jQuery框架

1.jQuery☞DOM操作

1.1 添加remove方法

- ☞ 删除每一个元素
- ☞ 实现思路: 遍历实例，把遍历到的每一个元素进行删除

```
   remove: function () {
       this.each(function (key,val) {

           //也可以这样书写
           //val.parentNode.removeChild(val);
```

```
           //this代表遍历到的所有值
           this.parentNode.removeChild(this);
       });
   }
```

1.2 添加appendTo方法

- ☞ 大概实现思路：

> 大概：遍历所有的元素，分别添加到selector中。
>
> 因为selector的类型太多，能否把它变成同一种类型处理呢？
>
> 可以，统一使用jQuery包装成实例即可。

- ☞ 具体实现思路：

> 1、定义一个数组，用来存储所有被添加的元素
>
> 2、遍历this中的所有元素，依次添加到 $selector 中的所有元素， 需要考虑遍历到元素，只能添加到$selector中的一个元素中，其余的元素添加的都是clone版本。 另外还需要考虑每次添加时，都需要先把要添加的元素存储到数组中。
>
> 3、最后通过jQuery(所有被添加元素组成的数组)包装成新实例返回。

```
/*
*function { appendTo } 把所有的元素添加到指定的selector中
*param { selector: DOM || 选择器 || jQuery实例 }
*return 所有被添加元素共同组成的新实例
**/
appendTo: function( selector ) {

    var result = [],
    $selector = jQuery( selector ),
    temp;

    // 遍历所有被添加的元素
    this.each( function() {
        var self = this;
        $selector.each( function( index ) {
            temp = index === 0? self: self.cloneNode( true );
                this.appendChild( temp );
                result.push( temp );
            } );
        } );
    return jQuery( result );
    }
```

1.3 添加append方法

```
/*
*function { append } 给所有的元素添加指定的内容
*param { context: DOM || jQuery实例 || 文本 }
```

```
    *return 给谁添加返回谁，说白了就是this
    **/
    append: function( context ) {

        // 如果是字符串，遍历每一个元素，把字符串累加进去
        if ( jQuery.isString( context ) ) {
            this.each( function() {
                this.innerHTML += context;
            });
        }
        // 否则借用appendTo把context添加到this中
        else {
            jQuery( context ).appendTo( this );
        }
        return this;
    }
```

1.4 添加prependTo方法

```
    /*
    *function { prependTo } 把所有的元素添加到指定的selector最前面
    *param { selector: DOM || 选择器 || jQuery实例 }
    *return 所有被添加元素共同组成的新实例
    **/
    prependTo: function( selector ) {
        var $selector = jQuery( selector ),
        result = [], temp;
        // 遍历所有被添加的元素
        this.each( function() {
            var self = this;
            // 遍历所有被添加元素的父节点
            $selector.each( function( index ) {
                // 只有给第一个父节点添加元素时，添加的是真实的，以后都是clone的
                temp = index === 0? self : self.cloneNode( true );
                this.insertBefore( temp, this.firstChild );
                result.push( temp );
            });
         });
        // 把所有被添加的元素包装成jQ对象返回
        return jQuery( result );
    }
```

1.5 添加prepend方法

```
    /*
    *function { prepent } 给所有的元素最前面添加指定的内容
    *param { context: DOM || jQuery实例 || 文本 }
    *return 给谁添加返回谁，说白了就是this
    **/
    prepend: function( context ) {
        // 如果是字符串，把它添加到所有元素的最前面
        if ( jQuery.isString( context ) ) {
            this.each( function() {
                this.innerHTML = context + this.innerHTML;
            } );
        }
        // 否则借用prependTo把context添加到this中
        else {
            $(context).prependTo( this );
        }
        return this;
    }
```

