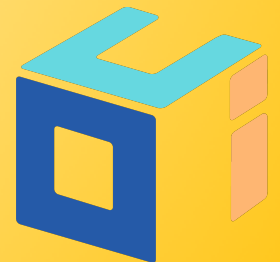




# Svelte

slides at <https://github.com/mvolkmann/talks>

**R. Mark Volkmann**  
Object Computing, Inc.  
<http://objectcomputing.com>  
Email: [mark@objectcomputing.com](mailto:mark@objectcomputing.com)  
Twitter: @mark\_volkman  
GitHub: mvolkmann



OCI | TRAINING

# What Is It?

- Alternative to web frameworks like React, Vue, and Angular
- A web application **compiler**, not a runtime library
  - implemented in TypeScript
  - compiles `.svelte` files to a single JavaScript file
  - no Svelte runtime **dependencies**, only **devDependencies**
- Doesn't use a virtual DOM
- Developed by **Rich Harris**
  - formerly at "The Guardian"; currently at "The New York Times"
  - previously created **Ractive** web framework - <https://ractive.js.org/>
    - used at "The Guardian"
    - inspired parts of Vue
  - created **Rollup** module bundler - <https://rollupjs.org/>
    - alternative to Webpack and Parcel

# Syntax Podcast Quotes

- From Scott Tolinski
- Currently creating a course on Svelte
- In episode 182: "Practical - How would we build Airbnb, Twitter, or Reddit?"
  - "I love Svelte man!"
  - "I just want to do everything in that now."
  - "There's so many cool little things about it."
  - "I can't believe what we're able to accomplish with so little code."
  - "It's really super fantastic!"

# An Example

- Since you are all experienced web developers,  
**let's look at an example app**  
so you can **compare** the look of the code  
to your current favorite web framework
- On to the classic ... todo app!

# Todo App ...

code and tests at  
<https://github.com/mvolkmann/svelte-todo>

## To Do List

1 of 2 remaining Archive Completed

Add

☒ learn Svelte Delete

☐ build a Svelte app Delete

a Todo component

```
src/main.js
import TodoList from './TodoList.svelte';

const app = new TodoList({target: document.body});

export default app;
```

# ... Todo App ...

script and  
style sections  
are optional

```
1 <script>
2   import {createEventDispatcher} from 'svelte';
3   import {fade} from 'svelte/transition';
4   const dispatch = createEventDispatcher();
5   export let todo; // the only prop
6 </script>
7
8 <style>
9   .done-true {
10     color: gray;
11     text-decoration: line-through;
12   }
13   li {
14     margin-top: 5px;
15   }
16 </style>
17
18 <li transition:fade>
19   <input
20     type="checkbox"
21     checked={todo.done}
22     on:change={ () => dispatch('toggleDone') }
23   />
24   <span class={'done-' + todo.done}>{todo.text}</span>
25   <button on:click={ () => dispatch('delete') }>Delete</button>
26 </li>
```

src/Todo.svelte

export makes it a prop

What is the name of this component?  
Can't tell.  
Names are assigned when other  
components import this one.

interpolation

# ... Todo App ...

src/TodoList.svelte

```
1 <script>
2   import Todo from './Todo.svelte';
3
4   let lastId = 0;
5   const createTodo = (text, done = false) => ({id: ++lastId, text, done});
6
7   let todoText = '';
8   let todos = [
9     createTodo('learn Svelte', true),
10    createTodo('build a Svelte app')
11  ];
12
13  $: uncompletedCount = todos.filter(t => !t.done).length;
14  $: status = `${uncompletedCount} of ${todos.length} remaining`;
15
16  function addTodo() {
17    todos = todos.concat(createTodo(todoText));
18    todoText = '';
19  }
20
21  const archiveCompleted = () => todos = todos.filter(t => !t.done);
22
23  const deleteTodo = todoId => todos = todos.filter(t => t.id !== todoId);
24
25  function toggleDone(todo) {
26    const {id} = todo;
27    todos = todos.map(t => t.id === id ? {...t, done: !t.done} : t);
28  }
29 </script>
```

Top-level variables ARE the component state!

reactive declarations

No methods, just functions.

