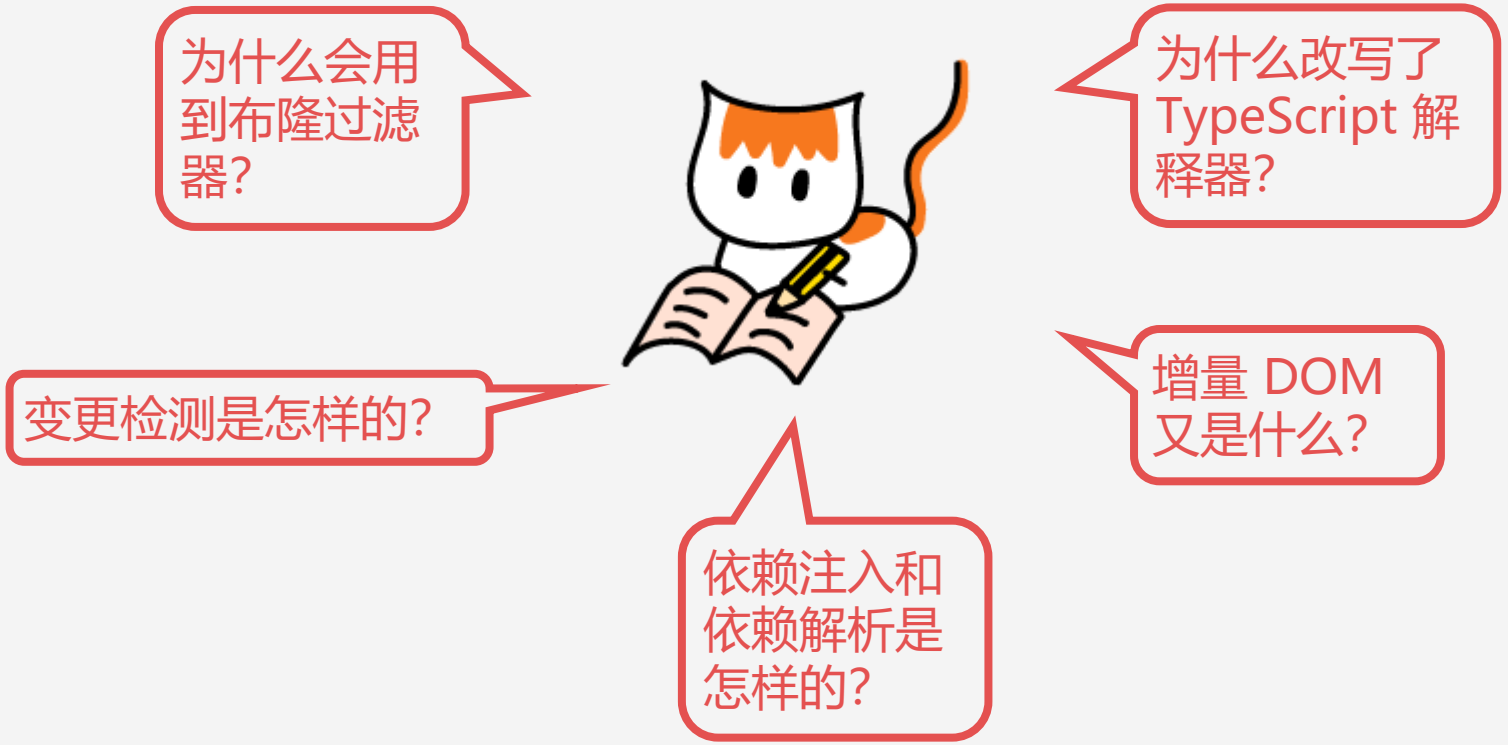
Angular 冷知识
之
ngZone

@被删

Angular 知多少

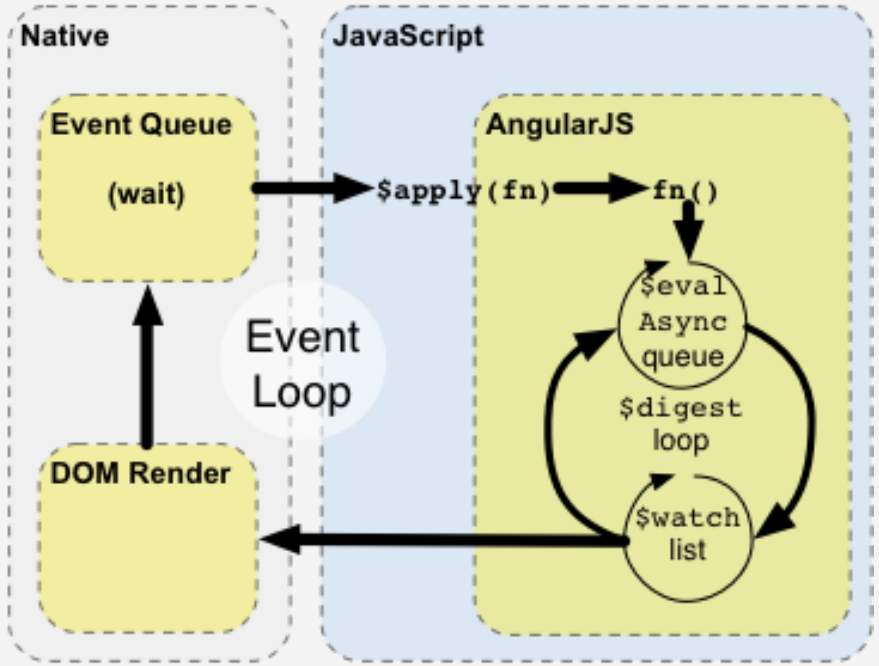

为什么会用到布隆过滤器?
为什么改写了 TypeScript 解释器?
变更检测是怎样的?
增量 DOM 又是什么?
依赖注入和依赖解析是怎样的?

Zone.js 和 ngZone 是做什么的?

1 AngularJS 脏检查

AngularJS(v1.x) 的变更检测

通过调用 scope.$apply 进入 AngularJS 执行上下文： AngularJS 执行 stimulusFn()，这通常会修改应用程序状态。



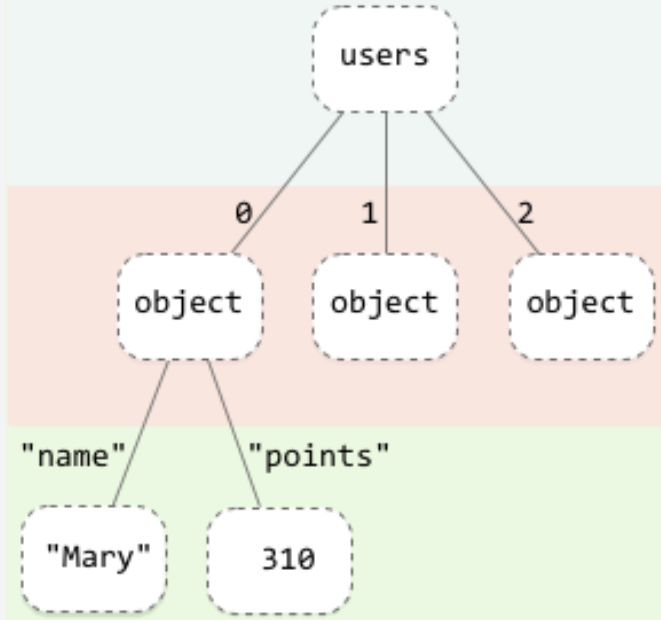
1. AngularJS 进入 $digest 循环，该循环由两个较小的循环组成，它们处理 $evalAsync 队列和 $watch 列表。
2. 在$digest 循环迭代持续直到模型稳定，这意味着 $evalAsync 队列为空，$watch 列表没有检测到任何变化。
3. 一旦 AngularJS $digest循环结束，执行就会离开 AngularJS 和 JavaScript 上下文。然后浏览器重新渲染 DOM 以反映任何更改。

- $evalAsync 队列：用于调度需要出现当前堆栈帧外、但在浏览器的视图之前渲染的工作
- $watch 列表：一组表达式，如果检测到更改，$watch 则调用该函数，该函数通常使用新值更新 DOM。

AngularJS(v1.x) 的变更检测

脏检查可以通过三种策略完成：按引用、按集合内容和按值。

1. 当 watch 表达式返回的整个值切换到新值时，通过引用观察(scope.$watch) 会检测到更改。如果值是数组或对象，则不会检测到其中的更改。
2. 按收集内容（scope.$watchCollection ） 。检测数组或对象内所发生的变化，检测很浅，不会进入嵌套集合。
3. 按值观察（scope.$watch ）检测任意嵌套数据结构中的任何更改。它是最强大的变化检测策略，但也是最昂贵的。每个摘要都需要完整遍历嵌套数据结构，并且需要将其完整副本保存在内存中。




```
$scope.users = [
  {name: "Mary", points: 310},
  {name: "June", points: 290},
  {name: "Bob", points: 300}
];
```

