

CIS 658 Web Architectures

TypeScript VI



GRAND VALLEY
STATE UNIVERSITY®

Lecturer: **Dr. Yong Zhuang**

Recall

- Type Alias vs. Interface vs. Class, Inheritance
- Generics in TypeScript

Practice 1: Interface

Answer

Practice 2: Generic Type

Answer

Using TypeScript in Browser

Browser Debugger (Chrome)

Debugger Controls
(step over, step into, ...)

Debugging breakpoints

The screenshot displays the Chrome DevTools interface. The 'Sources' panel is active, showing a file tree on the left and a code editor in the center. A breakpoint is set at line 9 of the file `history_cluster...mojom-webui.js`. The code in the editor is as follows:

```
1 // chrome/browser/new_tab_page/modules/history_clusters/history_clus
2 // Copyright 2020 The Chromium Authors
3 // Use of this source code is governed by a BSD-style license that c
4 // found in the LICENSE file.
5 import { mojo } from '../resources/mojo/mojo/public/js/bindings.js';
6 export const LayoutTypeSpec = { $: mojo.internal.Enum() };
7 export var LayoutType;
8 (function (LayoutType) {
9   LayoutType[LayoutType["kNone"] = 0] = "kNone";
10  LayoutType[LayoutType["kLayout1"] = 1] = "kLayout1";
11  LayoutType[LayoutType["kLayout2"] = 2] = "kLayout2";
12  LayoutType[LayoutType["kLayout3"] = 3] = "kLayout3";
13  LayoutType[LayoutType["kTextOnly"] = 4] = "kTextOnly";
14  LayoutType[LayoutType["kImages"] = 5] = "kImages";
15  LayoutType[LayoutType["MIN_VALUE"] = 0] = "MIN_VALUE";
16  LayoutType[LayoutType["MAX_VALUE"] = 5] = "MAX_VALUE";
17 }) (LayoutType || (LayoutType = {}));
18
```

The 'Debugger Controls' panel on the right shows the 'Paused on breakpoint' state. It includes sections for 'Threads', 'Watch', 'Breakpoints', 'Scope', 'Local', 'Module', and 'Call Stack'. The 'Breakpoints' section shows a list of breakpoints, with the current one highlighted:

| Breakpoint | Line |
|--|------|
| LayoutType[LayoutType["kNone"] = 0] = "kNone"; | 9 |
| LayoutType[LayoutType["MIN_VALUE"] = 0] = "MIN_VALUE"; | 15 |

The 'Variable Inspector' panel shows the current scope and local variables:

- Local: `this: undefined`
- LayoutType: `{}`
- Module: `LayoutType: {}`, `LayoutTypeSpec: { $: {...}}`
- mojo: `{internal: {...}, interfaceControl: {...}, pipeControl: {...}}`
- Global: `Window`
- Call Stack:
 - (anonymous) history_cluster...jom-webui.js:9
 - (anonymous) history_cluster...om-webui.js:17

Variable Inspector

Including JS code in HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="code1.js"></script>
  </head>
  <body>
    <!-- other HTML contents go here -->
    <script src="code2.js"></script>
  </body>
</html>
```

Scripts that do not modify page contents are placed in <head>

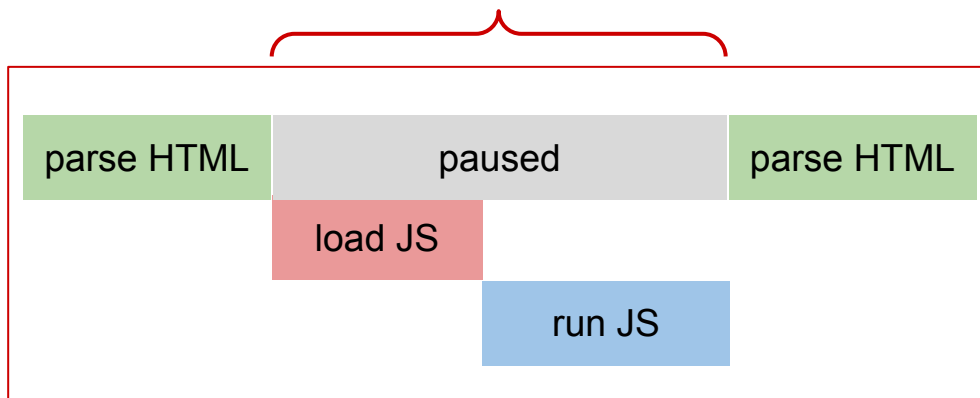
Scripts that do are placed towards the end of <body>

Script: Loading & Running

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>
  </head>
  <body>
    <!-- some HTML here -->
    <script src="...js"></script>
    <!-- more HTML here -->
  </body>
</html>
```

