CHANGE DETECTION

Ori Calvo, 2017

oric@trainologic.com

https://trainologic.com

Change Detection

- As user interacts with the application, state changes
- The big challenge is to exactly re-render the DOM
  - Update only relevant pieces
  - Don't touch the others
- Angular uses dirty checking mechanism
- Only data that might effect the DOM is checked

ApplicationRef.tick

- Executes change detection for the whole application
- You may monkey patch it and thus profile change detection

```
export class AppModule {
  constructor(appRef: ApplicationRef) {
    const originalTick = appRef.tick;

    appRef.tick = function() {
      const before = performance.now();
      originalTick.apply(this, arguments);
      const after = performance.now();

      console.log("tick", (after-before));
    }
  }
}
```

Performance

- Angular change detection is quite efficient
  - Few milliseconds for large applications
- The challenge is to reduce the number of change detection cycles per application activity
- Angular default approach is very aggressive since change detection is executed for every async operation

Trigger Changes

- What are the “things” that can trigger a change ?
  - User interaction
  - HTTP request
  - Timer
  - Any asynchronous operation
- How can Angular know of asynchronous operations running inside the application ?
  - ZoneJS

Zones

- Language feature in Dart
- zone.js is a JavaScript implementation of the Dart feature
- zone.js is a dependency of Angular
- You can use zone.js inside any application

Challenge

- How can you monitor the completion of below code

```
function run() {
   delay(1000).then(function() {
      console.log("STEP 1");

      delay(2000).then(function() {
         console.log("STEP 2");
      });
   });
}
```

- The function does not return any promise …

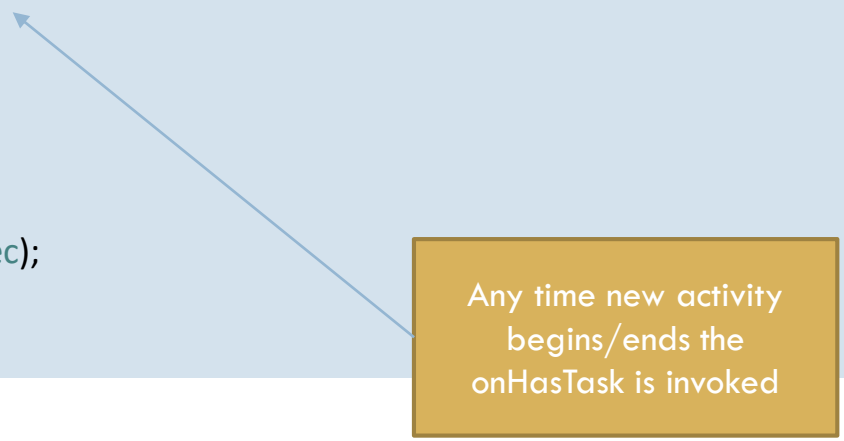
zone.js

- zone.js monkey patches any native asynchronous function
- Thus, it has knowledge of any asynchronous activity
- Only code that runs inside a zone is monitored and zone.js emits informative events

Run inside a Zone


```
function profile(func) {
   const spec: ZoneSpec = {
      name: "my",
      onHasTask: function(parentZoneDelegate, currentZone, targetZone, hasTaskState) {
         if(!hasTaskState.eventTask && !hasTaskState.macroTask && !hasTaskState.microTask) {
            console.log("DONE");
         }
      }
   };

   var zone = Zone.current.fork(spec);
   zone.run(func);
}
```



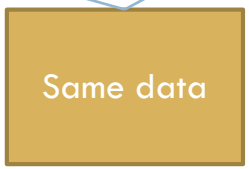
Zone state

- Like TLS, you can attach user defined data to a zone
- And then, fetch it from any method running inside the same zone

```
function profile(func) {
   const spec: ZoneSpec = {
      name: "my",
      properties: {
         id: 123,
      },
   };

   var zone = Zone.current.fork(spec);
   zone.run(func);
}
```

```
async function run() {
   delay(1000).then(function() {
      delay(2000).then(function() {
         console.log(Zone.current.get("id"));
      });
   });
}
```



Angular & zone.js

- As part of bootstrapping phase, Angular creates a new zone
- All components/services are executed under it
- Angular monitors all asynchronous activities and executes dirty checking

Angular change detection method

```
this._zone.onMicrotaskEmpty.subscribe({
  next: function () {
    this._zone.run(function () {
      this.tick();
    });
  }
});
```

Dirty Checking

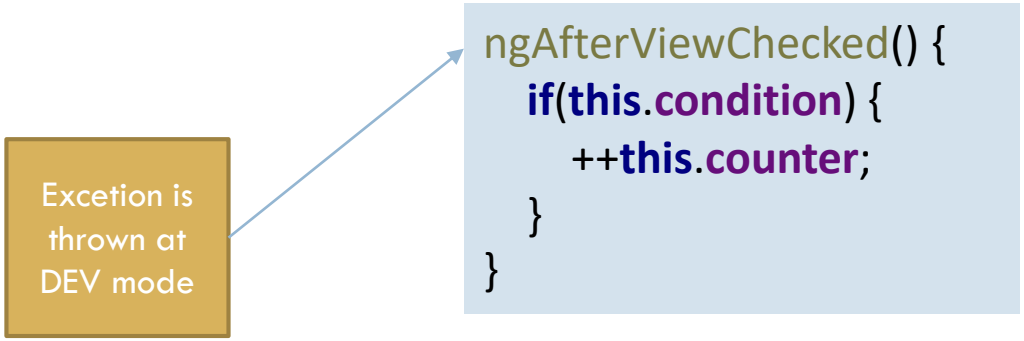
- zone.js helps Angular detecting user/IO activities
- Now, its Angular responsibility to detect changes
  - Phase 1: The developer is responsible for updating the application model
  - Phase 2: Angular via bindings is responsible for updating the view
- Change detection kicks at the end of a VM turn only if no pending Micro tasks

