MAGIC OF SVELTE COMPILER

Created by Mostafa Kheibary | Presented in tehlug 2024

SO, WHAT EXACTLY IS SVELTE ?

WHY IS IT SO POPULAR?

Svelte is a JavaScript framework that allows developers to build scalable and modular projects using a component-based system.

It’s similar to frameworks like React, Vue, and Angular, but with a key difference: Svelte doesn’t use a virtual DOM or shadow DOM.

I’ll explain more about this in the presentation.

THE BIRTH OF SVELTE

Svelte was created 8 years ago on Thanksgiving Day in 2016.

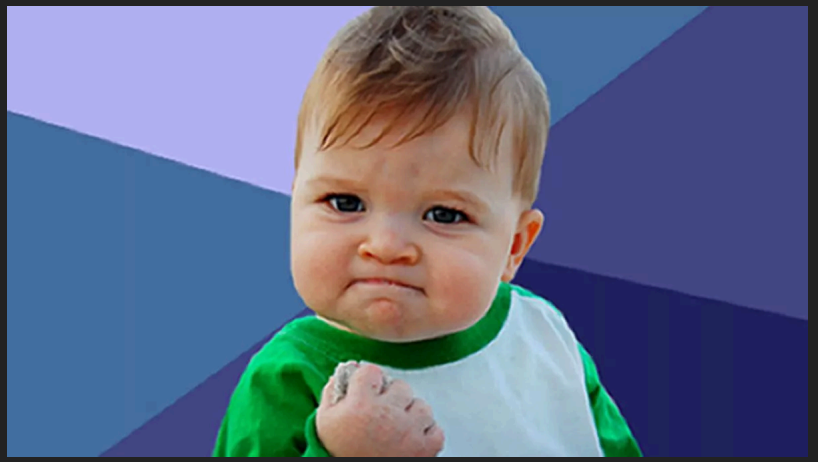
Rich Harris was frustrated with traditional frameworks and how they shipped large amounts of JavaScript to the client just to print a simple "Hello, World".

On that day, Rich Harris started a project that eventually became Svelte.

WHY IS SVELTE SO POPULAR?

- Svelte is a language that compiled to web components and native code
- It doesn't use a shadow DOM.
- Simple and intuitive reactivity system.
- Produces small JavaScript bundles: Svelte (1.6 KB) vs React (40 KB).

"It's best to learn how to use something before you figure out how it works."



SO, LETS BUILD A SIMPLE COUNTER APP

```
1 <script>
2   ....
3 </script>
4
5 <h1>Count : ...</h1>
6 <button>clicks</button>
```

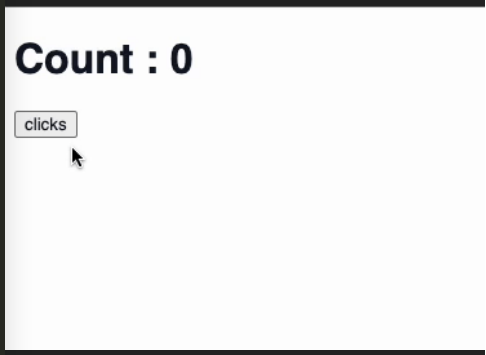
```
<script>
  let count = 0

  const handleIncrement = ()=> {
    count++;
  }
</script>

<h1>Count : ...</h1>
<button>clicks</button>
```

```
<script>
  let count = 0

  const handleIncrement = ()=> {
    count++;
  }
</script>

<h1>Count : ...</h1>
<button>clicks</button>
```

```
<script>
  let count = 0

  const handleIncrement = ()=> {
    count++;
  }
</script>

<h1>Count : {count}</h1>
<button on:click="{handleIncrement}">clicks</button>
```

COUNTER.SVELTE

```svelte
<script>
  let count = 0

  const handleIncrement = ()=> {
    count++;
  }
</script>

<h1>Count : {count}</h1>
<button on:click="{handleIncrement}">clicks</button>
```



HOW SVELTE CAN FIND OUT THAT THE COUNT IS CHANGE TO UPDATE?

```
1 const element = document.createElement('h1')
2 element.textContent = "Hello World"
3 document.body.appendChild(element)
```

It means we can write <>{someValue} declaratively and and we don't need to write imperative statements like element.textContent = someValue every time someValue changes. Svelte generates the synchronization code for us.

SVELTE COMPILER STAGES

- 1. Parsing JavaScript, CSS, and HTML
- 2. Placing Macros
- 3. Generating the Abstract Syntax Tree (AST)
- 4. Generating Optimized JavaScript Code

STAGE 1: PARSING JAVASCRIPT, HTML, AND CSS

The first step in Svelte's compilation process is to parse the different sections of the `.svelte` file:

- **JavaScript:** Extracts variables, functions, and reactive statements using `acorn` or a custom tokenizer.
- **HTML:** Parses the DOM structure (e.g., <h1>, <p>) using `parse5` or similar tools.
- **CSS:** Parses styles and ensures they are scoped by using tools like `css-tree` to tokenize and modify selectors.