Not really archiving in this simple implementation, just deleting.

No this anywhere, just plain functions!

# ... Todo App ...

```
1 <style>                                     src/ToDoList.svelte
2   button {
3     margin-left: 10px;
4   }
5
6   /* This removes bullets from a bulleted list. */
7   ul.unstyled {
8     list-style: none;
9     margin-left: 0;
10    padding-left: 0;
11  }
12 </style>
```



# ... Todo App

```
1 <div>
2   <h2>To Do List</h2>
3   <div>
4     {status}
5     <button on:click={archiveCompleted}>Archive Completed</button>
6   </div>
7   <form on:submit|preventDefault>
8     <input
9       type="text"
10      size="30"
11      autofocus
12      placeholder="enter new todo here"
13      bind:value={todoText}
14    />
15    <button disabled={!todoText} on:click={addTodo}>
16      Add
17    </button>
18  </form>
19  <ul class="unstyled">
20    {#each todos as todo}
21      <Todo
22        todo={todo}
23        on:delete={ () => deleteTodo (todo.id) }
24        on:toggleDone={ () => toggleDone (todo) }
25      />
26    {/each}
27  </ul>
28 </div>
```

src/ToDoList.svelte

not doing anything on submit

binds value of form element to a variable;  
simulates two-way data binding;  
provides current value and  
event handling for updating variable  
when user changes value

Mustache-style markup

# Logic in Markup

- Three approaches for conditional and iteration logic

- **React**

- uses JSX where logic is implemented by JavaScript code in curly braces

- **Angular and Vue**

- support framework-specific attributes for logic
  - ex. `ngIf`, `ngFor`, `v-if`, `v-for`, ...

- **Svelte**

- supports mustache-like custom syntax that wraps elements
  - ex. `{#if}` and `{#each}`
  - can wrap multiple elements without introducing a new, common parent

Why does it make sense to specify conditional and iteration logic INSIDE elements using attributes?

Imagine if you could do that with JavaScript functions.

```
doSomething(  
  arg1,  
  arg2,  
  if (arg1 > 10) ,  
  for (arg1 in someCollection));
```

Isn't that weird?

# Top Svelte Features

- It's fast!
  - see <https://krausest.github.io/js-framework-benchmark/current.html>
  - can select frameworks to compare
- Small bundle sizes
- File-based component definitions
- CSS scoped by default
- Clear place to put global CSS
- Easy component state management (reactivity)
- Reactive statements (\$ : )
- Easy app state management (stores)
- Easy way to pass data from components to descendant components (context)

# Small Bundle Sizes

- Delivered code is much smaller, so loads faster in browsers
- Uses Rollup by default for module bundling, but can also use Webpack or Parcel
- Create production build with `npm run build`
- A RealWorld Comparison of Front-End Frameworks with Benchmarks
  - <https://www.freecodecamp.org/news/a-realworld-comparison-of-front-end-frameworks-with-benchmarks-2019-update-4be0d3c78075/>

Gzipped App Size in KBs

**Angular+ngrx:** 134  
**React+Redux:** 193  
**Vue:** 41.8  
**Svelte:** 9.7

Lines of Code

**Angular+ngrx:** 4210  
**React+Redux:** 2050  
**Vue:** 2076  
**Svelte:** 1116

# File-based Component Defs

- **Angular** uses classes
- **React** uses functions or classes
- **Vue** uses object literals
- **Svelte** doesn't use any JavaScript container
  - JavaScript, CSS, and HTML in source files are combined to form the component definition which automatically becomes the default export
  - name is associated when imported and must start uppercase
  - lowercase names are reserved
    - for predefined elements like those in HTML and SVG

