Angular Elements

How to communicate between components

Denis Severin, Angular developer
Fundamentals team

Angular elements

Allows developers to wrap their components into standalone Web Components

fundamental conf

Benefits of Angular Elements

- Can be added to any existing application without time-consuming code refactoring;
- Optional self-contained Shadow DOM;
- In conjunction with Module Federation gives optimised bundle size;
- Easy to use for developers who aren’t familiar with Angular.

Dark side of Angular Elements

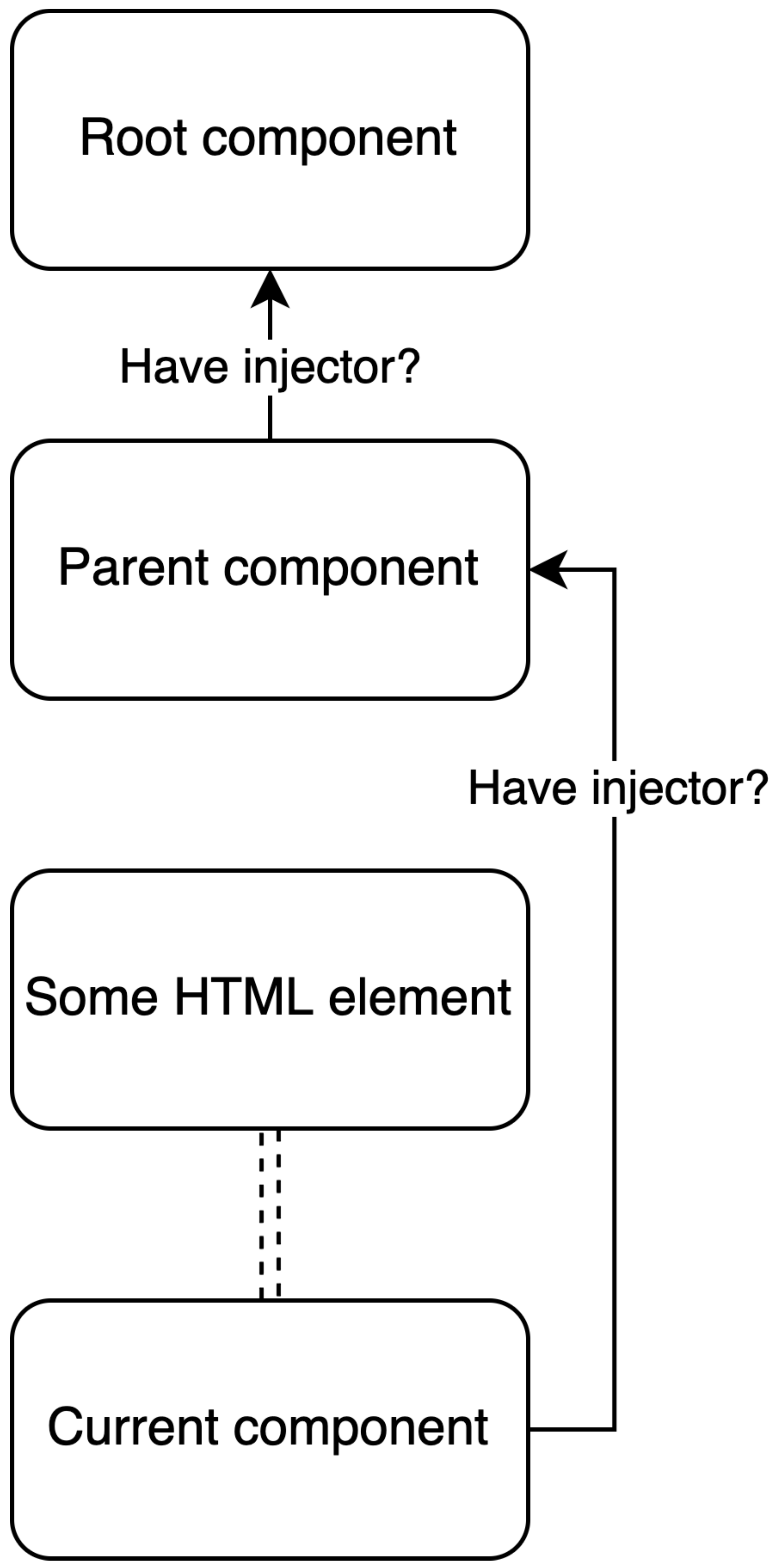
- Complex NgModule
- Components are bound to module injector

