.NET Conf TAIWAN 2024

Angular Signals 實戰演練

Mike Huang 黃升煌

https://fullstackladder.dev

https://github.com/wellwind

https://www.facebook.com/fullstackledder

.NET Conf TAIWAN 2024

DEMO

https://github.com/wellwind/dotnet-conf-2024-angular-signals

.NET Conf TAIWAN 2024

Reactive Programming

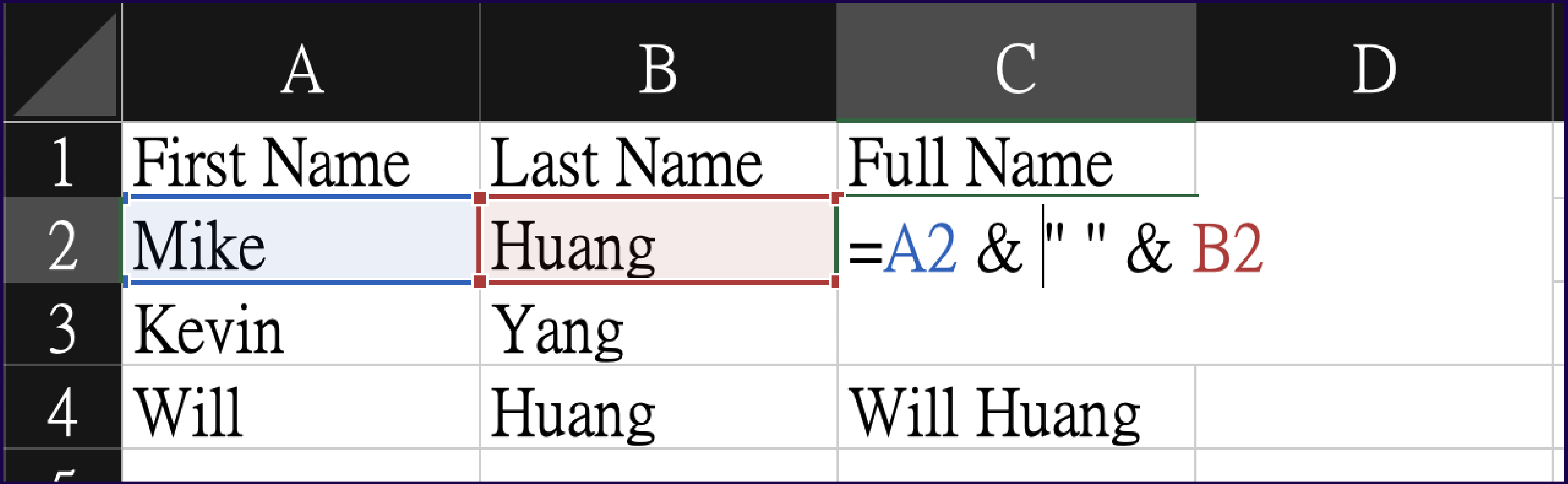
What is Reactive Programming

- 回應式程式設計

- 回應變化
  - 只針對特定的來源變化做出反應

- 你有變化，我才回應

Excel Example



	A	B	C	D
1	First Name	Last Name	Full Name	
2	Mike	Huang	=A2 & " " & B2	
3	Kevin	Yang		
4	Will	Huang	Will Huang	

Signal Example

.NET Conf TAIWAN 2024

```
firstName = 'John'
lastName = 'Doe'
fullName = 'John Doe'

updateFirstName(value: string) {
  this.firstName = value;
  this.fullName =
    `${this.firstName} ${this.lastName}`;
}

updateLastName(value: string) {
  this.lastName = value;
  this.fullName =
    `${this.firstName} ${this.lastName}`;
}
```

不是回應變化，單純的事件驅動後計算結果

```
firstName = signal('John');
lastName = signal('Doe');

fullName = computed(() =>
  `${this.firstName()} ${this.lastName()}`);

updateFirstName(value: string) {
  this.firstName.set(value);
}

updateLastName(value: string) {
  this.lastName.set(value);
}
```

