

jQuery框架


jQuery event

on(绑定事件)

-  初步版本

```
/*
 * function { on } 给所有元素绑定对应事件的事件句柄
 * param { type: string } 要绑定的事件类型
 * param { fn: Function } 要绑定的事件句柄
 */
on:function( type , fn ){
    this.each(function(){
        if(this.addEventListner){
            this.addEventListner( type , fn);
        }else{
            this.attachEvent( type , fn );
        }
    });


    return this;
}
```

-  添加事件兼容性之后

```
/*
 * function { on } 给所有元素绑定对应事件的事件句柄
 * param { type: string } 要绑定的事件类型
 * param { fn: Function } 要绑定的事件句柄
 */

on: function( type, fn ) {
    this.each( function() {
        **jQuery.addEvent(this,type,fn)**
    } );

    // 为了链式编程
    return this;
}
```

-  新版事件

```
on: function (type, fn) {
    /*
     * 实现思路:
     * 1、遍历所有元素,
     * 2、看看这个元素有没有$_event_cache属性, 有则继续使用, 没有则初始化为{}
     * 3、然后看看$_event_cache对象有没有以 type 命名的数组, 有则继续使用, 没有则初始化为[]
     * 4、最后把fn 存储到 这个数组中。
     * 需要注意: 如果是第一次给某个事件数组push函数, 那么还需要额外绑定一个调用这些数组函数的函数。
     */
    this.each(function () {

        // 这里的this, 指的是每一个元素
        this.$_event_cache = this.$_event_cache || {};
        this.$_event_cache[type] = this.$_event_cache[type] || [];

        // 判断是不是第一次添加事件绑定,
        // 如果是, 则额外绑定一个函数,
        if (this.$_event_cache[type].push(fn) == 1) {

            // 给当前元素(this)绑定事件,
            // 并把当前元素(this)使用一个变量存起来, 供事件处理函数使用。
            var self = this;
            jQuery.addEvent(this, type, function (e) {

                // IE6、7、8绑定事件处理函数, 里面的this, 并没有指向事件源,
                // 所以不能使用this, 只能现在外面把this使用另外一个变量保存起来,
```

```

        // 然后在这里引用。
        jQuery.each(self._event_cache[type], function () {

            // 这里的this, 代表数组中的函数
            this(e);

        });
    });
}

// 为了链式编程
return this;
}

```

for循环实现

```

_on: function (type, fn) {

    this.each(function () {
        this._event_cache = this._event_cache || {};
        this._event_cache[type] = this._event_cache[type] || [];

        // 判断是不是第一次添加事件绑定,
        // 如果是, 则额外绑定一个函数,
        if (this._event_cache[type].push(fn) == 1) {

            // 给当前元素(this)绑定事件,
            // 并把当前元素(this)使用一个变量存起来, 供事件处理函数使用。
            var self = this;
            // 给元素绑定事件
            jQuery.addEvent(this, type, function (e) {

                /**/ 该事件处理函数, 专门负责调用数组中的每一个函数
                for (var i = 0, len = self._event_cache[type].length; i < len; i++) {
                    self._event_cache[type][i](e);
                }
            });
        }
    });

    // 为了链式编程
    return this;
},

```

新版事件bug

```

on: function( type, fn ) {

    this.each( function() {

        // 这里的this, 指的是每一个元素
        this._event_cache = this._event_cache || {};
        /**/this._event_cache[ type ] = this._event_cache[ type ] || [];**

        /**/ 如果this._event_cache[ type ]有值, 那么直接push新值即可
        if ( this._event_cache[ type ] ) {
            this._event_cache[ type ].push( fn );
        }
        /**
        * 如果this._event_cache[ type ]没值,
        * 说明之前没有绑定过这个类型的事件,
        * 即现在是第一次绑定这个事件,
        * 第一次需要额外绑定一个调用 事件的处理函数数组 的函数。
        */
        **else {
            (this._event_cache[ type ] = []).push( fn );**

            // 给当前元素(this)绑定事件,
            // 并把当前元素(this)使用一个变量存起来, 供事件处理函数使用。
            var self = this;
            jQuery.addEvent( this, type, function( e ) {

                // IE6、7、8绑定事件处理函数, 里面的this, 并没有指向事件源,
                // 所以不能使用this, 只能现在外面把this使用另外一个变量保存起来,

