Svelte

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

R. Mark Volkmann
Object Computing, Inc.
http://objectcomputing.com
Email: mark@objectcomputing.com
Twitter: @mark_volkmann
GitHub: mvolkmann
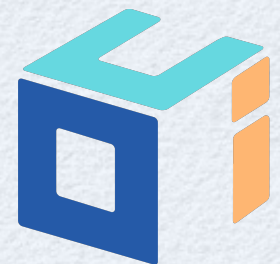
OCI | TRAINING

What Is It?

- Alternative to web frameworks like React, Vue, and Angular

- A web application **complier**, 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

2

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

enter new todo here   Add

☑ ~~learn Svelte~~ Delete ← a Todo component

☐ build a Svelte app Delete

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

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

export default app;
```

4

... Todo App ...

```
<script>
  import {createEventDispatcher} from 'svelte';
  const dispatch = createEventDispatcher();
  export let todo; // the only prop
</script>
```

src/Todo.svelte

export makes it a prop

```
<style>
  .done-true {
    color: gray;
    text-decoration: line-through;
  }
  li {
    margin-top: 5px;
  }
</style>
```

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

```
<li>
  <input
    type="checkbox"
    checked={todo.done}
    on:change={() => dispatch('toggleDone')}
  />
  <span class={'done-' + todo.done}>{todo.text}</span>
  <button on:click={() => dispatch('delete')}>Delete</button>
</li>
```

interpolation

5

... Todo App ...

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

  let lastId = 0;
  const createTodo = (text, done = false) => ({id: ++lastId, text, done});

  let todoText = '';
  let todos = [
    createTodo('learn Svelte', true),
    createTodo('build a Svelte app')
  ];

  $: uncompletedCount = todos.filter(t => !t.done).length;
  $: status = `${uncompletedCount} of ${todos.length} remaining`;

  function addTodo() {
    todos = todos.concat(createTodo(todoText));
    todoText = '';
  }

  const archiveCompleted = () => todos = todos.filter(t => !t.done);

  const deleteTodo = todoId => todos = todos.filter(t => t.id !== todoId);

  function toggleDone(todo) {
    const {id} = todo;
    todos = todos.map(t => t.id === id ? {...t, done: !t.done} : t);
  }
</script>
```

Top-level variables ARE the component state!

reactive declarations

No methods, just functions.

Not really archiving in this simple implementation, just deleting.

6

Svelte

... Todo App ...

```
<style>                                    src/TodoList.svelte
  button {
    margin-left: 10px;
  }

  /* This removes bullets from a bulleted list. */
  ul.unstyled {
    list-style: none;
    margin-left: 0;
    padding-left: 0;
  }
</style>
```

... Todo App

```
<div>
  <h2>To Do List</h2>
  <div>
    {status}
    <button on:click={archiveCompleted}>Archive Completed</button>
  </div>
  <form on:submit|preventDefault>        not doing anything on submit
    <input
      type="text"
      size="30"
      autofocus
      placeholder="enter new todo here"
      bind:value={todoText}
    />
    <button disabled={!todoText} on:click={addTodo}>
      Add
    </button>
  </form>
  <ul class="unstyled">
    {#each todos as todo}    Mustache-style markup
      <Todo
        todo={todo}    passing a prop; can be any JavaScript expression
        on:delete={() => deleteTodo(todo.id)}
        on:toggleDone={() => toggleDone(todo)}
      />
    {/each}
  </ul>
</div>
```

bind:value={todoText} ← 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

8

Svelte

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?

9

Markup `if` Statement

- Begin with `{#if condition}`
  - starting with # indicates a block opening tag

- Can use `{:else if condition}` and `{:else}`
  - starting with : indicates a block continuation tag

- End with `{/if}`
  - starting with / indicates a block ending tag

- Include markup to be conditionally rendered

- Example

```
{#if color === 'yellow'}
  <div>Nice color!</div>
{:else if color === 'orange'}
  <div>That's okay too.</div>
{:else}
  <div>Questionable choice.</div>
{/if}
```

Svelte

Markup `each` Statement

- Begin with `{#each iterable as element}`
  - include markup to be rendered for each element

- Optional `{:else}`
  - renders when iterable is empty

- End with `{/each}`

- Examples

```
{#each colors as color}
  <div style='color: {color}'>{color}</div>
{/each}
```

red
green
blue

```
{#each colors as color, index}
  <div>{index + 1}) {color}</div>
{/each}
```

