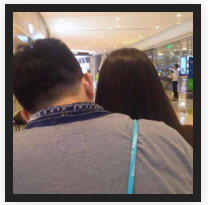
ANGULAR推进器

ANGULAR从原理到应用 - 变更检测

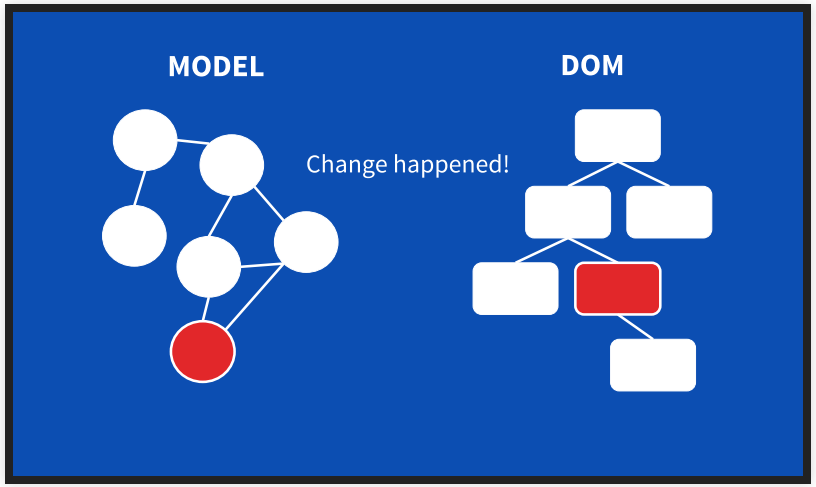
我是谁?

- Angular开发者 from Angular4 to Angular latest
- NodeJS开发者 from Express to Nestjs
- 我也是  和 
- 如果你觉得舒服就叫我Tommy,不方便就@我

什么是变更检测

- 获取程序的内部状态
- 使其以某种方式对用户界面可见
- 状态可以是: objects, Arrays, Primitives...
- Value Types: string number boolean null undefined
- Reference Types: array object function

当检测发生在运行时

- 模型中的变化 -> 更新DOM的位置
- 操作DOM开销昂贵
- 

多种解决方案

- HTTP 请求 + server 重新渲染
- 区分前后DOM的差异, 渲染不同的部分
- React Virtual DOM

回到ANGULAR

- 变化什么时候发生?

```
@Component({
template: `
  <h1>{{firstname}} {{lastname}}</h1>
  <button (click)="changeName()">Change name</bu
`
})
export class AppComponent {

firstname:string = 'Material';
lastname:string = 'Angular';

changeName() {
  this.firstname = 'Awesome';
  this.lastname = 'Angular';
}
```

另一个例子

```
@Component()
export class ContactsComponent implements OnInit{

  people:Person[] = [];

  constructor(private http: HttpClient) {}

  ngOnInit() {
    this.http.get('/people')
      .map(res => res.json())
      .subscribe(people => this.people = people);
  }
}
```

换句话说什么会触发变更

- Events - click, submit...事件
- XHR - 从远端服务器获取数据
- Timers - setTimeout(), setInterval() 浏览器web api

触发变更?

ASYNCHRONOUS

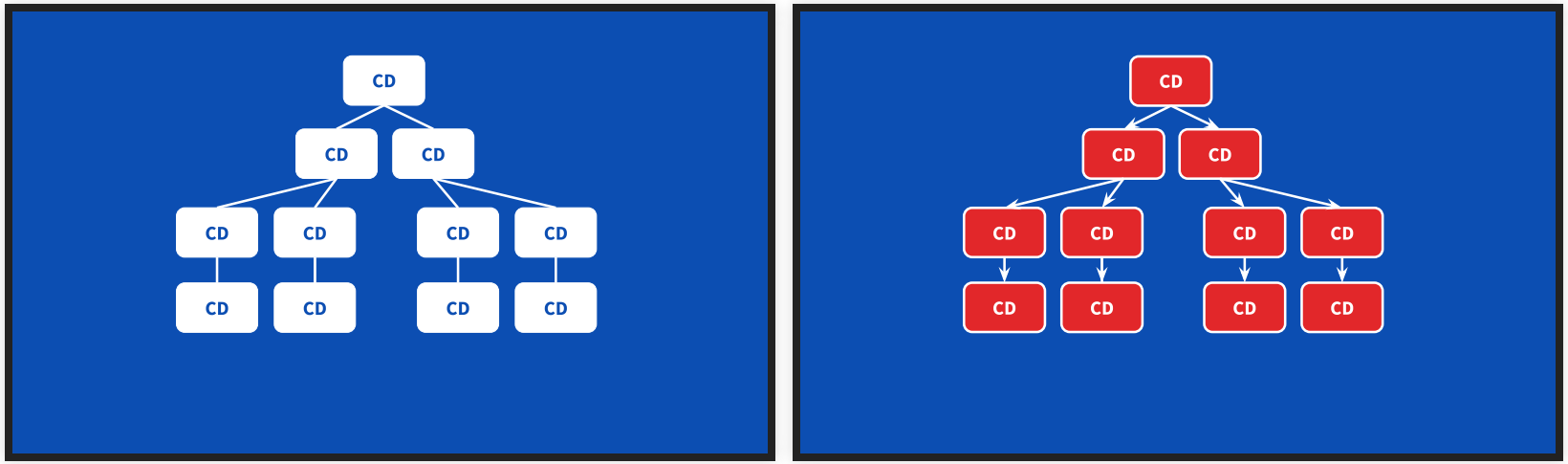
谁通知了ANGULAR?

