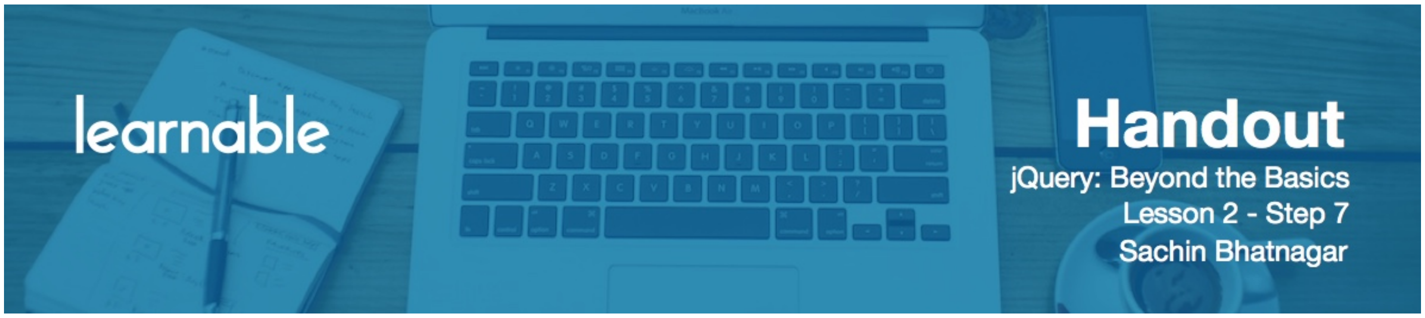

Introduction to jQuery animation queue

At this step we are going to discuss **animation queuing**. If we pipe in multiple effects on the same element, such as `slideDown`, `slideUp`, `animate` etc, these effects will queue up in the **fx queue** and be executed in a sequential manner.

Custom functions in animate queue:

You can employ the `queue` method to queue custom functions. However it won't be executed automatically - you rather have to specifically call the `dequeue` method. This lets you control the execution of the queue. Think of the `dequeue` method as a sort of a play button for the animation sequence.

Working with notification script demo

Let's build a small notification script which reads an array of messages and displays the messages in a slide out panel. This pane should slide out, show the first message, slide back in, and then slide out again with the next message. Once all messages in the array are displayed the script stops and clears out the array.

Go ahead and open *popnotify_Begin.htm* from the accompanying files.

You'll notice a tiny orange colored pane on the top left corner of the screen - this is actually a `div` with the class name of `messageBox` and it also contains an `h2` tag where the actual message text would be placed. Its `left` property is set to `-370px`, which takes the `div` off screen with only a tiny amount visible within the page. We want this `div` to animate back and forth between 0 pixels and -370 pixels on the `left`.

Begin by creating a function called `popNotify` and a variable `messageCounter`:

```
var messageCounter = 0;
function popNotify() {};
```

Next up, create two more variables to cache the selectors for the `messageBox` `div` and the `h2` tag inside the `messageBox`:

```
var messageBox = $('.messageBox');
var h2Tag = $('.messageBox > h2');
```

Code the `popNotify` function:

```
$(function() {
  var messages = [
    "Hello Everyone !",
    "I'm here to help you !",
    "You can use me to notify the user",
    "I use the jQuery queue method !"];
  var messageCounter = 0;
  var messageBox = $('.messageBox');
  var h2Tag = $('.messageBox > h2');
  (function popNotify() {
    h2Tag.text(messages[messageCounter]);
    messageBox
        .animate({
          left:0
        }, 500)
        .delay(3000)
        .animate({
          left:-370
        }, 500)
        .delay(200)
        .queue(function() {
          if(messageCounter === messages.length - 1) {
// Reset and Stop
            messages = [];
            messageCounter = 0;
          } else {
// Increment Counter and Keep going
            messageCounter ++;
            popNotify();
          }
          $(this).dequeue();
        });
  })();
});
```

To animate the `left` property I've written `0`, not `0px` - this way of providing values is equally acceptable since the `animate` method only animates CSS properties whose values are inherently numeric.

The `queue` allows us to insert custom functionality within the fx animation queue. Inside this function we check if the `messageCounter` is equal to the total length of the messages minus 1. If yes - clear the contents of the array, reset our counter, and stop. Otherwise, increment the `messageCounter` variable and keep repeating this animation queue until we hit the end of the array. By using the `dequeue` method, this custom functionality gets executed right away.

We want to run the `popNotify` function when the page loads, so it is turned into an immediately invoked function.

Go back to the browser and check the result.

Introduction to custom queues

I mentioned earlier that when you sequence multiple animation methods together, they go into the default fx queue. However you can also create your own animation queues and play them as needed.