Self-contained Web Components

How we can make Angular Elements be aware of parent/child components and communicate with them?

Communicating with parents

Ask for parent's element Injectors!


Root component
Have injector?
Parent component
Have injector?
Some HTML element
Current component

```
while (!injectorFound) {
    const parent = currentElement.parentElement as any | null;
    if (!parent) {
        break;
    }

    if (parent.__fdModuleInjector) {
        injectorFound = true;
        parentInjector.moduleInjector = parent.__fdModuleInjector;
        parentInjector.nodeInjector = parent.__fdInjector;
        break;
    }

    currentElement = parent;
}

// Maybe we are inside a web component shadow dom, so try to grab a host.
if (!injectorFound) {
    const host = (element.getRootNode() as ShadowRoot).host as any;
    if (host?.__fdModuleInjector) {
        parentInjector.moduleInjector = host.__fdModuleInjector;
        parentInjector.nodeInjector = host.__fdInjector;
    }
}

if (injectorFound && parentInjector.moduleInjector !== this.injector) {
    (this.injector as any).parent = parentInjector.nodeInjector;
}

element.__fdModuleInjector = this.injector;

const projectableNodes =
    extractProjectableNodes(element, this.componentFactory.ngContentSelectors);
this.componentRef = this.componentFactory.create(this.injector, projectableNodes, element);
```

Try to get parent injector

Update parent injector

Initialise component with updated injector

Communicating with Children

fundamental conf

Angular way

- @ViewChild (child element in template)
- @ViewChildren (child elements in template)
- @ContentChild (projected child element)
- @ContentChildren (projected child elements)

Benefits:

- Supports query selectors
- Supports Class name selectors
- Supports Injection token selectors

Web-Component way

```
> const childElements = document.querySelectorAll('web-component-child')
```