By Reference

```
$scope.$watch("users", …);
✔ $scope.users = newUsers;
✘ $scope.users.push(newUser);
✘ $scope.users[0].points = 320;
```

By Collection Items

```
$scope.$watchCollection("users", …);
✔ $scope.users = newUsers;
✔ $scope.users.push(newUser);
✘ $scope.users[0].points = 320;
```

By Value

```
$scope.$watch("users", …, true);
✔ $scope.users = newUsers;
✔ $scope.users.push(newUser);
✔ $scope.users[0].points = 320;
```

性能问题堪忧呀

Angular(2+) 版本的性能/设计改善措施

- 引入 ngZone 触发变更时机
- 模块化/树状的组件结构
- 增量 DOM 设计

2 Zone.js/ngZone 设计

JavaScript 运行时上下文

在 Javascript 中，代码执行过程中会产生堆栈，函数会在堆栈中执行。

对于异步操作来说，异步代码和函数执行的时候，上下文可能发生了变化，为此可能导致一些难题。比如：

1. 异步代码执行时，上下文发生了变更，导致预期不一致。
2. throw Error 时，无法准确定位到上下文。
3. 测试某个函数的执行耗时，但因为函数内有异步逻辑，无法得到准确的执行时间。

可以通过传参或是全局变量的方式来解决
但两种方式都不是很优雅（尤其全局变量）

Zone 的概念

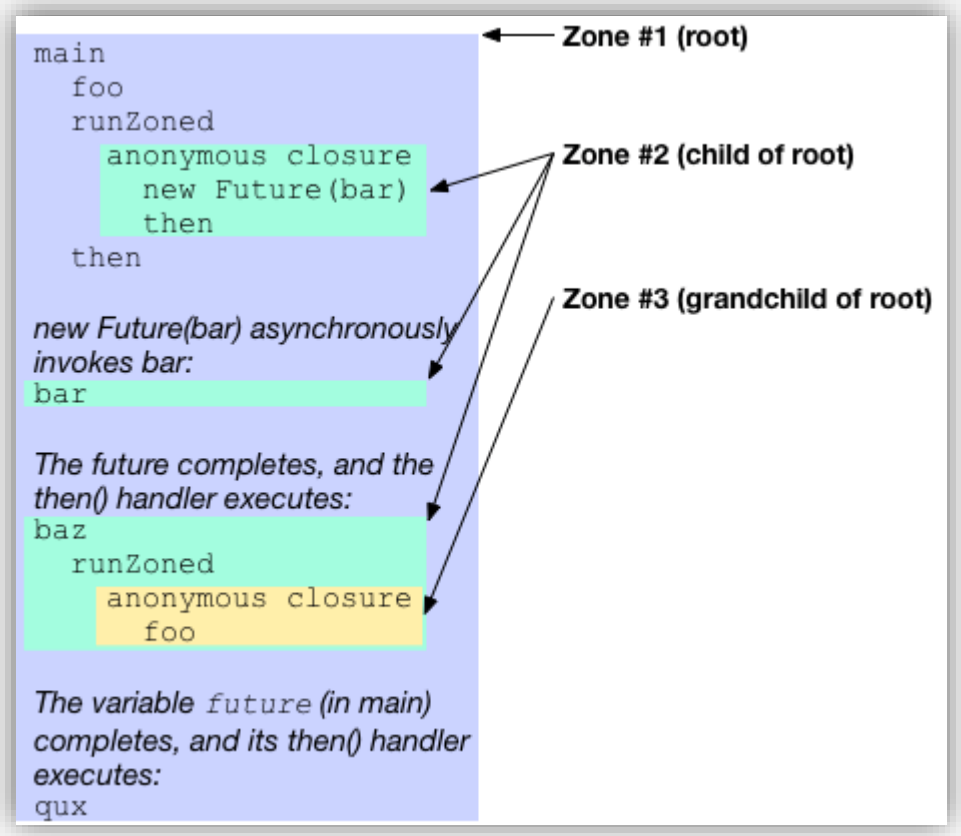
Zone 是跨异步任务而持久存在的执行上下文，具有当前区域的概念：

当前区域是随所有异步操作一起传播的异步上下文，它表示与当前正在执行的堆栈帧/异步任务关联的区域。

zone.js 提供以下能力：

1. 提供异步操作之间的执行上下文。
2. 提供异步生命周期挂钩。
3. 提供统一的异步错误处理机制。

设计灵感来自 Dart Zones



Zone 的使用

当前上下文可以使用 Zone.current 获取，可比作 JavaScript 中的 this

每个区域都有 name 属性，主要用于工具和调试目的，zone.js 还定义了用于操纵区域的方法：