<script>

HTML Parsing suspended



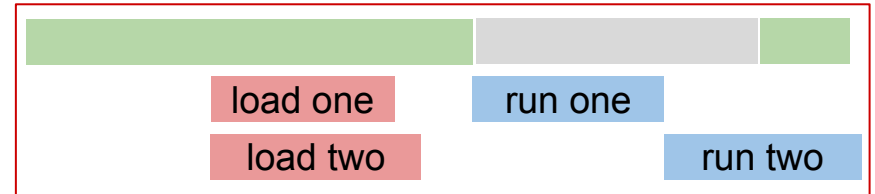
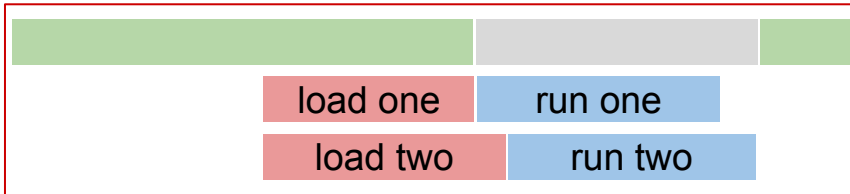
Defer vs. Async

```
<html>
  <body>
    <!-- some HTML here -->
    <script src="one" async></script>
    <script src="two" async></script>
    <!-- more HTML here -->
  </body>
</html>
```

`<script async>`

```
<html>
  <body>
    <!-- some HTML here -->
    <script src="one" defer></script>
    <script src="two" defer></script>
    <!-- more HTML here -->
  </body>
</html>
```

`<script defer>`



When should I use what?

Typically you want to use **async** where possible, then **defer** then no attribute. Here are some general rules to follow:

- If the script is modular and does not rely on any scripts then use `async`.
- If the script relies upon or is relied upon by another script then use `defer`.
- Avoid complex dependencies with `async`.

[async vs defer attributes](#)

How to transpile TypeScript code so that it can run in a browser

TS <script> option #1: Babel

with babel-standalone

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/babel" src="code1.ts"></script>
  </head>
  <body>
    <!-- HTML contents go here -->
    <script type="text/babel" src="code2.ts"></script>
  </body>
</html>
```

*DO NOT use Babel standalone for **production**
Use transpiled JS for production with bundler
(webpack, parcel, rollup, ...)*

TS <script> option #2: ParcelJS

```
npm init -y
npm install --save-dev parcel

# Create your-file.html with <script>
# Create one.ts and two.ts

npx parcel serve your-file.html

# Go to localhost:xxxx (in a browser)
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="one.ts"></script>
  </head>
  <body>
    <!-- HTML contents go here -->
    <script src="two.ts"></script>
  </body>
</html>
```

```
// one.ts
console.log("Hello from one");
```

```
// two.ts
console.log("Hello from two");
```

Browser predefined classes

- Classes associated with individual HTML tags

| Tag | Class |
|----------|---------------------------------|
| <a> | <u>HTMLAnchorElement</u> |
| <body> | <u>HTMLBodyElement</u> |
| <button> | <u>HTMLButtonElement</u> |
| | <u>HTMLImageElement</u> |
| <p> | <u>HTMLParagraphElement</u> |
| | <u><i>and many more ...</i></u> |

- Other classes: AudioBuffer, Bluetooth, ByteString, Promise, Request,...

Browser (Predefined) Objects

- Frequently used
 - screen: the computer screen occupied by the browser
 - document: the current HTML document that hosts the script
 - Provides functions for manipulating the DOM tree
 - window: the current window where the HTML doc is rendered
- Less frequently used
 - history: page visit history stack
 - localStorage: the browser persistent storage
 - location: the browser input box
 - and many more ...

```
for (const z in window) {  
    if (typeof window[z] === "object") {  
        console.log(z);  
    }  
}
```

Try this yourself

Browser window predefined functions

- `alert()`: show an info dialog on the browser
- `addEventListener()`: setup event listeners
- `confirm()`: show a yes/no dialog
- `prompt()`: show an input dialog
- `setInterval()`, `setTimeout()`: start a timer
- `clearInterval()`, `clearTimeout()`: reset existing timer
- ...
- and many more ...

```
for (const z in window) {  
    if (typeof window[z] === "function") {  
        console.log(z);  
    }  
}
```

Try this yourself

Complete documentations: [Web Windows API](#) (MDN: Mozilla Dev Network)

HTML Document CRUD methods/functions

| | |
|---------------|--|
| Create | <code>document.createElement()</code> , <code>document.createTextNode()</code> |
| Read | <code>___.getElementById()</code> // SINGULAR <code>___.getElementsByTagName()</code> // PLURAL <code>___.getElementsByClassName()</code> // PLURAL <code>___querySelector()</code> // SINGULAR: search by CSS selectors <code>___querySelectorAll()</code> // PLURAL: search by CSS selectors |
| Update | <code>___.appendChild()</code> |
| Delete | <code>___.removeChild()</code> |

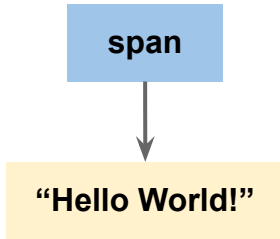
and many more ...