Updating Application Model

- Updating application model during phase 2 is disallowed
- Angular enforces that, by running 2nd binding check at each change detection
- If change was detected during 2nd run an exception is thrown
  - Only at development time ☺

Updating Application Model

- Once ngAfterViewChecked is invoked a component is not allowed to change its presentation state

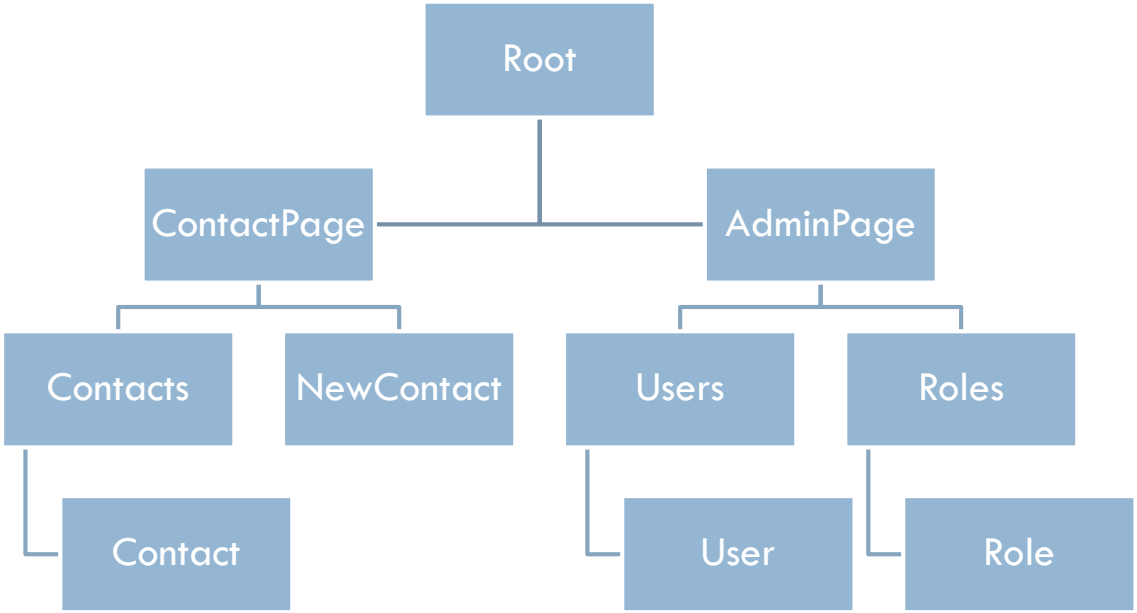


Conservative approach

- Angular has no understanding of your application model
- From Angular perspective an event triggered at one component may effect the state of another one
- Therefore, Angular by default checks every component

Component Tree

- Angular scans the whole component tree from top to bottom



Change Detector

- Each component has its own ChangeDetectorRef
- Using it, a component may customize the way Angular checks it

```
export class AppComponent {
  counter: number = 0;

  constructor(private cdr: ChangeDetectorRef) {
  }

  detach() {
    this.cdr.detach();
  }

  attach() {
    this.cdr.reattach();
  }

  inc() {
    ++this.counter;
  }
}
```

OnPush

- Tells Angular to dirty check the component only if one of its input changes
  - Or if an event is raised from its template

```
@Component({
  selector: "my-contact-list",
  templateUrl: "./contact-list.component.html",
  styleUrls: ["./contact-list.component.css"],
  moduleId: module.id,
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class ContactListComponent {
  @Input() contacts: Contact[];
}
```

Because of OnPush the DOM is not updated

```
export class AppComponent {
  contacts: Contact[];

  constructor() {
    this.contacts = [
      {"id": 1, "name": "Ori"},
      {"id": 2, "name": "Roni"},
    ];
  }

  change() {
    this.contacts.push({id:3, name: "Udi"});
  }
}
```

markForCheck

- Even when using the OnPush strategy you can force Angular to check a component for changes
- markForCheck marks current component and all it ancestors

```
export class ContactListComponent {
   @Input() contacts: Contact[];

   constructor(private cdr: ChangeDetectorRef) {
   }

   notifyChange() {
      this.cdr.markForCheck();
   }
}
```

detectChanges

- More challenging
- Enforce dirty checking for the current component and all its descendants
- You must be careful not to invoke too many times
- Unlike markForCheck it immediately executes

```
export class ContactListComponent {
   @Input() contacts: Contact[];

   constructor(private cdr: ChangeDetectorRef) {
   }

   notifyChange() {
      this.cdr.detectChanges();
   }
}
```

External State

- A component may be dependent on external state
  - Non @Input
- In that case the OnPush strategy does not work
- You can completely detach from Angular change detection and take full control

External State


```
export class ContactsPageComponent {
  contacts: Contact[];

  constructor(private cdr: ChangeDetectorRef, private contactService: ContactService) {
    this.cdr.detach();
  }

  ngDoCheck() {
    if(this.contacts != this.contactService.contacts) {
      this.contacts = this.contactService.contacts;

      this.cdr.detectChanges();
    }
  }
}
```

ngDoCheck is executed even when component is detached

Disable NgZone

- Starting Angular 5 you can disable NgZone
- Thus, no automatic change detection
- You must invoke ApplicationRef.tick manually

```
platformBrowserDynamic().bootstrapModule(AppModule, {
  ngZone: 'noop'
});
```

Summary

- Angular dirty checking can be optimized
- The primary tools for optimization are
  - OnPush
  - ChangeDetectorRef.detach
- Take into consideration that even without optimization Angular change detection is quite fast