- zone.fork(zoneSpec): 创建一个新的子区域，并将其 parent 设置为用于分支的区域
- zone.run(callback, ...)：在给定区域中同步调用一个函数
- zone.wrap(callback)：产生一个新的函数，该函数将区域绑定在一个闭包中，与 JavaScript 中的 Function.prototype.bind 工作原理类似

```
interface Zone {
  ...
  // 通过在任务区域中恢复 Zone.currentTask 来执行任务
  runTask<T>(task: Task, applyThis?: any, applyArgs?: any): T;
  // 安排一个 MicroTask
  scheduleMicroTask(source: string, callback: Function, data?: TaskData,
  // 安排一个 MacroTask
  scheduleMacroTask(source: string, callback: Function, data?: TaskData,
  // 安排一个 EventTask
  scheduleEventTask(source: string, callback: Function, data?: TaskData,
  // 安排现有任务（对重新安排已取消的任务很有用）
  scheduleTask<T extends Task>(task: T): T;
  // 允许区域拦截计划任务的取消，使用 ZoneSpec.onCancelTask 配置拦截
  cancelTask(task: Task): any;
}
```

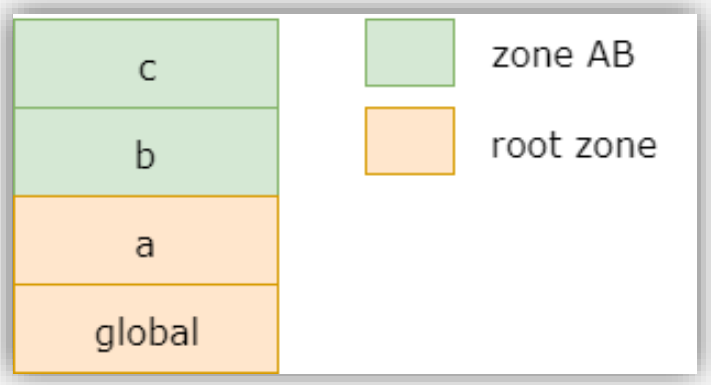
提供了许多方法来运行、计划和取消任务

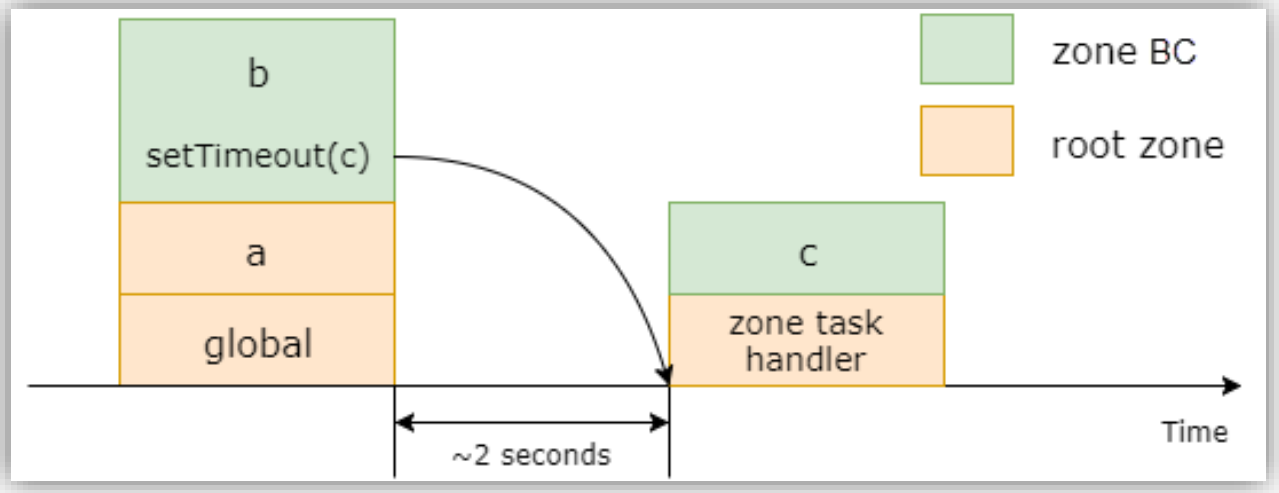
让异步逻辑运行在指定区域中

```
const zoneBC = Zone.current.fork({name: 'BC'});
function c() {
    console.log(Zone.current.name);  // BC
}
function b() {
    console.log(Zone.current.name);  // BC
    setTimeout(c, 2000);
}
function a() {
    console.log(Zone.current.name);  // <root>
    zoneBC.run(b);
}

a();
```