1) red
2) green
3) blue

```
{#each people as {name, age}}
  <div>{name} is {age} years old.</div>
{:else}
  <div>There are no people.</div>
{/each}
```

can use destructuring when elements are objects

11

Promises in Markup

- Can wait for promises to resolve or reject in markup and render different markup for each and while pending **a bit like React Suspense**

- Examples

  - assuming **getData** function returns a **Promise**

```
{#await getData()}
  <div>Waiting for data ...</div>
{:then result}
  <div>result = {result}</div>
{:catch error}
  <div class="error">Error: {error.message}</div>
{/await}
```
can use any name for variables after :then and :catch

```
{#await getData() then result}
  <div>result = {result}</div>
{:catch error}
  <div class="error">Error: {error.message}</div>
{/await}
```
can omit block to render while waiting for resolve or reject

can omit catch block

12

Svelte

Top Svelte Features

- Small bundle sizes

- File-based component definitions

- CSS scoped by default

- Clear place to put global CSS

- Easy component state management (reactivity)

- Easy app state management (stores)

- Easy way to pass data from components to descendant components (context)

- Reactive declarations

- Two-way data bindings

- Easy animations built-in

- Runtime warnings for accessibility issues

  missing an `alt` attribute on an `img` element

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`

- 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

Does It Disappear?

- Some say Svelte disappears once an app is built

- The Svelte library is mostly defined by `.js` files in `node_modules/svelte`

  - main functions are defined in `internal.js` which is currently ~1400 line of code

  - other library files used for specific features

    - `easing.js, motion.js, register.js, store.js, transition.js`

- `npm run build` produces files in `public` directory

  - including `bundle.js`

- Svelte library functions that are used by the app
  are copied to the top of `bundle.js`

  - in the case of the Todo app shown earlier, this is ~500 lines of code

- So Svelte doesn't disappear, it is just very small

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 <u>must start uppercase</u>

    - can't tell from looking at source file what names might be used

  - lowercase names are reserved

    - for predefined elements like those in HTML and SVG

Svelte

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
  - CSS rules for component only apply to elements with this class name
- Clear place for global CSS
  - `public/global.css`
- svelte3 ESLint plugin warns about unused CSS selectors
  - see https://github.com/sveltejs/eslint-plugin-svelte3

Easy Component State Mgmt.

("reactivity")

- Changes to <u>top-level variables</u> 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

18

Svelte

Easy App State Mgmt.

- "Stores" hold application state outside any component

- Alternative to using props or context ← described later
  to make data available in components

- Where to define?

  - for stores that should be <u>available to any component</u>,
    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 <u>available to descendants of a given component</u>,
    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
- **Readable**
  - handle computing their data
  - components cannot modify
- **Derived**
  - derive data from current values of other stores

Svelte

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

error handling omitted

can calculate new value from current with
update(currentValue => newValue)

21

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

uses **subscribe** method

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

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

uses auto-subscription

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

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

22

Svelte

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

23

More on Context

- Keys can be any kind of value, not just strings

- Values can be any kind of value
  including functions and objects with methods

- Context is not reactive!

  - descendant components are not re-rendered when
    context they use is changed in an ancestor component

  - use stores if reactivity is needed

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>
```

Svelte

Reactive Declarations

- `$:` is a "labeled statement" with label name "`$`" that Svelte treats as a "reactive declaration"

> 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.

- Add as a prefix on <u>top-level statements</u> that should be repeated whenever any referenced variables change

- Examples

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

> like "computed properties" in Vue

> great for debugging

> when applied to an assignment to an undeclared variable 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
}
```

> executes if any variables referenced in condition or body change, but of course the body only executes when condition is true

26

Two-way Data Bindings

- Form elements can be bound to a variable

- Simulates two-way data binding

- Provides current value and
  event handling for updating variable
  when user changes value

Svelte

Binding Example ...

Name Mark

Happy? ☑

Favorite Flavors ☑ vanilla ☐ chocolate ☑ strawberry

Favorite Season ◉ Spring ○ Summer ○ Fall ○ Winter

Favorite Color [yellow ▾]

Life Story Once upon a time ...

Mark likes yellow, Spring, and is happy.

Mark's favorite flavors are vanilla,strawberry.