Each section is tokenized into an Abstract Syntax Tree (AST), allowing the compiler to analyze and transform it later.

STAGE 2: PLACING MACROS FOR REACTIVITY

In Svelte, whenever a primitive value is modified (e.g., using =, +=, -=), the compiler inserts macros to ensure the DOM updates automatically.

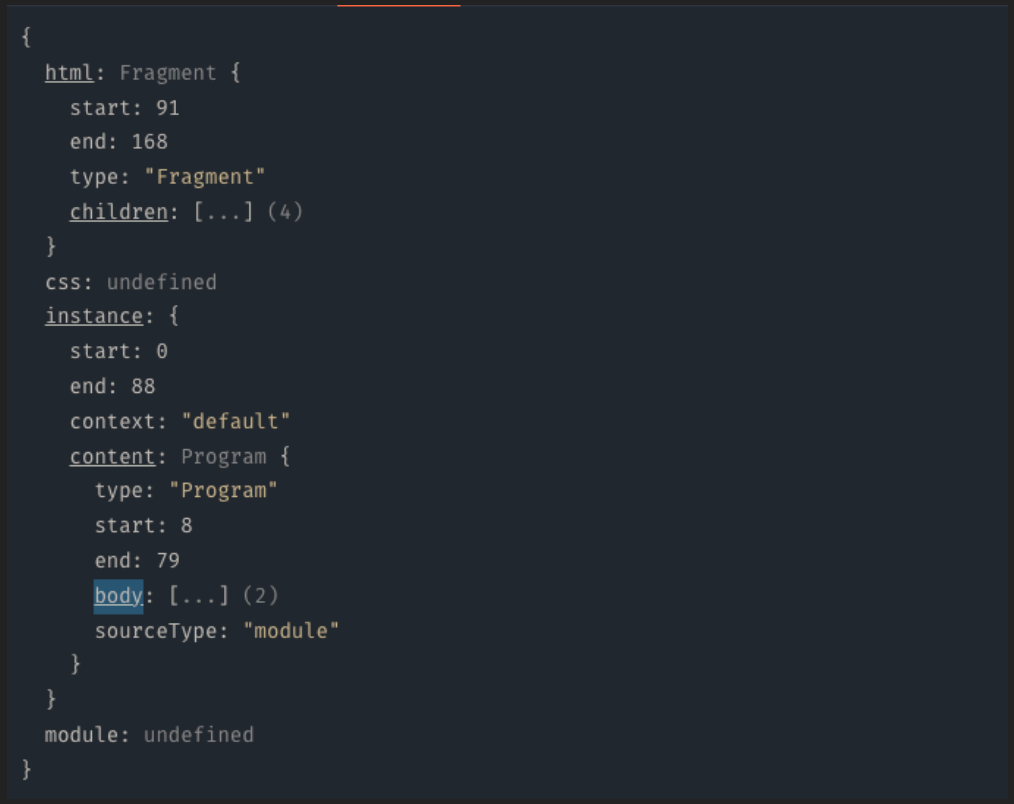
```
<script>
  let count = 0

  const handleIncrement = ()=> {
    count++;
  }
</script>

<h1>Count : {count}</h1>
<button on:click="{handleIncrement}">clicks</button>
```

STAGE 2: PLACING MACROS FOR REACTIVITY

In Svelte, whenever a primitive value is modified (e.g., using =, +=, -=), the compiler inserts macros to ensure the DOM updates automatically.

```
<script>
  let count = internalState(0)

  const handleIncrement = ()=> {
    setUpdateFunction('count',count+1)
  }
</script>

<h1>Count : {count}</h1>
<button on:click="{handleIncrement}">clicks</button>
```

STAGE 3: GENERATING THE ABSTRACT SYNTAX TREE (AST)



STAGE 3: GENERATING THE ABSTRACT SYNTAX TREE (AST)

```
body: [
  VariableDeclaration {
    type: "VariableDeclaration"
    start: 11
    end: 24
    declarations: [
      VariableDeclarator {
        type: "VariableDeclarator"
        start: 15
        end: 24
        id: Identifier {...}
        init: Literal {...}
      }
    ]
    kind: "let"
  }
  VariableDeclaration {
    type: "VariableDeclaration"
    start: 30
    end: 77
    declarations: [...] (1)
    kind: "const"
  }
]
```

