深入理解jQuery（5）——Data

Alex Sun
2014-11-15

1. 引入

有些时候，我们需要给DOM节点绑定一些数据，考虑如下例子：页面上有若干个缩略图，要求点击某个缩略图之后弹出浮层，显示相应的大图。假如每个缩略图都是一个<img>元素的话，其src属性值必然是缩略图地址，所以我们必须存储相应的大图地址，此时我们可以在每个<img>标签上添加一个属性，例如data-bigimg，如下所示：

```
<img alt="image" src="image/smallImage.png" data-
bigimg="image/bigImage.png" />
```

这样往往是比较有效的，但是也存在着一些问题：

1. 如果数据与展示无关，那么就违背了model和view分离的原则，DOM不友好；
2. 如果数据量比较大，会使得DOM结构复杂化，并且每次访问数据的时候都需要查找DOM，比较麻烦；
3. 属性值只能是字符串，无法绑定一个对象。

但是，以上问题也是可以解决的，我们知道，每个DOM节点本身就是一个对象，我们可以为对象添加属性（property）。这里需要区别开来attribute和property，一般来说，attribute会在DOM结构中展现出来的，而property是不渲染出来的（当然，这种说法并不严谨，主要指的是用户自定义的属性，此处不做深究）。看如下例子：

```
var test = document.getElementById("test");
test.setAttribute("owner", {
    name: "alex",
    age: 21
});
console.log(test.getAttribute("owner")); // [object Object]

test.owner = {
    name: "alex",
    age: 21
};
```

```
console.log(test.owner); // Object {name: "alex", age: 21}
```

在jQuery中，对attribute和property的操作分别为attr()和prop()，通过上面例子，我们觉得，似乎问题都得到了解决，但其实不然，使用property还存在着严重的问题，例如循环引用。

所谓循环引用，看如下的例子：

```
var a = {};
var b = {};
a.someProp = b;
b.someProp = a;
```

这里a和b互相引用了对方，当脚本执行停止的时候，导致a和b所占用的内存无法得到回收，有可能造成内存泄露。考虑上面的property的例子，也是有可能出现循环引用的，例如：

```
var test = document.getElementById("test");
test.owner = {
    name: test,
    age: 21
};
```

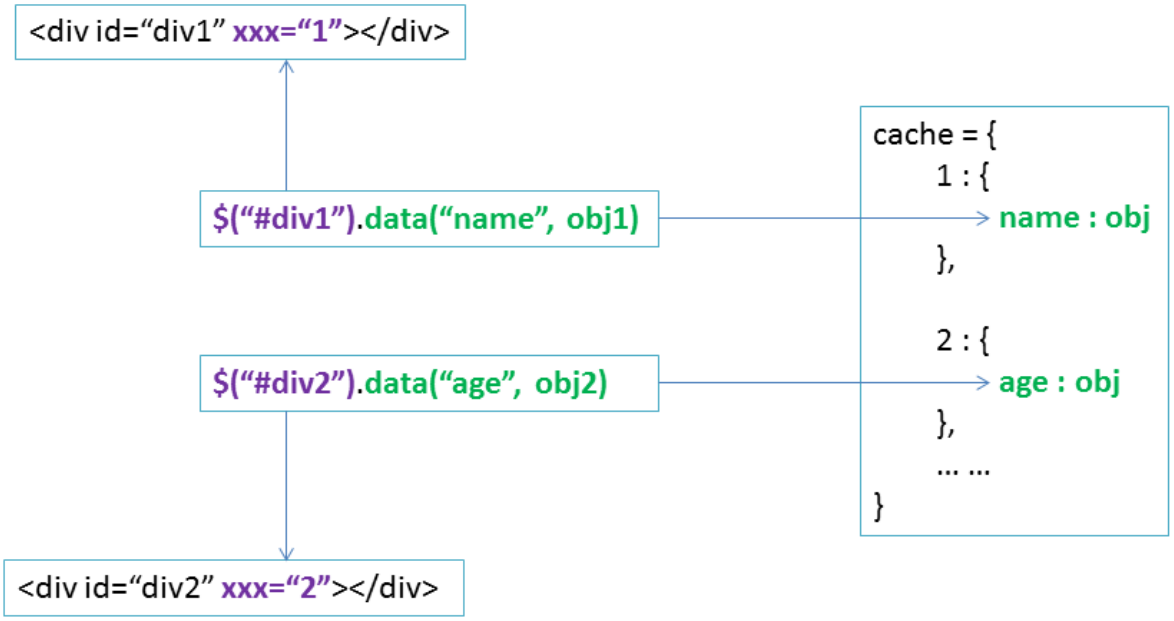
在jQuery中，解决方案是数据缓存，即Data，下面我们来进行分析。

2. Data原理

在jQuery中，使用data能有效地避免循环引用等一系列问题。首先来看data的用法：

```
var test = $("#test");
test.data("owner", {
    name: "alex",
    age: 21
});
console.log(test.data("owner")); //Object {name: "alex", age: 21}
```