- Zone.js
- NgZone implements from Zone.js

```
// 源码简化
class ApplicationRef {

changeDetectorRefs:ChangeDetectorRef[] = [];
// applicationRef在构造器中监听onTurnDone事件
constructor(private zone: NgZone) {
  this.zone.onTurnDone
    .subscribe(() => this.zone.run(() => this.tic
}
// tick函数遍历所有的探测器的接口/对象 对其执行检测
tick() {
  this.changeDetectorRefs
    .forEach((ref) => ref.detectChanges());
}
}
```

变更检测是如何进行的呢?

- Key: 每一个组件都有属于自己的变更检测器 (change detector)
- 

- 变更检测树 change detector tree: 有向图 数据流从上而下

数据流单向从上到下?

- 变更检测 from top to bottom
- BFS?DFS?
- 优点多多

效率如何?

- 感觉上很慢实际上很快 得益于 Angular 生成 VM 友好的代码
- VM 不喜欢动态不确定的代码 VM的优化得益于object的单态 而不是多态
- Angular在运行时 创造变更检测器 - 单态 - 确定的model
- Don't worry, Angular 帮我们处理好了这些复杂的部分

更聪明的变更检测

- Angular默认的变更检测是自动的
- 两个好帮手: Immutable(不可变) & Observables

理解可变和不可变(MUTABILITY)

- reference 没变 但是 property 改变 -> Angular负责地进行检测

```
@Component({
template: '<child [data]="data"></child>'
})
export class ParentComponent {

constructor() {
  this.data = {
    name: 'Button',
    email: 'github.com'
  }
}

changeData() {
  this.data.name = 'Tommy';
}
```

不可变对象

- reference change

```
var data = someAPIForImmutables.create({
              name: 'Button'
            });

var data2 = data.set('name', 'Tommy');

data === data2 // false reference are different
```

优化?

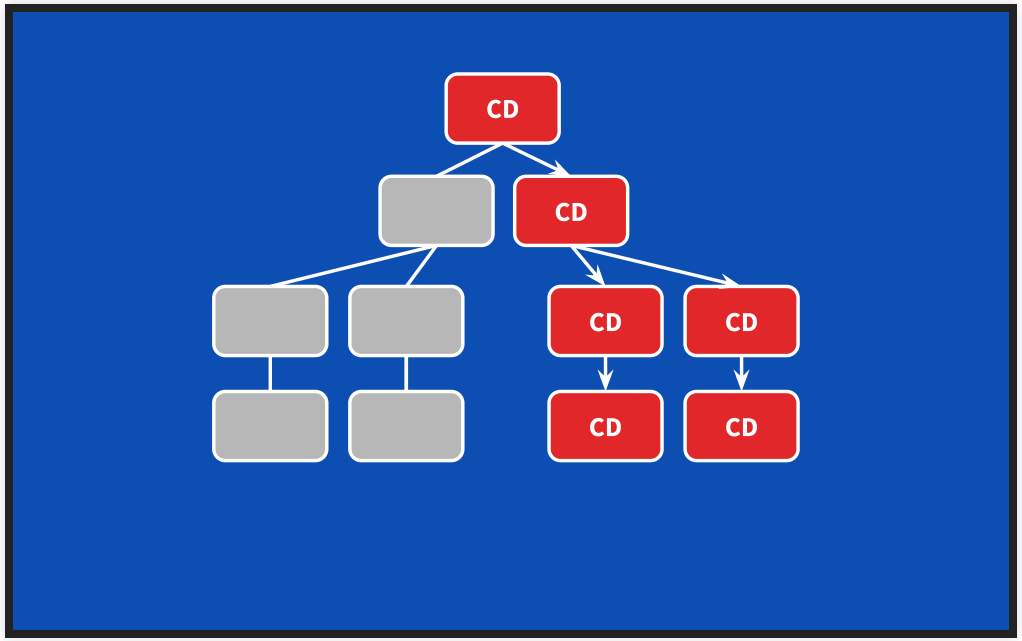
- 变更检测可以跳过某些component子树
- @Input()属性immutable
- 需要告诉angular

比如

- OnPush Strategy

```
@Component({
template: `
  <h2>{{data.name}}</h2>
  <span>{{data.email}}</span>
`
// onPush 策略会在@Input()的内容属性不变时生效
changeDetection: ChangeDetectionStrategy.On
})
class VCardCmp {
@Input() data;
}
```

结果?

