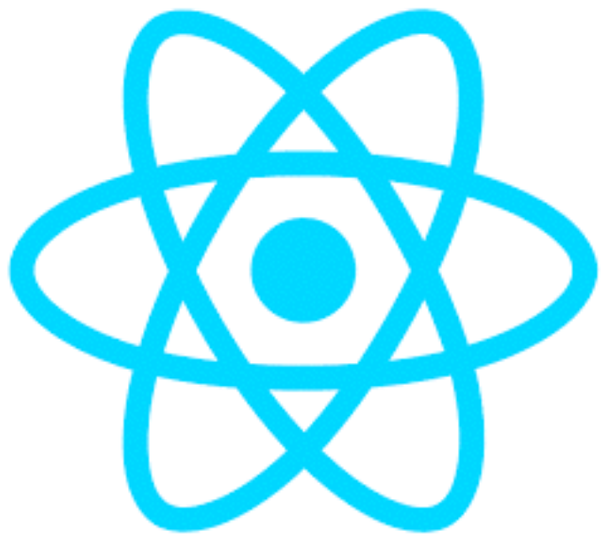
AngularJS

2010



React

2013

Angular

2015+


ANGULARJS
by Google

起源

- 2009年Google Feedback Project
- 将6个月开发的17000行前端代码，使用3周压缩到1500行

概念

- Template (模板)
- Directives (指令)
- Model (模型)
- Scope (作用域)
- Expressions (表达式)
- View (视图)
- Data Binding (数据绑定)
- Controller (控制器)
- Dependency Injection (依赖注入)
- Module (模块)
- Service (服务)

前端Model View Controller

- View:
  Document Object Model

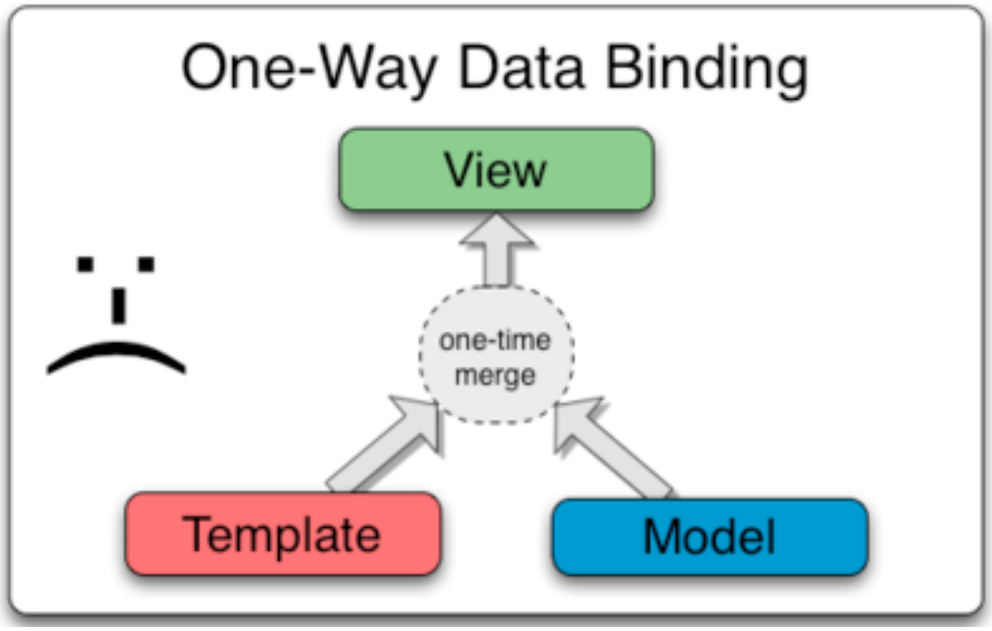
- Controller:
  JavaScript 类

- Model:
  JavaScript 对象

Data Binding(数据绑定)

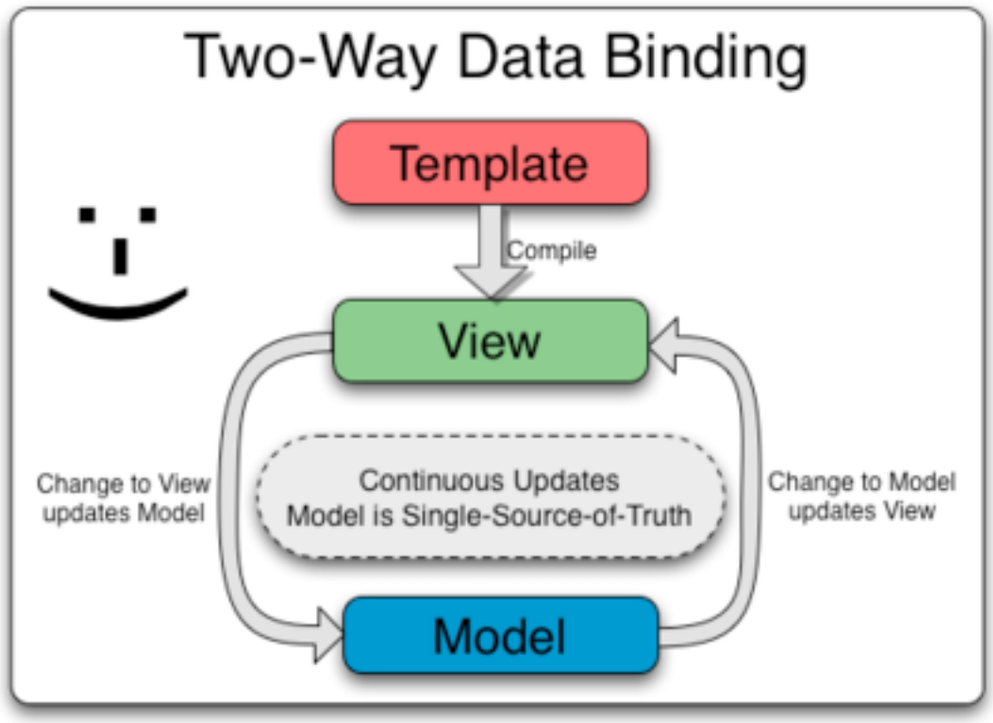
- AngularJS实现了双向数据绑定机制。
- 所谓的双向绑定，无非是从界面(view)的操作能实时反映到数据(model)，数据(model)的变更能实时展现到界面(view)。

经典模板系统中的数据绑定



- 大多数模板系统中的数据绑定都是单向的
- 把模板与model合并在一起变成view，如果在合并之后，model发生了变化，不会自动反映到view上。
- 用户在view上的交互也不会反映到model中，开发者必须写大量代码不断地在view与model之间同步数据。

AnguarJS 模板中的数据绑定



- 模板是在浏览器中编译的，在编译阶段产生了一个实时更新(live)的视图
- 不论在model或是view上发生了变化，都会立刻反映到对方。

$scope

- 是用来连接model和view
- model放在这个context中才能被控制器、指令和表达式等访问到
- ng-app创建$rootScope 顶层节点，所有的 $scope 都是继承自它

angular context

- $watch list
- $digest循环
- dirty checking
- TTL(Time To Live 生存时间值)

进入angular context

- $apply
- $digest

Directives (指令)

- 用于通过自定义属性和元素扩展HTML的行为

AngularJS优势

- 模板功能强大丰富，自带了丰富的Angular指令
- 自定义指令
- 完善：是一个比较完善的前端MVVM框架
- Google维护：有了一个强大的后台，社区非常活跃

