30 天精通RxJS (11)：实务范例- 完整拖拉应用

Dec 27th, 2016 . 5 mins read

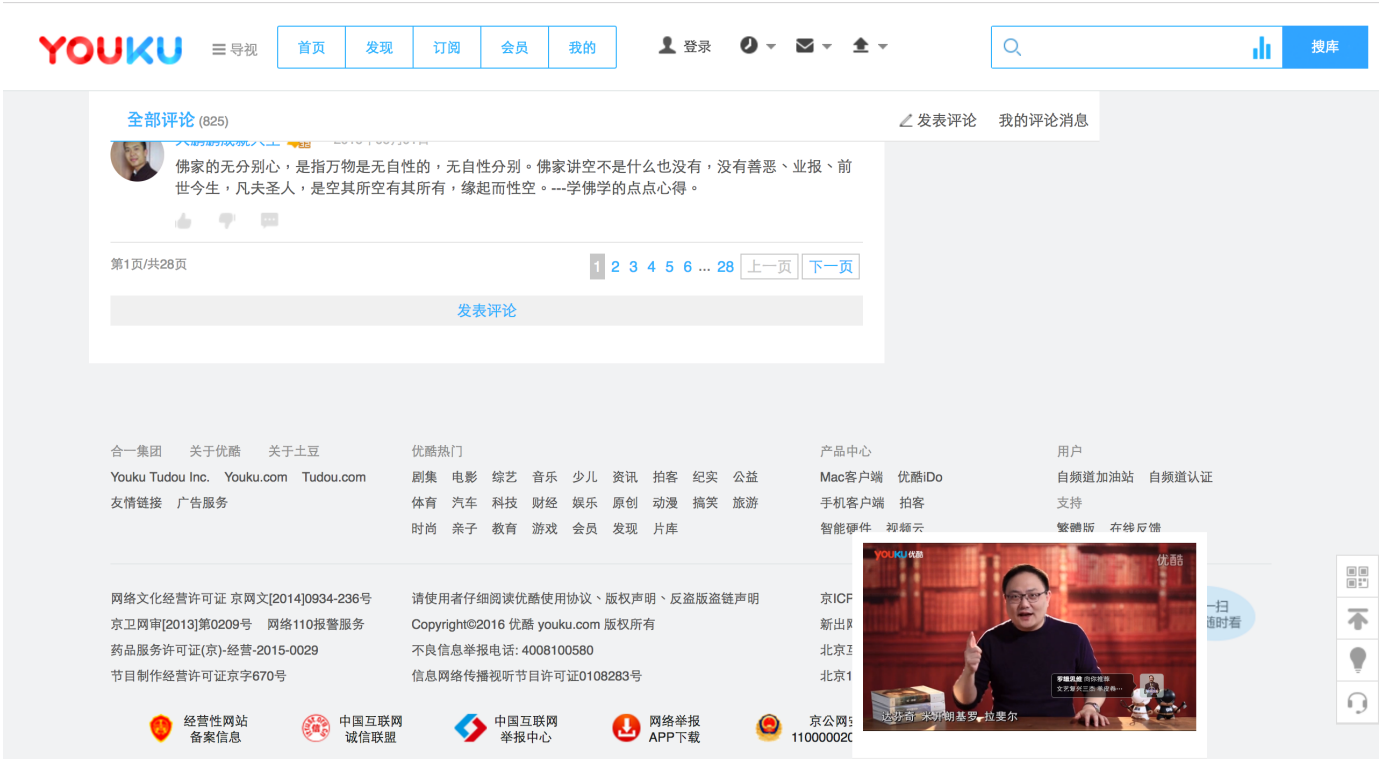
> 有次不小心进到了优酷，发现优酷有个不错的功能，能大大的提升用户体验，就让我们一起来实作这个效果吧！