- immutable object + OnPush
- 

OBSERVABLES

- 和immutable不同
- Observables + OnPush?

简单的购物车

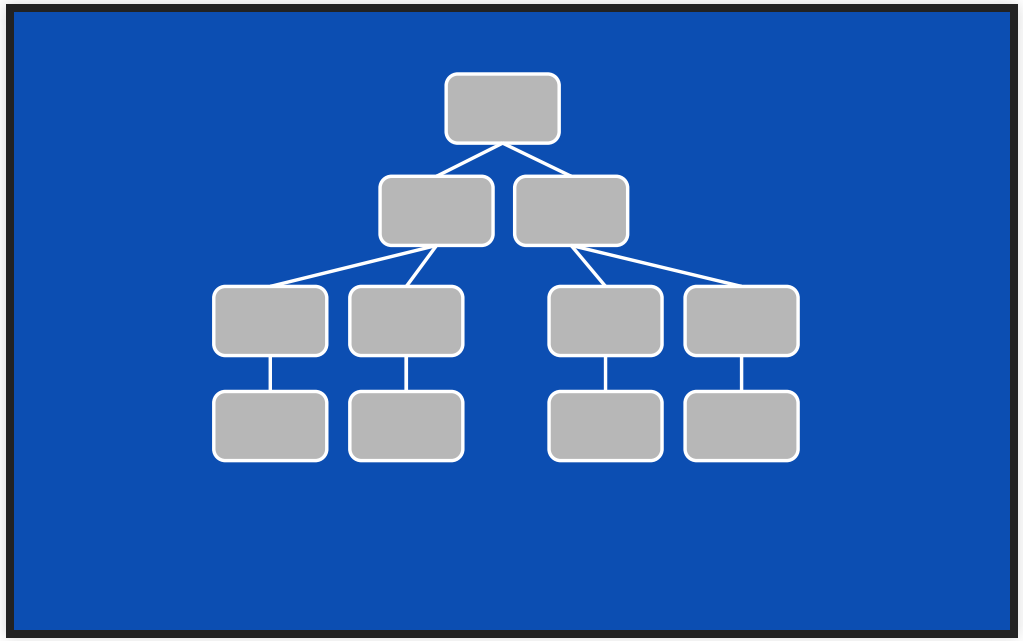
- @Input() addItemStream reference

```
@Component({
template: '{{count}}',
changeDetection: ChangeDetectionStrategy.On
})
export class CartComponent implements OnIni

@Input() addItemStream: Observable<any>;
count = 0;

ngOnInit() {
  this.addItemStream.subscribe(() => {
    this.count++; // 变更出现在OnInit hook
  })
}
}
```

