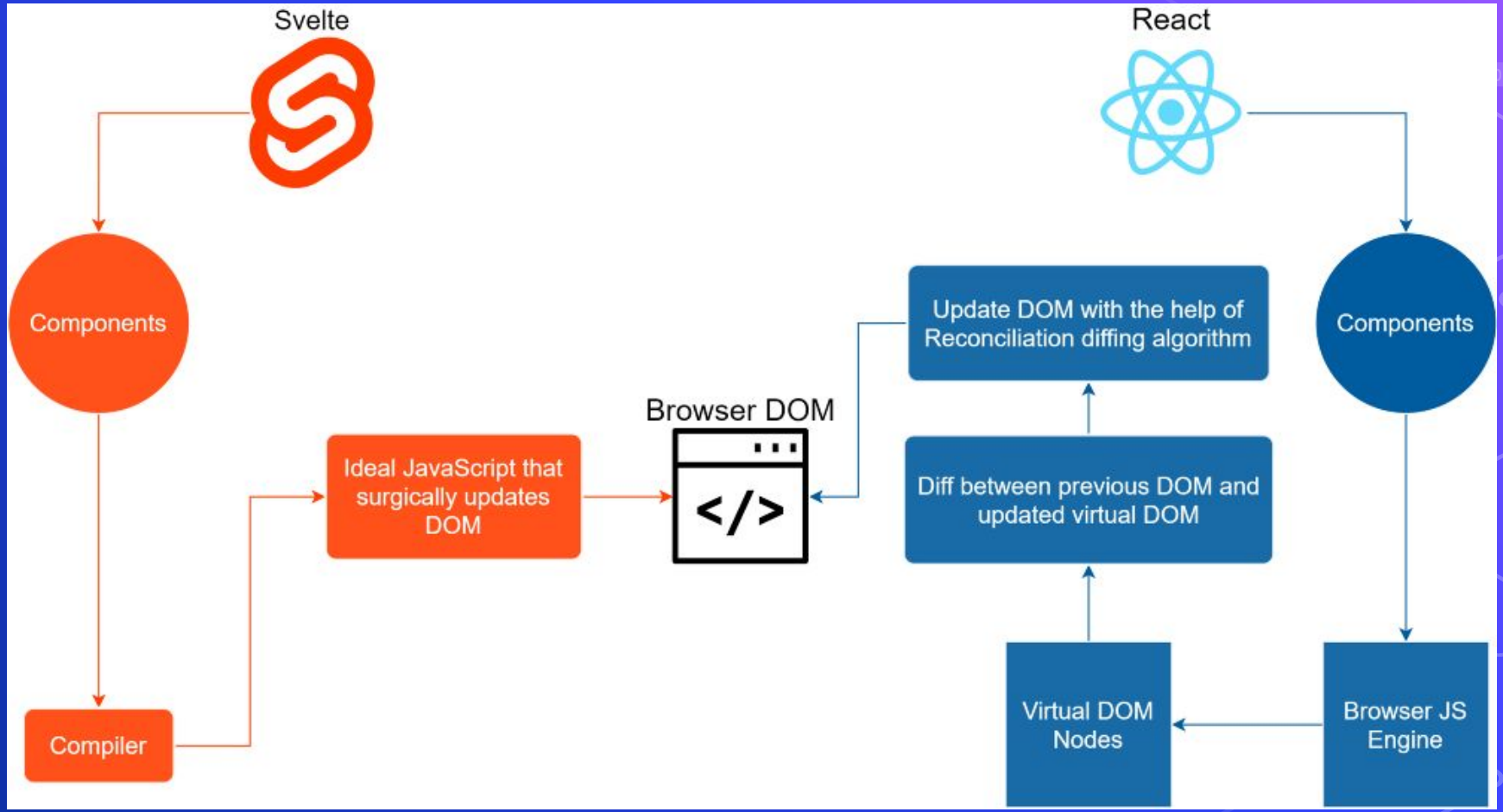
Version 3 is written in TypeScript and was released in 21 April 2019.

2. Why Svelte

React VS Svelte

Both provide a similar component-based architecture — that means both enable a bottom-up development, and both enable sharing their components between apps, via tools and platforms.

The difference between them is that Svelte is a compiler that converts your application into ideal JavaScript during build time as opposed to React, which uses a virtual DOM to interpret the application code during runtime.


Svelte
React
Components
Components
Update DOM with the help of Reconciliation diffing algorithm
Browser DOM
Ideal JavaScript that surgically updates DOM
</>
Diff between previous DOM and updated virtual DOM
Compiler
Virtual DOM Nodes
Browser JS Engine

“ Where Svelte is Strong

- Building Time
- Bundle Size
- Binding classes and variables
- Scoping CSS <style> within the component
- Easier to understand and get started
- More straightforward store implementation

“ Where Svelte Falls Behind

- Svelte won’t listen for reference updates and array mutations
- To use special syntaxes such as #if and #each within template
- Usage style for DOM events (onClick VS on:click)
- New and young framework with minimal community support
- Easier to understand and get started
- No additional improvements

Github

	React	Vue	Angular	Svelte	Ember
Watch	6.8K	6.4K	3.2K	786	958
Star	160K	176K	68.3K	39.8K	21.7K
Fork	31.7K	27.3K	18.1K	1.9K	4.2K

3. Svelte Concept

COMPONENT FORMAT

```
<script>
    // logic goes here
</script>
<style>
    /* styles go here */
</style>
<!-- markup (zero or more items) goes here -->
```

<Script>

A <script> block contains JavaScript that runs when a component instance is created. Variables declared (or imported) at the top level are 'visible' from the component's markup.

