Svelte Web Programming: Professional

(Integrative-Generative AI Edition)

Contents:

Author: Student Price Book Center

คำนำ

ในยุคที่เว็บแอปพลิเคชันมีความซับซ้อนและผู้ใช้คาดหวังประสิทธิภาพสูงสุดจากทุกคลิก Svelte ได้กลายมาเป็นทางเลือกที่น่าสนใจสำหรับนักพัฒนาเว็บยุคใหม่ ด้วยแนวคิด “compile-time framework” ที่แปลกใหม่ Svelte ไม่เพียงลดขนาดของ JavaScript บน client-side แต่ยังช่วยให้โค้ดกระชับ อ่านง่าย และมี performance ที่เหนือกว่า framework แบบเดิมอย่าง React หรือ Vue ในหลายสถานการณ์ หนังสือ *Svelte Web Programming: Professional* เล่มนี้จึงถือกำเนิดขึ้นเพื่อตอบโจทย์นักพัฒนามืออาชีพที่ต้องการยกระดับจากพื้นฐานสู่การใช้งาน Svelte และ SvelteKit ในระดับ production อย่างแท้จริง

โครงสร้างของหนังสือเริ่มต้นจากการนำ TypeScript เข้ามาใช้งานร่วมกับ Svelte อย่างเป็นระบบ (บทที่ 16) ผู้อ่านจะได้เรียนรู้ตั้งแต่การเปิดใช้งาน TypeScript ในโปรเจกต์ Svelte ไปจนถึงการ Typing องค์ประกอบสำคัญอย่าง props, store และ context รวมถึงการใช้ TypeScript กับ SvelteKit อย่างปลอดภัยในสภาพแวดล้อมจริง ซึ่งเนื้อหานี้ไม่เพียงช่วยลดข้อผิดพลาดขณะเขียนโค้ด แต่ยังช่วยให้ทีมพัฒนาทำงานร่วมกันได้อย่างมีประสิทธิภาพมากขึ้น

ต่อเนื่องด้วยบทที่ 17 ซึ่งมุ่งเน้นด้านการทดสอบแอปพลิเคชัน Svelte อย่างครอบคลุม ทั้ง Unit Testing, Integration Testing ไปจนถึง E2E Testing ผ่านเครื่องมือชั้นนำอย่าง Vitest, Jest, @testing-library/svelte และ Playwright/Cypress โดยให้ตัวอย่างชัดเจน พร้อมแนวทางการ mock store, events และการเรียก fetch เพื่อให้ผู้อ่านสามารถสร้างระบบที่เชื่อถือได้และตรวจสอบการทำงานในทุกระดับได้อย่างมั่นใจ

เมื่อระบบถูกสร้างขึ้นแล้ว การปรับปรุงประสิทธิภาพ (บทที่ 18) ถือเป็นหัวใจสำคัญในการเตรียมความพร้อมสู่ production หนังสือเล่มนี้จึงอธิบายวิธีลด bundle size, ใช้ lazy load, ตรวจสอบ memory leaks และ fine-tune update cycles ของ Svelte/SvelteKit อย่างละเอียด พร้อมแนวทางการ profiling และเทคนิค optimization จริงที่ใช้ในระบบระดับองค์กร

หลังจากระบบทำงานได้ดีและรวดเร็วแล้ว ขั้นตอนต่อไปคือการ deploy ซึ่งครอบคลุมอยู่ในบทที่ 19 ผู้อ่านจะได้เรียนรู้การ build โปรเจกต์ทั้งแบบ static และ SSR, การเลือกใช้ SvelteKit adapter ที่เหมาะสม, วิธี deploy ไปยังแพลตฟอร์มต่างๆ เช่น Netlify, Vercel หรือ Docker ตลอดจนการจัดการ secrets และ environment variables อย่างปลอดภัย ช่วยให้ระบบสามารถนำขึ้น production ได้อย่างราบรื่นและมั่นคง

ท้ายที่สุด บทที่ 20 ของหนังสือคือการบูรณาการความรู้ทั้งหมด เพื่อสร้างระบบระดับ production จริงอย่างมีระบบ ตั้งแต่การออกแบบโครงสร้างโปรเจกต์ การแยกโมดูลเพื่อการ reuse และ maintain ที่ดี การทำ code splitting อย่างชาญฉลาด ไปจนถึงการสร้าง CI/CD pipeline ที่อัตโนมัติ และการจัดการ error และ logging อย่างเป็นระบบ ซึ่งเนื้อหานี้เป็นสิ่งที่นักพัฒนาสายมืออาชีพไม่ควรมองข้าม

หนังสือ *Svelte Web Programming: Professional* จึงไม่ได้เป็นเพียงตำราเทคนิค แต่คือแนวทางสู่การสร้างซอฟต์แวร์ระดับองค์กรด้วย Svelte/SvelteKit อย่างมั่นคง มีประสิทธิภาพ และปลอดภัย ผู้เขียนหวังว่าหนังสือเล่มนี้จะเป็นเพื่อนคู่คิดสำหรับนักพัฒนาทุกคนในการต่อยอดจากพื้นฐานสู่ความเชี่ยวชาญ และร่วมผลักดันมาตรฐานการพัฒนาเว็บแอปพลิเคชันของคุณไปอีกระดับ