# CSS

- Scoped by default
  - CSS specified in a component `style` tag is automatically scoped to the component
  - achieved by adding the same generated CSS class name, `svelte-hash`, to each rendered element of the component affected by these CSS rules
- Clear place for global CSS
  - `public/global.css`

# Easy Component State Mgmt.

("reactivity")

- Changes to top-level variables referenced in interpolations automatically cause those interpolations to be reevaluated
- Example

```
<script>
  let count = 0;
  const increment = () => count++;
</script>

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

- Must assign a new value to trigger
  - pushing new elements onto an array doesn't do this

```
myArr = myArr.concat(newValue);
```

works

```
myArr = [...myArr, newValue];
```

works

```
// Alternative trick
myArr.push(newValue);
myArr = myArr;
```

works

# Reactive Statements

a.k.a. "destiny operator"

- `$:` is a "labeled statement" with label name "`$`" that Svelte treats as a "reactive statement"
- Add as a prefix on top-level statements that should be repeated whenever any referenced variables change
- Examples

Labeled statements can be used as targets of `break` and `continue` statements. It is not an error in JavaScript to use same label more than once in same scope.

```
$: average = total / count;  
$: console.log('count =', count);
```

like "computed properties" in Vue

great for debugging

When applied to an assignment to an undeclared variable it is called a "reactive declaration" and the `let` keyword is not allowed.

- Can apply to a block

```
$: {  
  // statements to repeat go here  
}
```

- Can apply to multiline statements like `if` statements

```
$: if (someCondition) {  
  // body statements  
}
```

re-evaluates condition if any variables it references change, and executes body only when true



# Easy App State Mgmt.

- “Stores” hold application state outside any component
- Alternative to using props or context to make data available in components
- Where to define?
  - for stores that should be available to any component, define and export them in a file like `src/stores.js` and import them from that file wherever needed
  - for stores that should only be available to descendants of a given component, define them in that component and pass them to descendants using props or context

# Kinds of Stores

- **Writable**

- only kind that can be modified by components
- methods
  - `set(newValue)`
  - `update(currentValue => newValue)`

calculates new value from current value

- **Readable**

- handle computing their data
- components cannot modify

- **Derived**

- derive data from current values of other stores

# Defining Writable Stores

stores.js

```
import {writable} from 'svelte/store';  
export const dogStore = writable([]);
```

initial value

```
export const fancyStore = writable(  
  initialValue,  
  async set => {  
    // Called when subscribe count goes from 0 to 1.  
    // Compute initial value and pass to set function.  
    const res = await fetch('/some/url');  
    const data = await res.json();  
    set(data);  
  
    return () => {  
      // Called when subscriber count goes to 0.  
    }  
  }  
);
```

using optional  
second argument

# Using Stores

- Option #1 - **subscribe** method - very verbose!
- Option #2 - **\$** auto-subscription shorthand - much better!
  - variables whose names begin with **\$** must be stores
  - automatically subscribes when first used and unsubscribes when removed from DOM

```
<script>
  import {onDestroy} from 'svelte';
  import {dogStore} from './stores.js';
  let dogs;
  const unsubscribe = dogStore.subscribe(value => dogs = value);
  onDestroy(unsubscribe);
</script>
```

uses **subscribe** method

```
<!-- Use dogs in HTML. -->
```

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

uses auto-subscription

```
<!-- Use $dogStore in HTML. -->
```

# Easy Passing Data to Descendants

- Use “context”
- Alternative to props and stores for making data available in descendant components

```
import {getContext, setContext} from 'svelte';
```

- Ancestor components set context associated with the component

```
setContext(key, value);
```

- must be called during component initialization
- Descendant components get context from closest ancestor that has context with given key

```
const value = getContext(key);
```

- must be called during component initialization
- Keys can be any kind of value, not just strings
- Values can be any kind of value including functions and objects with methods

# Context Example

A.svelte

