RAF-based promises & $animateRunner proposal

In ngAnimate 1.4 (the refactor code) there are two issues that need to be solved:

* We need to have all “callback” code be handled using rAF passes. This will ensure that no flickers happen and all animation sequencing code is handled at the right level.
* A runner object should be returned for each animation that is triggered using $animate or $animateCss. This would allow for animations to be paused/resumed.

# The current state of Matias’ giant refactored animation code

A service called **$$qRaf** exists which patches $q to use RAF to resolve itself. This works the same way as does $$q (with its own rAF tick), but in addition it also does something else to avoid the flicker problem.

When $$qRaf is resolved then it will check to see if **at least one RAF has passed** since the **time it was first created**. This ensures that **no extra RAF is fired when the promise is resolved**. So if no animation is found then there is always **one RAF** and if an animation does render then there will be no extra RAF upon resolution.

While this solves some issues in ngAnimate it causes some async/sync issues in code and the code itself cannot be made public to users since it can cause some larger issues with Angular apps. One issue for example is.

/\*\* normal behaviour \*\*/

var deferred = $$qRaf.defer();

deferred.resolve();

cleanup();

return deferred.runner;

// this results in cleanup being returned

// first and then the callbacks being fired

/\*\* unexpected behaviour \*\*/

var deferred = $$qRaf.defer();

someAnimation(function() {

deferred.resolve();

cleanup();

});

return deferred.runner;

// this results in the callbacks being

// fired first and then the cleanup

# The existing $animateRunner service

In the refactored code there is a runner service that extends an instance of a promise and adds in various methods.

var promise = $$qRaf.defer();

var runner = $animateRunner(promise);

runner.pause();

runner.end();

runner.resume();

runner.cancel();

This works in the same way that $httpPromise does and it adds issues such as promise cancellation and chaining issues (where .success() and .error() are not apart of the chain).

Perhaps we could merge the two together ($$qRaf + $animateRunner).

# The updated $animateRunner service

There is no need to bunch up the animation code into promises via a custom abstraction like $$qRaf. Instead what we can do is make a lightweight promise that works with all animation services ($animate, $animateCss and eventually $timeline) that internally manages itself with this RAF checking mechanism, but it also doesn’t consume promises in the way that $q does so it won’t cause the same kinds of problems that mixing $$qRaf and $q would.

Perhaps something like this:

var runner = $animateRunner(element);

runner.complete(pass, fail);

runner.end(); // this acts like defer.resolve();

But then since this abstraction doesn’t follow how promises work we can add the other runner-specific methods:

runner.complete();

runner.end();

runner.pause();

runner.resume();

runner.cancel(); // defer.reject()