Angular — 性能&变更检测

声明

本视频内容是对

Minko Gechev (ng-conf 2018)

和

Christian Liebel (NG-DE 2019)

“拙劣的模仿”，拾人牙慧，仅供交流学习，还望轻拍🙏

- 变更检测基础
- 变更检测和性能的关系
- 如何优化 Angular 应用性能（基础）

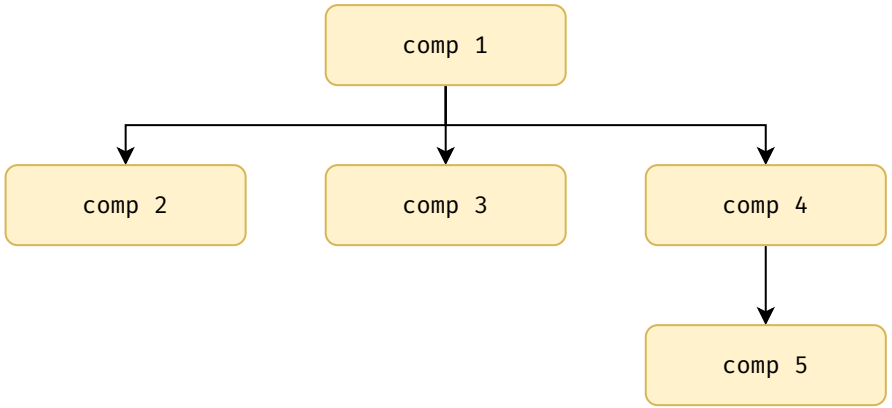
变更检测基础

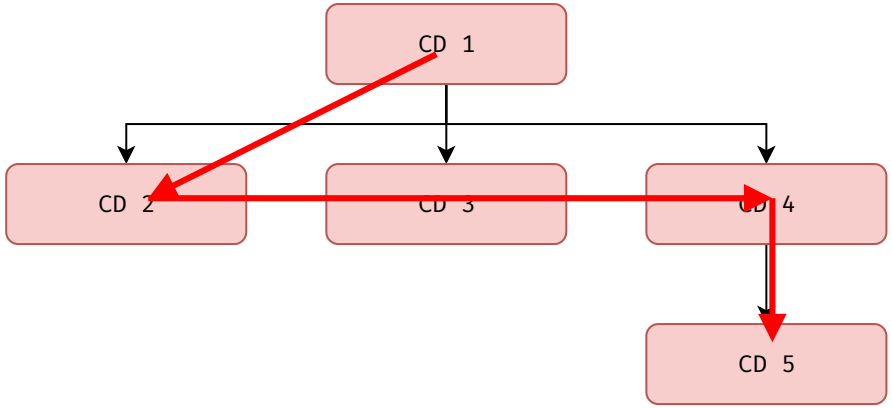
```
@Component({
  ...
  template: `
    ...
    <h2>{{ title }}</h2>
    ...
    <button (click)="updateTitle()">
      Update Title
    </button>
    ...
  `,
})
export class DemoComponent {
  title = 'Demo';

  updateTitle() {
    this.title = 'Demo - New';
  }
}
```

变更检测基础

```
abstract class ViewRef extends ChangeDetectorRef {
    abstract destroyed: boolean
    abstract destroy(): void
    abstract onDestroy(callback: Function): void

    // inherited from core/ChangeDetectorRef
    abstract markForCheck(): void
    abstract detach(): void
    abstract detectChanges(): void
    abstract checkNoChanges(): void
    abstract reattach(): void
}
```

变更检测基础





变更检测基础

Angular 怎么知道何时要触发变更检测呢？

变更检测基础

Angular 认为是“异步操作”导致应用状态变更，即：

- DOM 事件，如 click、 input
- 网络请求，即 XHR 和 fetch
- 计时器，如 setTimeout、 setInterval

Zone

Angular 内部使用了 Zone.js 给几乎所有的异步 API 都做了猴补丁（ Monkey Patching ）用来监听这些异步事件

在满足以下条件时会出发变更检测：

- When a sync or async function is executed
- When there is no microTask scheduled

https://angular.io/guide/zone

变更检测基础

```
@Component({
  ...
  template: `
    ...
    <h2>{{ title }}</h2>
    ...
    <button (click)="updateTitle()">
      Update Title
    </button>
    ...
  `,
})
export class DemoComponent {
  title = 'Demo';

  updateTitle() {
    this.title = 'Demo - New';
  }
}
```

Zone

```
@Injectable({providedIn: 'root'})
export class NgZoneChangeDetectionScheduler {
  ...
  initialize(): void {
    if (this._onMicrotaskEmptySubscription) { return; }
    this._onMicrotaskEmptySubscription = this.zone.onMicrotaskEmpty.subscribe({
      next: () => {
        this.zone.run(() => { this.applicationRef.tick(); });
      }
    });
  }
  ...
}
```

Zone

```
@Injectable({providedIn: 'root'})
export class ApplicationRef {
  ...
  tick(): void {
      ...
      for (let view of this._views) {
        view.detectChanges();
      }
  }
  ...
}
```

