Complex animation sequences

Prerequisites

A basic understanding of the following concepts:

- Introduction to Angular animations
- Transition and triggers

So far, we've learned simple animations of single HTML elements. Angular also lets you animate coordinated sequences, such as an entire grid or list of elements as they enter and leave a page. You can choose to run multiple animations in parallel, or run discrete animations sequentially, one following another.

Functions that control complex animation sequences are as follows:

- `query()` finds one or more inner HTML elements.
- `stagger()` applies a cascading delay to animations for multiple elements.
- `group()` runs multiple animation steps in parallel.
- `sequence()` runs animation steps one after another.

Animate multiple elements using query() and stagger() functions

The `query()` function allows you to find inner elements within the element that is being animated. This function targets specific HTML elements within a parent component and applies animations to each element individually. Angular intelligently handles setup, teardown, and cleanup as it coordinates the elements across the page.

The `stagger()` function allows you to define a timing gap between each queried item that is animated and thus animates elements with a delay between them.

The Filter/Stagger tab in the live example shows a list of heroes with an introductory sequence. The entire list of heroes cascades in, with a slight delay from top to bottom.

The following example demonstrates how to use `query()` and `stagger()` functions on the entry of an animated element.

- Use `query()` to look for an element entering the page that meets certain criteria.
- For each of these elements, use `style()` to set the same initial style for the element. Make it invisible and use `transform` to move it out of position so that it can slide into place.
- Use `stagger()` to delay each animation by 30 milliseconds.
- Animate each element on screen for 0.5 seconds using a custom-defined easing curve, simultaneously fading it in and un-transforming it.

src/app/hero-list-page.component.ts

```
animations: [
    trigger('pageAnimations', [
      transition(':enter', [
        query('.hero, form', [
          style({opacity: 0, transform:
'translateY(-100px)'}),
          stagger(-30, [
            animate('500ms cubic-
bezier(0.35, 0, 0.25, 1)', style({
opacity: 1, transform: 'none' }))
          ])
        ])
      ])
    ]),
  ]
})
export class HeroListPageComponent
implements OnInit {
  @HostBinding('@pageAnimations')
  public animatePage = true;

  _heroes = [];
```

```
  heroTotal = -1;
  get heroes() {
    return this._heroes;
  }

  ngOnInit() {
    this._heroes = HEROES;
  }

  updateCriteria(criteria: string) {
    criteria = criteria ?
criteria.trim() : '';

    this._heroes = HEROES.filter(hero
=>
hero.name.toLowerCase().includes(criteri

    const newTotal =
this.heroes.length;

    if (this.heroTotal !== newTotal) {
      this.heroTotal = newTotal;
    } else if (!criteria) {
      this.heroTotal = -1;
```

```
      }
    }
  }
```

Parallel animation using group() function

You've seen how to add a delay between each successive animation. But you may also want to configure animations that happen in parallel. For example, you may want to animate two CSS properties of the same element but use a different `easing` function for each one. For this, you can use the animation `group()` function.

> **Note:** The `group()` function is used to group animation *steps*, rather than animated elements.

In the following example, using groups on both `:enter` and `:leave` allow for two different timing configurations. They're applied to the same element in parallel, but run independently.

src/app/hero-list-groups.component.ts (excerpt)

```
animations: [
  trigger('flyInOut', [
    state('in', style({
      width: 120,
      transform: 'translateX(0)',
opacity: 1
    })),
    transition('void => *', [
      style({ width: 10, transform:
'translateX(50px)', opacity: 0 }),
      group([
        animate('0.3s 0.1s ease',
style({
          transform: 'translateX(0)',
          width: 120
        })),
        animate('0.3s ease', style({
          opacity: 1
        }))
      ])
    ]),
```

```
      transition('* => void', [
        group([
          animate('0.3s ease', style({
            transform:
  'translateX(50px)',
            width: 10
          })),
          animate('0.3s 0.2s ease',
  style({
            opacity: 0
          }))
        ])
      ])
    ])
  ]
```

Sequential vs. parallel animations

Complex animations can have many things happening at once. But what if you want to create an

animation involving several animations happening one after the other? Earlier we used `group()` to run multiple animations all at the same time, in parallel.

A second function called `sequence()` lets you run those same animations one after the other. Within `sequence()`, the animation steps consist of either `style()` or `animate()` function calls.

- Use `style()` to apply the provided styling data immediately.
- Use `animate()` to apply styling data over a given time interval.

Filter animation example

Let's take a look at another animation on the live example page. Under the Filter/Stagger tab, enter some text into the **Search Heroes** text box, such as `Magnet` or `tornado`.

The filter works in real time as you type. Elements leave the page as you type each new letter and the

filter gets progressively stricter. The heroes list gradually re-enters the page as you delete each letter in the filter box.

The HTML template contains a trigger called `filterAnimation`.

src/app/hero-list-page.component.html

```
<ul class="heroes"
[@filterAnimation]="heroTotal">
</ul>
```

The component file contains three transitions.

src/app/hero-list-page.component.ts

```
@Component({
  animations: [
    trigger('filterAnimation', [
      transition(':enter, * => 0, * =>
-1', []),
      transition(':increment', [
        query(':enter', [
          style({ opacity: 0, width:
'0px' }),
          stagger(50, [
            animate('300ms ease-out',
style({ opacity: 1, width: '*' })),
          ]),
        ], { optional: true })
      ]),
      transition(':decrement', [
        query(':leave', [
          stagger(50, [
            animate('300ms ease-out',
style({ opacity: 0, width: '0px' })),
          ]),
        ])
```

```
        ]),
      ]),
    ]
})
export class HeroListPageComponent
implements OnInit {
  heroTotal = -1;
}
```

The animation does the following:

- Ignores any animations that are performed when the user first opens or navigates to this page. The filter narrows what is already there, so it assumes that any HTML elements to be animated already exist in the DOM.
- Performs a filter match for matches.

For each match:

- Hides the element by making it completely transparent and infinitely narrow, by setting its opacity and width to 0.
- Animates in the element over 300 milliseconds. During the animation, the element assumes its default width and opacity.
- If there are multiple matching elements, staggers in each element starting at the top of the page, with a 50-millisecond delay between each element.

Animation sequence summary

Angular functions for animating multiple elements start with `query()` to find inner elements, for example gathering all images within a `<div>`. The remaining functions, `stagger()`, `group()`, and `sequence()`, apply cascades or allow you to control how multiple animation steps are applied.

More on Angular animations

You may also be interested in the following:

- Introduction to Angular animations
- Transition and triggers
- Reusable animations
- Route transition animations