怎么办?很慌

- 全部component设置为OnPush
- 

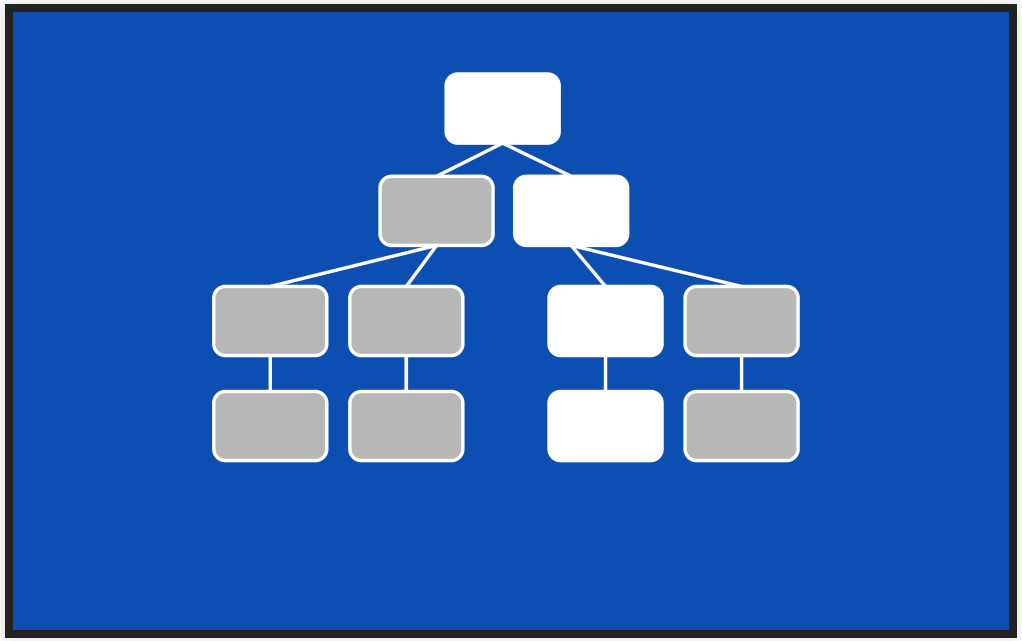
ANGULAR不知道 但我们知道

- markForCheck from ChangeDetectorRef

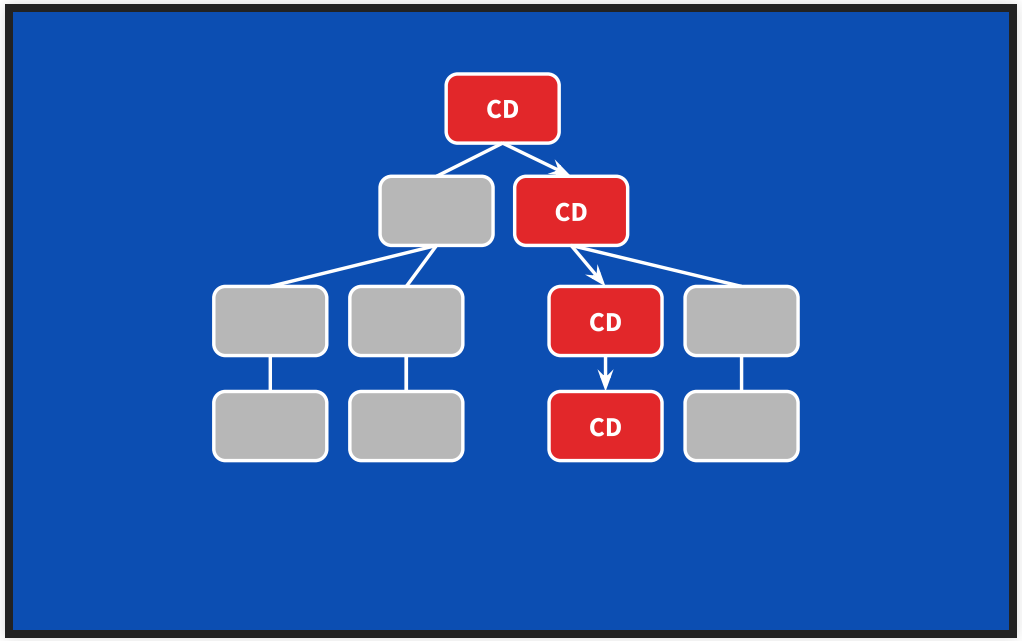
```
@Component({
  template: '{{count}}',
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class CartComponent implements OnInit {

  @Input() addItemStream: Observable<any>;
  count = 0;
  // 注入ChangeDetectorRef
  constructor(private cdr: ChangeDetectorRef)
  ngOnInit() {
    this.addItemStream.subscribe(() => {
      this.count++; // 变更出现在OnInit hook
      this.cdr.markForCheck(); // 人为通知angular检测这个componer
    })
```

不慌了!

- Observables 事件已经被触发了(变更检测前)
- 

MESSI不用慌了 我们也不用(变更检测后)

- Observables 没凉
- 

另一个应用场景

- setTimeout&setInterval

```
@Component({
changeDetection: ChangeDetectionStrategy.OnPush
})
export class AppComponent implements OnInit{
data = [{name: 'Button'}];
constructor(public cdr: ChangeDetectorRef) {}

ngOnInit() {
  setTimeout(() => {
    this.data.push({name: 'Tommy'});
    this.cdr.markForCheck(); // setTimeout + OnPush也需要配合 markForC
  }, 2000);
}
}
```

变更检测的种类

- CheckOnce(Depreciated)
- Checked(Depreciated)
- CheckAlways(Depreciated)
- Detached(Depreciated)
- OnPush(In using)
- Default(In using)
- 自己去探索

一个可能会踩到的坑

- Pure pipe

```
{{data | CustomizedPipe}}
// data is a reference type, customized pipe may not be trigg
```

- data的属性发生了变化 但是reference没变

解决方案

- Impure pipe

```
@Pipe({
name: 'CustomizedPipe',
pure: false
})
```

- 类似于markForCheck
- 会频繁对传入的data进行检测
- options: 使用immutable type data

谢谢