一样建议大家可以直接看影片 Yes

在第08 篇的时候，我们已经成功做出简易的拖拉效果，今天要来做一个完整的应用，而且是实务上有机会遇到但不好处理的需求，那就是优酷的影片效果！

> 如果还没有用过优酷的读者可以先前往 这里 试用。

当我们在优酷看影片时往下滚动画面，影片会变成一个小视窗在右下角，这个视窗还能够拖拉移动位置。这个功能可以让使用者一边看留言同时又能看影片，且不影响其他的资讯显示，真的是很不错的feature。



就让我们一起来实作这个功能，同时补完拖拉所需要注意的细节吧！

需求分析

影片上方(hover)会有遮罩(masker)与鼠标变化(cursor)，可以拖拉移动(drag)，且移动范围不超过可视区间！

上面可以拆分成以下几个步骤

- 准备static 样式与fixed 样式
- HTML 要有一个固定位置的锚点(anchor)
- 当滚动超过锚点，则影片变成fixed
- 当往回滚动过锚点上方，则影片变回static
- 影片fixed 时，要能够拖拉
- 拖拉范围限制在当前可视区间

基本的HTML 跟CSS 笔者已经帮大家完成，大家可以直接到下面的连结接着实作：

- JSBin
- JSFiddle

先让我们看一下HTML，首先在HTML 里有一个div(#anchor)，这个div(#anchor) 就是待会要做锚点用的，它内部有一个div(#video)，则是滚动后要改变成fixed 的元件。

CSS的部分我们只需要知道滚动到下方后，要把div(#video)加上 `video-fixed` 这个class。

接着我们就开始实作滚动的效果切换class 的效果吧！

第一步，取得会用到的DOM

因为先做滚动切换class，所以这里用到的DOM 只有#video, #anchor。

```
const video = document.getElementById('video');
const anchor = document.getElementById('anchor');
```

第二步，建立会用到的observable

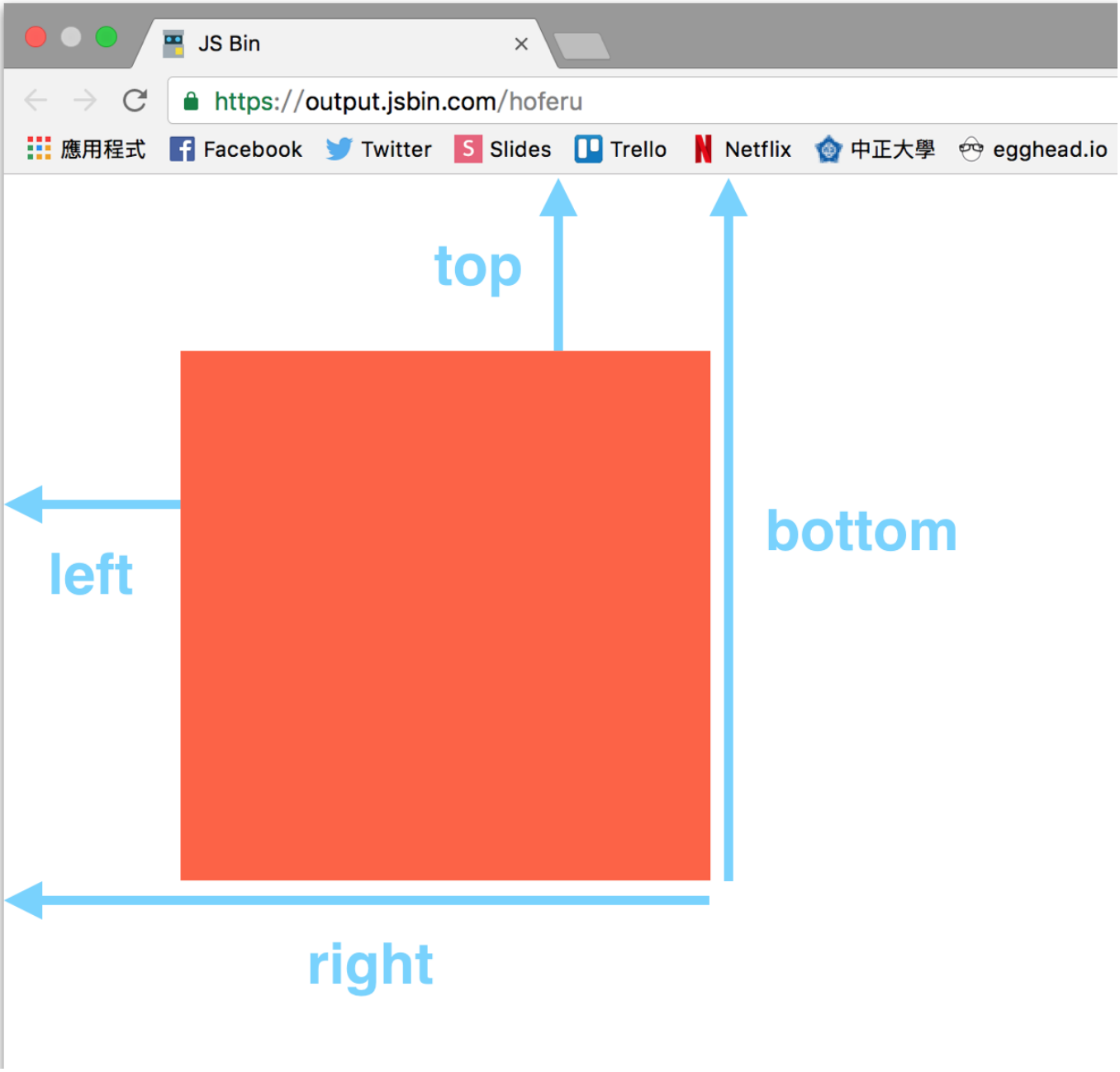
这里做滚动效果，所以只需要监听滚动事件。

```
const scroll = Rx.Observable.fromEvent(document, 'scroll');
```

