{CSS}

浅谈 CSS 与
web 界面动效开发模式

尤雨溪

@尤小右　@yyx990803　知 @尤雨溪

之前：Google Creative Lab
HTML5 交互原型开发

现在：METEOR 工程师

Node.js 全栈开发

- 界面动效正变得越来越重要，尤其在移动端
- web app 提升用户体验和挑战原生应用的必经之路

在 Creative Labs 期间参与了很多对界面动效有较高要求的交互原型的开发，对这一块积累了一些经验。

如何提升界面动效开发者的开发体验?

在这个短短的演讲里，我没有办法给大家一个答案。同时我认为，在 web app 的这个领域我们还没有找到一个完美的解决方案。今天我只是和大家探讨一下一些我们面对的问题和已经存在的解决方案，希望能够抛砖引玉，激发大家在这个领域的思考。

API 设计：命令式 vs. 声明式

首先我想探讨的就是我们作为开发者书写界面动效的思维模式。

这里有两大类别：命令式和声明式。

命令式

- jQuery.animate
- Velocity.js
- GSAP (GreenSocks)

命令式

显式调用动画函数来触发效果

```
$( "#clickme" ).click(function() {
  $( "#book" ).animate({
    opacity: 0.25,
    left: "+=50",
    height: "toggle"
  }, 5000, function() {
    // Animation complete.
  });
});
```

声明式

- CSS Transition
- CSS Animation
- ngAnimate
- Vue.js transitions

声明式

- 声明式地定义各个 “状态” 下的 CSS 规则
- 通过状态的变化 (切换 CSS class) 来触发动画效果

· 为了示例代码的简洁，我们在下面的代码都省略前缀并假设已经包含了

```
* { transition: all .5s ease }
```

- 切换一个 class 可以触发嵌套的多个 transition

```
/* 状态1 */
#app {
  opacity: 0;
}
#app h1 {
  transform: translate(0, 30px);
}

/* 状态2 */
#app.show {
  opacity: 1;
}
#app.show h1 {
  transform: translate(0, 0);
}
```

基于 CSS 预处理器可以让声明式的样式更易维护

（下图为 LESS 代码示例）

```less
// 状态1
#app {
  opacity: 1;
  h1 {
    transform: translate(0, 0);
  }
}

// 状态2
#app.show {
  opacity: 0;
  h1 {
    transform: translate(0, 30px);
  }
}
```

Angular / Vue.js:

通过数据绑定直接将 CSS class 与应用状态挂钩

HTML

```
<div
  class="tooltip"
  v-if="showToolTip"
  v-transition="fade">
</div>
```