ขอให้การเดินทางสู่โลกของ Svelte ที่ทรงพลังนี้ เต็มไปด้วยแรงบันดาลใจ ความเข้าใจ และความสำเร็จในทุกโปรเจกต์ของคุณ

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 16

TypeScript กับ Svelte

(TypeScript and Svelte)

เนื้อหา

- TypeScript กับ Svelte
- TypeScript กับ Svelte (เชิงลึก)
- TypeScript กับ Svelte อย่างละเอียด
- การเพิ่ม TypeScript ในโปรเจกต์ Svelte
- การ Typing Props, Store, และ Context ใน Svelte + TypeScript
- การใช้ SvelteKit + TypeScript อย่างปลอดภัย

บทนำบทที่ 16: TypeScript กับ Svelte

ในช่วงไม่กี่ปีที่ผ่านมา TypeScript ได้กลายเป็นเครื่องมือสำคัญสำหรับนักพัฒนาเว็บสมัยใหม่ที่ต้องการโค้ดที่ปลอดภัย ตรวจสอบได้ล่วงหน้า และสามารถขยายตัวได้ดี ในขณะเดียวกัน Svelte ซึ่งเป็นเฟรมเวิร์ก JavaScript ที่เน้นการคอมไพล์ในขั้นตอน build-time ก็ได้รับความนิยมเพิ่มขึ้นอย่างต่อเนื่อง เนื่องจากประสิทธิภาพสูงและแนวคิดที่ลดความซับซ้อนของโค้ด การผสมผสานระหว่าง TypeScript และ Svelte จึงกลายเป็นแนวทางที่น่าสนใจสำหรับผู้ที่ต้องการพัฒนาแอปพลิเคชันที่มีคุณภาพสูงทั้งในด้านโครงสร้างและประสิทธิภาพ

บทนี้จะพาผู้อ่านเข้าสู่โลกของการพัฒนา Svelte ด้วย TypeScript อย่างเป็นระบบ โดยเริ่มจากพื้นฐานของการติดตั้งและตั้งค่าโปรเจกต์ให้รองรับ TypeScript อย่างถูกต้อง ไม่ว่าจะเป็นการสร้างโปรเจกต์ใหม่ด้วย create-svelte หรือการปรับโปรเจกต์เดิมให้รองรับ TypeScript พร้อมแนะนำไฟล์สำคัญ เช่น tsconfig.json และ svelte.config.js ที่เกี่ยวข้องกับการกำหนดค่าการคอมไพล์

เมื่อโปรเจกต์พร้อมใช้งานแล้ว บทนี้จะลงลึกไปที่การกำหนดชนิดข้อมูล (typing) สำหรับองค์ประกอบหลักของ Svelte ได้แก่ props, store และ context ซึ่งถือเป็นหัวใจสำคัญของการสื่อสารและการจัดการข้อมูลระหว่าง component ต่าง ๆ ผู้อ่านจะได้เรียนรู้วิธีระบุประเภทข้อมูลให้ชัดเจน รวมถึงเทคนิคเพื่อหลีกเลี่ยงข้อผิดพลาดที่มักเกิดขึ้นเมื่อใช้ชนิดข้อมูลที่ไม่แน่นอน

ในส่วนของ props บทนี้จะอธิบายการกำหนดชนิดของ prop ที่ component ภายนอกส่งเข้ามา โดยใช้ไวยากรณ์ของ TypeScript ที่ผสมผสานกับ Svelte อย่างกลมกลืน ทั้งยังแสดงตัวอย่างการใช้ interface และ generics เพื่อเพิ่มความยืดหยุ่นและความปลอดภัยของโค้ด

ถัดไปคือ store ซึ่งเป็นกลไกจัดการ state ของ Svelte ผู้อ่านจะได้เรียนรู้การสร้าง store แบบ typed ด้วย writable และ readable พร้อมทั้งการใช้ชนิดข้อมูลเพื่อควบคุมค่าที่สามารถส่งผ่านไปยังส่วนต่าง ๆ ของแอปพลิเคชัน นอกจากนี้ยังมีตัวอย่างการประยุกต์ใช้กับ derived store และ custom store เพื่อแสดงศักยภาพของ TypeScript ในการจัดการ state ที่ซับซ้อน

context API ซึ่งใช้แชร์ข้อมูลระหว่าง component แบบลึกซ้อนก็สามารถกำหนดชนิดข้อมูลได้เช่นกัน บทนี้จะเสนอแนวทางการใช้ setContext และ getContext ควบคู่กับ TypeScript เพื่อให้การแลกเปลี่ยนข้อมูลมีความปลอดภัย ปราศจากข้อผิดพลาดที่เกิดจากการใช้ key หรือ type ไม่ตรงกัน

ท้ายที่สุด บทนี้จะปิดท้ายด้วยการนำ TypeScript มาใช้ร่วมกับ SvelteKit ซึ่งเป็นเฟรมเวิร์กเต็มรูปแบบของ Svelte สำหรับการสร้างแอปพลิเคชันแบบ SSR และ SPA ผู้อ่านจะได้เข้าใจรูปแบบการ typing ของ routing, load function, endpoint, และ environment variables พร้อมแนวทางปฏิบัติที่ปลอดภัยและยืดหยุ่นสำหรับโค้ด production

