深入理解jQuery（6）——Queue

Alex Sun
2014-12-28

在jQuery中，同Deferred，Data一样，队列同样是非常重要的一个工具，是实现动画的一个关键。下面进行详细分析。

1. 源码结构

```
jQuery.extend({
    queue: function() {},
    dequeue: function() {},
    _queueHooks: function() {}
});

jQuery.fn.extend({
    queue: function() {},
    dequeue: function() {},
    delay: function() {},
    clearQueue: function() {},
    promise: function() {}
});
```

2. 基本使用

事实上，一个Queue即为一个数组，数组中的每个元素都是一个函数。queue操作相当于数组的push操作，是向数组中添加一个函数；dequeue操作相当于数组的shift操作，将数组的第一项取出，不过dequeue操作同时还会执行该函数。可以使用工具方法或者原型方法来管理对象的队列，下面通过例子来查看：

```
function a() {
    console.log("a");
}

function b() {
    console.log("b");
```

```
}

var test = $("#test");
$.queue(test, "q1", a);
$.queue(test, "q1", b);
console.log($.queue(test, "q1")); //[function, function]
$.dequeue(test, "q1"); //a
$.dequeue(test, "q1"); //b
```

其中q1相当于是队列的名字，向队列中添加了两个函数后，依次执行出队操作，就会依次执行函数。

同样的效果可以通过原型方法来实现，如下：

```
var test = $("#test");
test.queue("q1", a);
test.queue("q1", a);
console.log(test.queue("q1")); //[function, function]
test.dequeue("q1"); //a
test.dequeue("q1"); //b
```

不过这两种方法存在着细微差别，看如下例子：

```
var test1 = $("#test");
var test2 = $("#test");

test1.queue("q1", a);
test2.queue("q1", b);
console.log(test1.queue("q1")); //[function, function]

$.queue(test1, "q1", a);
$.queue(test2, "q1", b);
console.log($.queue(test1, "q1")); //[function]
```

之所以有这样的差异，是因为在使用原型方法的时候，队列是加在DOM对象上的；而使用工具方法的时候，队列是加在第一个参数对象上的。在这里，test1和test2选择的是同一个DOM对象，但是它们两个却是不同的jQuery对象。事实上，Data（数据缓存）在使用的时候也有类似的差异。

3. 源码分析

（1）jQuery.queue()源码分析

```
queue: function(elem, type, data) {
    var queue;

    if (elem) {
        // 默认队列名称是fx
        type = (type || "fx") + "queue";
        // 队列的实现依赖于数据缓存
        queue = data_priv.get(elem, type);

        // 如果没有data，则直接返回queue，否则进行入队操作
        // Speed up dequeue by getting out quickly if this is just a
lookup
        if (data) {
            // 如果队列不存在，则创建队列并以数组形式存储data；
            // 如果data本身就是数组，则会覆盖掉之前的队列内容
            // 如果队列存在并且data不为数组，则进行push操作
            if (!queue || jQuery.isArray(data)) {
                queue = data_priv.access(elem, type,
jQuery.makeArray(data));
            } else {
                queue.push(data);
            }
        }
        return queue || [];
    }
}
```

（2）jQuery._queueHooks()源码分析

```
// not intended for public consumption - generates a queueHooks object,
or returns the current one
_queueHooks: function(elem, type) {
    var key = type + "queueHooks";
    // 如果存在key为(type + "queueHooks")的数据缓存，则直接返回
    // 否则创建该数据缓存，其值为一个对象，该对象的empty属性是一个Callbacks对象
    return data_priv.get(elem, key) || data_priv.access(elem, key, {
        empty: jQuery.Callbacks("once memory").add(function() {
            data_priv.remove(elem, [type + "queue", key]);
        })
    });
}
```

_queueHooks()是一个私有函数，起作用是清理数据缓存。因为队列的实现是基于数据缓存的，当出队操作全部完成后，数据缓存中还存在着该队列，只不过队列中没有元素罢了，此时需要清理