STAGE 4: CODE GENERATION

```
export default function component({ target }) {


// variables are declared for each element and text node:
let e0,e1, t1, b2, t3;
let c0=0;

function handleIncrement() {
  update({ count: c0++ });
}
return {
  create() {
    e0 = document.createElement("h1");
    e1 = document.createElement("button");
    t1 = document.createTextNode("Count ");
    t3 = document.createTextNode("clicks");

    e1.addEventListener("click", handleIncrement);
  },

```

```
export default function component({ target }) {


// variables are declared for each element and text node:
let e0,e1, t1, b2, t3;
let c0=0;

function handleIncrement() {
  update({ count: c0++ });
}
return {
  create() {
    e0 = document.createElement("h1");
    e1 = document.createElement("button");
    t1 = document.createTextNode("Count ");
    t3 = document.createTextNode("clicks");

    e1.addEventListener("click", handleIncrement);
  },

```

```
export default function component({ target }) {


// variables are declared for each element and text node:
let e0,e1, t1, b2, t3;
let c0=0;

function handleIncrement() {
  update({ count: c0++ });
}
return {
  create() {
    e0 = document.createElement("h1");
    e1 = document.createElement("button");
    t1 = document.createTextNode("Count ");
    t3 = document.createTextNode("clicks");

    e1.addEventListener("click", handleIncrement);
  },

```

```
6   let c0=0;
7
8   function handleIncrement() {
9     update({ count: c0++ });
10  }
11  return {
12    create() {
13      e0 = document.createElement("h1");
14      e1 = document.createElement("button");
15      t1 = document.createTextNode("Count ");
16      t3 = document.createTextNode("clicks");
17
18      e1.addEventListener("click", handleIncrement);
19    },
20
21    mount() {
22      e0.appendChild(t1);
23      e0.appendChild(b2);
24      e1.appendChild(t3);
25
```

```
15      t1 = document.createTextNode("Count ");
16      t3 = document.createTextNode("clicks");
17
18      e1.addEventListener("click", handleIncrement);
19    },
20
21    mount() {
22      e0.appendChild(t1);
23      e0.appendChild(b2);
24      e1.appendChild(t3);
25
26      target.append(e0);
27      target.append(e1);
28    },
29
30    update(changes) {
31      // check if name changed
32      if (changes.count) {
33        // update `name` variable and all binding to `name`
34        b2.data = c0 = changes.count;
```

```
    e0.appendChild(b2);
    e1.appendChild(t3);

    target.append(e0);
    target.append(e1);
  },

  update(changes) {
    // check if name changed
    if (changes.count) {
      // update `name` variable and all binding to `name`
      b2.data = c0 = changes.count;
    }
  },

  // called to remove the component from the DOM
  detach() {
    e1.removeEventListener("click", handleIncrement);
    target.removeChild(e0);
    target.removeChild(e1);
  }
```

```
export default function component({ target }) {


// variables are declared for each element and text node:
let e0,e1, t1, b2, t3;
let c0=0;

function handleIncrement() {
  update({ count: c0++ });
}
return {
  create() {
    e0 = document.createElement("h1");
    e1 = document.createElement("button");
    t1 = document.createTextNode("Count ");
    t3 = document.createTextNode("clicks");

    e1.addEventListener("click", handleIncrement);
  },

```

```
      target.append(e0);
      target.append(e1);
    },

    update(changes) {
      // check if name changed
      if (changes.count) {
        // update `name` variable and all binding to `name`
        b2.data = c0 = changes.count;
      }
    },

    // called to remove the component from the DOM
    detach() {
      e1.removeEventListener("click", handleIncrement);
      target.removeChild(e0);
      target.removeChild(e1);
    },
  };
}
```

```
export default function component({ target }) {


// variables are declared for each element and text node:
let e0,e1, t1, b2, t3;
let c0=0;

function handleIncrement() {
  update({ count: c0++ });
}
return {
  create() {
    e0 = document.createElement("h1");
    e1 = document.createElement("button");
    t1 = document.createTextNode("Count ");
    t3 = document.createTextNode("clicks");

    e1.addEventListener("click", handleIncrement);
  },

```

APP.JS

```
1 import Counter from './counter'
2
3 const component = Counter({
4   target: document.body,
5 })
6
7 component.create()
8
9 component.mount()
```

AFTER YOU INITIATE THE ROOT COMPONENT, EVERY CHILD WILL FALLOW THIS EXACT PATTERN AND MOUNT THEIR CHILDREN

AND THATS IT

But don't forget we just show you the basics and the svelte compiler have so many feature that is gonna take time to have it in this presentation

BUILD YOUR OWN COMPILER IN JS


THANK YOU
FOR YOUR ATTENTION

CONTACT ME :)

- linkedin : mostafa kheibary
- twitter : mosii_kh