其基本原理如图所示：



首先是一个全局的对象cache作为中介，在每个需要绑定数据的DOM节点上新增一个属性，这里我们使用的是xxx，属性值为唯一的ID，在每次需要绑定数据或者读取数据的时候，首先找到相应的DOM节点，然后得到其xxx属性值，然后用这个属性值去cache中查找该DOM节点上绑定的数据。这里可以看到，cache与DOM节点是没有任何关系的，所以也就不会产生生循环引用了。

3. 源码分析1

```
function Data() {
    // Support: Android < 4,
    // Old WebKit does not have Object.preventExtensions/freeze method,
    // return new empty object instead with no [[set]] accessor
    Object.defineProperty(this.cache = {}, 0, {
        get: function() {
            return {};
        }
    });

    this.expando = jQuery.expando + Math.random();
}
```

```
Data.uid = 1;

Data.accepts = function(owner) {
    // Accepts only:
    //  - Node
    //    - Node.ELEMENT_NODE
    //    - Node.DOCUMENT_NODE
    //  - Object
    //    - Any
    // 对document或元素节点以及对象有效
    return owner.nodeType ?
        owner.nodeType === 1 || owner.nodeType === 9 : true;
};
```

上面是Data相关的一部分代码，在Data的构造函数中，我们可以看到使用了Object.defineProperty方法。执行完后，我们初始化得到的Data对象如下所示：

```
var data = {
    cache: {
        0: {

        }
    }
};
```

在Object.defineProperty的第三个参数中，只有get，因此我们只能够访问cache[0]，而不能修改它。data对象有一个expando属性，该属性值为一个非常随机的字符串，因此可以看做是唯一的。事实上，该属性即为之前我们为DOM节点增加的xxx，例如data.expando=”jQuery203038306501042097807 0.22824297472834587”，则对于某个DOM节点dom，即有dom[“jQuery203038306501042097807 0.22824297472834587”]=x，其中x为一个唯一的ID。事实上，该ID即为Data.uid，Data.uid是一个递增的数字，每分配一次，就会增加1。

接下来是Data的原型，概括如下：

```
Data.prototype = {
    key(owner)
    set(owner, data, value)
    get(owner, key)
    access(owner, key, value)
    remove(owner, key)
    hasData(owner)
    discard(owner)
}
```

其中key()方法接受一个参数，该参数标明的是需要添加数据缓存的对象，首先使用Data.accepts()

方法判断该对象是否被接受，如果不被接受则返回0，然后判断该对象上是否已经有xxx（即Data.expando）属性，如果有，直接返回该属性值，否则为该对象添加xxx属性，属性值为Data.uid，之后Data.uid自增1，然后返回该对象的xxx属性值。其实，也就是说，key()方法接受一个对象，然后判断该对象是否有数据缓存的标识，如果有则直接返回，如果没有则为其创建一个并返回。具体代码分析在此不作赘述。

set()方法是为owner对象设置数据缓存，首先通过key()方法得到owner的缓存id，然后通过该id去cache中得到owner的数据缓存对象。如果data为字符串，则增加data=value属性；如果data为对象，则将data扩展到缓存上。

get()方法返回owner的缓存对象或其属性值。同样，首先通过key()方法得到owner的缓存id，然后通过id去cache中得到owner的所有数据。如果key没有声明，则返回owner的缓存对象，否则返回缓存对象的key属性值。

access()方法是对get()和set()的封装，类似于$().attr()的操作。

remove()方法删除owner的某个缓存数据，如果没有指定key，则清空owner的所有缓存数据，如果指定了key（key可以使一个字符串，也可以是一个字符串数组），则依次删除所指定的属性值。

hasData()判断owner是否有缓存数据，当从未设置缓存数据（null，undefined）或者缓存为空（{}）时，返回false，否则返回true。

discard()删除数据缓存中owner的缓存。即找到owner的缓存id，然后删除cache的该id属性。

4. 源码分析2

在jQuery中，创建了两个Data对象，代码如下：

```
// These may be used throughout the jQuery core codebase
data_user = new Data();
data_priv = new Data();
```

其中data_user是对外公开的，data_priv是内部私有的。

然后是对jQuery工具方法的扩展，源码如下。可以看到所有的操作都是调用Data的方法。

```
jQuery.extend({
    acceptData: Data.accepts,

    hasData: function( elem ) {
        return data_user.hasData( elem ) || data_priv.hasData( elem );
    },

    data: function( elem, name, data ) {
```

```
            return data_user.access( elem, name, data );
        },

        removeData: function( elem, name ) {
            data_user.remove( elem, name );
        },

        // TODO: Now that all calls to _data and _removeData have been
    replaced
        // with direct calls to data_priv methods, these can be deprecated.
        _data: function( elem, name, data ) {
            return data_priv.access( elem, name, data );
        },

        _removeData: function( elem, name ) {
            data_priv.remove( elem, name );
        }
    });
```