回應變化，根據來源自動組合結果

Angular Signal 實戰

zoneless

.NET Conf TAIWAN 2024

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideExperimentalZonelessChangeDetection(),
  ],
};
```

```
{
  "projects": {
    "your-app": {
      ...,
      "architect": {
        "build": {
          "options": {
            ...,
            "polyfills": [
              ~~"zone.js"~~
            ]
          }
        }
      }
    }
  }
}
```

(實驗性質)不使用 zone.js
之後可移除 polyfills

Computed 被執行時機 (1)

```
@Component({
  template: `
    <div>Current Page: {{ page() }}</div>
    <!-- <div>Next Page: {{ nextPage() }}</div> -->
    <button (click)="goNext()">Next Page</button>
  `,
})
export class AppComponent {
  page = signal(1);

  nextPage = computed(() => {
    const currentPage = this.page();
    console.log(`currentPage: ${currentPage}`);
    return currentPage + 1;
  });

  goNext() {
    this.page.update((current) => current + 1);
    // console.log(`nextPage: ${this.nextPage()}`);
  }
}
```

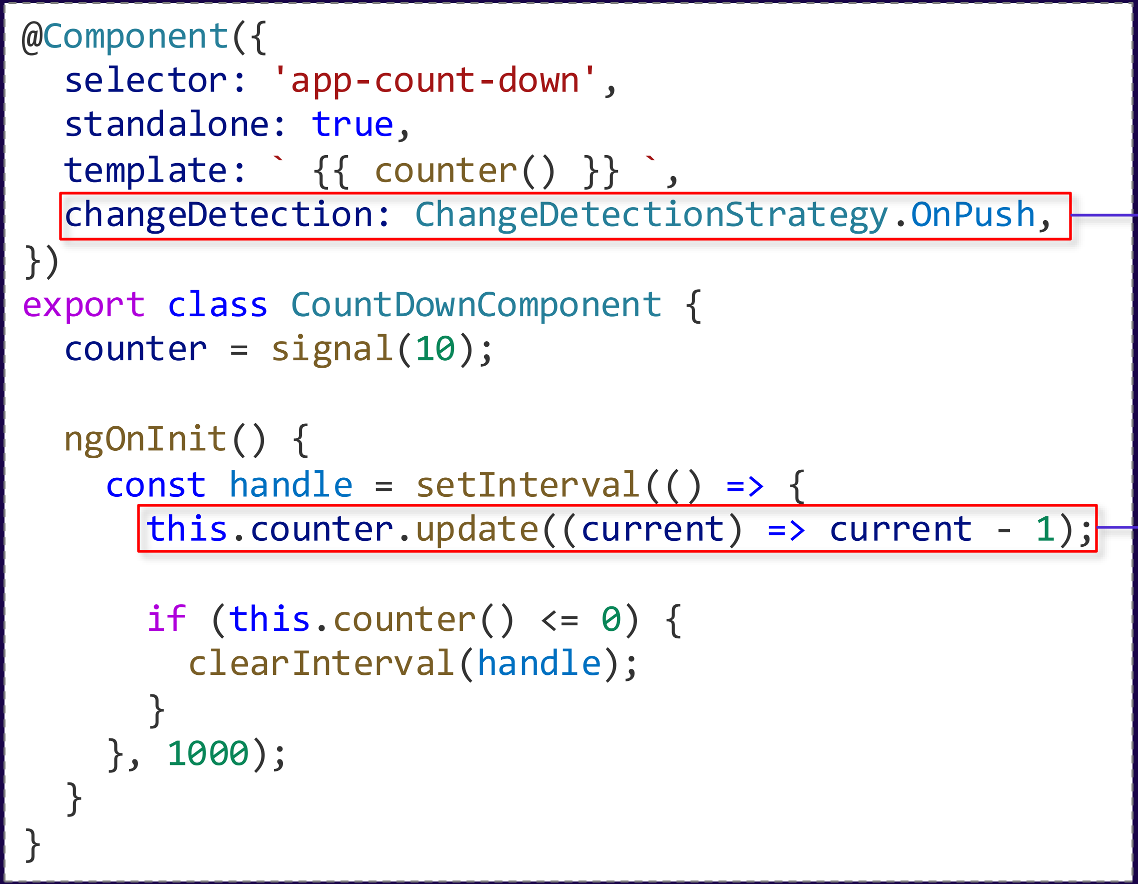
沒被使用到，不會被執行

Computed 被執行時機 (2)

