Gestures in an Angular Application

Ryan Kara Follow
Apr 4, 2018 · 4 min read

In this post I will attempt to explain how to use hammerjs gesture recognizers provided by the @angular/platform-browser package. I'll be referencing *@angular/platform-browser@5.2.0* within my code samples, but there are some changes coming to 6.0.0 that will be discussed later.

Background

If you are working on a mobile project that requires gestures, hammerjs has the gestures to get you started. Out of the box, hammerjs includes pan,pinch, press, rotate, swipe, and tap gesture recognition. Each of these gesture recognizers may be wired up to any element within the DOM in order to detect the specific gesture and allow you to handle it.

Hammerjs relies on pointer events (pointermove, pointerup, pointerdown, and pointercancel) to perform all of its gestures. If Internet Explorer is being used, MSPointer events are used instead of the standard pointer events.

@angular/platform-browser handles the binding of the hammerjs gestures automatically within the HammerGestureConfig addEventListener method. This call will initialize hammerjs outside of angular and perform the necessary checks to determine if hammerjs is loaded within the angular application.

Getting started with hammerjs

hammerjs can easily be installed via npm by executing the following command within your angular project:

```
npm install hammerjs
```

Next, you will need to add `import 'hammerjs';` to your main.ts file. If you forget to add the import statement to your main.ts file you will

see an error within the console that will stop your application from running.

```
Error: Hammer.js is not loaded, can not bind to x event
```

Difficulties in hammerjs

Currently, in version 5.2.0, an exception is thrown, when you forget to include hammerjs, that stops the remainder of your angular project to load. *@angular/platform-browser@6.0.0* changes the thrown exception over to a `console.warn`, so that the remainder of the project continues to load without gestures.

The original issue can be tracked within the github issue tracker.

hammerjs Gestures

@angular/platform-browser includes a set of events to attach to DOM elements.

The following events are included, but the up to date list can be found on github:

- pan
- panstart
- panmove
- panend
- pancancel
- panleft
- panright
- panup
- pandown
- pinch
- pinchstart

- pinchmove
- pinchend
- pinchcancel
- pinchin
- pinchout
- press
- pressup
- rotate
- rotatestart
- rotatemove
- rotateend
- rotatecancel
- swipe
- swipeleft
- swiperight
- swipeup
- swipedown
- tap

Gesture Recognizers

- Pan : A Pan gesture is recognized when a pointer is down and moved within a set direction. The pan gesture is commonly used when scrolling through a set of items.
- Pinch : A Pinch gesture is recognized when two or more pointers are moving toward or away from each other. The pinch gesture is commonly used for zooming in or out.
- Press : A Press gesture is recognized when the pointer is being held down for a set amount of time. This is commonly used for long presses.

- Rotate : A Rotate gesture is recognized when a set amount of pointers, minimum of 2, are moving in a circular motion. This is commonly used to rotate items.
- Swipe : A Swipe gesture is recognized when a pointer is moving at a set speed for a set minimum amount of distance. This is commonly used to flip between items within a UI. Instead of scrolling, it is more useful to swap out items in a set direction.
- Tap : A Tap gesture is recognized when a user taps the screen. This is commonly used for button presses.

Swipe vs Pan

Swipe and Pan can almost be used interchangeably, but the main difference is that a pan event will fire off as the panning occurs, whereas the swipe event only fires off at the end of the swipe. The pan is more useful for smoothly scrolling an item as you have your cursor down, but a swipe is more useful for scrolling an item after the swipe occurs.

hammerjs without @angular/platform-browser

Without *@angular/platform-browser*, you will be required to create your own custom directives to add gesture support to your application.