CSS

```
.fade-enter {
  opacity: 0;
}
.fade-leave {
  opacity: 0;
}
```

JS

```
app.showToolTip = true;
```

命令式 vs. 声明式

- 命令式:
  - 动画状态和应用状态混杂在一起，
    逻辑复杂后不易维护，容易导致 bug
- 声明式:
  - 将应用状态和动画状态分离，易于维护

可以基于命令式的 API 构筑声明式的上层 API

具体实现：JavaScript vs. CSS

JavaScript Animations

- 通过逐帧计算并设置 CSS property 来模拟动画
- 可以进行更精确的控制：
  时间轴、缓动曲线、物理模拟、增量动画

- 需要在 js 主线程进行大量计算
- 通常伴随着命令式的 API

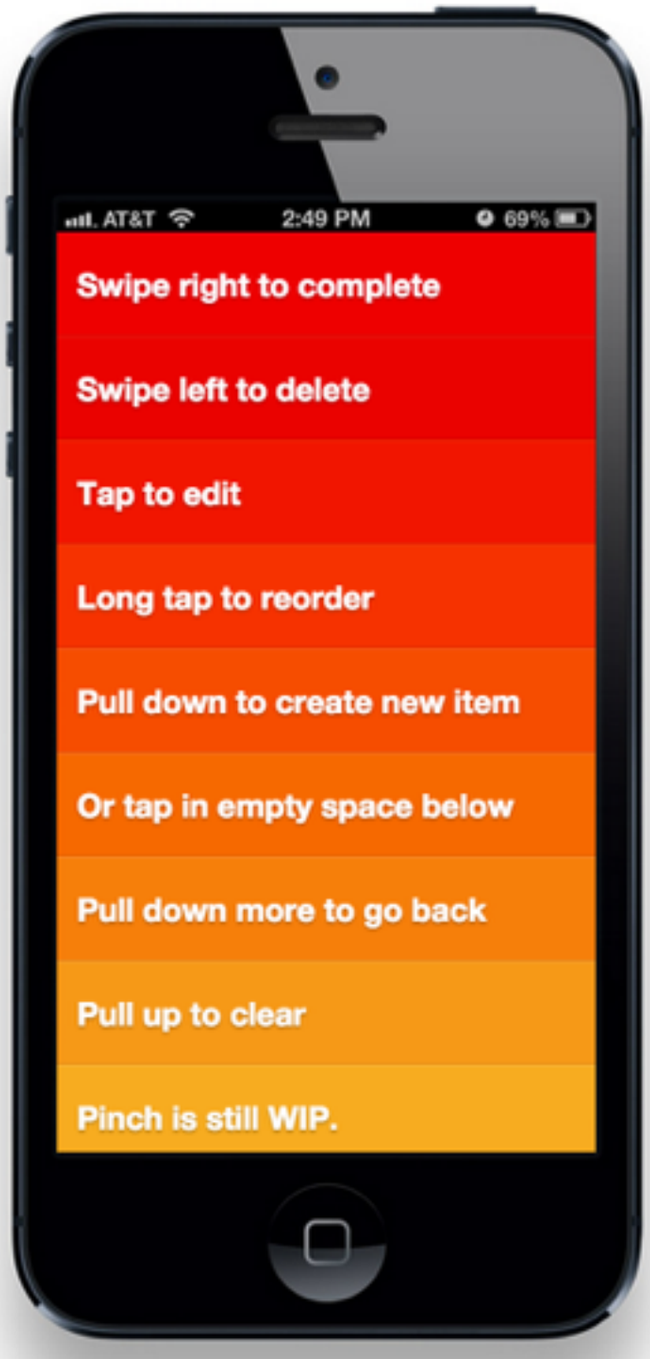
CSS Transitions / Animations

- 通过 CSS class / property 的变更来触发动画
- 对 js 主线程的计算消耗小
- 通常伴随着声明式的 API
- 受限制于 cubic-bezier 曲线或是 keyframes
  受限制于固定状态之间的切换，无法处理增量动画

针对使用场景选择解决方案

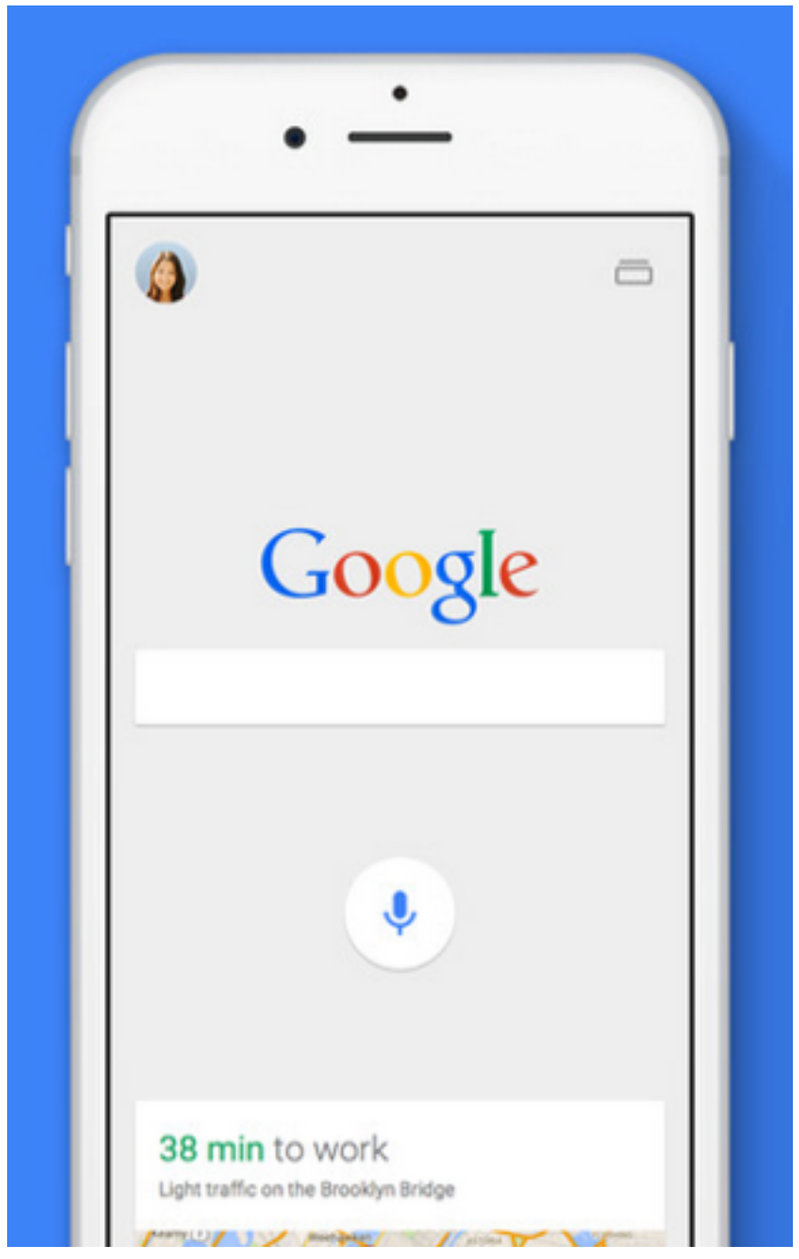
可维护性 vs. 细节要求

现有的声明式解决方案已经
可以在保证可维护性的基础上
应对基本的动效需求



clear.evanyou.me

细节要求 >>> 可维护性的典型例子



Google iOS App

HTML5 原型开发 - 声明式+命令式混合

理想的动效解决方案

- 提供声明式的 API
- 可以组构（composable）：顺序、并发、时间轴
- 支持状态切换，也支持增量动画
- 支持缓动曲线，也支持物理模拟

值得关注的一些新方案

Famo.us

- 彻底绕过浏览器的 CSS 布局，全部依赖 CSS transform 自己实现一套布局系统，从而避免 reflow 的问题
- 类似游戏的渲染机制
- 强大的缓动和物理模拟系统
- 混合了命令式和声明式的 API
  Modifier + Transitionable

WebAnimations

- Google 在 web 动效标准化方面的尝试
- 应用在 Polymer 之中

WebAnimations

- 通过 JavaScript 提供了可组构的、面向对象的底层抽象，便于构造声明式的上层 API

```
var animationGroup = new AnimationGroup([
  new AnimationSequence([
    new Animation(...),
    new Animation(...),
  ]),
  new Animation(...)
]);
```

{ THANKS }