首先我们会需要滚动事件发生时，去判断是否**滚过#anchor最底部**，所以把原本的滚动事件变成是否滚过最底部的true or false。

```
scroll.map(e => anchor.getBoundingClientRect().bottom < 0)
```

这里我们用到了 `getBoundingClientRect` 这个浏览器原生的API，他可以取得DOM物件的宽高以及上下左右离萤幕可视区间上(左)的距离，如下图



当我们可视范围区间滚过#anchor底部时，`anchor.getBoundingClientRect().bottom` 就会变成负值，此时我们就改变#video的class。

```
.subscribe(bool => {
    if(bool) {
        video.classList.add('video-fixed');
    } else {
        video.classList.remove('video-fixed');
    }
})
```

到这里我们就已经完成滚动变更样式的效果了！

全部的JS 程式码，如下

```
const video = document.getElementById('video');
const anchor = document.getElementById('anchor');

const scroll = Rx.Observable.fromEvent(document, 'scroll');

scroll
.map(e => anchor.getBoundingClientRect().bottom < 0)
.subscribe(bool => {
    if(bool) {
        video.classList.add('video-fixed');
    } else {
        video.classList.remove('video-fixed');
    }
})
```

> 当然这段还能在用debounce/throttle 或requestAnimationFrame 做优化，这个部分我们日后的文章会在提及。

接下来我们就可以接着做**拖拉的行为**了。

第一步，取得会用到的DOM

这里我们会用到的DOM 跟前面是一样的(#video)，所以不用多做什么。

第二步，建立会用到的observable

```
const mouseMove = Rx.Observable.fromEvent(document, 'mousemove')
```

第三步，撰写程式逻辑

跟上次是差不多的，首先我们会点击#video 元件，点击(mousedown)后要变成移动事件(mousemove)，而移动事件会在滑鼠放开(mouseup)时结束(takeUntil)

```
mouseDown
.map(e => mouseMove.takeUntil(mouseUp))
.concatAll()
```

因为把mouseDown observable发送出来的**事件**换成了mouseMove observable，所以变成了observable(mouseDown)送出observable(mouseMove)。因此最后用concatAll把后面送出的元素变成mouse move的事件。