接下来是对jQuery实例方法的扩展，在此之前，先来了解一些HTML5中自定义数据的相关知识。在HTML5中，可以在标签中添加“data-”的数据，并且可以通过js访问到。例如：

```
    <!DOCTYPE html>
    <html>
    <head>
    <meta charset="utf-8">
    <title>Examples</title>
    </head>
    <body>
        <div id="div1" data-hello="hello world"></div>
        <div id="div2" data-size-x="5"></div>
        <script>
            var div1=document.getElementById("div1");
            console.log(div1.dataset.hello); // hello world
            console.log(div1.getAttribute("data-hello")); // hello world

            var div2=document.getElementById("div2");
            console.log(div2.dataset.sizeX); // 5
            console.log(div2.getAttribute("data-size-x")); // 5
        </script>
    </body>
    </html>
```

在jQuery中，调用实例方法data()访问数据的时候，不仅会返回缓存中的数据，也会返回标签中的自定义数据，例如上面的例子，可以这样访问：

```
console.log($("#div1").data("hello")); // hello world
console.log($("#div2").data("size-x")); // 5
console.log($("#div2").data("sizeX")); // 5
```

下面来看源码：

```
jQuery.fn.extend({
    data: function(key, value) {
        // 首先应当注意的是如果对多个元素设置数据，则会对每个元素设置数据
        // 如果对多个元素获取数据，则只会获取第一个元素的数据
        var attrs, name,
            elem = this[0],
            i = 0,
            data = null;

        // Gets all values
        // 如果没有指定key，则返回所有缓存数据
        if (key === undefined) {
            // 如果没有元素，则直接返回
            if (this.length) {
                // 得到第一个元素在缓存中的所有
                data = data_user.get(elem);

                // 如果elem是一个元素节点，则对于其每一个attribute，获取所有的自定义数据
                // 第一次访问时无hasDataAttrs，会进入if语句，之后hasDataAttrs为true，不会再进入if语句
                if (elem.nodeType === 1 && !data_priv.get(elem, "hasDataAttrs")) {
                    attrs = elem.attributes;
                    for (; i < attrs.length; i++) {
                        name = attrs[i].name;

                        if (name.indexOf("data-") === 0) {
                            name = jQuery.camelCase(name.slice(5));
                            // 将HTML5的data-字段加入数据缓存
                            dataAttr(elem, name, data[name]);
                        }
                    }
                    // hasDataAttrs放在私有区域，不会对外造成影响
                    data_priv.set(elem, "hasDataAttrs", true);
                }
            }

            return data;
        }
```

```
        // Sets multiple values
        // 当key为一个对象的时候，对每个元素设置缓存数据
        if (typeof key === "object") {
            return this.each(function() {
                data_user.set(this, key);
            });
        }

        return jQuery.access(this, function(value) {
            var data,
                camelKey = jQuery.camelCase(key);

            // The calling jQuery object (element matches) is not empty
            // (and therefore has an element appears at this[ 0 ]) and
the
            // `value` parameter was not undefined. An empty jQuery
object
            // will result in `undefined` for elem = this[ 0 ] which
will
            // throw an exception if an attempt to read a data cache is
made.
            // value为空，即操作位数据的查询
            if (elem && value === undefined) {
                // Attempt to get data from the cache
                // with the key as-is
                // 首先查找aaa-bbb形式的属性值，如果有则返回，否则继续
                data = data_user.get(elem, key);
                if (data !== undefined) {
                    return data;
                }

                // Attempt to get data from the cache
                // with the key camelized
                // 查找驼峰式的属性值，例如aaaBbb，如果有则返回，否则继续
                data = data_user.get(elem, camelKey);
                if (data !== undefined) {
                    return data;
                }

                // Attempt to "discover" the data in
                // HTML5 custom data-* attrs
                // 在HTML5的自定义数据汇总进行查找，有则返回，否则继续
                data = dataAttr(elem, camelKey, undefined);
                if (data !== undefined) {
                    return data;
                }

                // We tried really hard, but the data doesn't exist.
```

```
                return;
            }

            // Set the data...
            // 添加数据缓存
            this.each(function() {
                // First, attempt to store a copy or reference of any
                // data that might've been store with a camelCased key.
                var data = data_user.get(this, camelKey);

                // For HTML5 data-* attribute interop, we have to
                // store property names with dashes in a camelCase form.
                // This might not apply to all properties...*
                data_user.set(this, camelKey, value);

                // *... In the case of properties that might _actually_
                // have dashes, we need to also store a copy of that
                // unchanged property.
                if (key.indexOf("-") !== -1 && data !== undefined) {
                    data_user.set(this, key, value);
                }
            });
        }, null, value, arguments.length > 1, null, true);
    },

    // 删除数据缓存
    removeData: function(key) {
        return this.each(function() {
            data_user.remove(this, key);
        });
    }
});
```