```
page = signal(1);
displayPagination = signal(false);

nextPage = computed(() => {
  console.log('calculate next page');
  return this.page() + 1;
});

pagination = computed(() => {
  console.log('calculate pagination');
  return this.displayPagination()
    ? { page: this.page(), nextPage: this.nextPage() }
    : undefined;
});
```

當 displayPagination 為 false 時
page 和 nextPage 變更不會被追蹤

Angular Signal 搭配 OnPush 策略

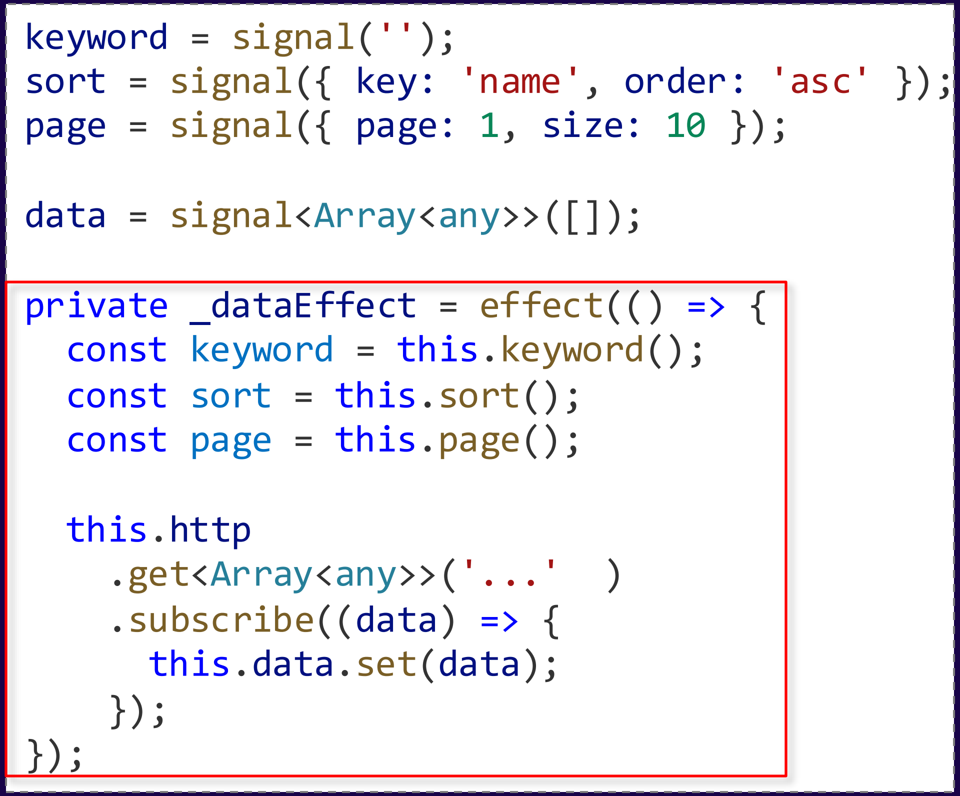


```
@Component({
  selector: 'app-count-down',
  standalone: true,
  template: ` {{ counter() }} `,
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class CountDownComponent {
  counter = signal(10);

  ngOnInit() {
    const handle = setInterval(() => {
      this.counter.update((current) => current - 1);

      if (this.counter() <= 0) {
        clearInterval(handle);
      }
    }, 1000);
  }
}
```

使用 OnPush 策略

Signal 變更自動觸發畫面變更

非同步 effect