AngularJS缺点

- 大而全：学习起来有难度，学习曲线很曲折
- 开发人员思想的转变
- 性能问题
- 数据流

https://github.com/Pasvaz/bindonce

http://ionicframework.com/


React

起源

- React 起源于 Facebook 的内部项目，因为该公司对市场上所有 JavaScript MVC 框架，都不满意，就决定自己撸一套。

- 做出来以后，发现这套东西很好用，就在2013年5月开源了。

概念

- 仅仅是UI
- 虚拟DOM
- 单向响应的数据流

react 优势

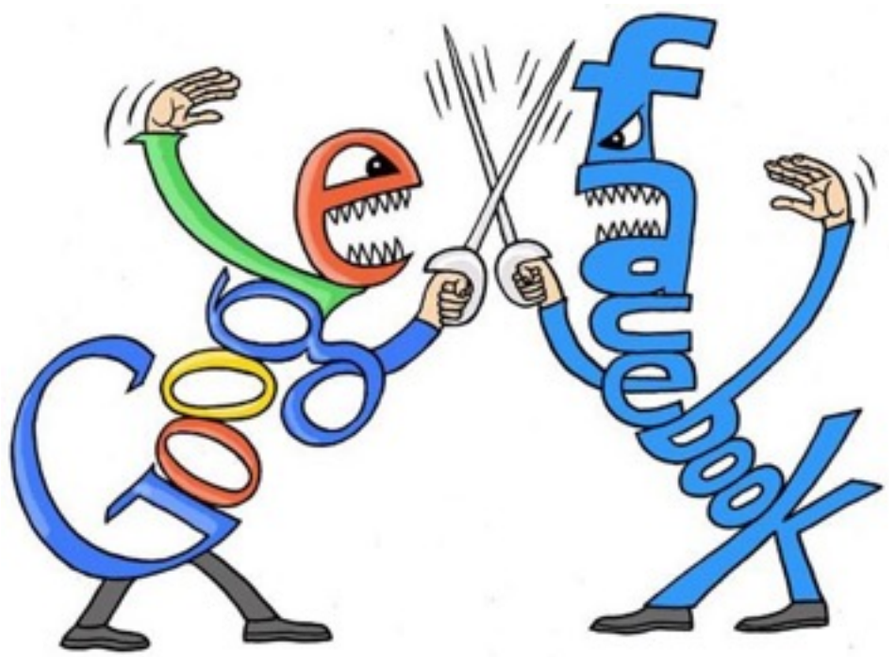
- 轻
- 虚拟DOM
- 学习简单

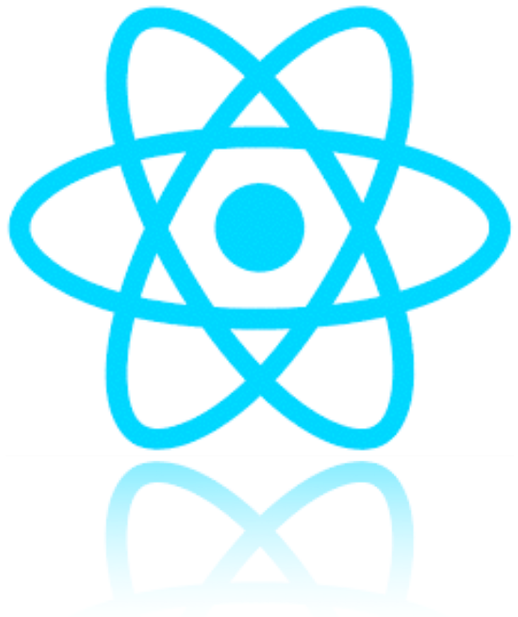
react 缺点

- 只是一个V而已，当然这也可能也是优势
- 官方文档不完善
- 更多的在于概念

https://facebook.github.io/react-native/

https://github.com/Flipboard/react-canvas


Google
Facebook



React 比 AngularJS 快?