---

TypeScript กับ Svelte

หัวข้อที่ครอบคลุม

1. ☐ การเพิ่ม TypeScript ในโปรเจกต์ Svelte/SvelteKit
2. ☐ การ typing props, store, และ context
3. ☐ การใช้ SvelteKit + TypeScript อย่างปลอดภัย

☐ **1. การเพิ่ม TypeScript ในโปรเจกต์ Svelte**

☐ **สำหรับ SvelteKit (แนะนำ):**

หากคุณเริ่มจาก 0 ให้ใช้ template แบบ TypeScript ได้เลย:

```
npm create svelte@latest my-app
# เลือก: TypeScript + ESLint + Prettier (แนะนำ)
cd my-app
npm install
npm run dev
```

☐ **ถ้าเริ่มจากโปรเจกต์ JavaScript:**

สามารถเพิ่ม TS แบบ manual:

```
npm install --save-dev typescript svelte-preprocess
touch tsconfig.json
```

จากนั้นปรับ svelte.config.js:

```
import preprocess from 'svelte-preprocess';

export default {
```

```
  preprocess: preprocess(),
  kit: {
    // ...
  }
}
```

เพิ่มไฟล์ tsconfig.json:

```
{
  "extends": "./.svelte-kit/tsconfig.json",
  "compilerOptions": {
    "strict": true,
    "types": ["svelte"]
  }
}
```

เปลี่ยนนามสกุลไฟล์จาก .svelte ⟶ .svelte และสามารถใช้ <script lang="ts"> ได้เลย

□ 2. การ typing props, store และ context

✳ □ Typing Props (ใน Component)

Parent.svelte

```
<script lang="ts">
  import Child from './Child.svelte';
</script>

<Child name="John" age={25} />
```

Child.svelte

```
<script lang="ts">
  export let name: string;
  export let age: number;
</script>

<p>{name} is {age} years old.</p>
```

✳ □ Typing Store

```
// src/stores/counter.ts
```

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

export const count = writable<number>(0);
<!-- ใช้งานใน .svelte -->
<script lang="ts">
  import { count } from '../stores/counter';
</script>

<p>Count is {$count}</p>
```

✳ □ **Typing Custom Store ด้วย interface**

```
// src/stores/user.ts
import { writable } from 'svelte/store';

export interface User {
  id: number;
  name: string;
}

export const user = writable<User | null>(null);
```

✳ □ **Typing Context (setContext / getContext)**

```
// src/lib/auth.ts
import { getContext, setContext } from 'svelte';
import type { Writable } from 'svelte/store';

const key = Symbol();

export function initAuth(userStore: Writable<string | null>) {
  setContext<Writable<string | null>>(key, userStore);
}

export function useAuth() {
  return getContext<Writable<string | null>>(key);
```

```
}
```

การใช้งานใน component:

```
<script lang="ts">
  import { initAuth } from '$lib/auth';
  import { writable } from 'svelte/store';

  initAuth(writable('John'));
</script>
```

□ 3. การใช้ SvelteKit + TypeScript อย่างปลอดภัย

SvelteKit มีระบบ TypeScript support โดยตรงในหลายจุด เช่น:

□ การใช้ load แบบ type-safe

```
// +page.ts (client-side)
import type { PageLoad } from './$types';

export const load: PageLoad = async ({ fetch }) => {
  const res = await fetch('/api/user');
  const user: { id: number; name: string } = await res.json();
  return { user };
};
```

□ การใช้ +page.server.ts

```
// +page.server.ts
import type { PageServerLoad } from './$types';

export const load: PageServerLoad = async () => {
  return { msg: 'Hello from server' };
};
```

□ การใช้งาน locals / cookies แบบ type-safe

แก้ hooks.server.ts:

```
// src/hooks.server.ts
import type { Handle } from '@sveltejs/kit';

export const handle: Handle = async ({ event, resolve }) => {
  event.locals.user = 'admin';
```

```
  return resolve(event);
};
```

เพิ่มใน src/app.d.ts:

```
// src/app.d.ts
declare namespace App {
  interface Locals {
    user: string;
  }
}
```

☐ **ตัวอย่างผลลัพธ์**

จุดที่ใช้ TypeScript	ตัวอย่าง Type	ผลลัพธ์ / ประโยชน์
Props ของ component	export let name: string	ช่วยให้ใช้ component แบบปลอดภัย
Store	writable<User>()	Store มี type ชัดเจน ป้องกันการใช้ผิด
Context API	setContext<MyType>(key,…)	ได้ IntelliSense + ป้องกัน null
ฟังก์ชันโหลดข้อมูล (load)	PageLoad, PageServerLoad	โหลดข้อมูลแบบปลอดภัยจาก fetch/api

---

TypeScript กับ Svelte (เชิงลึก)

☐ **เหตุผลที่ควรใช้ TypeScript กับ Svelte**

- ☐ ป้องกัน bug จากการใช้ตัวแปรผิดชนิด
- ☐ ช่วยให้โค้ดอ่านง่ายขึ้น (auto-completion, IntelliSense)
- ☐ ใช้ร่วมกับ IDE ได้เต็มที่ (เช่น VS Code)
- ☐ รองรับการพัฒนาโปรเจกต์ขนาดใหญ่ที่ต้องการความปลอดภัยของ type และการแยกส่วน

☐ **1. การเพิ่ม TypeScript ในโปรเจกต์**

☐ **1.1 สำหรับ SvelteKit**