执行的效果





实际上，每个异步任务的调用堆栈会以根区域开始

如何识别出异步任务?

zone.js 主要是通过猴子补丁拦截异步 API（包括 DOM 事件、XMLHttpRequest和 NodeJS 的 API 如EventEmitter、fs等）来实现这些功能：

```
// 为指定的本地模块加载补丁
static __load_patch(name: string, fn: _PatchFn, ignoreDuplicate = false): void {
  // 检查是否已经加载补丁
  if (patches.hasOwnProperty(name)) {
    if (!ignoreDuplicate && checkDuplicate) {
      throw Error('Already loaded patch: ' + name);
    }
  // 检查是否需要加载补丁
  } else if (!global['__Zone_disable_' + name]) {
    const perfName = 'Zone:' + name;
    // 使用 performance.mark 标记时间戳
    mark(perfName);
    // 拦截指定异步 API，并进行相关处理
    patches[name] = fn(global, Zone, _api);
    // 使用 performance.measure 计算耗时
    performanceMeasure(perfName, perfName);
  }
}
```

比如定时器

```
Zone.__load_patch('timers', (global: any) => {
  const set = 'set';
  const clear = 'clear';
  patchTimer(global, set, clear, 'Timeout');
  patchTimer(global, set, clear, 'Interval');
  patchTimer(global, set, clear, 'Immediate');
});
```

计时器相关的 Timer 会被创建 MacroTask 任务并添加到 Zone 的任务中进行处理

```
type TaskType = 'microTask'|'macroTask'|'eventTask';
```

任务执行的生命周期

zone.js 提供了异步操作生命周期钩子，有了这些钩子，Zone 可以监视和拦截异步操作的所有生命周期：

```
interface ZoneSpec {
  // 允许拦截 Zone.fork，对该区域进行 fork 时，请求将转发到此方法以进行拦截
  onFork?: (parentZoneDelegate: ZoneDelegate, currentZone: Zone, targetZ
  // 允许拦截回调的 wrap
  onIntercept?: (parentZoneDelegate: ZoneDelegate, currentZone: Zone, ta
  // 允许拦截回调调用
  onInvoke?: (parentZoneDelegate: ZoneDelegate, currentZone: Zone, targe
  // 允许拦截错误处理
  onHandleError?: (parentZoneDelegate: ZoneDelegate, currentZone: Zone,
  // 允许拦截任务计划
  onScheduleTask?: (parentZoneDelegate: ZoneDelegate, currentZone: Zone,
  // 允许拦截任务回调调用
  onInvokeTask?: (parentZoneDelegate: ZoneDelegate, currentZone: Zone, t
  // 允许拦截任务取消
  onCancelTask?: (parentZoneDelegate: ZoneDelegate, currentZone: Zone, t
  // 通知对任务队列为空状态的更改
  onHasTask?: (parentZoneDelegate: ZoneDelegate, currentZone: Zone, targ
}
```

NgZone 的设计

NgZone 基于 zone.js 之上再做了一层封装，通过 fork 创建出子区域作为 Angular 区域：

1. zone.js 处理了大多数异步 API，比如 setTimeout()、Promise.then() 和 addEventListener() 等。
2. 对于一些 zone.js 无法处理的第三方 API，NgZone 服务的 run() 方法可允许在 angular Zone 中执行函数。
3. 通过使用 Angular Zone，函数中的所有异步操作会在正确的时间自动触发变更检测。
4. 不想触发变更检测（比如像 scroll 等事件过于频繁），此时可以使用 NgZone 的 runOutsideAngular() 方法。

NgZone自动触发变更检测

```
@Injectable()
export class ApplicationRef {
  ...
  constructor(
      private _zone: NgZone, private _injector: Injector, private _exceptionHandler:
      private _componentFactoryResolver: ComponentFactoryResolver,
      private _initStatus: ApplicationInitStatus) {
    // Microtask 为空时，触发变更检测
    this._onMicrotaskEmptySubscription = this._zone.onMicrotaskEmpty.subscribe({
      next: () => {
        this._zone.run(() => {
          // tick 为变更检测的逻辑，会重新进行 template 的计算和渲染
          this.tick();
        });
      }
    });
    ...
}
```

