jQuery框架

1. 解决问题

1.1 为什么要把函数传入jQuery中

- 原因:在dom元素之前，通过js是无法获取这些dom元素的,常用的解决方案是，监听load事件.但是load通常比较慢，后来有了DOMContentLoaded事件，大家都监听这个事件。 console.log(document.getElementsByTagName('span')[0]);

```
document.addEventListener('DOMContentLoaded', function () {
    console.log(document.getElementsByTagName('div')[0]);
});
```

1.2 异步监听load事件不触发

```
var isLoad = false;
window.onload = function () {
    isLoad = true;
    console.log(000);
};

setTimeout(function () {
    if ( isLoad ) {
        console.log(111);
    }else {
        // 事件句柄一定是在事件被触发的时候才会执行
        window.onload = function () {
            console.log(111);
        };
    }
}, 3000);
```

1.3 IE8中apply无法平铺自定义的伪数组对象

```
var obj = {};
var arr = ['aa','bb','cc'];
var likeArray = { 0: '111', 1: '222', length: 2 };
var scripts = document.getElementsByTagName('script');
[].push.apply( obj, arr );
[].push.apply( obj, scripts );
[].push.apply( obj, likeArray );
console.log( obj );
```

1.4 错误捕获

- 错误捕获语句
- try { 要捕获的代码 }catch(e){ 对错误进行处理 }finally{ 无论错误是否发送，一定执行代码 }

```
try {
    console.log(b);
}catch (e){
    // 如果发生错误，该如何解决
    console.log(e);
}finally {
    console.log( '无论错误是否发送，我一定执行' );
}
console.log(222);
```

2. jQuery原型上的核心方法

- jQuery原型中的核心成员：

> 属性:
>
> 1、jquery属性，存储jQuery当前的版本

2、constructor属性，存储jQuery工厂函数

3、selector属性，jQuery实例的一个标识，也代表默认的selecotr

4、length属性，jQuery实例默认的length

```
    // 属性
    console.log($().jquery);
    console.log($().constructor);
    console.log($().selector);
    console.log($().length);
```

方法:

5、toArray方法，把实例转换为数组返回

```
console.log($('script').toArray());
```

6、get方法，获取指定下标的元素，支持负数倒着取，如果传null或undefined则把实例转换为数组返回

```
console.log($('span').get(0)); // 得到原生DOM对象
console.log($('span').get(-1)); // 得到原生DOM对象
console.log($('span').get()); // 得到原生DOM对象组成的数组
```

7、**each方法**，遍历实例所有的元素，把遍历到的数据分别传给回调

8、**map方法**，遍历实例所有的元素，把遍历到的数据分别传给回调，然后把回调的返回值组成一个数组返回

9、slice方法，截取实例中部分元素，重新构成新的实例返回

```
console.log($('span').slice(1, 2)); // 得到一个新的jQuery实例
```

10、eq方法，获取指定下标的元素，支持负数倒着取，最后把获取到的元素封装成新的实例返回

```
console.log($('span').eq(0)); // 得到一个新的jQuery实例，包含第一个DOM对象
console.log($('span').eq(-1)); // 得到一个新的jQuery实例，包含最后一个DOM对象
console.log($('span').eq()); // 得到一个新的jQuery空实例
```

11、first方法，获取第一个元素，把获取到的元素封装成新的实例返回

```
console.log($('span').first()); // 得到一个新的jQuery实例，包含第一个DOM对象
```

12、last方法，获取最后一个元素，把获取到的元素封装成新的实例返回

```
console.log($('span').last()); // 得到一个新的jQuery实例，包含最后一个DOM对象
```

13、push方法，给实例添加新元素

```
var $span = $('span');
$span.push( $('script').get(0) )
console.log( $span );  // 把scripeDOM添加到了$span这个实例中
```

14、sort方法，给实例进行排序

```
var $span2 = $('span');
$span2.sort(function ( span1, span2 ) {
    return span1.innerHTML < span2.innerHTML;
});
console.log($span2); // 排序后的实例
```