1. export creates a component prop

```
<script>
	// these are readonly
	export const thisIs = 'readonly';

	export function greet(name) {
		alert(`hello ${name}!`);
	}

	// this is a prop
	export let format = n => n.toFixed(2);
</script>
```

2. Assignments are 'reactive'

```
<script>
	let count = 0;

	function handleClick () {
		// calling this function will trigger an
		// update if the markup references `count`
		count = count + 1;
	}
</script>
```

3. $: marks a statement as reactive

```
<script>
	export let num;

	// we don't need to declare `squared` and `cubed`
	// — Svelte does it for us
	$: squared = num * num;
	$: cubed = squared * num;
</script>
```

4. Prefix stores with $ to access their values

```
<script>
    import { writable } from 'svelte/store';

    const count = writable(0);
    console.log($count); // logs 0

    count.set(1);
    console.log($count); // logs 1

    $count = 2;
    console.log($count); // logs 2
</script>
```

<Style>

CSS inside a <style> block will be scoped to that component.

To apply styles to a selector globally, use the :global(...) modifier.

```
<style>
p {
	/* only affect <p> in component */
	color: burlywood;
}

:global(body) {
	/* this will apply to <body> */
	margin: 0;
}

</style>
```

Attributes and props

<input type=checkbox> // unquoted

page {p} // contain JS expressions

<Widget {...things}/> // Spread Attribute

{expression} // Text Expression

<!-- this is a comment! -->

Svelte Block

```
{#if porridge.temperature > 100}
	<p>too hot!</p>
{:else if 80 > porridge.temperature}
	<p>too cold!</p>
{:else}
	<p>just right!</p>
{/if}
```

```
<ul>
	{#each items as item}
		<li>{item.name} x {item.qty}</li>
	{/each}
</ul>
```

```
{#await promise}
	<p>waiting to resolve...</p>
{:then value}
	<p>The value is {value}</p>
{:catch error}
	<p>{error.message}</p>
{/await}
```

```
{@html post.content}
```

```
{@debug user}
```

Element directives

```
on:eventname={handler}
on:eventname|modifiers={handler}

bind:property={variable}

bind:group={variable}

class:name={value}

use:action={parameters}
action = (node: HTMLElement, parameters: any) => {
        update?: (parameters: any) => void,
        destroy?: () => void
}
```

Element directives

transition:fn={params}

```
transition = (node: HTMLElement, params: any) => {
	delay?: number,
	duration?: number,
	easing?: (t: number) => number,
	css?: (t: number, u: number) => string,
	tick?: (t: number, u: number) => void
}
```

```
<div transition:fade>
	fades in and out
</div>
```

in:fn|local={params}

out:fn|local={params}

animate:name={params}

Slot

```
<slot name="x"><!-- optional fallback --></slot>
<slot prop={value}></slot>

<svelte:self>

<svelte:component this={expression}/>

<svelte:window on:event={handler}/>

<svelte:body on:event={handler}/>

<svelte:head>...</svelte:head>
```

RUN TIME

Svelte - onMount / beforeUpdate / afterUpdate / onDestroy / createEventDispatcher / getContext / setContext / hasContext

svelte/store - writable / readable / derived / get

svelte/motion - tweened / spring

svelte/transition - fade, blur, fly, slide, scale, draw and crossfade

svelte/animate - flip

svelte/easing

svelte/register

4. How to Setup Svelte

```
npx degit sveltejs/template my-svelte-project
cd my-svelte-project
npm install
npm run dev
```

5. Example

Example1

```
<script>
    import { spring } from 'svelte/motion';

    let coords = spring({ x: 50, y: 50 }, {
        stiffness: 0.1,
        damping: 0.25
    });

    let size = spring(10);
</script>


<style>
    svg { width: 100%; height: 100%; }
    circle { fill: #ff3e00 }
</style>
```

```
<div>
    <label>
        <h3>stiffness ({coords.stiffness})</h3>
        <input bind:value={coords.stiffness} type="range"
min="0" max="1" step="0.01">
    </label>
</div>


<svg
    on:mousemove="{e => coords.set({ x: e.x, y: e.y })}"
    on:mousedown="{() => size.set(30)}"
    on:mouseup="{() => size.set(10)}">
        <circle cx={$coords.x} cy={$coords.y} r={$size}/>
</svg>
```

Example2

```
<script>
    import { count } from './stores.js';
    import Incrementer from './Incrementer.svelte';
    let count_value;

    const unsubscribe = count.subscribe(value => {
        count_value = value;
    });
</script>

<h1>The count is {count_value}</h1>

<Incrementer />
```

```
<script>
    import { count } from './stores.js';

    function increment() {
        count.update(n => n + 1);
    }
</script>

<button on:click={increment}>
    +
</button>
```

```
import { writable } from 'svelte/store';

export const count = writable(0);
```

Thanks!

Any questions?