```
keyword = signal('');
sort = signal({ key: 'name', order: 'asc' });
page = signal({ page: 1, size: 10 });

data = signal<Array<any>>([]);

private _dataEffect = effect(() => {
  const keyword = this.keyword();
  const sort = this.sort();
  const page = this.page();

  this.http
    .get<Array<any>>('...'   )
    .subscribe((data) => {
      this.data.set(data);
    });
});
```

在變更偵測的過程中，effect 本身是非同步被執行的，
會在一次同步週期內的 signal 全部更新完畢才會執行

如果同步地更新 keyword、sort 和 page，
effect 只會被執行一次

儘量避免在 effect 內設定狀態

```
firstName = signal('Mike');
lastName = signal('Huang');

fullName = computed(() =>
  `${this.firstName()} ${this.lastName()}`);
```

```
firstName = signal('Mike');
lastName = signal('Huang');
fullName = signal('');

fullNameEffect = effect(() => {
  this.fullName.set(
    `${this.firstName()} ${this.lastName()}`);
});
```

利用 untracked 解除對 signal 的相依關係

userProfile 變更時，effect 不會被執行

```
userId = signal(1);
userProfile = signal({ name: 'Mike' });

private _userEffect = effect(() => {
  const userId = this.userId();
  const userProfile = untracked(() => this.userProfile());

  console.log(`User Changed, ID = ${userId}, Name = ${userProfile.name}`);
});
```

toSignal

```
data$ = this.http.get('https://jsonplaceholder.typicode.com/todos/1');

data = toSignal(this.data$);
```

注意！此時 data$ 會立刻被訂閱

如果是在元件內使用 toSignal，在元件 destroy 時會取消訂閱
如果是在 root service 內使用，則不會隨著元件 destroy 取消訂閱
可能造成 memory leak

除非你很清楚你在做什麼，否則應該盡量避免在 service 內使用 toSignal

toObservable

```
todoId = signal<number>(-1);
todoId$ = toObservable(this.todoId);

ngOnInit() {
  this.todoId$.subscribe((id) => {
    console.log(id);
    // (A) -1, 2, 3
    // (B) 2, 3
    // (C) 3
  });

  this.todoId.set(2);
  this.todoId.set(3);
}
```

在同步週期內只會處理 "最後一次變化" 的值

透過 RxJS 避免在 Effect 內更新 Signal

.NET Conf TAIWAN 2024

```
protected loading = signal(false);

_loadingEffect = effect(() => {
  const _ = this.searchCondition();
  this.loading.set(true);
}, { allowSignalWrites: true });

_loadedEffect = effect(() => {
  const _ = this.result();
  this.loading.set(false);
}, { allowSignalWrites: true });
```

Angular 18
預設在 effect 裡面更新 Signal 是不允許的，
但可以透過設定 `allowSignalWrites: true` 來放行

（請確認定你知道你在做什麼）

（Angular 19 不需要，預設為 true）

```
private loading$ =
this.searchCondition$.pipe(
  switchMap(() => this.result$.pipe(
    map(() => false),
    startWith(true)
  )),
);

protected loading = toSignal(
  this.loading$,
  { initialValue: false }
);
```

透過 RxJS 有時候可以寫出更合理的程式

linkedSignal (Angular 19)

```
protected keyword = signal('ng');

protected pageNumber = linkedSignal({

  source: () => this.keyword(),

  computation: (source, prev) => {
    if (prev?.source !== source) {
      return 1;
    }
    return prev.value ?? 1;
  },

});