fundamental conf

Benefits

- At least it works

Leveraging both approaches

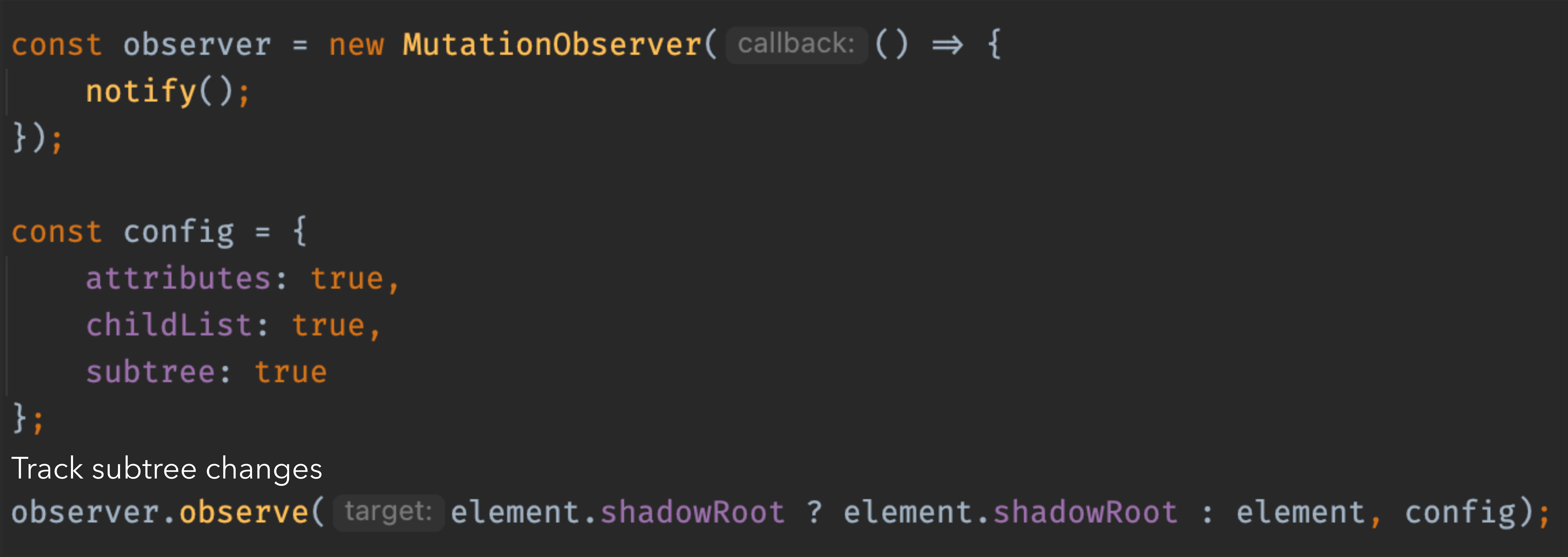
Fundamental conf

Technicalities

- React inner DOM tree change;
- Ask child custom elements if it fits the selector (query selector, Class name, Injection token);

Reacting on inner DOM changes

With the help of MutationObserver we can actually listen on inner Sub Tree changes and react appropriately (collect new data).



```
const observer = new MutationObserver( callback: () => {
    notify();
});

const config = {
    attributes: true,
    childList: true,
    subtree: true
};
```

Track subtree changes

```
observer.observe( target: element.shadowRoot ? element.shadowRoot : element, config);
```

Checking if element fits the conditions

With simple string selector we can use querySelectorAll.

With Class references and Injection tokens we can ask element if it has injector, and if so, try to get fitting element from the injector itself.

```
/**
 * Searches for the elements that fits the selector conditions
 * @param root Root element to search in
 * @param selector Either a string (Query Selector), or a Type.
 */
function getItems(root: HTMLElement | ShadowRoot, selector: string | Type<any>): Element[] | Type<any>[] {
    if (typeof selector === 'string') {
        return Array.from(root.querySelectorAll(selector));
    }

    // Go through each item and check if it's the needed class
    const elements: FdWebComponent[] = [];

    root.querySelectorAll('*').forEach( callbackfn: (childElement : Element ) => {
        if (!isFdWebComponent(childElement)) {
            return;
        }

        // eslint-disable-next-line no-bitwise
        if (childElement.__fdInjector.get(selector,  notFoundValue: null,  flags: InjectFlags.Self | InjectFlags.Optional)) {
            elements.push(childElement);
        }
    });

    return elements;
}
```

Return array of elements from query selector

Doesn't affect the performance due to Shadow DOM

If element's injector has such token, include it to resulting array

Final result

No need for heavy code refactoring

```
@WebComponentQuery(TAB_PANEL_TOKEN)
@ContentChildren(forwardRef( forwardRefFn: () ⇒ TabPanelComponent))
tabPanels: QueryList<TabPanelComponent>;
```