```
<script>
  import {setContext} from 'svelte';
  import B from './B.svelte';
  setContext('favorites', {color: 'yellow', number: 19});
</script>

<div>
  This is in A.
  <B />
</div>
```

## Output

```
This is in A.
This is in B.
This is in C.
favorite color is yellow
favorite number is 19
```

B.svelte

```
<script>
  import C from './C.svelte';
</script>

<div>
  This is in B.
  <C />
</div>
```

C.svelte

```
<script>
  import {getContext} from 'svelte';
  const {color, number} = getContext('favorites');
</script>

<div>
  This is in C.
  <div>favorite color is {color}</div>
  <div>favorite number is {number}</div>
</div>
```

# Outstanding Issues

- TypeScript support
  - it's coming, but not ready yet
  - <https://github.com/sveltejs/svelte/issues/1639>
- Popularity
  - perhaps Svelte will soon be considered the #4 most popular approach for building web apps
  - isn't easy to find developers that already know it
  - but it's very easy to learn and there is less to learn than other approaches

# Related Tools

- **Svelte VS Code extension**
- **Sapper** - <https://sapper.svelte.dev/>
  - “application framework powered by Svelte”
  - similar to Next and Gatsby
  - provides routing, server-side rendering, and code splitting
- **Svelte Native** - <https://svelte-native.technology/>
  - for implementing native mobile apps
  - based on NativeScript
  - community-driven project
- **Svelte GL** - <https://github.com/Rich-Harris/svelte-gl>
  - in-work Svelte version of Three.js
- **Svelte Testing Library** - <https://testing-library.com/docs/svelte-testing-library/intro>
- **Storybook** with Svelte - <https://storybook.js.org/docs/guides/guide-svelte/>



# Topics Not Covered Here

but covered at <https://objectcomputing.com/resources/publications/sett/july-2019-web-dev-simplified-with-svelte>

- Two-way data bindings
  - more options than shown here
- Easy animations built-in
- Inserting HTML
- Slots
  - for passing child elements to a component
- Event details
  - handling, modifiers, dispatching
- Lifecycle functions
  - `onMount`, `beforeUpdate`, `afterUpdate`, and `onDestroy`
- Actions
  - register a function to be called when a specific element is added to DOM
  - ex. moving focus
- Routing
  - can use `page` on npm or Sapper
- Module Context
  - to run JavaScript code in a component source file only once instead of once for each component instance created
- Special Elements
  - `<svelte:name ...>`
- Debugging with `{@debug}`
  - debugger breaks on state changes
- Unit tests
  - with Jest and Svelte Testing Library
- End-to-end tests
  - with Cypress
- Compiling to custom elements
  - can be used with any framework

# Svelte Resources

- **“Rethinking Reactivity” talk by Rich Harris**
  - delivered multiple times, most recently at “Shift Conference” June 20, 2019
  - explains motivation for Svelte and compares to React
- **Home page** - <https://svelte.dev>
  - contains **Tutorial**, **API Docs**, **Examples**, online **REPL**, **Blog**, and **Sapper** link
    - REPL is great for trying small amounts of Svelte code
    - REPL can save for sharing and submitting issues
- **Discord chat room** - <https://discordapp.com/invite/yy75DKs>
- **GitHub** - <https://github.com/sveltejs/svelte>
- **Awesome Svelte** - <https://github.com/CalvinWalzel/awesome-svelte>
- **Awesome Svelte Resources** - <https://github.com/ryanatkn/awesome-svelte-resources>

# Conclusion

- **Svelte is a worthy alternative** to the current popular options of React, Vue, and Angular
- For more, see **my long article**
  - <https://objectcomputing.com/resources/publications/sett/july-2019-web-dev-simplified-with-svelte>
- **Radiohead** says you should leave your current approach for developing web applications and try Svelte!
  - Weird Fishes / Arpeggi
  - **"Yeah, everybody leaves if they get the chance and this is my chance."**