```

```

        // 然后在这里引用。
        jQuery.each( self._event_cache[ type ], function() {

            // 这里的this，代表数组中的函数
            this( e );

        } );

    } );

} );

// 为了链式编程
return this;

}

```

☞ off(解除绑定事件)

- ☞ 初步版本

```

/*
 * function { off } 删除所有元素指定的事件句柄
 * param { type: string } 要删除的事件类型
 * param { fn: Function } 要删除的事件句柄
 * */

off: function( type, fn ) {
    this.each( function() {
        // 处理事件绑定兼容性
        if ( this.removeEventListener ) {
            this.removeEventListener( type, fn );
        } else {
            this.detachEvent( 'on' + type, fn );
        }
    });

    // 为了链式编程
    return this;
}

```

- ☞ 添加事件兼容性之后

```

/*
 * function { off } 删除所有元素指定的事件句柄
 * param { type: string } 要删除的事件类型
 * param { fn: Function } 要删除的事件句柄
 * */

off: function( type, fn ) {
    this.each( function() {
        // 处理事件绑定兼容性
        jQuery.removeEvent(this,type,fn)
    });

    // 为了链式编程
    return this;
}

```

☞ click(点击事件)

- ☞ 初步形式

```

/*
 * function { click } 给所有元素绑定点击事件
 * param { fn: Function } 事件句柄
 * */

click: function( fn ) {
    this.each( function() {
        // 处理事件绑定兼容性
        if ( this.addEventListener ) {
            this.addEventListener( 'click', fn );
        } else {
            this.attachEvent( 'onclick', fn );
        }
    });
}

```

```

});

// 为了链式编程
return this;
},

//另一种较为简便的方法
_click: function( fn ) {
    return this.on( 'click', fn );
}

```

- 添加事件兼容性之后

```

/*
 * function { click } 给所有元素绑定点击事件
 * param { fn: Function } 事件句柄
 * */
click: function( fn ) {
    this.each( function() {
        jQuery.addEvent(this, 'click', fn);
    });

    // 为了链式编程
    return this;
}

```

- 新版事件

```

/*
 * function { off } 删除所有元素指定的事件句柄
 * param { type: string } 要删除的事件类型
 * param { fn: Function } 要删除的事件句柄
 * */
off: function (type, fn) {
    /*
     * 实现思路:
     * 1、如果没有传参数, 删除所有事件的处理函数( 把元素的$_event_cache里面存储的事件函数数组重置 )
     * 2、如果传入1个参数, 那么解除对应事件的处理函数( 把元素的$_event_cache里面对应事件的函数数组重置 )
     * 3、如果传入2个参数, 那么解除对应事件的对应处理函数( 遍历对应事件的数组, 依次和fn比较, 如果相等那么从数组中剔除 )
     */

    var len=arguments.length;

    this.each(function () {
        var $_event_cache=this.$_event_cache,
            key,i;
        // if have no $_event_cache, do nothing;
        if(!$_event_cache){
            return;
        }

        //if have no arguments , clear all arguments in $_event_cache
        if (len==0){
            for (key in $_event_cache){
                $_event_cache[key]=[];
            }
        }

        //if have one of arguments ,clear appointed argument in $_event_cache
        if (len==1){
            $_event_cache[type]=[];
        }

        //if have two of arguments ,clear appointed array of incident constructor in $_event_cache
        else if(len==2){
            for(i=$_event_cache[type].length-1;i>=0;i--){
                if($_event_cache[type][i]==fn){
                    $_event_cache[type].splice(i,1);
                }
            }
        }
    })
    // 为了链式编程
}

```

```

    return this;
}

```

☞ 添加事件兼容性

```

$.extend({
    //☞ 给指定元素添加事件绑定句柄
    addEvent: function (ele,type,fn) {
        //如果参数不够三个或者ele不是DOM, 那么不作处理
        if(!jQuery.isDOM(ele)||arguments.length!==3){
            return;
        }

        if(jQuery.isFunction(ele.addEventListener)){
            ele.addEventListener(type,fn);
        }else {
            ele.attachEvent('on'+type,fn);
        }
    },

    //☞ 解除指定元素指定的事件绑定句柄
    removeEvent: function (ele,type,fn) {
        if (!jQuery.isDOM(ele)||arguments.length!==3){
            return;
        }

        if(jQuery.isFunction(ele.removeEventListener)){
            ele.removeEventListener(type,fn);
        }else {
            ele.detachEvent('on'+type,fn);
        }
    }
});