1.6 appendTo、append、prependTo、prepent之间的比较

- ☞ appendTo和prependTo有很大相同之处，

  > ☞ 不同点在于appendTo把自己添加到某元素的后面，
  >
  > ☞ prependTo把自己添加到某元素的最前面。

- ☞ append和prepent有很大相同之处，

> ☞ 不同点在于append给自己的后面添加元素，
>
> ☞ prepend给自己的最前面添加元素。

- ☞ To和不To的区别：

> ☞ 1、添加方向相反，
>
> ☞ 2、不To的方法，对于字符串会当做文本添加；
>
> ☞ 而To的方法，会把字符串当做选择器处理。
>
> ☞ 也就是说，append和prepent支持给元素添加文本，appendTo和prependTo不支持。

2.属性操作☞class

2.1 添加hasClass

- ☞ 判断元素中是否含有指定的class
- ☞ 实现思路：

> ☞ 遍历所有的元素，只要有一个元素存在指定的className，
>
> ☞ 那么就返回true，否则返回false

```
    hasClass: function( className ) {
        for ( var i = 0, len = this.length; i < len; i++ ) {
            if ( (' ' + this[i].className + ' ').indexOf( ' ' + className + '' ) > -1 ) {
                return true;
                }
            }

            return false;
        },

        // 判断元素中是否含有指定的class(第二种方式)
        _hasClass: function( className ) {

            // 用一个变量记录是否存在
            var has = false;

            // 遍历所有的元素
            this.each( function() {
                if ( (' ' + this.className + ' ').indexOf( ' ' + className + '' ) > -1 ) {
                    has = true; // 如果有一个元素存在，就把这个变量改为true
                    return false; // 如果发现有一个元素存在，后面的就不用再遍历了，所以返回false
                }
            } );

            return has;
        },
```

2.2 添加addClass

- ☞ 给所有的元素添加指定的className
- ☞ 实现思路：

> ☞ 遍历所有的元素，先看这个元素有没有指定的className，
>
> ☞ 如果没有就添加，已经有了就无视。

```
    addClass: function (className) {
        // 遍历所有的元素
        for (var i = 0, len = this.length; i < len; i++) {
            // 如果这个元素不存在className，那么拼接上
            if ((' ' + this[i].className + ' ').indexOf(' ' + className + ' ') == -1) {
                //this[i].className += ' ' + className;
                this[i].className = (this[i].className + ' ' + className).replace(/^\s*|\s*$/g, '');
            }
        }

        // 为了链式编程
        return this;
    },
```

```
// 给所有的元素添加指定的className(第二种方式)
_addClass: function (className) {
    this.each(function () {
        // 这里面的this代表的是val值
        // 如果遍历到的每一个元素，不存在className，那么就进行添加操作
        if (!jQuery(this).hasClass(className)) {
            //this.className += ' ' + className;
            this.className = (this.className + ' ' + className).replace(/^\s*|\s*$/g, '');
        }
    });

    // 为了链式编程
    return this;
}
```

2.3 添加removeClass

- ☞ 删除所有元素中指定的className
- ☞ 实现思路：

> ☞ 1、如果className === undefined，那么删除所有元素的所有className ☞ 2、如果传参了，删除所有元素指定的className。

```
removeClass: function (className) {
    // 如果没有传参，清除全部className
    if (className === undefined) {
        this.each(function () {
            this.className = '';
        });
    } else {
        // 遍历所有的元素
        this.each(function () {
            // 利用正则替换匹配到的className，最后trim一下。
            this.className = (" " + this.className + " ")
                    .replace(' ' + className + ' ', " ")
                    .replace(/^\s*|\s*$/g, '');
        });
    }

    // 为了链式编程
    return this;
}
```

2.4 添加toggleClass

- ☞ 有就删除，没有就添加

```
toggleClass: function (className) {
    this.each(function () {
        var $this = jQuery(this);
        if ($this.hasClass(className)) {
            $this.removeClass(className);
        } else {
            $this.addClass(className);
        }
    });

    // 为了链式编程
    return this;
}
```