当 NgZone 满足以下条件时，会创建一个名为 angular 的 Zone 来自动触发变更检测：

- 当执行同步或异步功能时（zone.js 内置变更检测，最终会通过 onMicrotaskEmpty 来触发）
- 已经没有已计划的 Microtask（onMicrotaskEmpty）

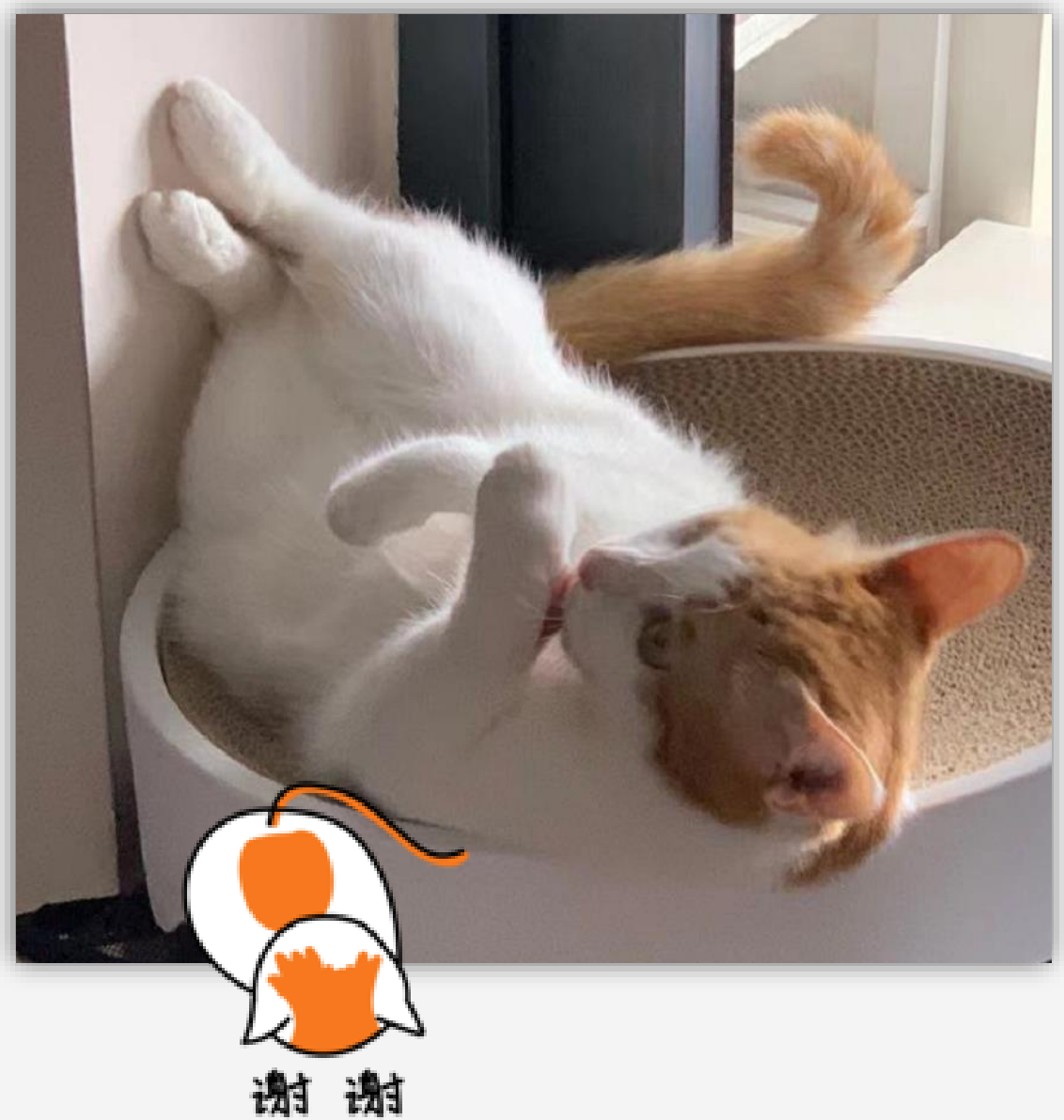
变更检测的时机更加精准

Angular(2+) 版本的性能/设计改善措施

- 引入 ngZone 触发变更时机
- 模块化/树状的组件结构
- 增量 DOM 设计

Angular 框架源码分析和解读

- https://github.com/godbasin/godbasin.github.io
- https://github.com/godbasin/front-end-playground



Github: godbasin
@被删