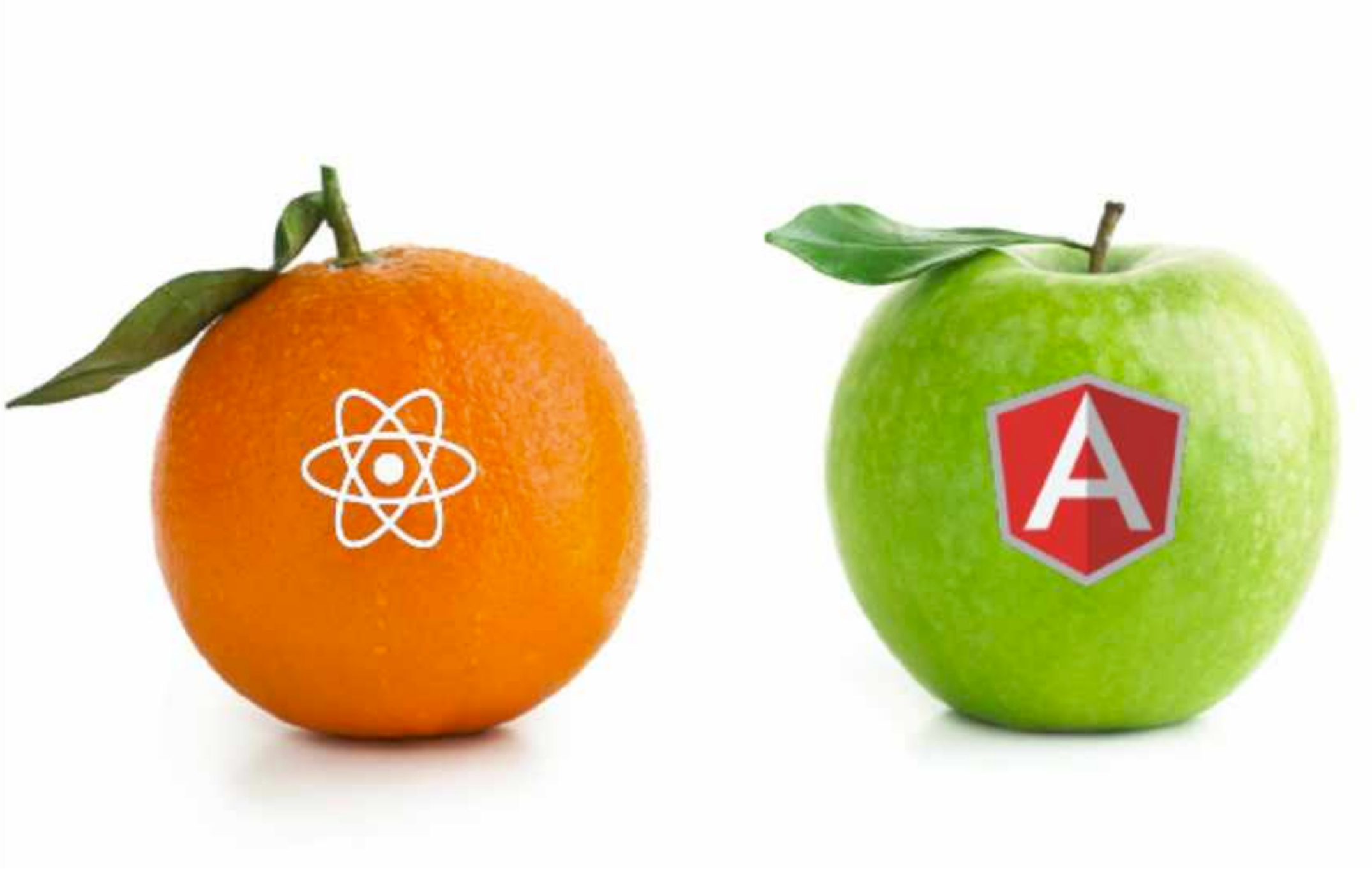
结论

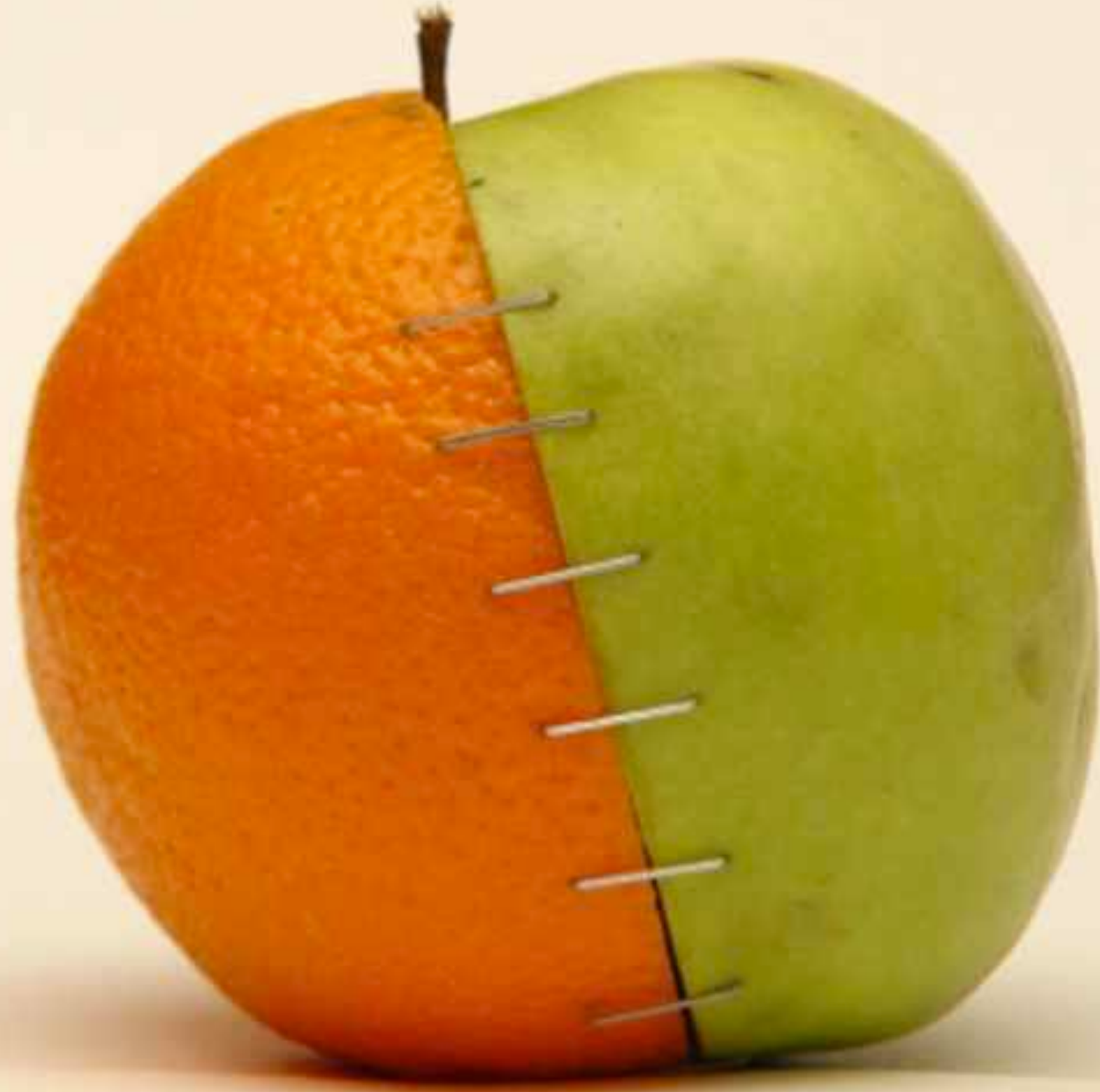
React 比 AngularJS 快

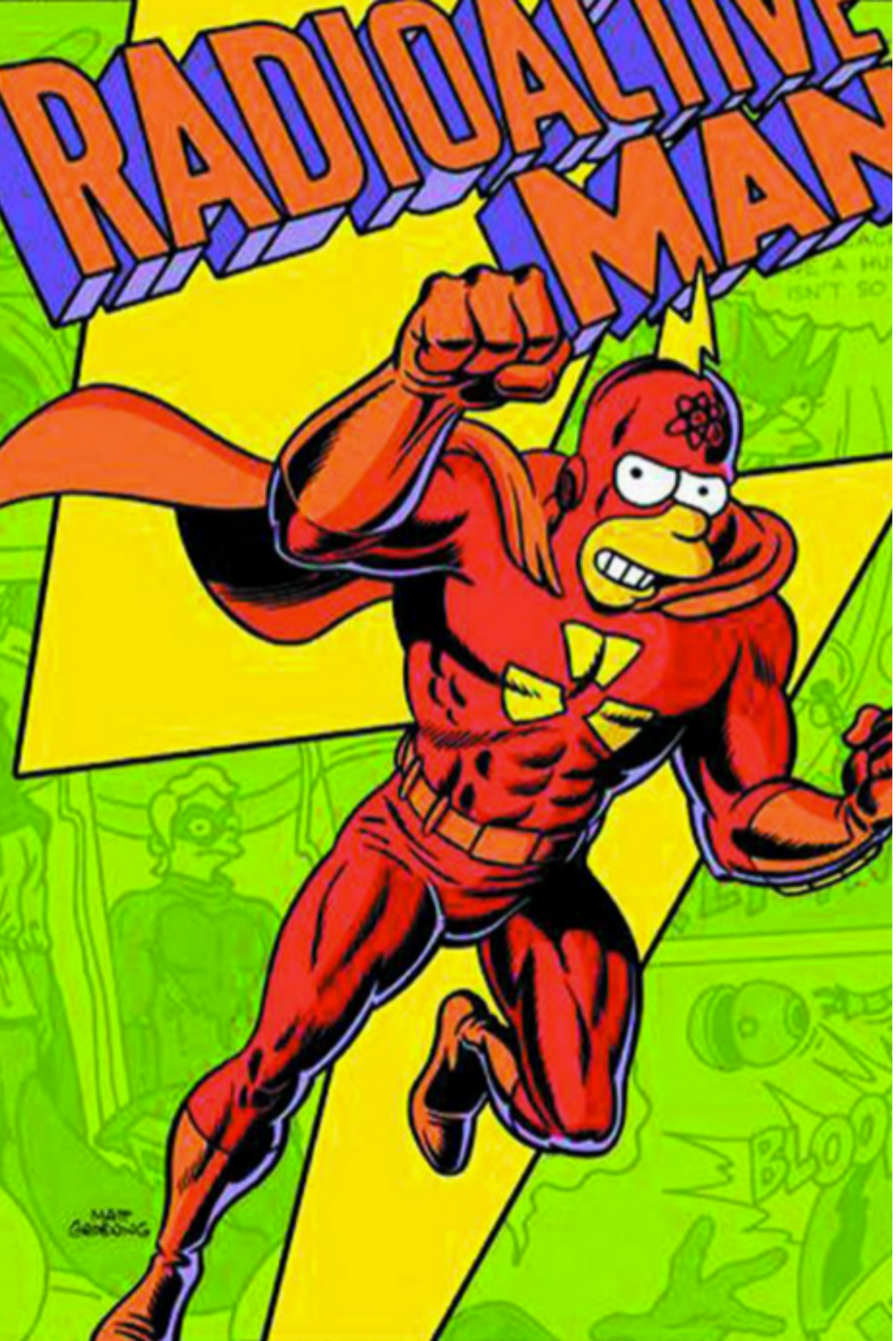
结论

React 比 AngularJS 快

愚蠢的结论



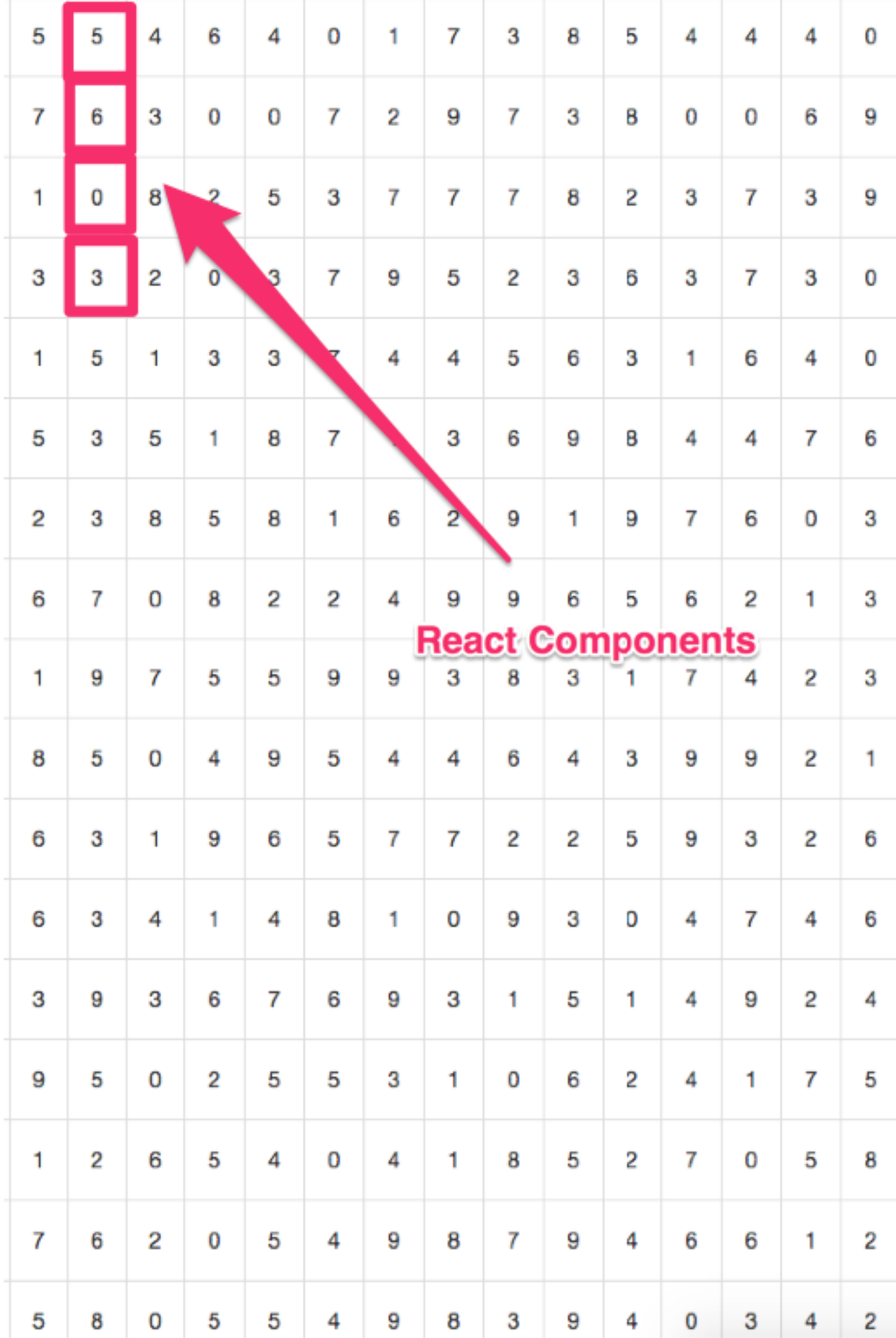




让
React
来
拯救
AngularJS

定义一个AngularJS指令，

使用React来渲染每一个td



Result:

Why

- React无法控制真实的DOM

- appendChild(),
  React
  appendChild()
  React...

http://davidchang.github.io/ngReact


ANGULAR
by Google

2015年 3月5号

在ng-conf 2015上

官方发布了Angular的alpha版

https://angular.io/

这是超越1.x版本的一次彻底更新

破坏性升级

怎么将AngularJS的项目升级到Angular呢?