```
npm create svelte@latest my-app
# เลือก template: "SvelteKit + TypeScript"
```

โปรเจกต์ที่ได้จะมี:

```
src/
  routes/
  lib/
```

app.d.ts □ type global

stores/ □ custom store

svelte.config.js

tsconfig.json □ ควบคุม TS ทั้งหมด

□ ไม่ต้องตั้งค่าเอง ทุกอย่างพร้อมใช้

□ 1.2 สำหรับ Svelte ปกติ (non-SvelteKit)

npm install --save-dev typescript svelte-preprocess

เพิ่มใน rollup.config.js หรือ vite.config.ts:

```
import sveltePreprocess from 'svelte-preprocess';

export default {
  preprocess: sveltePreprocess(),
  // ...
}
```

เพิ่ม tsconfig.json (ตัวอย่าง):

```
{
  "compilerOptions": {
    "target": "es6",
    "strict": true,
    "moduleResolution": "node",
    "types": ["svelte"]
  },
  "include": ["src/**/*"]
}
```

□ 2. Typing Props, Store และ Context (เชิงลึก)

□ Typing Props ใน Component

Svelte ไม่ใช้ interface Props แบบ React แต่ใช้ export let พร้อม type:

```
<script lang="ts">
  export let title: string;
  export let count?: number; // optional prop
</script>
```

คุณสามารถตั้งค่า default ได้เช่นกัน:

```
export let count: number = 0;
```

☐ Typing Store

☐ *Writable*

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

const count = writable<number>(0);
```

☐ *Readable*

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

const time = readable<Date>(new Date(), (set) => {
  const interval = setInterval(() => set(new Date()), 1000);
  return () => clearInterval(interval);
});
```

☐ *Derived*

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

const double = derived(count, $count => $count * 2);
```

☐ *Custom store แบบมี interface*

```
interface User {
  id: number;
  name: string;
}

const user = writable<User | null>(null);
```

☐ Typing Context API

```
// lib/context.ts
import { setContext, getContext } from 'svelte';
import type { Writable } from 'svelte/store';

const key = Symbol();
```

```
export function initTheme(store: Writable<string>) {
  setContext<Writable<string>>(key, store);
}

export function useTheme(): Writable<string> {
  return getContext<Writable<string>>(key);
}
```

☐ เทียบเท่า Dependency Injection แบบ type-safe

☐ **3. การใช้ SvelteKit + TypeScript อย่างปลอดภัย**

☐ **การ typing load function**

```
// +page.ts
import type { PageLoad } from './$types';

export const load: PageLoad = async () => {
  return { name: 'Alice' };
};
```

ช่วยให้ return type, params, error, และ fetch เป็น type-safe

☐ **Global Typing (เช่น Locals, Session)**

app.d.ts

```
declare namespace App {
  interface Locals {
    user: {
      id: number;
      name: string;
    };
  }

  interface Session {
    token: string;
  }
}
```

ใช้งานใน hooks.server.ts

```
export const handle = async ({ event, resolve }) => {
  event.locals.user = { id: 1, name: 'Admin' };
  return resolve(event);
};
```

☐ **Type module context (layout, error page)**

- +layout.ts ⟶ LayoutLoad
- +page.server.ts ⟶ PageServerLoad
- +layout.server.ts ⟶ LayoutServerLoad
- +error.svelte ⟶ มี access ไปยัง error: Error & { message: string }

☐ **เคล็ดลับ TypeScript กับ SvelteKit (Best Practice)**

เทคนิค	คำแนะนำ
lang="ts"	อย่าลืมใส่ใน <script lang="ts"> ทุกไฟล์
ใช้ strict mode	เปิด strict เพื่อความแม่นยำ (ใน tsconfig.json)
ใช้ $types	อย่าพิมพ์ type ด้วยตัวเอง ใช้ import จาก ./$types แทน
แยก Store และ Interface	เก็บใน src/stores/, src/types/ เพื่อแยก concerns
หลีกเลี่ยง any	ใช้ union type, optional, generics แทน any

☐ **สรุปภาพรวมแบบมืออาชีพ**

ฟีเจอร์	ตัวอย่าง	ประโยชน์หลัก
Props typing	export let id: number	ป้องกันการส่ง prop ผิดชนิด
Store typing	`writable<User	null>()`
Context typing	setContext<Writable<string>>()	ช่วยให้ code ใช้งานซ้ำ และ type-safe
load() typing	export const load: PageLoad = …	ป้องกัน fetch และ return ผิดโครงสร้าง
app.d.ts	declare Locals, Session	ทำให้ระบบ auth / permission type-safe