```

☞ 给原型批量添加事件

```

// 批量给原型添加事件监听函数
var events = ( "blur focus focusin focusout load resize scroll unload click dblclick " +
"mousedown mouseup mousemove mouseover mouseout mouseenter mouseleave " +
"change select submit keydown keypress keyup error contextmenu " ).split( " " );
jQuery.each( events, function( index, val ) {
    $.fn[ val ] = function( fn ) {
        return this.on( val, fn );
    }
});

```

☞ jQuery上的extend方法

```

/*
 * 如果传入1个对象, 把这个对象的内容copy到this身上,
 * 如果传入多个对象, 把后面所有对象的内容copy到第一个对象身上。
 */
function extend() {
    var arg = arguments,
        argLen = arg.length,
        key, i = 1;

    // 1个对象, 把这个对象的内容copy到this身上
    if ( argLen == 1 ) {
        for ( key in arg[ 0 ] ) {
            this[ key ] = arg[ 0 ][ key ];
        }
    }
    // 多个对象, 把后面所有对象的内容copy到第一个对象身上。
    else if( argLen > 1 ) {
        for ( ; i < argLen; i++ ) {
            for ( key in arg[ i ] ) {
                arg[ 0 ][ key ] = arg[ i ][ key ];
            }
        }
    }
}

```

```
}
```

jQuery框架

ajax

☞ 添加一个创建xhr对象的静态方法

```
// 兼容获取XMLHttpRequest对象
getXhr: function () {
    if (window.XMLHttpRequest) {
        return new XMLHttpRequest();
    } else {
        return new ActiveXObject('Microsoft.XMLHTTP');
    }
},

// ajax默认的配置项
ajaxSettings: {
    url: location.href,
    method: "GET",
    async: true,
    contentType: "application/x-www-form-urlencoded; charset=UTF-8",
    timeout: 0,
    dataType: null,
    success: function () {
    },
    error: function () {
    },
    complete: function () {
    }
},

// 发送ajax请求
ajax: function (options) {
    var config = {}, xhr;

    // ajaxSettings ==> config
    // options ==> config
    jQuery.extend(config, jQuery.ajaxSettings, options);

    xhr = jQuery.getXhr();
    xhr.open(config.method, config.url, config.async);
    xhr.onreadystatechange = function () {
        // 请求完毕，则调用complete方法
        if (xhr.readyState == 4) {
            config.complete();

            // 请求成功，则调用success方法
            if (( xhr.status >= 200 && xhr.status < 300 ) || xhr.status == 304) {
                config.success();
            }
            // 请求失败，则调用error方法
            else {
                config.error();
            }
        }
    };
    xhr.send();
}
```

☞ ajax

```
// 兼容获取XMLHttpRequest对象
getXhr: function() {
    if ( window.XMLHttpRequest ) {
        return new XMLHttpRequest();
    }else {
        return new ActiveXObject('Microsoft.XMLHTTP');
    }
},
```

```

// ajax默认的配置项
ajaxSettings: {
    url: location.href,
    method: "GET",
    async: true,
    contentType: "application/x-www-form-urlencoded; charset=UTF-8",
    timeout: 0,
    dataType: null,
    success: function() {},
    error: function() {},
    complete: function() {}
},

/**// 把数据转换为url参数**
urlStringify: function( data ) {
    var key, result = '';

    for ( key in data ) {
        result += key + '=' + data[ key ] + '&';
    }

    // 剔除最后一个&符号
    return result.slice( 0, -1 );
},

// 发送ajax请求
ajax: function( options ) {
    var config = {}, xhr;

    // ajaxSettings ==> config
    // options ==> config
    jQuery.extend( config, jQuery.ajaxSettings, options );

    xhr = jQuery.getXhr();
    xhr.open( config.method, config.url + '?' + jQuery.urlStringify( config.data ), config.async );
    xhr.onreadystatechange = function() {
        // 请求完毕，则调用complete方法
        if ( xhr.readyState == 4 ) {
            config.complete();

            // 请求成功，则调用success方法
            if ( ( xhr.status >= 200 && xhr.status < 300 ) || xhr.status == 304 ) {
                config.success();
            }
            // 请求失败，则调用error方法
            else {
                config.error();
            }
        }
    };
    xhr.send();
}

```