At a bare minimum you need to bind to the window's hammerjs manager and bind to the on tap event that is provided by hammerjs.

```
import { Directive, ElementRef, EventEmitter, Input, N

interface HammerManager {
  new (element: HTMLElement | SVGElement, options?: an
  destroy(): void;
  add(recognizer: Recognizer): void;
  on(eventName: string, callback: Function): void;
}

interface Recognizer {
  new (options?: any): Recognizer;
  recognizeWith(otherRecognizer: Recognizer | string):
}

@Directive({
  selector: '[customTapGesture]',
})
export class TapGestureDirective implements OnInit, On
  constructor(private elementRef: ElementRef, private

  /**
   * Return the hammerjs library if it's available
   */
  private get hammerLib() {
    return typeof window !== 'undefined' ? (window as
  }

  private manager?: HammerManager;

  /**
   * Event fired when the element is tapped
   */
  @Output() cTap = new EventEmitter<any>();

  /**
   * Binds HammerJS Instances
   */
  ngOnInit() {
    if (this.hammerLib) {
      this.manager = this.bindHammer();
    }
  }

  /**
```

```
45      * Unbinds HammerJS instances
```

A live version of this is available at https://stackblitz.com/edit/tap-gesture-directive.

HammerJS with @angular/platform-browser

Using the hammerjs library through *@angular/platform-browser* allows developers to easily configure gestures for mobile input without the use of custom directives. Each of the gestures events relies on custom defined DOM event plug-ins. These events are triggered outside of Angular's Zone.js instance and will only re-enter the zone when the proper event is fired. More information about the DOM event plug-ins can be found in Ben Nadel's blog. Below I will wire up gestures on a simple div for each of the gesture recognizers.

Documentation on the event object that is returned from hammerjs triggers can be found here.

Pan

Wiring up the DOM Element within the Angular Component:

```
<div
    (pan)="onPan($event)"
    (panstart)="onPanStart($event)"
    (panmove)="onPanMove($event)"
    (panend)="onPanEnd($event)"
    (pancancel)="onPanCancel($event)"
    (panleft)="onPanLeft($event)"
    (panright)="onPanRight($event)"
```

A live stackblitz is avaiable at https://stackblitz.com/edit/pan-gesture.

Pinch

Wiring up the DOM Element within the Angular Component:

```
<div
    (pinch)="onPinch($event)"
    (pinchstart)="onPinchStart($event)"
    (pinchmove)="onPinchMove($event)"
    (pinchend)="onPinchEnd($event)"
    (pinchcancel)="onPinchCancel($event)"
    (pinchin)="onPinchIn($event)"
```

A live stackblitz is avaiable at https://stackblitz.com/edit/pinch-gesture.

Press

Wiring up the DOM Element within the Angular Component:

```
<div
    (press)="onPress($event)"
    (pressup)="onPressUp($event)">
</div>
```

A live stackblitz is avaiable at https://stackblitz.com/edit/press-gesture.

Rotate

Wiring up the DOM Element within the Angular Component:

```
<div
    (rotate)="onRotate($event)"
    (rotatestart)="onRotateStart($event)"
    (rotatemove)="onRotateMove($event)"
    (rotateend)="onRotateEnd($event)"
    (rotatecancel)="onRotateCancel($event)">
```

A live stackblitz is avaiable at https://stackblitz.com/edit/rotate-gesture.

Swipe

Wiring up the DOM Element within the Angular Component:

```
<div
    (swipe)="onSwipe($event)"
    (swipeleft)="onSwipeLeft($event)"
    (swiperight)="onSwipeRight($event)"
    (swipeup)="onSwipeUp($event)"
    (swipedown)="onSwipeDown($event)"
```

A live stackblitz is avaiable at https://stackblitz.com/edit/swipe-gesture.

Tap

Wiring up the DOM Element within the Angular Component:

```
<div
    (tap)="onTap($event)">
 </div>
```

A live stackblitz is avaiable at https://stackblitz.com/edit/tap-gesture.

Configuring the Gestures

If you want to override any of the default settings for gestures within a module, you will need to provide a custom HammerGestureConfig class.

The `HammerGestureConfig` is configured like so:

```
import { HammerGestureConfig, HAMMER_GESTURE_CONFIG } f
export class MyHammerConfig extends HammerGestureConfig
  overrides = <any> {
    pan: { direction: Hammer.DIRECTION_All },
    swipe: { direction: Hammer.DIRECTION_VERTICAL },
```

The `MyHammerConfig` class defined above sets the direction for the pan and swipe gesture recognizers. The settings for each of the recognizers is defined within the hammerjs documentation.

Finally, you can add your gesture configuration to the module by adding the following provider:

```
{
    provide: HAMMER_GESTURE_CONFIG,
    useClass: MyHammerConfig
}
```

See something I missed? Reach out to me in the comments below or on Twitter.