---

TypeScript กับ Svelte อย่างละเอียด

☐ **บทที่ 16: TypeScript กับ Svelte**

☐ ความสำคัญของ TypeScript กับ Svelte

Svelte รองรับการใช้งาน TypeScript โดยตรงโดยไม่ต้องใช้ wrapper หรือ plugin ภายนอก ทำให้นักพัฒนาได้รับประโยชน์จาก static typing, auto-complete และการตรวจสอบข้อผิดพลาดที่ compile time ได้อย่างเต็มที่ โดยไม่สูญเสียความเรียบง่ายของ Svelte

---

☐ หัวข้อที่ 1: การเพิ่ม TypeScript ในโปรเจกต์ Svelte

วิธีที่ 1: สร้างโปรเจกต์ใหม่พร้อม TypeScript (แนะนำ)

```
npm create vite@latest my-app -- --template svelte-ts
cd my-app
npm install
npm run dev
```

Vite จะสร้างโครงสร้างโปรเจกต์พร้อมใช้งาน TypeScript และตั้งค่าทั้งหมดให้อัตโนมัติ เช่น:

- tsconfig.json
- ใช้ .svelte ที่รองรับ <script lang="ts">
- มีไฟล์ app.d.ts สำหรับ global type

วิธีที่ 2: เพิ่ม TypeScript ภายหลัง

หากคุณมีโปรเจกต์ Svelte อยู่แล้ว:

```
npx svelte-add@latest typescript
```

คำสั่งนี้จะเพิ่มไฟล์ config และ dependency ที่จำเป็นให้โดยอัตโนมัติ

โค้ดตัวอย่าง .svelte ที่ใช้ TypeScript:

```
<script lang="ts">
  let count: number = 0;

  function increment(): void {
    count += 1;
  }
</script>

<button on:click={increment}>
  Count is {count}
</button>
```

☐ หัวข้อที่ 2: การ Typing Props, Store และ Context

Typing Props ใน Svelte

ใน <script lang="ts"> เราสามารถใช้ export let พร้อมระบุ type ได้เลย:

```
<script lang="ts">
  export let name: string;
  export let age: number;
</script>

<p>Hello {name}, age {age}</p>
```

หากส่ง prop ผิดประเภท จะเกิดข้อผิดพลาดที่ compile time (ช่วยลด bug ได้มาก)

Typing Store

Svelte store สามารถใช้กับ TypeScript ได้ง่าย:

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

export const counter = writable<number>(0);
```

หากต้องการ store ที่ซับซ้อน:

```
type User = {
  id: number;
  name: string;
};

export const user = writable<User | null>(null);
```

Typing Context

การใช้ context ใน Svelte ใช้ setContext และ getContext ซึ่งสามารถ type ได้ชัดเจนมากขึ้น:

```
// context.ts
export const ThemeContextKey = Symbol();

export type Theme = 'light' | 'dark';
// Parent.svelte
<script lang="ts">
  import { setContext } from 'svelte';
  import { ThemeContextKey, Theme } from './context';
```

```
  const theme: Theme = 'dark';
  setContext(ThemeContextKey, theme);
</script>
// Child.svelte
<script lang="ts">
  import { getContext } from 'svelte';
  import { ThemeContextKey, Theme } from './context';

  const theme = getContext<Theme>(ThemeContextKey);
</script>
```

□ หัวข้อที่ 3: การใช้ SvelteKit + TypeScript อย่างปลอดภัย

SvelteKit ออกแบบให้รองรับ TypeScript ตั้งแต่ต้น โดยเมื่อสร้างโปรเจกต์จะมีตัวเลือกให้ใช้ TypeScript ทันที

```
npm create svelte@latest my-app
# แล้วเลือก "TypeScript" ตอน setup
```

□ *การใช้ TypeScript ใน Routing*

SvelteKit ใช้ +page.ts, +layout.ts, +server.ts แยก logic:

```
// +page.ts
export const load = async () => {
  const res = await fetch('/api/products');
  const products: Product[] = await res.json();
  return { products };
};
// +page.svelte
<script lang="ts">
  export let data: {
    products: Product[];
  };
</script>

<ul>
  {#each data.products as product}
```

```
    <li>{product.name}</li>
  {/each}
</ul>
```

✅ *ประโยชน์จากการใช้ TypeScript:*

- ป้องกัน bug จาก prop / store / endpoint ที่ไม่ตรง type
- ทำงานร่วมกับ IDE ได้ดี (IntelliSense, Jump to Definition)
- ปลอดภัยมากขึ้นในการ refactor โค้ด
- เพิ่มความมั่นใจเวลาทำงานในทีม

✅ สรุป

การใช้ TypeScript กับ Svelte/SvelteKit ทำให้การพัฒนา:

- **ปลอดภัยขึ้น**
- **ชัดเจนขึ้น**
- **บำรุงรักษาโค้ดง่ายขึ้น**

ทั้งยัง **ไม่เพิ่มความซับซ้อนมากนัก** เพราะ Svelte ออกแบบมาให้ TS integration เป็นธรรมชาติ หากคุณทำงานในทีม หรือเขียนโปรเจกต์ที่มีความซับซ้อน TypeScript คือเครื่องมือสำคัญที่ควรใช้อย่างยิ่ง

การเพิ่ม TypeScript ในโปรเจกต์ Svelte

✅ การเพิ่ม TypeScript ในโปรเจกต์ Svelte

Svelte รองรับ TypeScript แบบ built-in โดยไม่ต้องใช้ loader ภายนอกเพิ่มเติม และคุณสามารถเริ่มใช้ได้ง่ายมากไม่ว่าจะสร้างโปรเจกต์ใหม่หรือเพิ่มเข้าไปในโปรเจกต์ที่มีอยู่แล้ว

✅ วิธีที่ 1: สร้างโปรเจกต์ใหม่พร้อม TypeScript (แนะนำที่สุด)

ใช้ Vite ซึ่งเป็นวิธีที่รวดเร็วและเป็นมาตรฐานใน Svelte community:

```
npm create vite@latest my-svelte-app -- --template svelte-ts
cd my-svelte-app
npm install
npm run dev
```

หรือถ้าใช้ SvelteKit:

```
npm create svelte@latest my-sveltekit-app
```

เลือก Skeleton project แล้วเลือก TypeScript เมื่อ wizard ถาม

ระบบจะสร้างไฟล์ TypeScript ทั้งหมดให้อัตโนมัติ เช่น:

- tsconfig.json – การตั้งค่า TypeScript

- app.d.ts – สำหรับประกาศ global types
- .svelte ไฟล์จะรองรับ <script lang="ts">

□ วิธีที่ 2: เพิ่ม TypeScript เข้าไปในโปรเจกต์ Svelte ที่มีอยู่แล้ว

หากคุณมีโปรเจกต์ Svelte อยู่แล้วและต้องการเพิ่ม TypeScript:

1. ติดตั้ง dependency ที่จำเป็น:

```
npm install --save-dev typescript svelte-preprocess @tsconfig/svelte
```

2. สร้างไฟล์ tsconfig.json:

```
{
  "extends": "@tsconfig/svelte/tsconfig.json",
  "include": ["src/**/*"],
  "exclude": ["node_modules/*", "__sapper__/*", "public/*"]
}
```

3. ติดตั้ง svelte-preprocess และตั้งค่าใน svelte.config.js:

```
import sveltePreprocess from 'svelte-preprocess';

export default {
  preprocess: sveltePreprocess(),
};
```

4. เปลี่ยน <script> ในไฟล์ .svelte เป็น <script lang="ts"> เช่น:

```
<script lang="ts">
  let count: number = 0;
  function increment(): void {
    count += 1;
  }
</script>

<button on:click={increment}>Count is {count}</button>
```

□ ตัวอย่างโครงสร้างไฟล์โปรเจกต์ที่ใช้ TypeScript

```
my-svelte-app/
│
├── src/
│   ├── App.svelte
```

```
│   ├── main.ts
│   └── stores.ts
│
├── svelte.config.js
├── tsconfig.json
├── vite.config.ts
└── package.json
```

□ ข้อดีของการใช้ TypeScript กับ Svelte

- □ ตรวจสอบ type ผิดพลาดล่วงหน้า
- □ IntelliSense จาก editor อย่าง VSCode
- □ ปลอดภัยต่อการ refactor
- □ เหมาะกับทีมที่ทำงานร่วมกันหลายคน
- □ รองรับการทำงานกับ REST API, Store, และ Component ที่ซับซ้อน

□ ตัวอย่าง: เพิ่ม TypeScript ในไฟล์ Svelte

```
<script lang="ts">
  export let name: string;

  function greet(): void {
    alert(`Hello, ${name}`);
  }
</script>

<button on:click={greet}>Greet</button>
```

หากคุณส่งค่า name ที่ไม่ใช่ string เข้ามา จะมี error แจ้งทันที!

□ สรุป

วิธี	รายละเอียด
□ วิธีที่ 1	สร้างโปรเจกต์ใหม่ด้วย template svelte-ts หรือ SvelteKit + TypeScript
□ วิธีที่ 2	เพิ่ม TypeScript เข้าโปรเจกต์ Svelte ที่มีอยู่ ด้วยการติดตั้ง typescript และ svelte-preprocess

วิธี	รายละเอียด
☐ Tips	ใช้ <script lang="ts"> ในทุกไฟล์ .svelte และใช้ export let พร้อม type เพื่อความปลอดภัย

ด้านล่างนี้คือ **ตัวอย่างโปรแกรม Svelte + TypeScript** จำนวน **6 โปรแกรม** แบ่งเป็น:

- ☐ **3 โปรแกรมพื้นฐาน**: ตัวอย่างง่าย เหมาะกับผู้เริ่มต้น
- ☐ **3 โปรแกรมแนวประยุกต์**: ตัวอย่างซับซ้อนขึ้น ใช้กับโปรเจกต์จริงได้

☐ หมวดที่ 1: โปรแกรมพื้นฐาน Svelte + TypeScript

☐ โปรแกรมที่ 1: Counter App

☐ *โครงสร้าง*

```
counter-app/
├── src/
│   ├── App.svelte
│   └── main.ts
├── tsconfig.json
├── svelte.config.js
├── vite.config.ts
└── package.json
```

☐ *src/App.svelte*

```
<script lang="ts">
  let count: number = 0;

  const increment = (): void => {
    count += 1;
  };
</script>

<h1>Count: {count}</h1>
<button on:click={increment}>Increment</button>
```

☐ *src/main.ts*