Story: Once upon a time ...

```
<script>
  const colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple'];
  const flavors = ['vanilla', 'chocolate', 'strawberry'];
  const seasons = ['Spring', 'Summer', 'Fall', 'Winter']
  let favoriteColor = '';
  let favoriteFlavors = [];
  let favoriteSeason = '';
  let happy = true;
  let name = '';
  let story = '';
</script>
```

these variables
are bound to
form elements on
next two slides

Svelte

... Binding Example ...

```
<div class="form">
  <div>
    <label>Name</label>
    <input type="text" bind:value={name} />
  </div>
  <div>
    <label>Happy?</label>
    <input type="checkbox" bind:checked={happy} />
  </div>
  <div>
    <label>Favorite Flavors</label>
    {#each flavors as flavor}
      <label>
        <input type="checkbox" value={flavor} bind:group={favoriteFlavors} />
        {flavor}
      </label>
    {/each}
  </div>
```

for checkboxes, bind to `checked` property rather than `value`

using `bind:group` with a set of related checkboxes makes the value an array of strings

29

... Binding Example

```
<div>
  <label>Favorite Season</label>
  {#each seasons as season}
    <label>
      <input type="radio" value={season} bind:group={favoriteSeason} />
      {season}
    </label>
  {/each}
</div>
<div>
  <label>Favorite Color</label>
  <select bind:value={favoriteColor}>
    <option />
    {#each colors as color}
      <option>{color}</option>
    {/each}
  </select>
</div>
<div>
  <label>Life Story</label>
  <textarea bind:value={story} />
</div>
```

using `bind:group` with a set of related radio buttons makes the value a single string

to change a select to a scrollable list that allows selecting multiple options add `multiple` attribute;
makes the value an array of selected values

30

Easy Animations Built-in

- **svelte/animate** provides
  - **flip**

- **svelte/motion** provides
  - **spring**
  - **tweened**

- **svelte/transition** provides
  - **crossfade**
  - **draw** - for SVG elements
  - **fade**
  - **fly** - set **x** and/or **y**
  - **scale**
  - **slide**

basic example

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

<li transition:fade>
  <!-- some content -->
</li>
```

this fades in when mounted and fades out when destroyed

- Also see **svelte/easing**

31

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 that other approaches

32

Creating a Project

- Install Node.js from https://nodejs.org
  - installs `node`, `npm`, and `npx` commands

- Approach #1
  - `npx degit sveltejs/template` *app-name*
  - **degit** is useful for project scaffolding

- Approach #2
  - browse https://svelte.dev/repl
  - click download button
  - unzip downloaded zip file
  - demo this!

> `degit` was created by Rich Harris.
> It downloads a git repo,
> by default the master branch.
> In this case `sveltejs` is the user name
> and `template` is the repo.
> The second argument is the
> name of the directory to create.

Svelte

Installing and Running

- **cd *app-name***

- **npm install**

- **npm run dev**

  - provides live reload, unlike **npm run start**

  - syntax errors are reported in window where this is running, not in browser because it doesn't produce a new version of the app if there are errors

- Browse **localhost:5000**

- Just renders "Hello world!"

Svelte

Topics Not Covered Here

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

- Inserting HTML

- Slots
  - for passing child elements to a component

- Event details
  - handling, modifiers, dispatching

- Lifecycle functions
  - `onMount, beforeUpdate, afterUpdate, onDestroy`

- Actions
  - register a function to be called when a specific element is added to DOM

- 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}`

- 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

Related Tools

- **"Svelte" VS Code extension**
  - provides syntax highlighting and intellisense
- **Sapper** - https://sapper.svelte.dev/
  - "application framework powered by Svelte"
  - name may be a contraction of "Svelte" and "Application"
  - 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-vue
  - community-driven project
- **Svelte Testing Library** - https://testing-library.com/docs/svelte-testing-library/intro
- **Storybook** with Svelte - https://storybook.js.org/docs/guides/guide-svelte/

Svelte Resources

- **"Rethinking reactivity"** - https://www.youtube.com/watch?v=AdNJ3fydeao
  - **talk by Rich Harris** at "You Gotta Love Frontend (YGLF) Code Camp 2019"
- **Home page** - https://svelte.dev
- **Tutorial** - https://svelte.dev/tutorial
- **API Docs** - https://svelte.dev/docs
- **Examples** - https://svelte.dev/examples
- **Online REPL** - https://svelte.dev/repl
  - great for trying small amounts of Svelte code; can save for sharing and submitting issues
- **Blog** - https://svelte.dev/blog
- **Discord chat room** - https://discordapp.com/invite/yy75DKs
- **GitHub** - https://github.com/sveltejs/svelte

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