> 这段如果不清楚的可以回去看一下08 篇的讲解

但这里会有一个问题，就是我们的这段拖拉事件其实只能做用到video-fixed 的时候，所以我们要加上 `filter`

```
mouseDown
.filter(e => video.classList.contains('video-fixed'))
.map(e => mouseMove.takeUntil(mouseUp))
.concatAll()
```

这里我们用filter如果当下#video没有 `video-dragable` class的话，事件就不会送出。

再来我们就能跟上次一样，把mousemove 事件变成{ x, y } 的物件，并订阅来改变#video 元件

```
mouseDown
    .filter(e => video.classList.contains('video-fixed'))
    .map(e => mouseMove.takeUntil(mouseUp))
    .concatAll()
    .map(m => {
        return {
            x: m.clientX,
            y: m.clientY
```





```
        video.style.left = pos.x + 'px';
    })
```

到这里我们基本上已经完成了所有功能，其步骤跟08 篇的方法是一样的，如果不熟悉的人可以回头看一下！

但这里有两个大问题我们还没有解决

1. 第一次拉动的时候会闪一下，不像优酷那么顺
2. 拖拉会跑出当前可视区间，跑上出去后就抓不回来了

让我们一个一个解决，首先第一个问题是因为我们的拖拉直接给元件滑鼠的位置(clientX, clientY)，而非给滑鼠相对移动的距离！

所以要解决这个问题很简单，我们只要把点击目标的左上角当作(0,0)，并以此改变元件的样式，就不会有闪动的问题。

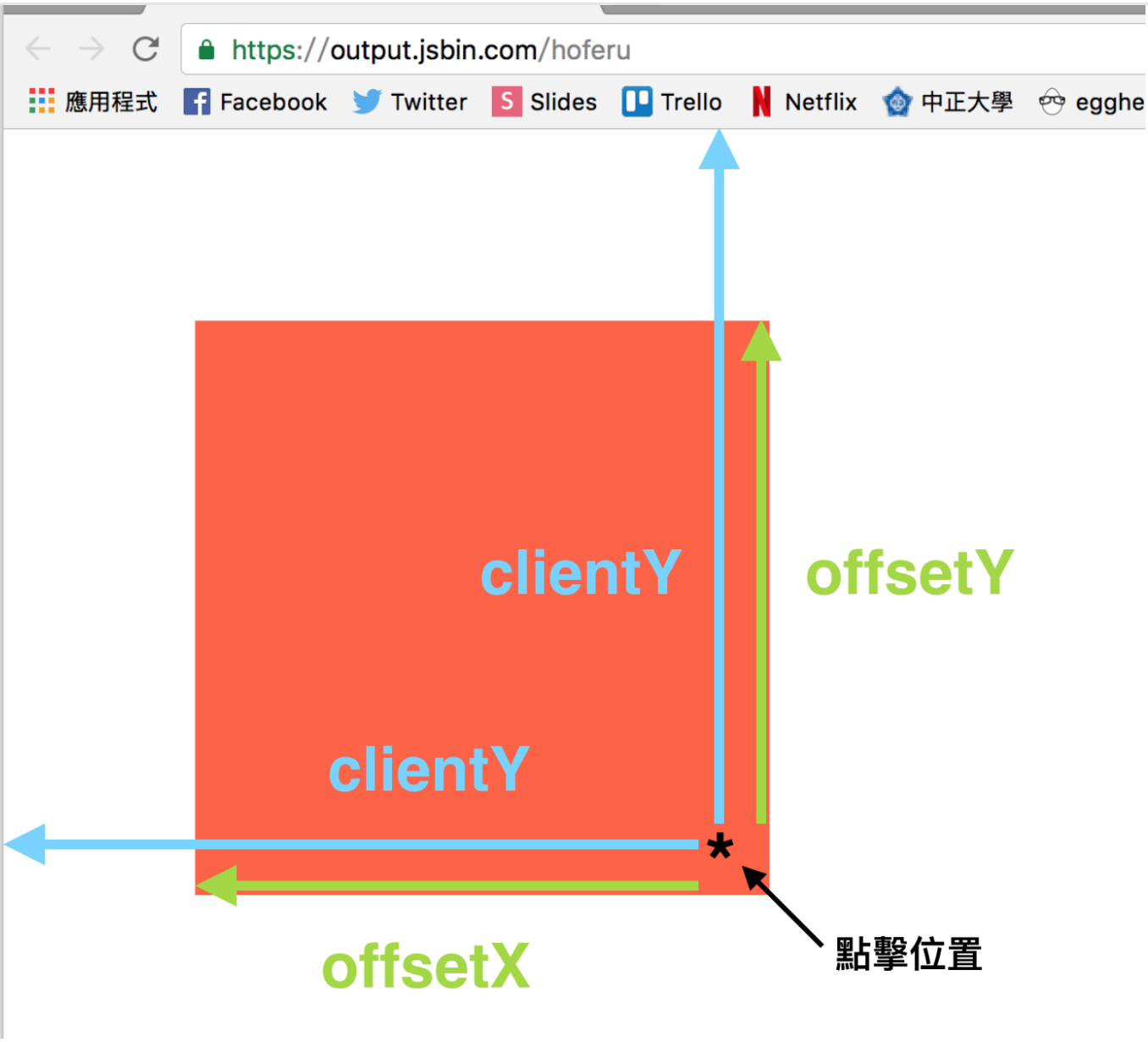
这个要怎么做呢？很简单，我们在昨天讲了一个operator 叫做withLatestFrom，我们可以用它来把mousedown 与mousemove 两个Event 的值同时传入callback。

```
mouseDown
    .filter(e => video.classList.contains('video-fixed'))
    .map(e => mouseMove.takeUntil(mouseUp))
    .concatAll()
    .withLatestFrom(mouseDown, (move, down) => {
        return {
            x: move.clientX - down.offsetX,
            y: move.clientY - down.offsetY
        }
    })
    .subscribe(pos => {
        video.style.top = pos.y + 'px';
        video.style.left = pos.x + 'px';
    })
```