Automatic web component generation

Fundamental conf

Technicalities

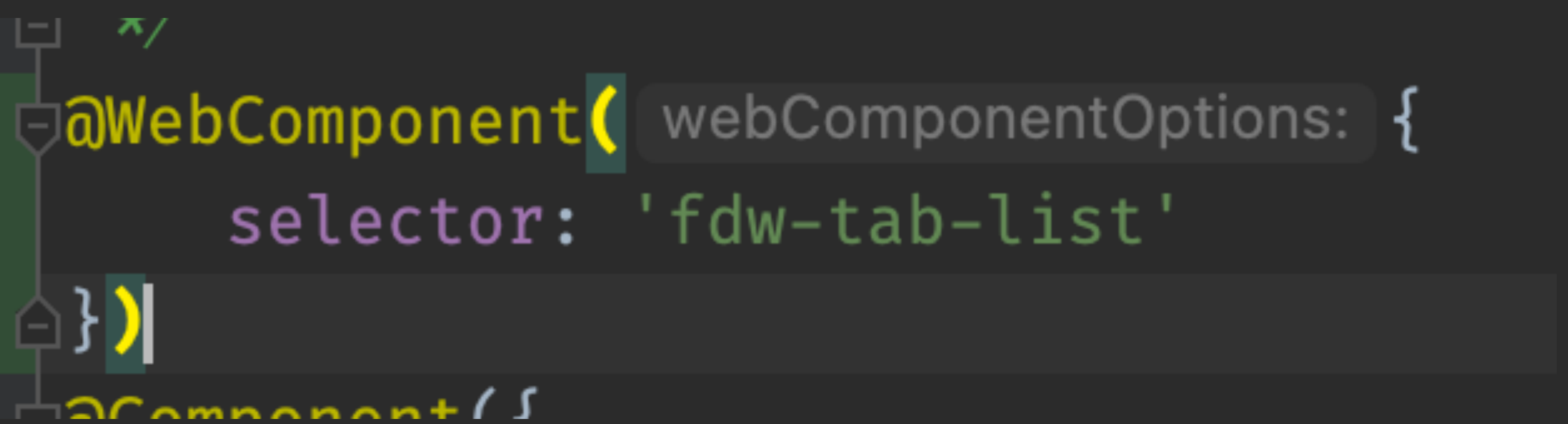
fundamental conf

- Backwards capability
- Agnostic approach
- Automated generator

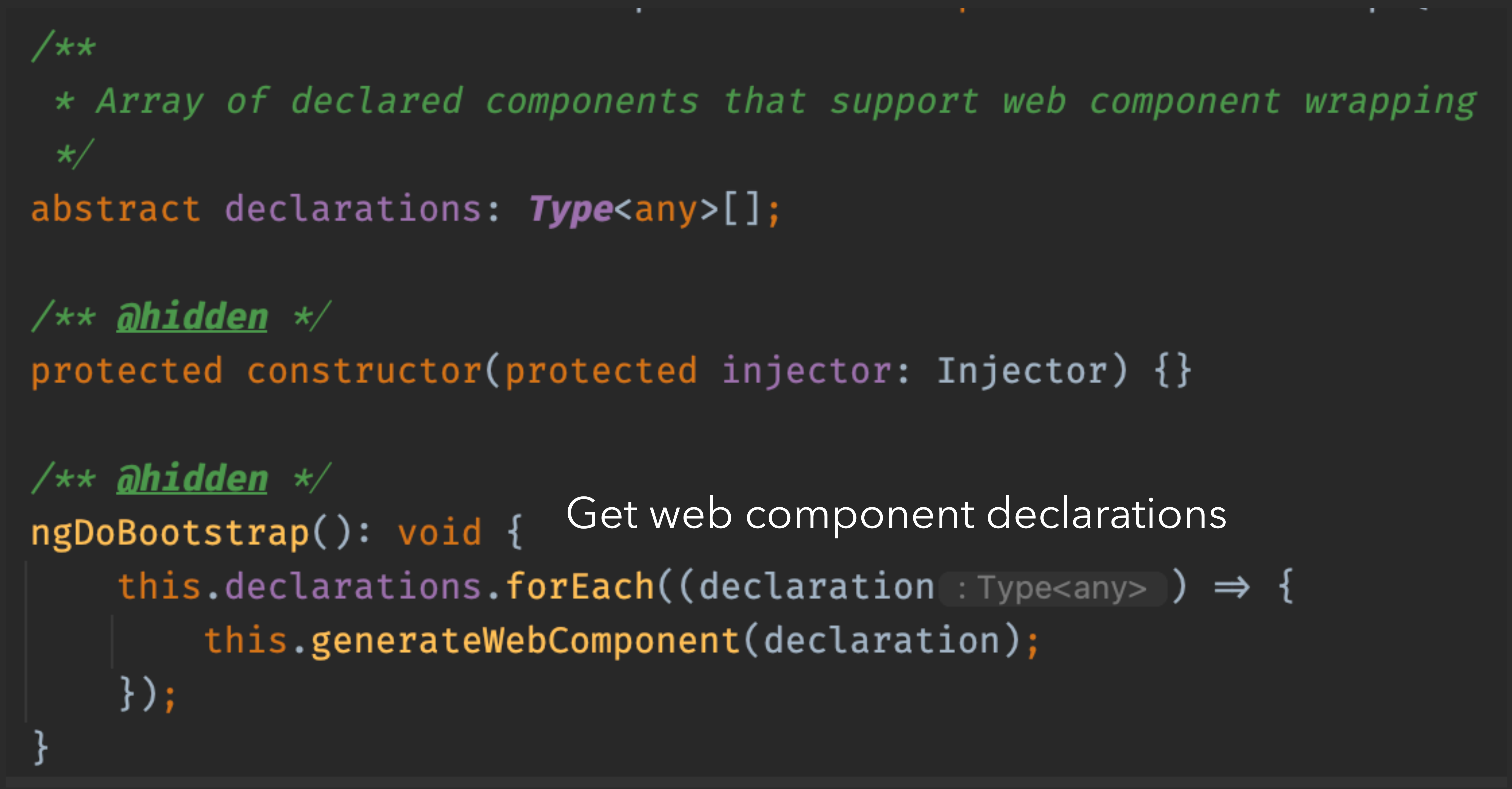
```
export function WebComponent(webComponentOptions: WebComponentOptions): (constructor: Type<any>) => void {

    return function(constructor : Type<any> ) {
        constructor.prototype['__fdWebComponentSelector'] = webComponentOptions.selector;
        defineLifecycle(constructor);
    };
}
```