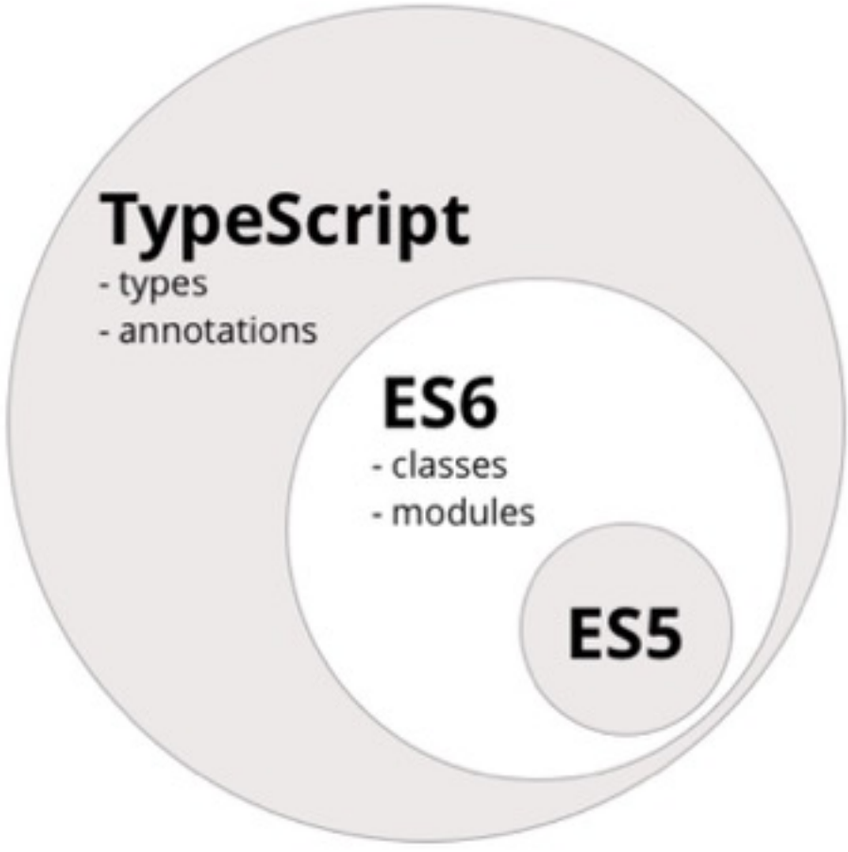
怎么将AngularJS的项目升级到Angular呢?

bower install react

Angular与AngularJS是雷峰与雷峰塔的区别。

TypeScript

- Angular 完全基于TypeScript编写
- TypeScript是由微软开发
- TypeScript并不是一个完全新的语言，它是ES6的一个超集
- 微软和谷歌之间的官方合作
- Angular 应用不是一定需要用TypeScript开发



ES5, ES6, and TypeScript

浏览器支持

- 很少有浏览器支持ES6,更不用说TypeScript
- TypeScript => ES6
- https://github.com/google/traceur-compiler
- https://babeljs.io/

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template: `
  <div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template: `
  <div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template: `
  <div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template: `
  <div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

Component

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template: `
  <div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

Component

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

Component

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

Component

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

Component

View

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

Component

View

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

Component

View

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

Component

View

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

Component

View

*语法

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

Component

View

*语法

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

class

Component

View

*语法

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

class

Component

View

*语法

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

class

Component

View

*语法

事件

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

class

Component

View

*语法

事件

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>`
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

class

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Component

View

*语法

事件

类型

Annotations

多行字符串

属性

class

Component

View

*语法

事件

类型

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>
  `
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Annotations

多行字符串

属性

class

```
@Component({
  selector: 'products-list',
  properties: {
    products: 'products'
  },
  events: ['click']
})
@View({
  directives: [For, ProductRow],
  template:
  `<div class="products-list">
    <product-row *for="#product of products"
      [product]="product"
      (click)='clicked(product)'>
    </product-row>
  </div>`
})
class ProductsList {
  products: Array<Product>;
  click: EventEmitter;

  constructor() {
    this.click = new EventEmitter();
  }

  clicked(product) {
    this.click.next(product);
  }
}
```