> 15、splice方法，从指定下标删除指定数量的元素，或者从指定下标替换指定数量的元素。

```
var $span3 = $('span');
$span3.splice(1,3,'a','b');
console.log($span3); // 删除替换后的实例
```

3. jQuery中的each方法

- jQuery的each方法可以用来遍历数组和对象，并且会把遍历到的下标、值、原数据返回给回调函数，供其使用。

```
var arr = ['aaaa', 'bbbbbb', 'ccccc'];
$.each( arr, function ( index, val ) {
    console.log( index, val );
});

var obj = { name: '嘿', age: 66 };
$.each(obj, function ( key, val ) {
    console.log( key, val );
});
```

- 自己实现each方法

> ⑴.如果传入的是对象，则通过for in遍历它，把得到每一项key和val分别传给回调，供其使用。
>
> ⑵.如果传入的是数组，则遍历每一项下标与值，分别传给回调，供其使用。

```
function each( obj, fn ) {
    if ( obj instanceof Array ) {
        for ( var i = 0, len = obj.length; i < len; i++ ) {
            fn( i, obj[i] );
        }
    }else {
        for ( var key in obj ) {
            fn( key, obj[key] );
        }
    }

}

// 测试数组与对象的遍历
var arr = ['aaaa', 'bbbbbb', 'ccccc'];
var obj = { name: '嘿', age: 66 };
each( obj, function ( key, val ) {
    console.log( key, val );
});
each( arr, function ( index, val ) {
    console.log( index, val );
});
```

- 传入jQuery each方法的回调，里面的this指向val。 //测试this function each(obj, fn) { if (obj instanceof Array) { for (var i = 0, len = obj.length; i < len; i++) { fn.call(obj[i], i, obj[i]); } }else { for (var key in obj) { fn.call(obj[key], key, obj[key]); } } }

```
var obj2 = { val: [1], va2: {a:1}, va3: new Date() } ;
each(obj2, function () {
    console.log( this );
});
```

- 如果传入jQ each方法的回调，执行时返回false，那么中断遍历。

```
//测试中断
function each( obj, fn ) {
    if ( obj instanceof Array ) {
        for ( var i = 0, len = obj.length; i < len; i++ ) {
            if ( fn.call( obj[i], i, obj[i] ) === false ) {
                break;
            }
        }
    }else {
```

```
            for ( var key in obj ) {
                if ( fn.call( obj[key], key, obj[key] ) === false ) {
                    break;
                }
            }
        }

        return obj;
    }

    var obj2 = { val: [1], va2: {a:1}, va3: new Date() } ;
    each(obj2, function () {
        console.log( this );
        if ( this[0] == 1 ) {
            return false;
        }
    });
```

4.jQuery中的map方法

- 定义:jQeury有一个静态map方法，用来遍历数组或对象，把遍历到的数据传给回调函数，供其使用。

```
> 和each的区别在于，map会接收回调的返回值，返回把接收到返回值组成数组返回。
>
> map方法的作用就通过一个对象得到一个新数组。

    var obj = { a: 111, b: 222, c: 333 };
    console.log($.map(obj, function (val, key) {
        console.log(val, key, this);
        return val * 10;
    }));
```

- 自己实现map方法：

> 1、先定义一个用来接收回调返回值的数组
>
> 2、按照如下方式遍历对象
>
> 2.1、如果传入的obj是数组，那么使用传统的for循环遍历每一项值和下标，传给回调，然后把回调的结果存储到预定义好的数组中。
>
> 2.2、如果传入的obj是对象，那么使用for in循环遍历每一项val和key，传给回调，然后把回调的结果存储到预定义好的数组中。
>
> 3、返回存储了结果的数组

```
    function map( obj, fn ) {
        var result = [],
            i = 0, len,
            temp, key;

        if ( obj instanceof Array ) {
            for ( len = obj.length; i < len; i++ ) {
                // 过滤null和undefined
                temp = fn( obj[i], i );
                if ( temp != null ) {
                    result.push( temp );
                }
            }
        }else {
            for ( key in obj ) {
                // 过滤null和undefined
                temp = fn( obj[key], key );
                if ( temp != null ) {
                    result.push( temp );
                }
            }
        }

        return result;
    }

    var obj = { a: 111, b: 222, c: 333 };
    console.log(map(obj, function (val, key) {
        console.log(val, key, this);
        return val * 10;
```

```
}));
```