Store web component metadata

Use on top of Angular's @Component decorator

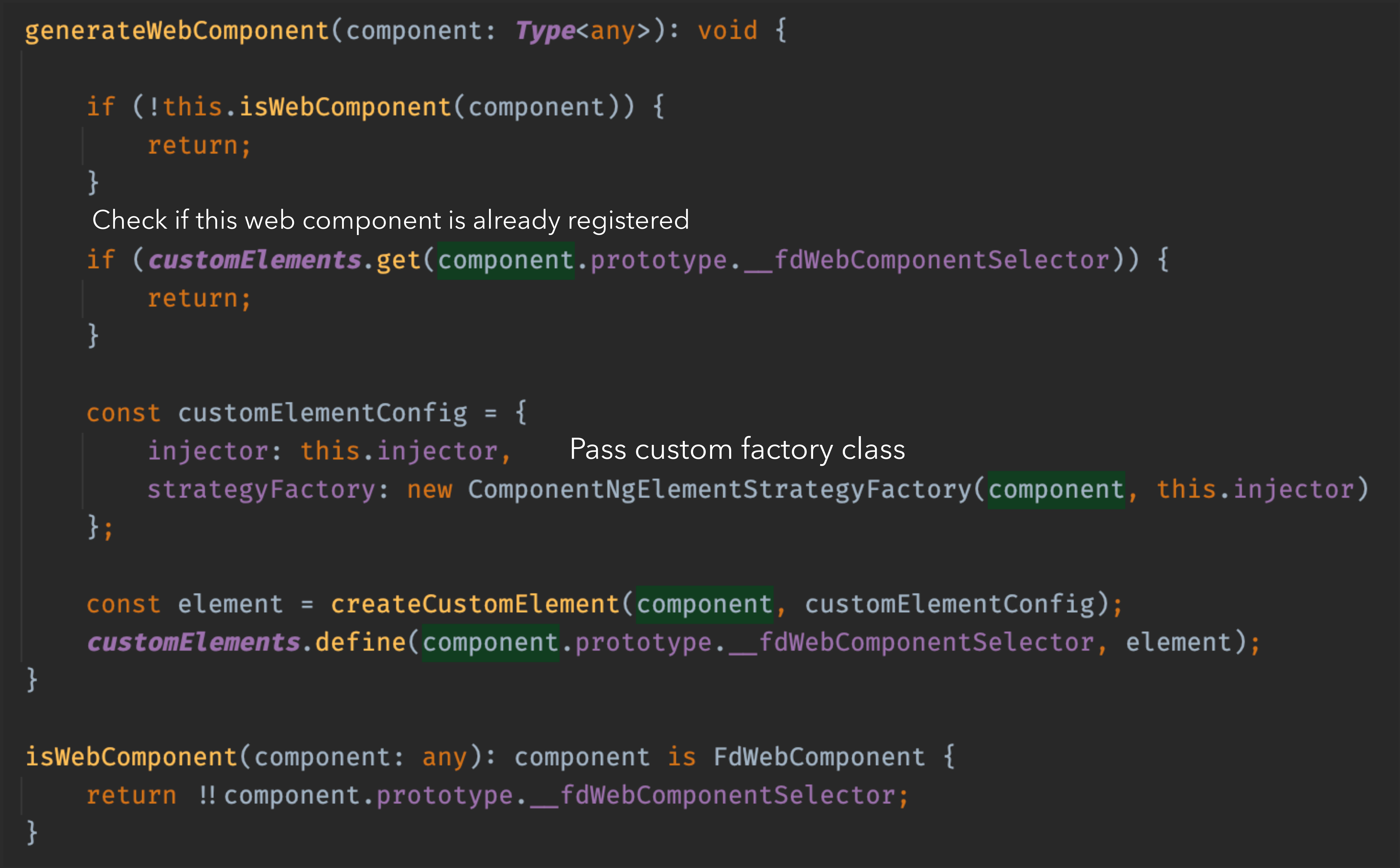


```
 */
@WebComponent( webComponentOptions: {
    selector: 'fdw-tab-list'
})
@Component({
```



```
/**
 * Array of declared components that support web component wrapping
 */
abstract declarations: Type<any>[];

/** @hidden */
protected constructor(protected injector: Injector) {}

/** @hidden */
ngDoBootstrap(): void {
    this.declarations.forEach((declaration : Type<any> ) => {
        this.generateWebComponent(declaration);
    });
}
```

Get web component declarations



```
generateWebComponent(component: Type<any>): void {

    if (!this.isWebComponent(component)) {
        return;
    }
```

Check if this web component is already registered

```
    if (customElements.get(component.prototype.__fdWebComponentSelector)) {
        return;
    }

    const customElementConfig = {
        injector: this.injector,
        strategyFactory: new ComponentNgElementStrategyFactory(component, this.injector)
    };
```

Pass custom factory class

```
    const element = createCustomElement(component, customElementConfig);
    customElements.define(component.prototype.__fdWebComponentSelector, element);
}

isWebComponent(component: any): component is FdWebComponent {
    return !! component.prototype.__fdWebComponentSelector;
}
```

Using Web Components in app

Fundamental conf

How to consume Angular elements

There are two options:

1. Build elements as a standalone application and simply load it’s JS.
2. Build elements as a library and include angular compiler to the app.

Build as application

Benefits:

- No need to load external compilers;
- Easy to include into existing application;
- With module federation can share @angular/* libs across multiple components

Downfalls:

- Resulting bundle is compiled JS code which does not include any metadata;
- Due to unknown usage scenarios resulting bundle may contain non-tree-shakeable code.

Build as a library

Benefits:

- Expose components metadata, interfaces;
- Tree-shaking;

Downfalls:

- Angular compiler is **required**;
- Existing application builders may need to be adjusted

Fundamental conf

Demo application

Conclusion

fundamental conf

Useful links

Fundamentals library

Demo application