变更检测和性能的关系

提升性能 = 做更少的事情

- 降低变更检测调用的频率
- 降低每次变更检测所耗费的时间

变更检测和性能的关系

降低变更检测调用的频率

- 避免 Zone 污染（ Zone Pollution ）
- 谨慎地监听高频事件，如 scroll 、 mouseover 等

变更检测和性能的关系

降低每次变更检测所耗费的时间

- 使用 OnPush 变更检测策略
- 减少绑定的数量（在列表中尤为重要）
- 不要在模板上做昂贵耗时的绑定

变更检测和性能的关系

防止 Zone 污染 DEMO

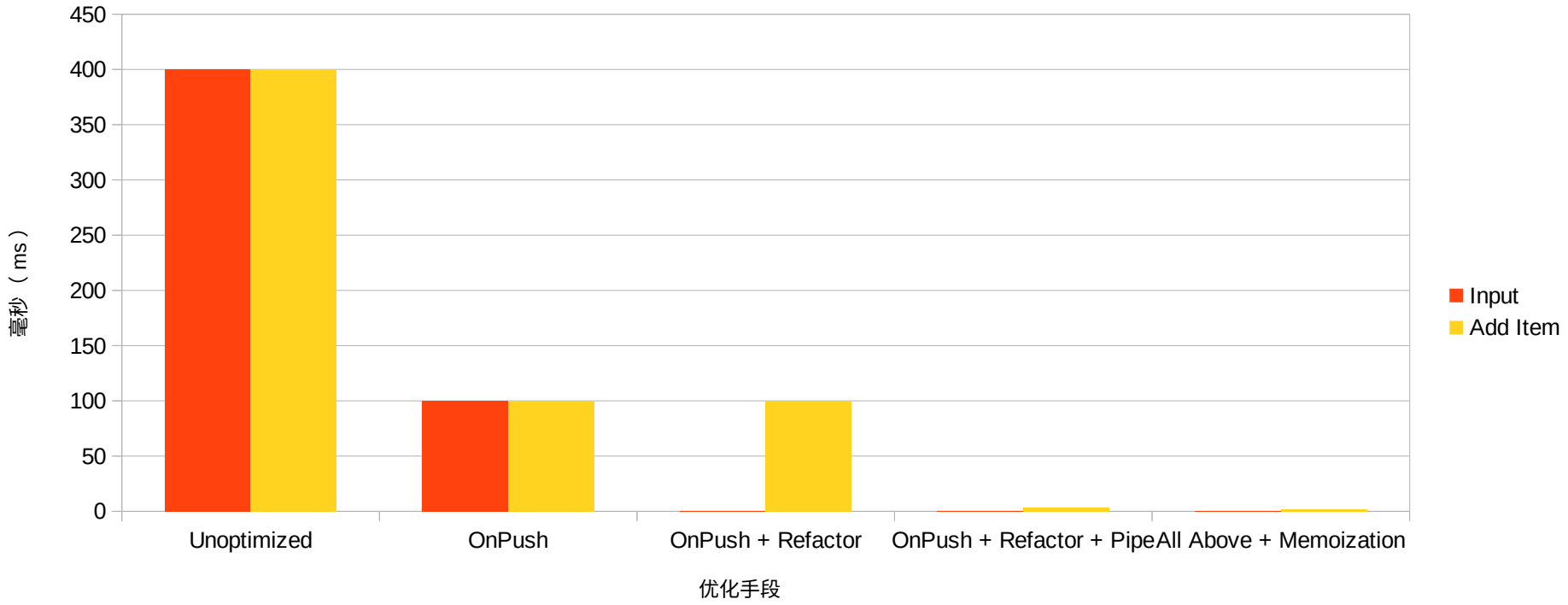
变更检测和性能的关系

OnPush 变更检测策略 DEMO

如何优化 Angular 应用性能

DEMO

如何优化 Angular 应用性能



如何优化 Angular 应用性能

- 先测量，再优化
- 重构：把触发事件（变更检测）和渲染耗时的地方分开
- 善用 OnPush “ 保护” 组件
- 善用 Pure Pipe
- 善用缓存、 memoization 等技巧

总结

- 变更检测是 UI 框架同步数据和视图必要的机制
- 高频 + 耗时的变更检测 = 性能差
- OnPush + Async Pipe 是手牵手的好朋友
- Less is more
  - Do less = More performance

参考资料

- Optimizing an Angular application - Minko Gechev (https://www.youtube.com/watch?v=ybNj-id0kjY)
  - B 站搬运： https://www.bilibili.com/video/BV1HK4y1k7ez/
- Angular Performance: Your App at the Speed of Light - Christian Liebel | NG-DE 2019 (https://www.youtube.com/watch?v=moUCZoJfhwY)
- 4 Runtime Performance Optimizations (https://www.youtube.com/watch?v=f8sA-i6gkGQ)
- Performance optimizations in Angular | Mert Değirmenci | #AngularConnect （ https://www.youtube.com/watch?v=Tlmx1PbP8Qw)
- https://danielwiehl.github.io/edu-angular-change-detection/
- https://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html
- https://blog.thoughtram.io/angular/2016/02/01/zones-in-angular-2.html