```
for (const z in document) {  
  if (typeof document[z] === "function") {  
    console.log(z);  
  }  
}
```

Try this yourself

Create Text Nodes

```
<span>Hello world!</span>
```

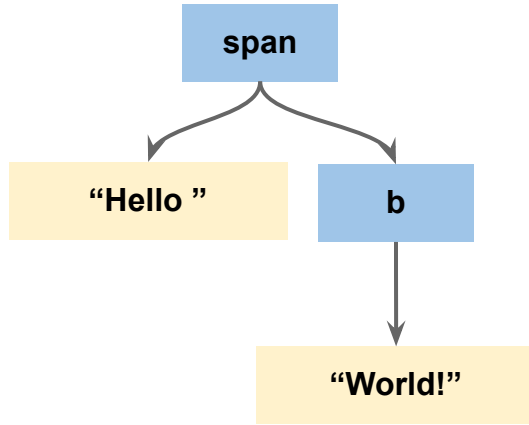


```
// Option 1
const spanParent = document.createElement("span");
const hello = document.createTextNode("Hello World");
spanParent.appendChild(hello);
```

```
// Option 2
const spanParent = document.createElement("span");
spanParent.innerText = "Hello World";
```

Add Multiple Children

```
<span>Hello <b>world!</b></span>
```



```
const spanTop = document.createElement("span");
const txt1 = document.createTextNode("Hello");
spanTop.appendChild(txt1);

const bChild = document.createElement("b");
bChild.innerText = "World";
spanTop.appendChild(bChild);
```

Insert Contents into Existing DOM

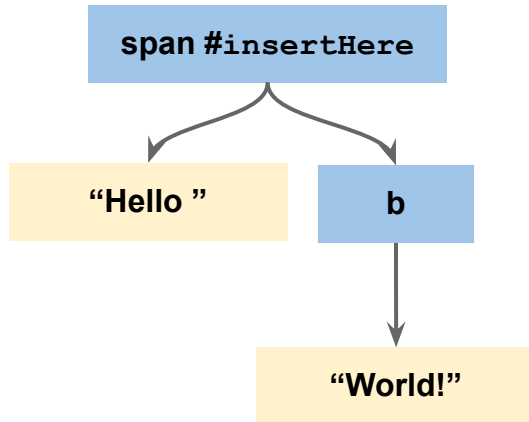
before

```
<span id="insertHere"> </span>
```

after

```
<span id="insertHere">  
  Hello <b>world!<b>  
</span>
```

```
const spanTop = document.getElementById("insertHere");  
const txt1 = document.createTextNode("Hello");  
spanTop.appendChild(txt1);  
  
const bChild = document.createElement("b");  
bChild.innerText = "World";  
spanTop.appendChild(bChild);
```

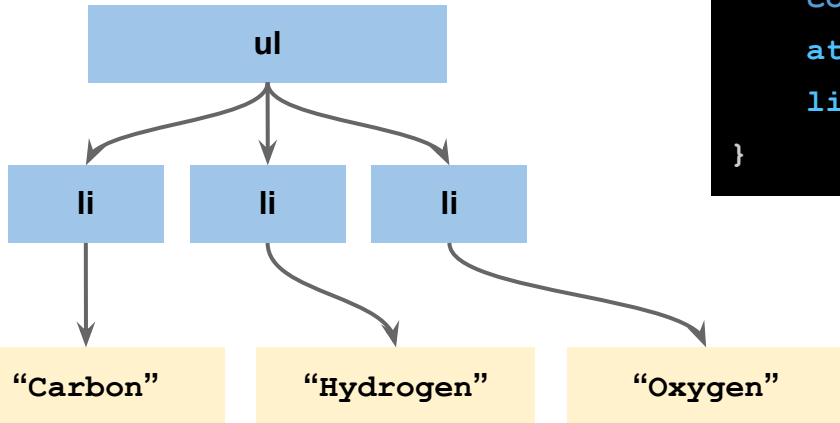


Add Multiple Children from Array

```
<ul>
  <li>Carbon</li>
  <li>Hydrogen</li>
  <li>Oxygen</li>
</ul>
```

```
const atoms = ["Carbon", "Hydrogen", "Oxygen"]
const listTop = document.createElement("ul");

for (let a of atoms) {
  const atm = document.createElement("li");
  atm.innerText = a;
  listTop.appendChild(atm);
}
```



Setting attributes