5.较全面的jQuery属性与方法

```
// 为了全局变量污染，把代码写到自调函数中
(function ( w ) {

    var version = "1.0.0";

    var document = w.document;

    var arr = [],
        push = arr.push,
        slice = arr.slice;

    var obj = {},
        toString = obj.toString,
        hasOwn = obj.hasOwnProperty;

    // 为了用户使用方便，提供一个工厂函数
    function jQuery( selector ) {
        return new init( selector );
    }

    // 原型简写&原型默认拥有的属性与方法
    jQuery.fn = jQuery.prototype = {
        jquery: version,
        constructor: jQuery,
        isReady: false,
        length: 0,
        toArray: function () {
            return slice.call( this );
        },
        get: function ( num ) {
            /*
            * 如果为整数，则返回this[num]
            * 如果为负数，则返回this[this.length + num]
            * 如果为null或undefined，则调用toArray返回数组
            * */
            if ( num == null ) {
                return this.toArray();
            }

            return num >= 0? this[num] : this[this.length + num];
        },

        slice: function ( ) {
            /*
            * slice返回的是一个新的实例：可以通过jQuery()得到
            * 截取的功能可以借用数组的slice方法实现。
            * */
            /*var $new = jQuery();
            var arr = slice.apply( this, arguments );
            push.apply( $new, arr );
            return $new;*/

            // 简写
            return jQuery( slice.apply( this, arguments ) );
        },

        eq: function ( num ) {
            var dom;

            // 如果传入null或undefined，则返回一个新的实例
            if ( num == null ) {
                return jQuery();
            }

            // 如果传入数字，得到对应下标的元素，包装成新的实例返回
            // 如果没有得到，则直接返回新的实例。
            /*dom = this.get( num );
            if ( dom ) {
                return jQuery( dom );
            }else {
                return jQuery();
            }*/
```

```
            // 简写
            return (dom = this.get( num ))? jQuery( dom ): jQuery();
        },

        first: function() {
            return this.eq( 0 );
        },

        last: function() {
            return this.eq( -1 );
        },

        // 原型上的each方法，是为实例准备的，
        // 所以借用静态的each遍历实例即可。
        each: function( fn ) {
            return jQuery.each( this, fn );
        },

        // 原型上的map方法，是为实例准备的，
        // 所以借用静态的map遍历实例，
        // 然后把静态map方法返回的数组再返回即可。
        map: function( fn ) {
            return jQuery.map( this, fn );
        },

        push: push,
        sort: arr.sort,
        splice: arr.splice
    };

    // 给jQuery自身以及原型添加一个extend方法
    jQuery.extend = jQuery.fn.extend = function ( obj ) {
        for ( var key in obj ) {
            this[key] = obj[key];
        }
    };

    // 添加静态方法
    jQuery.extend({

        // 判断是不是函数
        isFunction: function( func ) {
            return typeof func === 'function';
        },

        // 判断是不是字符串
        isString: function( str ) {
            return typeof str === 'string';
        },

        // 判断是不是DOM
        isDOM: function( dom )  {
            return !!dom && !!dom.nodeType;
        },

        // 判断是不是html字符串
        isHTML: function( html ) {
            return html.charAt(0) === '<' &&
                    html.charAt(html.length - 1) === '>' &&
                    html.length >= 3;
        },

        // 判断是不是window
        isWindow: function( win ) {
            return !!win && win.window === win;
        },

        // 判断是不是伪数组或数组
        isLikeArray: function( likeArray ) {

            // function & window 返回 false
            if ( jQuery.isFunction( likeArray ) || jQuery.isWindow( likeArray ) ) {
                return false;
            }

            // 如果likeArray是对象，并有length属性，length属性值为0或者拥有length-1的属性
            return !!likeArray && typeof likeArray === 'object' && 'length' in likeArray &&
```

```
            ( likeArray.length === 0 || [likeArray.length - 1] in likeArray );
    },

    // 解析html
    parseHTML: function( html ) {
        var tempDiv = document.createElement('div');
