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.

The functions that control complex animation sequences are:

- `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.

{@a complex-sequence}

The query() function

Most complex animations rely on the `query()` function to find child elements and apply animations to them, basic examples of such are:

- `query()` followed by `animate()` used to query simple HTML elements and directly apply animations to them.
- `query()` followed by `animateChild()` used to query child elements which themselves have animations metadata applied to them and trigger such animation (which would be otherwise be blocked by the current/parent element's animation)

The first argument of `query()` is a css selector string which can also contain the following angular-specific tokens:

- `:enter` / `:leave` for entering/leaving elements
- `:animating` for elements currently animating
- `@*` / `@triggerName` for elements with any or a specific trigger
- `:self` the animating element itself

Entering and Leaving Elements

Not all child elements are actually considered as entering/leaving, this can at times be counterintuitive and confusing, please see the query api docs for more information.

You can also see an illustration of this in the animations live example (introduced in the animations introduction section) under the Querying tab.

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

After having queried child elements via `query()`, the `stagger()` function lets you define a timing gap between each queried item that is animated and thus animates elements with a delay between them.

The following example demonstrates how to use the `query()` and `stagger()` functions to animate a list (of heroes) adding each in sequence, with a slight delay, from top to bottom.

- 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 transparent 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.

Parallel animation using group() function

You've seen how to add a delay between each successive animation. But you might also want to configure animations that happen in parallel. For example, you might 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.

The following example, uses `group()` s on both `:enter` and `:leave` for two different timing configurations, thus applying two independent animations to the same element in parallel.

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 you 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

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` .

The `filterAnimation` in the component's decorator contains three transitions.

The code in this example performs the following tasks:

- Skips animations when the user first opens or navigates to this page (the filter animation narrows what is already there, so it only works on elements that already exist in the DOM).
- Filters heroes based on the search input's value.

For each change:

- Hides an element leaving the DOM by setting its opacity and width to 0.
- Animates an element entering the DOM over 300 milliseconds. During the animation, the element assumes its default width and opacity.
- If there are multiple elements entering or leaving the DOM, staggers each animation starting at the top of the page, with a 50-millisecond delay between each element.

Animating the items of a reordering list

Although Angular animates correctly `*ngFor` list items out of the box, it will not be able to do so if their ordering changes. This is because it will lose track of which element is which, resulting in broken animations. The only way to help Angular keep track of such elements is by assigning a `TrackByFunction` to the `NgForOf` directive. This makes sure that Angular always knows which element is which, thus allowing it to apply the correct animations to the correct elements all the time.

Rule of Thumb: If you need to animate the items of an `*ngFor` list and there is a possibility that the order of such items will change during runtime, always use a `TrackByFunction` .

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 lets you control how multiple animation steps are applied.

More on Angular animations

You might also be interested in the following:

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