Component

View

*语法

事件

类型

事件系统

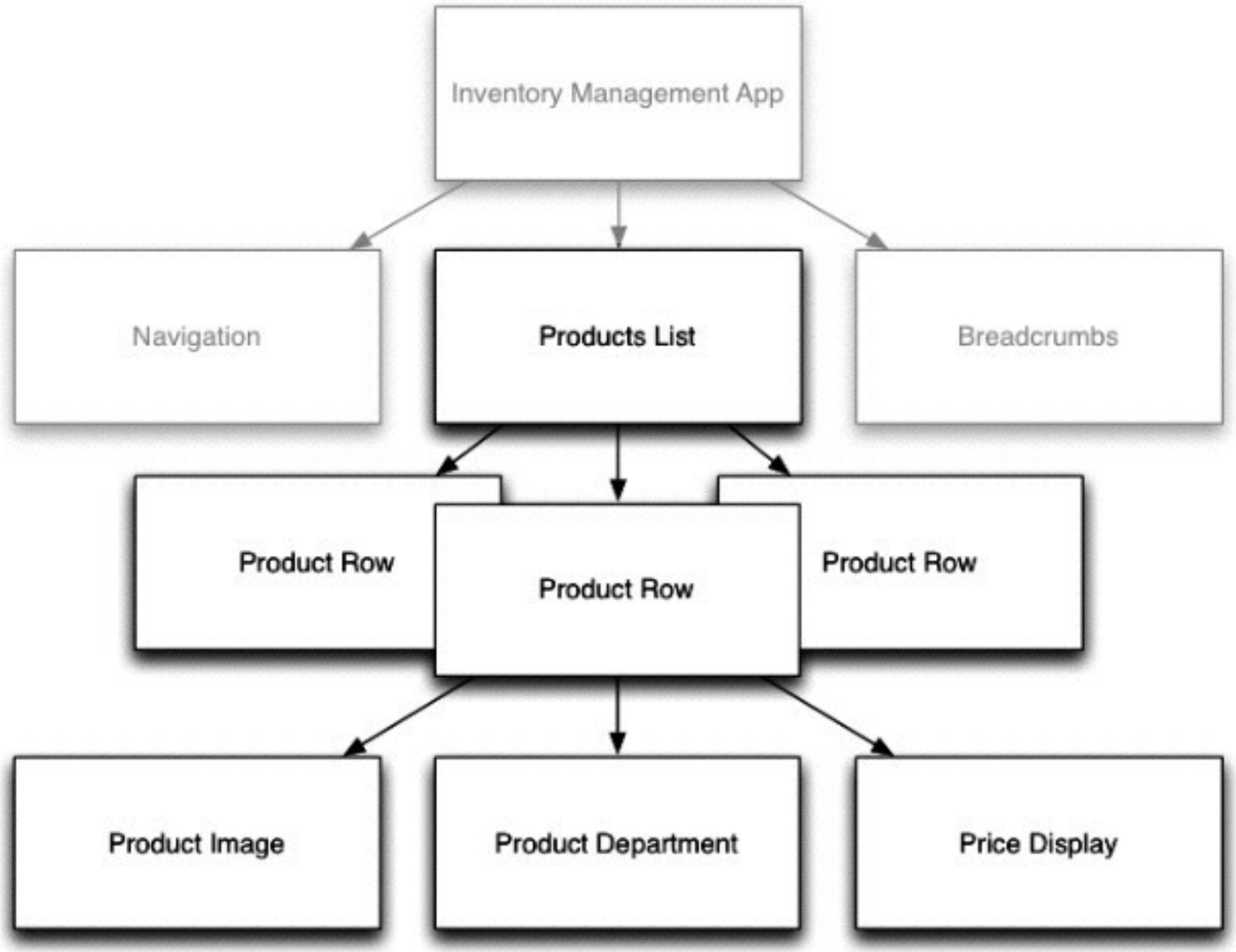
核心概念

- 组件
- 数据流
- 变化检查
- 依赖注入

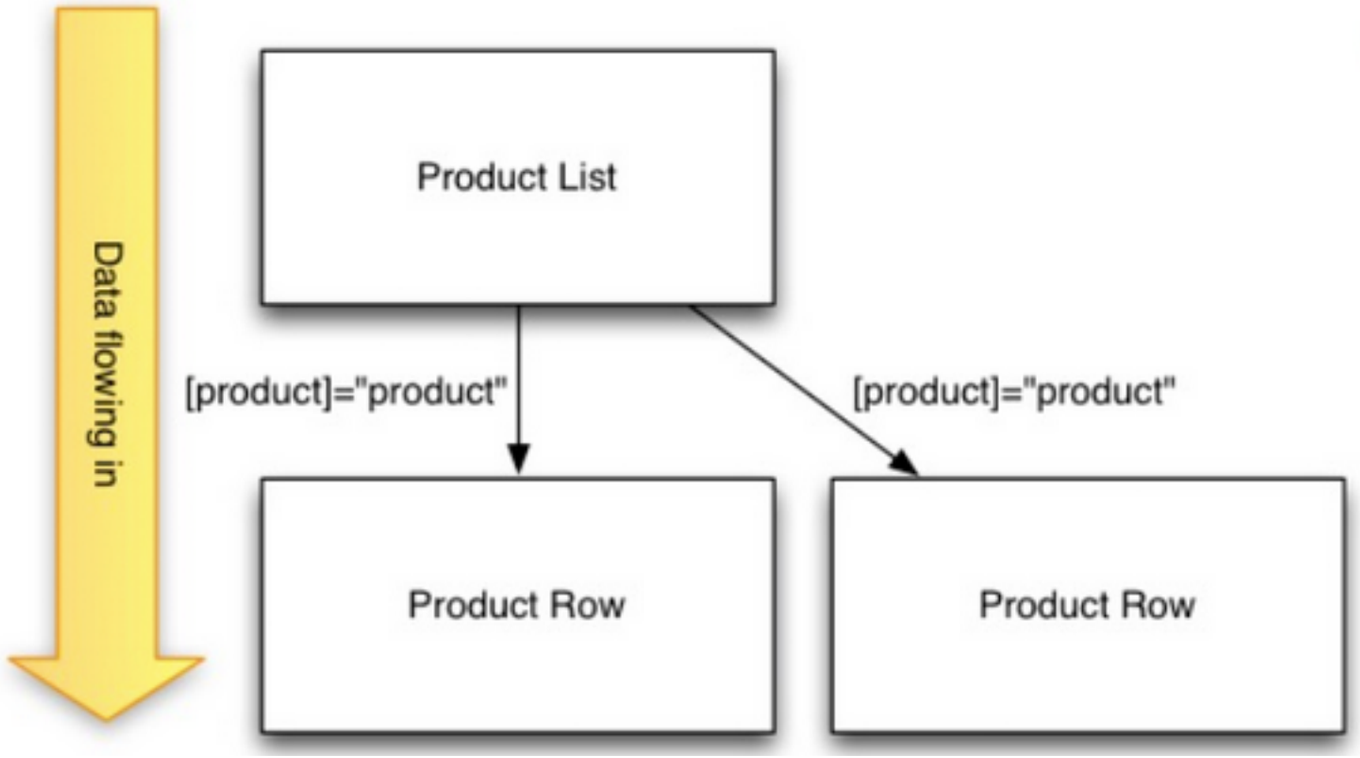
组件

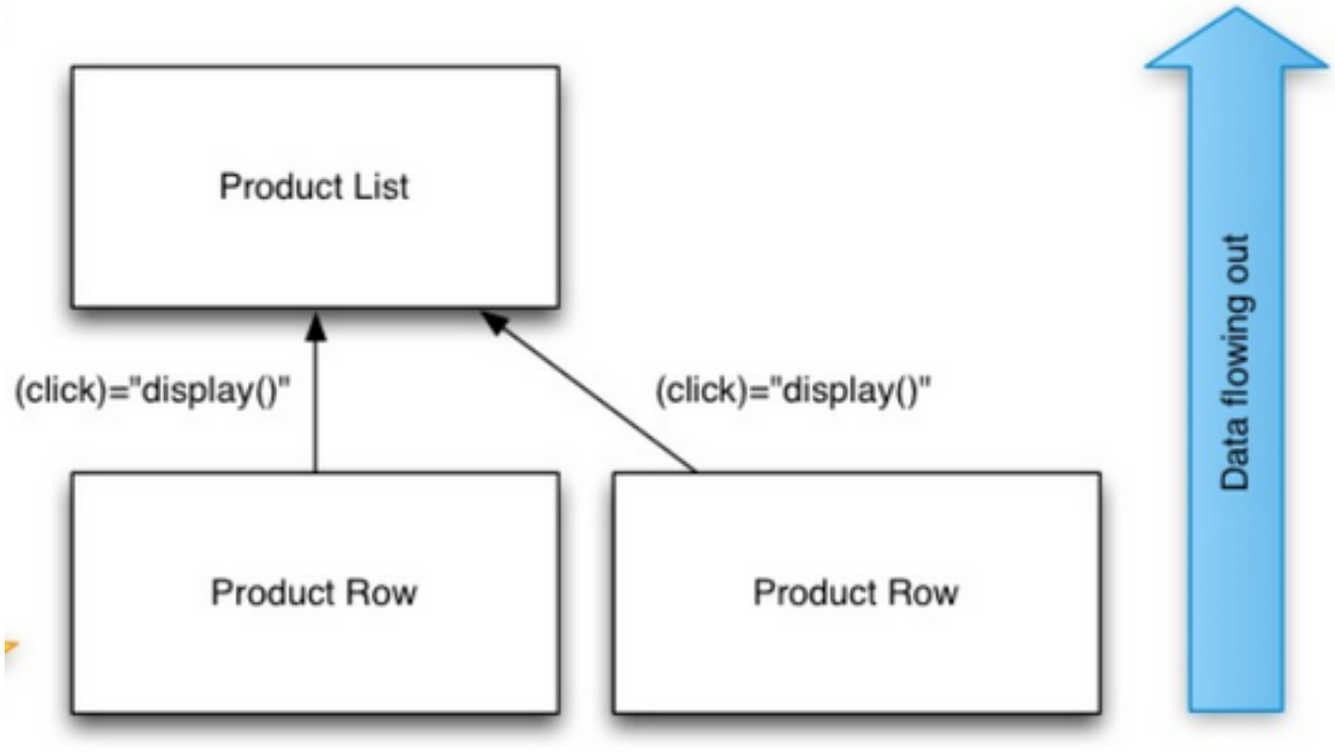
- Angular 的应用是由一系列的组件构成的
- 始终有一个包含其他组件的根组件
- 组件就是AngularJS中的指令

组件树

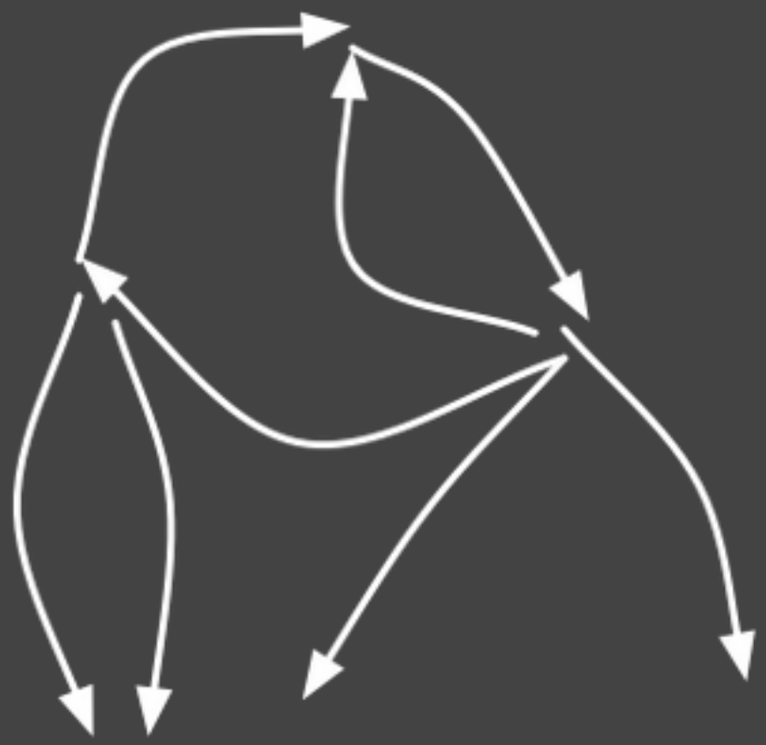

Inventory Management App
Navigation
Products List
Breadcrumbs
Product Row
Product Row
Product Row
Product Image
Product Department
Price Display

数据流


Data flowing in
Product List
[product]="product"
[product]="product"
Product Row
Product Row


Product List
(click)="display()"
(click)="display()"
Product Row
Product Row
Data flowing out

AngularJS



Angular

AngularJS

```
<parent prop1="exp1" prop2="exp2">
    <child prop3="exp3" prop4="exp4"></child>
</parent>
```

哪个指令先更新?