        tempDiv.innerHTML = html;
        return tempDiv.children;
    },

    // 封装一个兼容的DOMContentLoaded方法
    ready: function( fn ) {

        // 如果页面已经触发了DOMContentLoaded事件，那么直接执行fn，
        // 再监听DOMContentLoaded事件已经无用了。
        if ( jQuery.fn.isReady ) {
            return fn();
        }

        // IE9以及现代浏览器使用addEventListener以及DOMContentLoaded事件
        if ( document.addEventListener ) {
            document.addEventListener('DOMContentLoaded', function () {
                jQuery.fn.isReady = true;
                fn();
            });
        }

        // IE8使用attachEvent以及onreadystatechange事件
        else {
            document.attachEvent('onreadystatechange', function () {
                if ( document.readyState === 'complete' ) {
                    jQuery.fn.isReady = true;
                    fn();
                }
            });
        }
    },

    each: function( obj, fn ) {
        var i = 0, len, key;
        if ( jQuery.isLikeArray( obj ) ) {
            for ( len = obj.length; i < len; i++ ) {
                if ( fn.call( obj[i], i, obj[i] ) === false ) {
                    break;
                }
            }
        }else {
            for ( key in obj ) {
                if ( fn.call( obj[key], key, obj[key] ) === false ) {
                    break;
                }
            }
        }

        return obj;
    },

    map: function( obj, fn ) {
        var result = [],
            i = 0, len,
            temp, key;

        if ( jQuery.isLikeArray( obj ) ) {
            for ( len = obj.length; i < len; i++ ) {
                temp = fn( obj[i], i );
                if ( temp != null ) {
                    result.push( temp );
                }
            }
        }else {
            for ( key in obj ) {
                temp = fn( obj[key], key );
                if ( temp != null ) {
                    result.push( temp );
                }
            }
        }
        return result;
```

```
            }
        });

        // 构造函数
        var init = jQuery.prototype.init = function ( selector ) {

            // 空处理 ==> 直接返回this
            if ( !selector ) {
                return this;
            }

            // 函数 ==> 添加到DOMContentLoaed事件中
            if ( jQuery.isFunction( selector ) ) {
                jQuery.ready( selector );
            }

            // 字符串 ==> 要么解析为DOM，要么作为选择器获取页面中的DOM
            else if ( jQuery.isString( selector ) ) {
                // html片段
                if ( jQuery.isHTML( selector ) ) {
                    push.apply(this, jQuery.parseHTML( selector ));
                }
                // 选择器
                else {
                    try {
                        push.apply( this, document.querySelectorAll( selector ) );
                    }catch(e){}
                }
            }

            // dom ==> 直接添加到this中
            else if ( jQuery.isDOM( selector ) ) {
                push.call( this, selector );
            }

            // 数组或伪数组 ==> 把每一项都添加到this中
            else if ( jQuery.isLikeArray( selector ) ) {
                push.apply( this, slice.call( selector ) );
            }

            // 其他 ==> 直接添加到this中
            else {
                push.call( this, selector );
            }
        };

        // 为了第三方扩展(即jQ插件)
        init.prototype = jQuery.fn;

        // 对外暴漏
        w.jQuery = w.$ = jQuery;

        // 解决DOMContentLoaded不触发的问题
        $(function () {});

    }( window ));
```

6.jQuery中一些DOM操作

6.1 jQuery中的empty方法

```
    // 添加DOM操作
    $.fn.extend({

        // 清空每一个元素的内容
        empty: function () {
            /*
            * 实现思路:
            * 遍历实例，把遍历到的每一个元素innnerHTML = ''即可。
            * */
            this.each( function() {
                // 这里的this代表遍历到val值
                this.innerHTML = '';
            });

            // 为了链式编程
```

```
            return this;
        }
    });

    alert(1);
    $('div').empty();
```