当我们能够同时得到mousemove 跟mousedown 的事件，接着就只要把滑鼠相对可视区间的距离(client) 减掉点按下去时滑鼠相对元件边界的距离(offset) 就行了。这时拖拉就不会先闪动一下啰！

> 大家只要想一下，其实client - offset 就是元件相对于可视区间的距离，也就是他一开始没动

Series / 30 天精通RxJS



接着让我们解决第二个问题，拖拉会超出可视范围。这个问题其实只要给最大最小值就行了，因为需求的关系，这里我们的元件是相对可视居间的绝对位置(fixed)，也就是说

- top 最小是 0
- left 最小是 0
- top最大是**可视高度**扣掉**元件本身高度**
- left最大是**可视宽度**扣掉**元件本身宽度**

这里我们先宣告一个function 来处理这件事

```
const validValue = (value, max, min) => {
    return Math.min(Math.max(value, min), max)
```

再来我们就可以直接把这个问题解掉了

```
mouseDown
    .filter(e => video.classList.contains('video-fixed'))
    .map(e => mouseMove.takeUntil(mouseUp))
    .concatAll()
    .withLatestFrom(mouseDown, (move, down) => {
        return {
            x: validValue(move.clientX - down.offsetX, window.innerWidt
            y: validValue(move.clientY - down.offsetY, window.innerHeig
        }
    })
    .subscribe(pos => {
        video.style.top = pos.y + 'px';
        video.style.left = pos.x + 'px';
    })
```

这里我偷懒了一下，直接写死元件的宽高(320, 180)，实际上应该用 `getBoundingClientRect` 计算是比较好的。

现在我们就完成整个应用啰！

> 这里 有最后完成的结果。

今日结语

我们简单地用了不到35 行的程式码，完成了一个还算复杂的功能。更重要的是我们还保持了整支程式的可读性，让我们之后维护更加的轻松。

今天的练习就到这边结束了，不知道读者有没有收获呢？如果有任何问题欢迎在下方留言给我！

如果你喜欢本篇文章请帮我按个like 跟星星。

Tags

JavaScript RxJS Observable Drag&Drop RxJS 30 Days

Series / 30 天精通RxJS