Angular

```
<parent [prop1]="exp1" [prop2]="exp2">
    <child [prop3]="exp3" [prop4]="exp4"></child>
</parent>
```

parent一定先更新

Digest TTL = 1

变化检查

是一颗树

是可预测的

TTL = 1

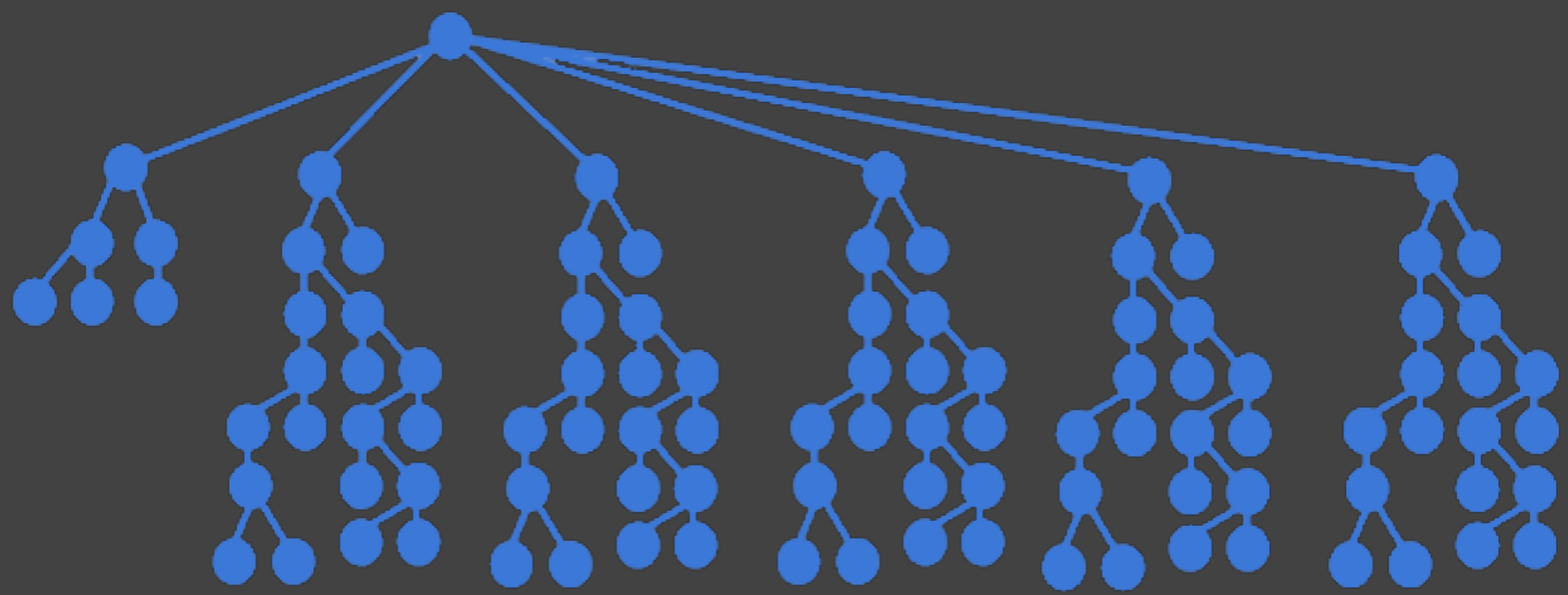
到底有多快?