```
import App from './App.svelte';
```

```
const app = new App({
  target: document.body,
});

export default app;
```

☐ ***ผลการรัน:***

หน้าเว็บแสดงตัวเลข Count พร้อมปุ่ม “Increment” เมื่อคลิกจะเพิ่มค่า count

---

☐ **โปรแกรมที่ 2: Greeting Props**

☐ ***src/App.svelte***

```
<script lang="ts">
  export let name: string;
</script>

<h1>Hello, {name}!</h1>
```

☐ ***src/main.ts***

```
import App from './App.svelte';

const app = new App({
  target: document.body,
  props: {
    name: 'Svelte Developer'
  }
});

export default app;
```

☐ ***ผลการรัน:***

แสดงข้อความ: **Hello, Svelte Developer!**

☐ **โปรแกรมที่ 3: Custom Typed Store**

☐ ***src/stores.ts***

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

export const name = writable<string>('Guest');
```

 src/App.svelte

```
<script lang="ts">
  import { name } from './stores';
</script>

<h1>Welcome, {$name}!</h1>
<input bind:value={$name} placeholder="Enter your name" />
```

 ผลการรัน:

มี input สำหรับกรอกชื่อ แสดงคำว่า: **Welcome, ...!** และอัปเดตอัตโนมัติเมื่อพิมพ์

 หมวดที่ 2: โปรแกรมแนวประยุกต์ Svelte + TypeScript

 โปรแกรมที่ 4: Form Validation (Email Checker)

 src/App.svelte

```
<script lang="ts">
  let email: string = '';
  let error: string = '';

  function validate(): void {
    const pattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    error = pattern.test(email) ? '' : 'Invalid email address';
  }
</script>

<input type="email" bind:value={email} placeholder="Enter your email" />
<button on:click={validate}>Validate</button>

{#if error}
  <p style="color:red">{error}</p>
{:else if email}
  <p style="color:green">Valid email </p>
{/if}
```

 ผลการรัน:

กรอก email แล้วกด Validate จะแสดงว่า valid หรือไม่ (ตรวจสอบด้วย regex)

☐ โปรแกรมที่ 5: Fetch API ด้วย TypeScript

☐ *src/App.svelte*

```
<script lang="ts">
  import { onMount } from 'svelte';

  type User = {
    id: number;
    name: string;
    email: string;
  };

  let users: User[] = [];

  onMount(async () => {
    const res = await fetch('https://jsonplaceholder.typicode.com/users');
    users = await res.json();
  });
</script>

<h1>User List</h1>
<ul>
  {#each users as user}
    <li>{user.name} ({user.email})</li>
  {/each}
</ul>
```

☐ *ผลการรัน:*

แสดงรายชื่อผู้ใช้จาก API (jsonplaceholder.typicode.com/users) พร้อมอีเมล

☐ โปรแกรมที่ 6: To-do List (ใช้ Store + TypeScript)

☐ *src/stores.ts*

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

export type Todo = {
```

```
  id: number;
  text: string;
  done: boolean;
};

export const todos = writable<Todo[]>([]);
```

src/App.svelte

```
<script lang="ts">
  import { todos, Todo } from './stores';
  let newTodo: string = '';

  function addTodo(): void {
    if (newTodo.trim()) {
      todos.update(current => [
        ...current,
        { id: Date.now(), text: newTodo, done: false }
      ]);
      newTodo = '';
    }
  }

  function toggleDone(id: number): void {
    todos.update(current =>
      current.map(todo =>
        todo.id === id ? { ...todo, done: !todo.done } : todo
      )
    );
  }
</script>

<h1>My Todo List</h1>

<input bind:value={newTodo} placeholder="New task..." />
<button on:click={addTodo}>Add</button>
```

```
<ul>
  {#each $todos as todo}
    <li>
      <input type="checkbox" checked={todo.done} on:change={() => toggleDone(todo.id)} />
      <span style="text-decoration: {todo.done ? 'line-through' : 'none'}">
        {todo.text}
      </span>
    </li>
  {/each}
</ul>
```

☐ *ผลการรัน:*

ผู้ใช้สามารถเพิ่มรายการงาน กด checkbox เพื่อขีดฆ่าเมื่อเสร็จงาน

☐ **สรุป**

ประเภท	โปรแกรม	ฟีเจอร์หลัก
พื้นฐาน	Counter	ตัวแปร + ฟังก์ชัน
พื้นฐาน	Greeting Props	ส่ง props แบบ typed
พื้นฐาน	Store	ใช้ writable พร้อม type
ประยุกต์	Email Validation	ฟอร์ม + ตรวจ email
ประยุกต์	API Fetch	ใช้ TypeScript + onMount
ประยุกต์	To-do List	Store, Add, Toggle, Type-safe

การ Typing Props, Store, และ Context ใน Svelte + TypeScript

☐ **1. การ Typing Props (ใน Component)**

Svelte ใช้ export let สำหรับรับ props จาก component ภายนอก และเมื่อใช้ TypeScript เราสามารถกำหนดชนิดข้อมูลให้ props ได้ง่ายมาก

☐ *ตัวอย่างพื้นฐาน*