```
<a id="intro" class="deepIndent noAds" href="http://go.org">  
  Some text here  
</a>
```

```
const sample = document.createElement("a");  
sample.innerText = "Some text here";  
sample.id = "intro"  
sample.classList.add("deepIndent");  
sample.classList.add("noAds");  
sample.setAttribute("href", "http://go.org");
```

```
const sample = document.createElement("a");  
sample.innerText = "Some text here";  
sample.setAttribute("id", "intro");  
sample.setAttribute("class", "deepIndent noAds");  
sample.setAttribute("href", "http://go.org");
```

querySelector(): select ONE element

```
<body>
  <p class="title">Ice Cream Flavor:</p>
  <ul>
    <li>Death by Chocolate</li>
    <li>Mint Chocolate Chip</li>
    <li>Strawberry</li>
  </ul>
  <script src="ice.ts">
</body>
```

Ice Cream Flavors:

- Too much Chocolate
- Mint Chocolate Chip
- Strawberry

```
const item:Element = document.querySelector("ul > li");
#the first one will be returned
item.textContent = "Too much Chocolate";
```

querySelectorAll(): select MULTIPLE elements

```
<body>
  <p class="title">Ice Cream Flavor:</p>
  <ul>
    <li>Death by Chocolate</li>
    <li>Mint Chocolate Chip</li>
    <li>Strawberry</li>
  </ul>
  <script src="ice.ts">
</body>
```

Ice Cream Flavors:

- Death by Chocolate (on sale)
- Mint Chocolate Chip (on sale)
- Strawberry

```
let items:NodeListOf<Element>;
items = document.querySelectorAll("ul > li");
for (let flav of items) {
  if (flav.textContent.includes("Chocolate")) {
    flav.textContent = flav.textContent + " (on sale)";
  }
}
```

CSS Selector and querySelector(All)

```
<body>
  <h2>Some heading</h2>
  <p>First paragraph</p>
  <ol>
    <li class="fruit">Strawberry</li>
    <li class="device">Raspberry Pi</li>
    <li class="singer">Barry Manilow</li>
  </ol>
  <p>Second paragraph</p>
</body>
```

```
const q1 = document.querySelector("h2 + p");
q1.classList.add("red"); // Affect "First paragraph"
const q2 = document.querySelector("h2 ~ ol > li:first-child");
q2.classList.add("red"); // Affect "Strawberry"
const q3 = document.querySelector("li:last-child");
q3.classList.add("red"); // Affect "Barry Manilow"
```

```
const pars = document.querySelectorAll("h2 ~ p");
for (let x of pars) {
  // Apply to "First paragraph" and "Second paragraph"
  x.setAttribute("__", "__");
}
const who = document.querySelectorAll("ol > li.singer");
for (let x of who) {
  // Apply to "Barry Manilow"
}
```

Using Timer

```
<body>
  <p>Ice Cream Flavor:</p>
  <ul>
    <li>Death by Chocolate</li>
    <li>Mint Chocolate Chip</li>
    <li>Strawberry</li>
    <li>Bluemoon</li>
  </ul>
  <script src="ice.ts">
</body>
```

```
function choco() {
  const item:Element = document.querySelector("ul > li");
  item.textContent = "Too much Chocolate";
}
setTimeout(choco, 2000);
```

Ice Cream Flavors:

- Too much Chocolate
- Mint Chocolate Chip
- Strawberry
- BlueMoon

2 seconds later



setTimeout(someFunc, delayInMillisec)

JavaScript Events

| Source of Event | Events |
|--|---|
| Window | onload, onresize, onunload, ... |
| Document | onkeydown, onkeyup, onmousedown, onmouseup, onmouseenter, onmouseleave, ... |
| Input field | onblur, onfocus, onchange, |
| Button | onclick, ondblclick |
| Complete Reference: Event APIs | |

Setting Up Event Handlers

- Which Event?
- Who is the event source?
 - Resize => window
 - Key presses => document
 - Load => document
 - Click => button, image,
 - Focus => input elements
 - Mouse => elements
- Details of the event object properties
(MouseEvent, KeyboardEvent,).

Refer to online API

```
function keyHandler(ev: KeyboardEvent): void {  
    // put code here  
}  
  
function clickHandler(ev:MouseEvent): void {  
    // put code here  
}  
  
document.addEventListener("keypress", keyHandler);  
  
const myLogo = document.getElementById("myLogo");  
myLogo.addEventListener("click", clickHandler);
```

addEventListener

```
<button onclick="clickFunction()">Click Me</button>
```

inline event attributes

```
function clickFunction() {  
    // put code here  
}
```

Practice: Event Handling

Counting Click



Given the HTML and JavaScript code in the above link, implement functionality for the `<h1 id="h1counter">` element so that when it is clicked, the `<p id="outputH1counter">` displays 'Clicked ... times'.