变化检查所需的时间 = C * N

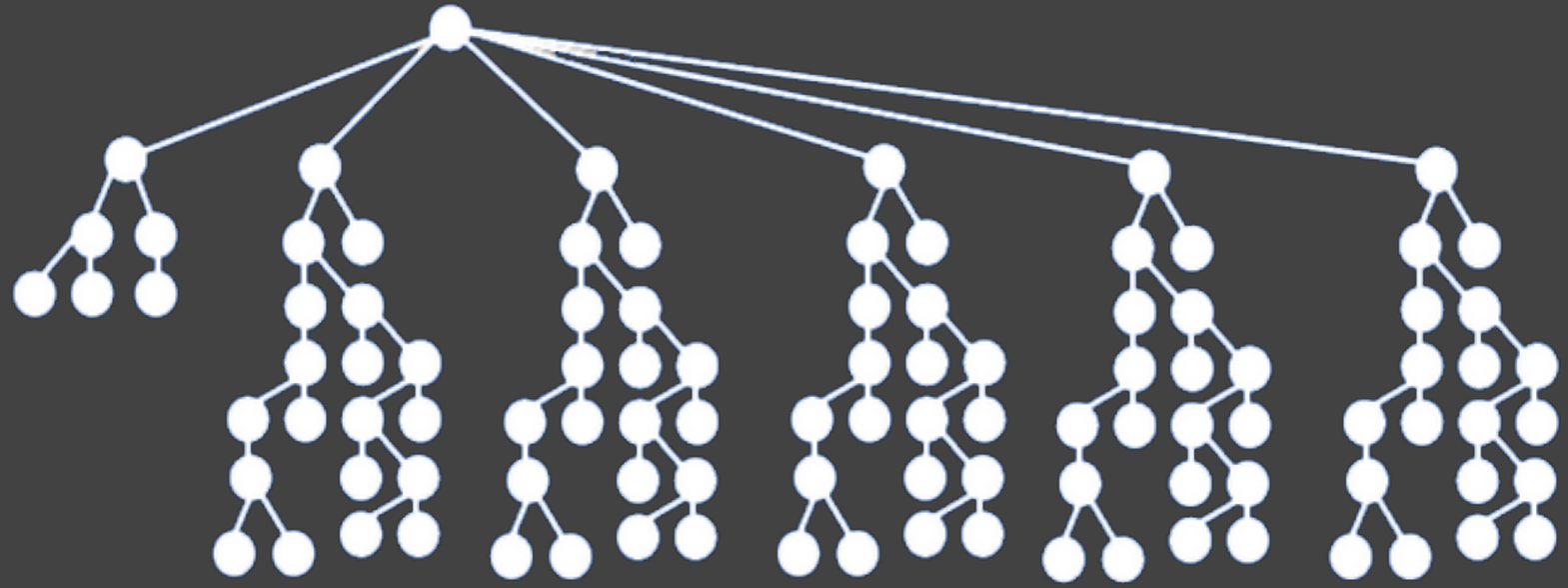
C - 检查绑定数据的所需时间

N - 绑定数据的数量

经过树的每个节点检查是否改变



经过树的每个节点检查是否改变



Result

3-10x

更聪明的变化检测

Immutable Objects

不可变对象

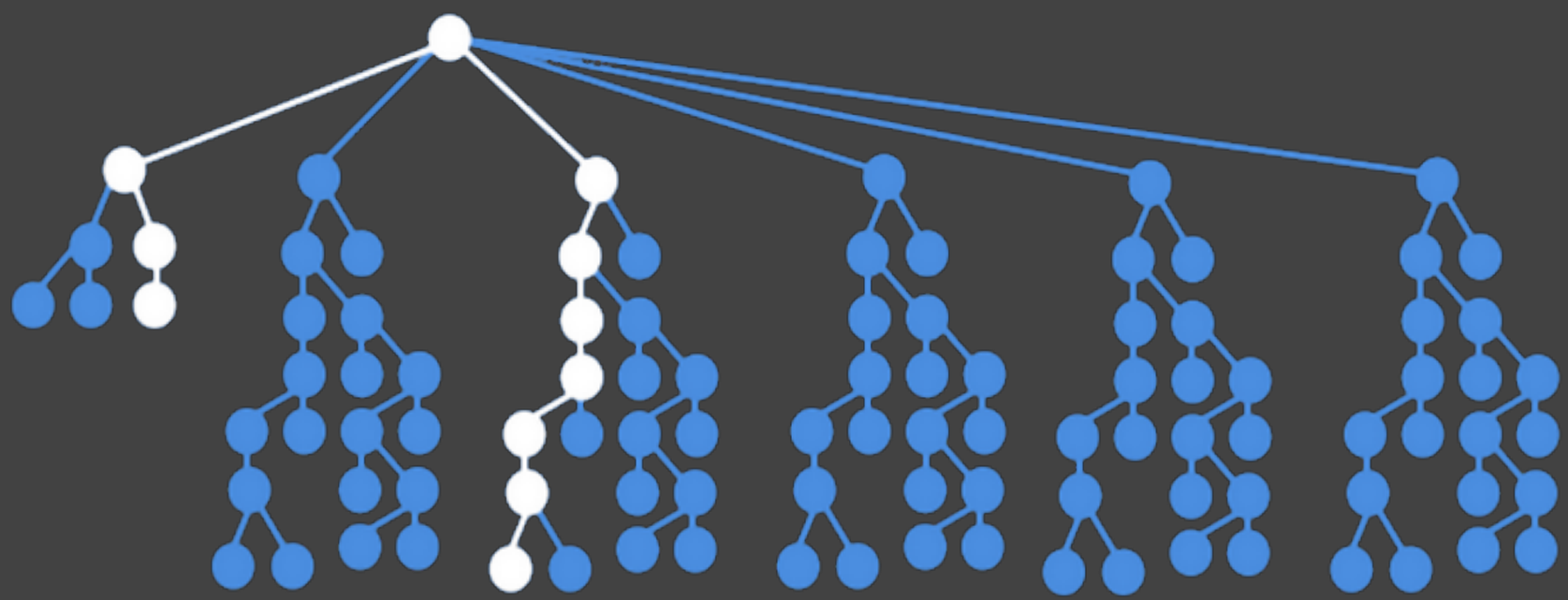
old === new

old.title === new.title

old.xxx === new.xxx

组件仅仅依赖其绑定的属性

组件仅仅依赖其绑定的属性



变化检测所需的时间 ≃ C * M

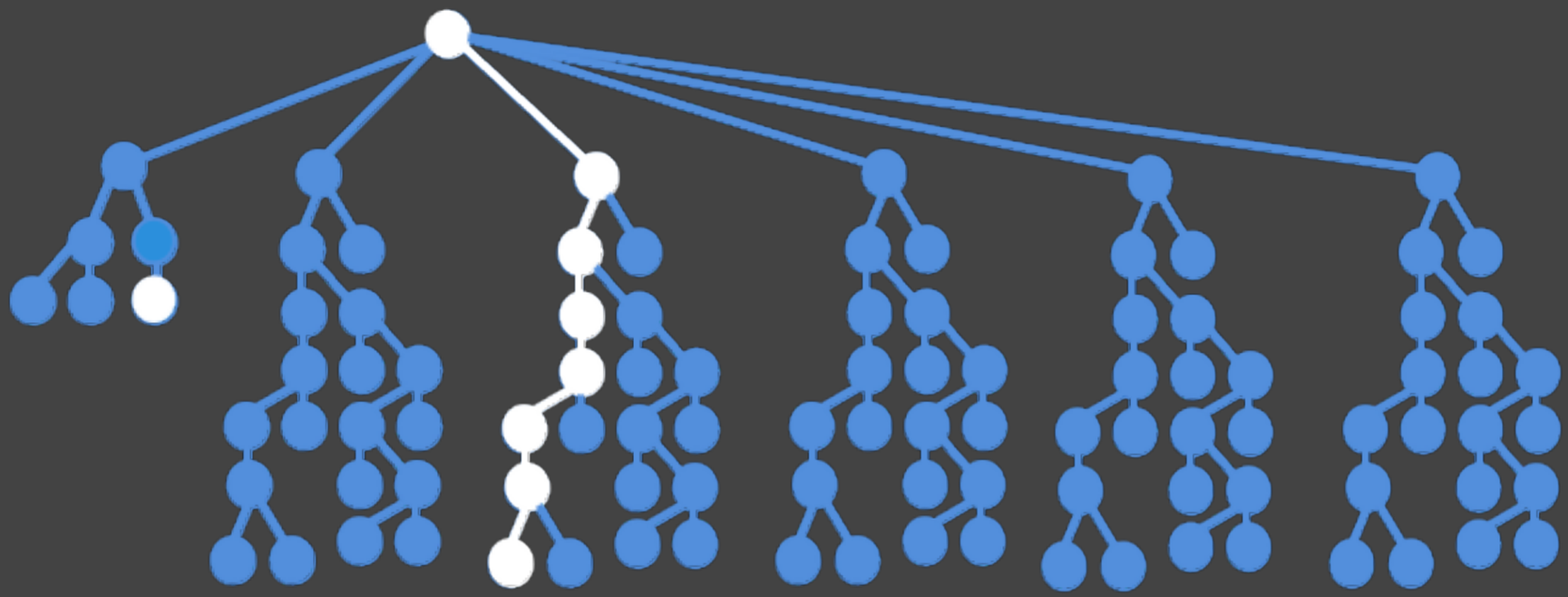
C - 检查绑定数据的所需时间

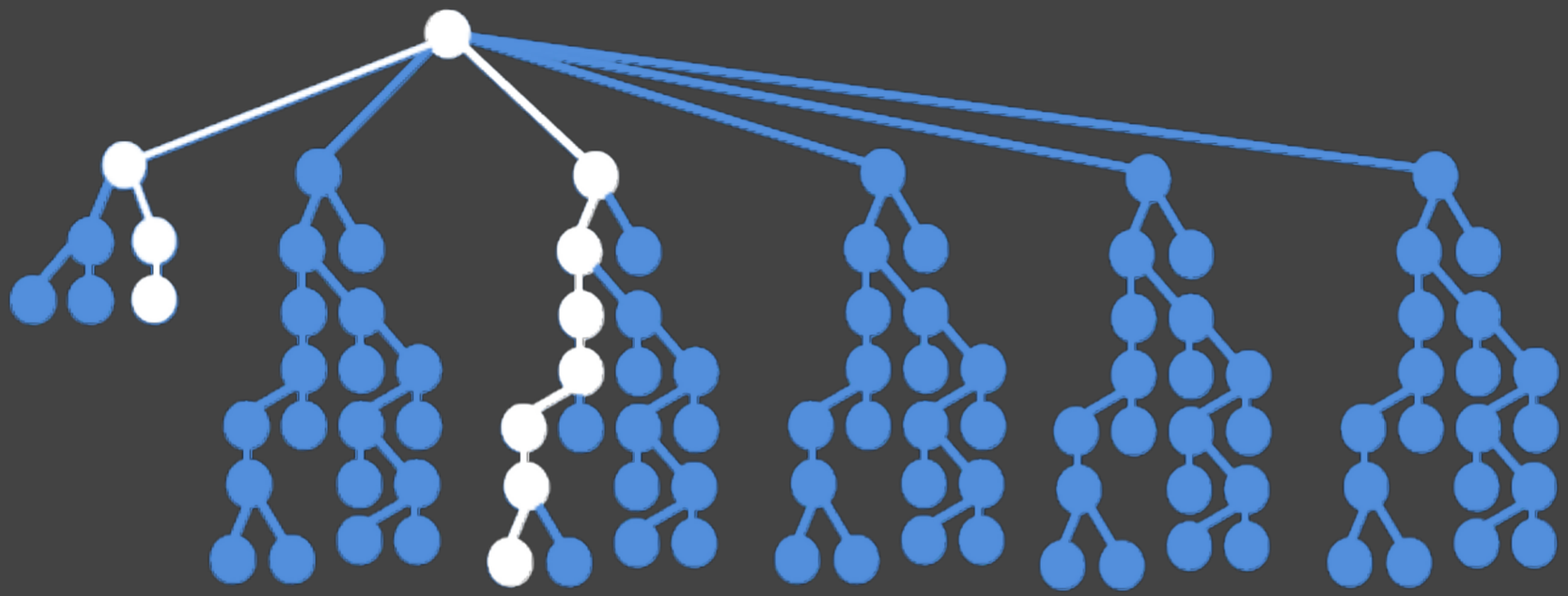
M <= N

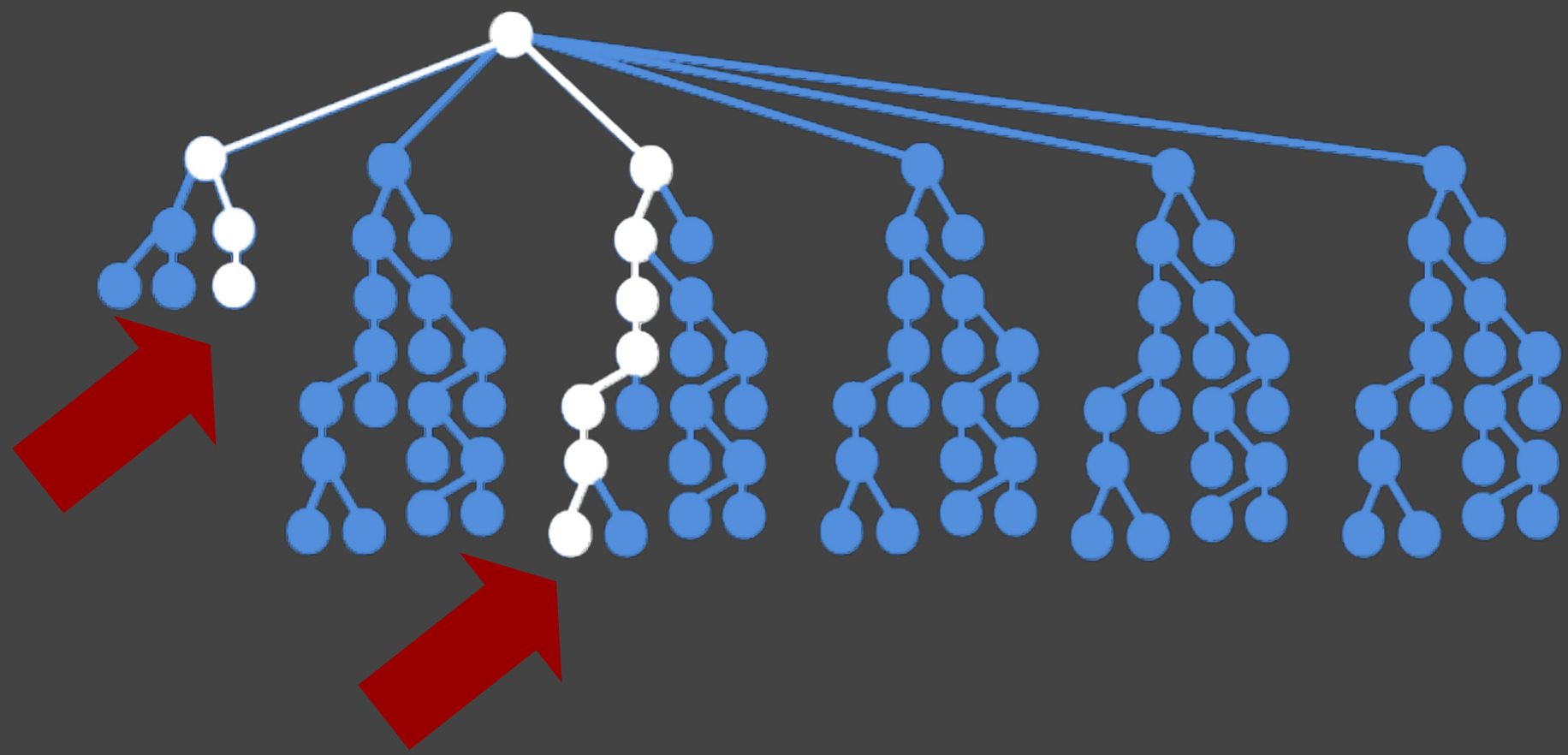
Observable Objects

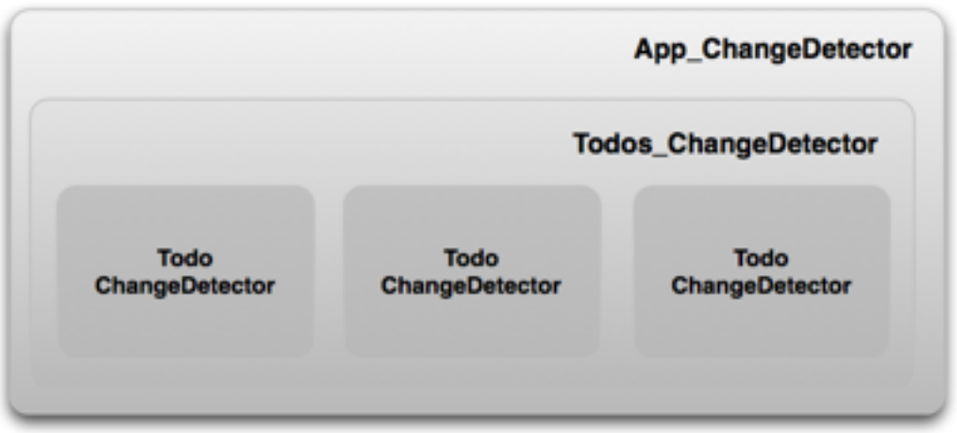
Observable

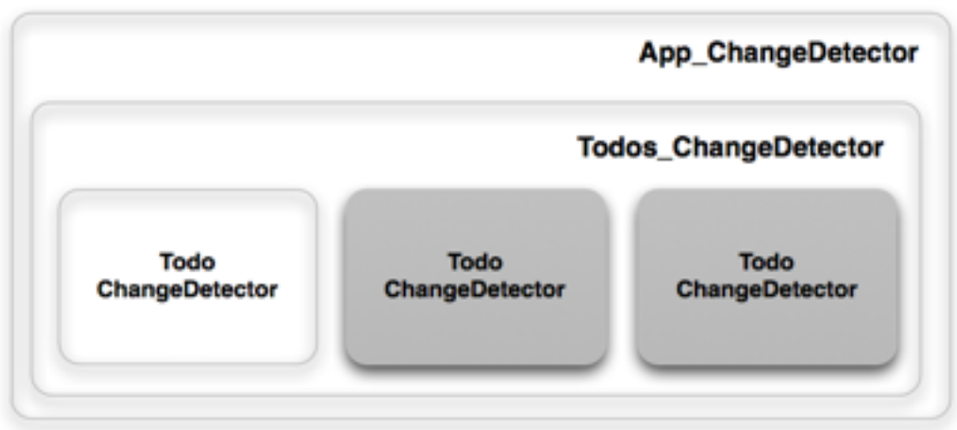
- 将绑定的数据设定成可观察对象
- 变化检测树中我们就可以跳过该组件的子树