Open up *timeSphere_Begin.htm* file.

There is a `div` that should expand and morph into a shape of a circle and reveal a logo when you click on the "Open" button. When you click on the "Close" button, it should shrink back to its original shape as a thin vertical line.

We will create two custom queues. The first one - `openQ` - would consist of a series of animated steps that morph a thin vertical shaped `div` into a circle, whereas the `closeQ` would be used to quickly morph the `div` back to its original thin form.

In the `openQ` we want to execute four separate animation effects simultaneously, at the same time. Each of these effects have different animation durations as well.

In the `closeQ` queue we will have multiple animation steps that are simultaneously executed as in the `openQ`.

Working with custom queues demo

Begin by caching the selector for the `logoBox`:

```
var logoBox = $('.logoBox');
```

Now build the first animation queue:

```
function loadQ() {
  logoBox
      .queue("openQ", function (next) {
        $(this).animate({
          width: '400px'
        }, {
          duration: 500,
          queue: false
        });
        next();
      })
      .queue("openQ", function (next) {
        $(this).animate({
          opacity: 1,
          height: '400px'
        }, {
          duration: 900,
          queue: false
        });
        next();
      })
      .queue("openQ", function (next) {
        $(this).animate({
          'border-radius': '200px'
        }, {
          duration: 1500,
          queue: false
        });
        next();
      })
}
```

Setting the property `queue` to `false` prevents the `animate` method from queuing up by default in the fx queue since we want this to execute simultaneously with other animate methods that we'll queue up in a moment.

We have to call the `next` method to dequeue animation method and proceed to the next method in the chain.

Now to fade in the logo image:

```
var logo = $('#logo');

function loadQ() {
  logoBox
      .queue("openQ", function (next) {
        $(this).animate({
          width: '400px'
        }, {
          duration: 500,
          queue: false
        });
        next();
      })
      .queue("openQ", function (next) {
        $(this).animate({
          opacity: 1,
          height: '400px'
        }, {
          duration: 900,
          queue: false
        });
        next();
      })
      .queue("openQ", function (next) {
        $(this).animate({
          'border-radius': '200px'
        }, {
          duration: 1500,
          queue: false
        });
        next();
      })
      .queue("openQ", function (next) {
        logo.delay(1000).fadeIn(400);
        next();
      })
}
```

To run this as a timeline, call `dequeue` in the click event handler:

```
$('#openLogo').on("click", function() {
  $('.logoBox').dequeue("openQ");
});
```

Now lets build the `closeQ` queue:

```
logoBox.queue("closeQ", function(next) {
  logo.fadeOut(400);
  next();
})
    .delay(400,"closeQ")
    .queue("closeQ", function(next) {
      $(this).animate({
        'border-radius':'0px',
        opacity:0.4
      },{
        duration:400,
        queue:false
      });
      next();
    })
    .queue("closeQ", function(next) {
      $(this).animate({
        width:'2px',
        height:'100px'
      },{
        duration:500,
        queue:false
      });
      next();
    });
```

Return to the browser, click "Open" and then "Close" to fire animations. This will work, however clicking on the buttons again makes no effect. This is because the animation timeline has played out. To rewind our animation to the first frame, we should put the entire thing inside a function called `loadQ` and add one last `queue` method to call the `loadQ` function once the entire `closeQ` timeline plays out:

```
(function loadQ() {
  logoBox
      .queue("openQ", function(next) {
        $(this).animate({
          width:'400px'
        },{
          duration:500,
          queue:false
        });
        next();
      })
      .queue("openQ", function(next) {
        $(this).animate({
          opacity:1,
          height:'400px'
        },{
          duration:900,
          queue:false
        });
        next();
      })
      .queue("openQ", function(next) {
        $(this).animate({
          'border-radius':'200px'
        },{
          duration:1500,
          queue:false
        });
        next();
      })
      .queue("openQ", function(next) {
        logo.delay(1000).fadeIn(400);
        next();
      })
      .queue("closeQ", function(next) {
        logo.fadeOut(400);
        next();
      })
      .delay(400,"closeQ")
      .queue("closeQ", function(next) {
        $(this).animate({
          'border-radius':'0px',
          opacity:0.4
        },{
          duration:400,
          queue:false
        });
        next();
      })
      .queue("closeQ", function(next) {
        $(this).animate({
          width:'2px',
          height:'100px'
        },{
          duration:500,
          queue:false
        });
        next();
      })
      .queue("closeQ", function() {
        loadQ();
        $(this).dequeue();
      });
})();
```

As you see I've also turned our `loadQ` function into an immediately invoked function.

Go back to the browser and try that out!