数据缓存，将该空队列移除掉。

（3）jQuery.fn.queue()源码分析

```
queue: function(type, data) {
    var setter = 2;

    // 第一个参数不为字符串的时候，使用默认队列
    if (typeof type !== "string") {
        data = type;
        type = "fx";
        setter--;
    }

    // queue("q1", data), setter=2, arguments.length=2
    // queue("q1"), setter=2, arguments.length=1
    // queue(data), setter=1, arguments.length=1
    // queue(), setter=1, arguments.length=0
    // 当arguments.length < setter的时候，说明是要获取队列，而不是入队
    if (arguments.length < setter) {
        // 如果jQuery对象中有多个DOM对象，则只返回第一个DOM对象的队列
        return jQuery.queue(this[0], type);
    }

    // 入队操作
    return data === undefined ?
        this :
        this.each(function() {
            // 对每个DOM对象进行入队操作
            var queue = jQuery.queue(this, type, data);

            // ensure a hooks for this queue
            // 添加hooks
            jQuery._queueHooks(this, type);

            // 如果为默认队列，并且队列中第一个元素不是inprogress字符串，则立即进行出队操作
            if (type === "fx" && queue[0] !== "inprogress") {
                jQuery.dequeue(this, type);
            }
        });
}
```

在最后为什么要立即进行出队操作呢？看如下例子：

```
var test = $("#test");
```

```
test.click(function() {
    test.animate({
        width: "300px"
    }, 2000).animate({
        height: "300px"
    }, 2000);
});
```

由于动画的实现是基于队列的，因此在调用animate方法的时候，事实上是进行了入队操作，但是我们并没有显式进行出队操作。事实上，也就是源码中inprogress的判断，在进行完入队操作后，如果队列中第一项不是inprogress字符串，则立即进行出队操作。

（4）jQuery.fn.delay()源码分析

```
// Based off of the plugin by Clint Helfers, with permission.
// http://blindsignals.com/index.php/2009/07/jquery-delay/
delay: function(time, type) {
    // 得到延迟时间和队列名称
    time = jQuery.fx ? jQuery.fx.speeds[time] || time : time;
    type = type || "fx";

    // 向队列中添加一个函数，next为该函数的回调函数
    return this.queue(type, function(next, hooks) {
        // 在time时间过后，执行该回调函数
        var timeout = setTimeout(next, time);
        hooks.stop = function() {
            clearTimeout(timeout);
        };
    });
}
```

（5）jQuery.dequeue()源码分析

```
dequeue: function(elem, type) {
    // 默认队列名称fx
    type = type || "fx";

    // 得到队列，队列长度，队列的第一个元素
    // hooks(用来清理数据缓存)
    // next用于进行下一个出队操作
    var queue = jQuery.queue(elem, type),
        startLength = queue.length,
        fn = queue.shift(),
```

```
            hooks = jQuery._queueHooks(elem, type),
            next = function() {
                jQuery.dequeue(elem, type);
            };

        // If the fx queue is dequeued, always remove the progress sentinel
        // 如果fn为inprogress字符串，则继续出队
        if (fn === "inprogress") {
            fn = queue.shift();
            startLength--;
        }

        if (fn) {

            // Add a progress sentinel to prevent the fx queue from being
            // automatically dequeued
            // 将inprogress字符串放在队列第一项
            if (type === "fx") {
                queue.unshift("inprogress");
            }

            // clear up the last queue stop function
            delete hooks.stop;
            // 执行出队函数
            fn.call(elem, next, hooks);
        }

        // 如果队列为空，则清理数据缓存
        if (!startLength && hooks) {
            hooks.empty.fire();
        }
    }
```

（6）jQuery.fn.promise()

该方法是当jQuery对象的所有队列方法执行完后，再执行回调。具体代码不做赘述。例子如下：

```
var test = $("#test");

test.click(function() {
    test.animate({
        width: "300px"
    }, 2000).animate({
        height: "300px"
    }, 2000);
```

```
    test.promise().done(function() {
        console.log("done");
    });
});
```