- 该组件的变化检测只有通过观察者对象触发









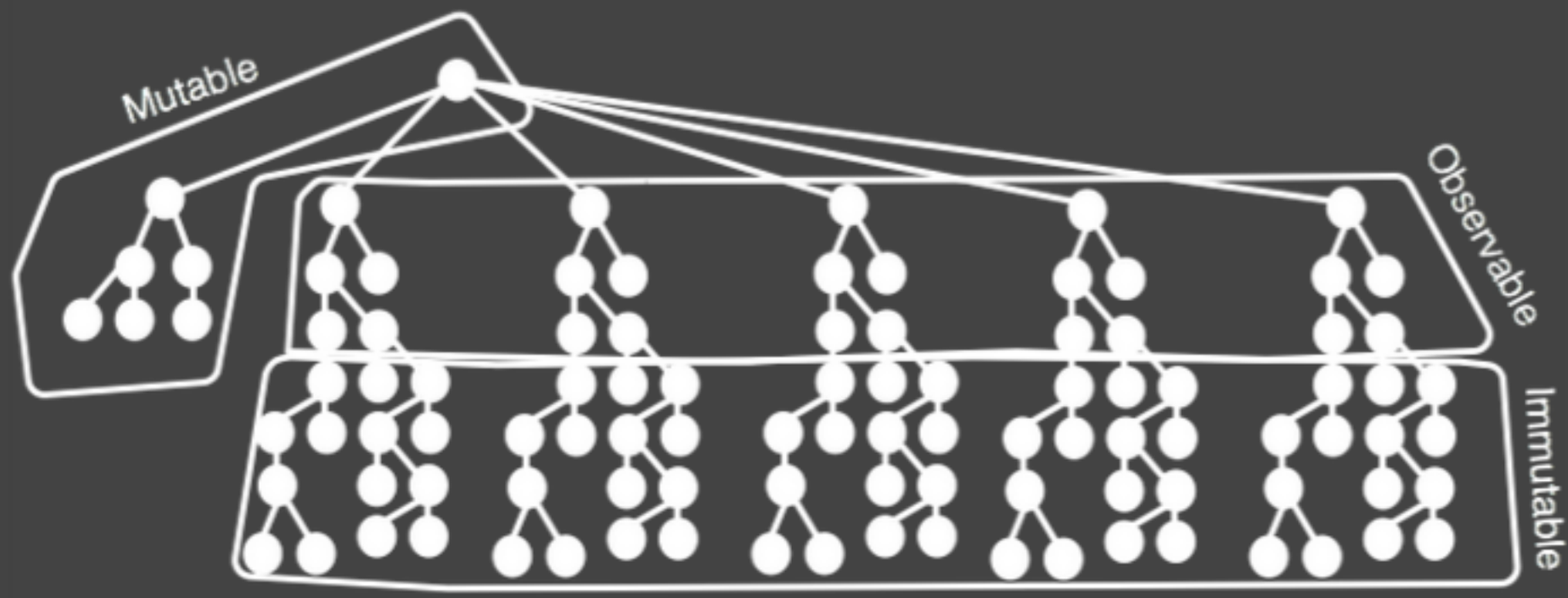


触发后变化检测系统将从根开始检查

Observable会引发联级更新吗?

- 可观察对象有不爽之处,因为它们会导致联级更新
- 使系统非常难推理
- 观察者通过触发事件只是标志着下一次需要检查的路径

混用



Angular.Model ?

```
<input [(ng-model)]="todo.text"></input>
```

依赖注入

AngularJS DI的问题

- 依赖是单例的
- 没有方便的方式来解决加载异步依赖
- 命名冲突
- AngularJS DI被嵌入在整个框架中，我们无法使用它作为一个独立的系统

token

- bind(Car).toClass(Car)
- injector.get(Car)

TypeScript直接通过type来注入

```
class Zilong {
    constructor(Pigu:pigu) {
    }
}
```

提供decorators来注入

```
class Zilong {
  constructor(@InjectPromise(Pigu) pigu) {

  }
}
```

- @Inject
- @InjectPromise
- @InjectLazy
- @Optional