updatePage(num: number) {
  this.pageNumber.set(num);
}
```

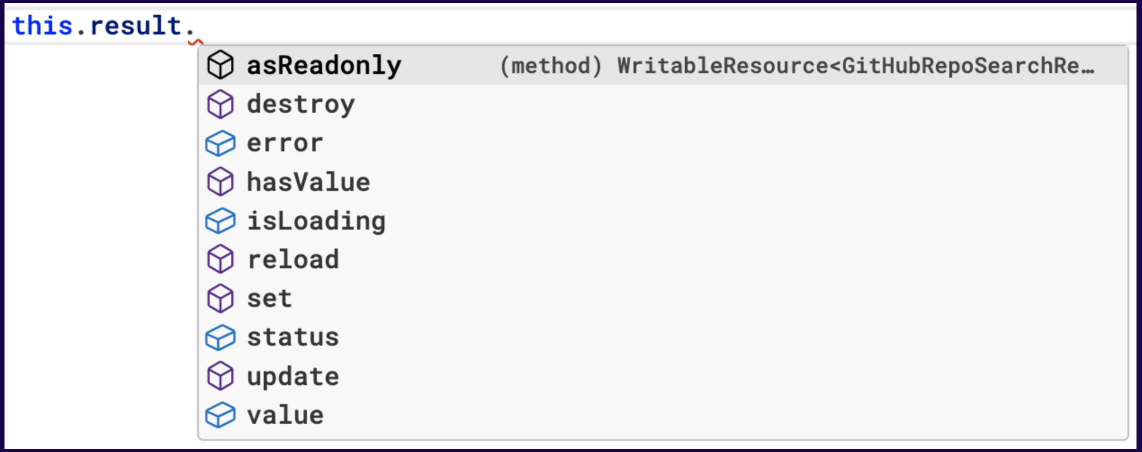
依照指定來源，回傳一個運算結果
（類似 computed）

使用 `linkedSignal` 得到的依然是一個 `WriteableSignal<T>`
因此可以更新它的值
（減少使用 `effect` 更新 `Signal` 的機會）

resource (Angular 19)

```
protected result = resource({

  request: () => this.searchConditionWithDebounce(),

  loader: (condition) =>
    lastValueFrom(this.service.search(condition.request)),
});
```

依照指定來源，回傳一個**非同步**的結果（Promise）



```
this.result.
asReadonly    (method) WritableResource<GitHubRepoSearchRe…
destroy
error
hasValue
isLoading
reload
set
status
update
value
```

提供多種方法輔助我們設定/取得目前狀態

Signal in Service

```
@Injectable({ providedIn: 'root' })
export class FileStore {
  private _selectedFile = signal<Array<string>>([]);

  get selectedFile() {
    return this._selectedFile.asReadonly();
  }

  selectedFileCount = computed(() =>
    this._selectedFile().length);

  selectFile(fileId: string) {
    this._selectedFile
      .update((fileIds) => [...fileIds, fileId]);
  }
}
```

對外隱藏 Writable Signal

對外只公開 Read Only Signal
避免被任意改動

限制更新 Signal 的方法

除非你很清楚你在做什麼，否則應該盡量避免在 service 內使用 toSignal

.NET Conf TAIWAN 2024

outputFromObservable

```
userService = inject(UserService);

userId = input.required<number>();
userId$ = toObservable(this.userId);

user$ = this.userId$.pipe(
  switchMap((id) => this.userService.getUserProfile(id)),
  shareReplay(1)
);
user = toSignal(this.user$);

userLoaded = outputFromObservable(this.user$);
```

toSignal 會訂閱一次

注意！當外部元件使用這個 output 事件時也會訂閱此 Observable 物件

避免重複訂閱的技巧，使用 share or shareReplay operator

signal queries

```
@Component({
  ...,
  template: `
    <ul>
      @for(todoItem of todoItems(); track todoItem.id) {
        <li>
          <app-todo-item #item [todoItem]="todoItem"></app-todo-item>
        </li>
      }
    </ul>
`,
})
export class AppComponent {
  todoItemComponents = viewChildren<TodoItemComponent>('item');

  ...
}
```

直接就是 Array<TodoItemComponent>，不用再處理 QueryList

Signal 思考

- [相對]簡單，且解決了部分複雜的 RxJS 狀態
- 不是拿來取代 RxJS 的，混在一起用體驗最好
- 思考方向需要在 RxJS 與 Signal 中靈活切換
- 當你主動變更一個狀態時，想想這個狀態的來源是什麼
- 開始用！就對了！！

.NET Conf TAIWAN 2024

Thank You

https://fullstackladder.dev

https://github.com/wellwind

https://www.facebook.com/fullstackledder

.NET Conf
TAIWAN
2024

STUDY4

為 學 習 而 生