```
<script lang="ts">
  export let name: string;
  export let age: number;
```

```
</script>

<p>Hello {name}, age {age}</p>
```

หาก component อื่นส่ง props แบบผิดชนิดเข้ามา TypeScript จะแจ้งเตือนทันที

☐ *ตัวอย่างการส่ง props จาก main.ts*

```
import App from './App.svelte';

const app = new App({
  target: document.body,
  props: {
    name: 'Alice',
    age: 25
  }
});
```

☐ *เทคนิค: การรวม Props เป็น Interface*

หาก props หลายตัว แนะนำให้สร้าง interface แยกเพื่อให้ type ชัดเจน:

```
// types.ts
export interface UserProps {
  name: string;
  email: string;
  age?: number; // optional
}
<script lang="ts">
  import type { UserProps } from './types';
  export let name: UserProps['name'];
  export let email: UserProps['email'];
</script>
```

☐ 2. การ Typing Store

Svelte store (writable, readable, derived) รองรับ TypeScript แบบสมบูรณ์ และสามารถกำหนดชนิดข้อมูลให้กับ state ได้โดยตรง

☐ *ตัวอย่าง writable<T>()*

```
// src/stores.ts
```

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

export const count = writable<number>(0);

export const name = writable<string>('Guest');

export interface Todo {
  id: number;
  text: string;
  done: boolean;
}

export const todos = writable<Todo[]>([]);
```

ใน component:

```
<script lang="ts">
  import { count } from './stores';

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

<h1>Count: {$count}</h1>
<button on:click={increment}>Add</button>
```

☐ *Custom Store แบบมี type*

```
// src/customStore.ts
import { writable } from 'svelte/store';

interface AuthState {
  user: string;
  token: string;
}
```

```
function createAuthStore() {
  const { subscribe, set, update } = writable<AuthState | null>(null);

  return {
    subscribe,
    login: (user: string, token: string) =>
      set({ user, token }),
    logout: () => set(null)
  };
}

export const authStore = createAuthStore();
```

☐ 3. การ Typing Context (setContext / getContext)

Context ใช้ในการส่งค่าจาก component แม่ไปยัง component ลูกโดยไม่ต้องผ่าน props
ต้องใช้ Symbol() เพื่อป้องกัน key ซ้ำ และใช้ generic ใน getContext เพื่อ type ที่ปลอดภัย

☐ *ตัวอย่าง: สร้าง Context แบบ Typed*

```
// context.ts
export const ThemeKey = Symbol();
export type Theme = 'light' | 'dark';
```

☐ *ใน component แม่ (Parent)*

```
<script lang="ts">
  import { setContext } from 'svelte';
  import { ThemeKey, Theme } from './context';

  const theme: Theme = 'dark';
  setContext(ThemeKey, theme);
</script>

<slot />
```

☐ *ใน component ลูก (Child)*

```
<script lang="ts">
  import { getContext } from 'svelte';
  import { ThemeKey, Theme } from './context';
```

```
  const theme = getContext<Theme>(ThemeKey);
</script>

<p>Current theme: {theme}</p>
```

หากใช้ getContext โดยไม่ได้กำหนด type (<Theme>) จะไม่สามารถตรวจสอบได้ว่าค่าที่ได้ตรงชนิดหรือไม่

☐ **สรุป**

จุดใช้งาน	วิธี typing	ตัวอย่าง
Props	export let name: string;	รับ props อย่างปลอดภัย
Store	writable<T>(), interface	ใช้กับ state หลายแบบ
Context	getContext<Type>(key)	ใช้กับ global settings เช่น theme, user

---

☐ **ข้อดีของการ typing ที่ดี:**

- ป้องกันการส่งค่าผิดประเภท
- Editor แนะนำค่าได้ (IntelliSense)
- ปลอดภัยในการ refactor
- ทำงานร่วมกับทีมได้ง่าย

ด้านล่างนี้คือ **ตัวอย่างโปรแกรม Svelte + TypeScript** ที่แสดงการใช้ **Props, Store และ Context แบบ typed** แบ่งเป็น 2 หมวด:

☐ **โปรแกรมพื้นฐาน (3 โปรแกรม)**

แสดงวิธีใช้งานพื้นฐานของ Props, Store และ Context

☐ **โปรแกรมที่ 1: Props แบบ TypeScript**

☐ *โครงสร้าง*

```
props-demo/
├── src/
│   ├── App.svelte
│   └── main.ts
└── tsconfig.json
```

```
└── package.json
```

src/App.svelte

```
<script lang="ts">
  export let username: string;
</script>

<h1>Hello, {username}!</h1>
```

src/main.ts

```
import App from './App.svelte';

const app = new App({
  target: document.body,
  props: {
    username: 'SvelteDev'
  }
});

export default app;
```

คำอธิบาย:

- ใช้ export let พร้อมกำหนด type (string)
- ส่งค่า username แบบ type-safe

ผลการรัน:

```
Hello, SvelteDev!
```

โปรแกรมที่ 2: Writable Store แบบมี Type

โครงสร้าง

```
store-demo/
├── src/
│   ├── App.svelte
│   ├── stores.ts
│   └── main.ts
```

src/stores.ts

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