JS自定义数据提交处理方案

# 问题

在Ajax技术流行的今天，我遇到了一个很头疼的问题，我需要维护一个前人开发的问题单模块功能，并且不停的迭代开发，而这个问题就是问题单字段特别多，而且除了更新问题单外，问题单的提交审核的操作几乎要对每个字段进行校验，不满足校验时提示用户并取消提交操作。

要知道只C#的后台模型这么简单的属性写法加上注释的话也差不多四五百行代码了，何况要在前台获取表单数据，然后对获取的结果进行校验，检验通过后才能提交表单。这样的代码开发及维护起来工作量那是相当的大，而且极容易出错，举个例子，需要提交员工的数据如下：

<input type="hidden" id="id" value="1">

<input type="text" id="number">

<input type="text" id="Name">

<input type="text" id="Age">

<input type="text" id="BirthDay">

# 传统的处理方式

也是我接手这个模块时是这样获取并校验的：

function getCon(){

Number: $('#number').val(),

Name: $('#name').val(),

Age: $('#age').val(),

BirthDay: $('#birthDay').val(),

}

function check(con){

if(!con.Number){

return '工号不能为空!';

}

if(!con.Name){

return '姓名不能为空!';

}

if(con.Age == null){

return '年龄不能为空!';

}

if(!/^\d{2}$/.test(con.Age)){

return '请输入两位数字';

}

if(con.BirthDay == null){

return 'BirthDay不能为空!';

}

if(con.BirthDay < '1988-02-01'){

return '年龄太大！';

}

}

var con = getCon();

var res = check(con);

if(res){

alert(res);

return;

}

$.post('action', con, function(data){

alert(data);

}, 'json')

这个例子字段很少，都可以看出来，代码量却不少，而且各个地方分散的太严重，又相互联系，新增一个属性会在页面改动很多添加代码，很容易漏，而且很难找到。因此，我便着手写自己的框架来处理这样的问题。

# 封装JS模型

我首先设计了一个JS模型用来存储，键为后台模型的属性，值包含前台属性绑定的ID，类型valueType以及校验的条件checkTip，这样的好处是，将页面也后台模型通过JS模型联系起来，当需求变化的时候只需要改变对应属性的模型就好，例子如下：

var model = {

id: 1,

number: {

id:"number",

valueType:function(id){return parseInt($(id).val())},

checkTip:"工号不能为空"

},

Name:{

id:"name",

valueType:"string",

checkTip:"姓名不能为空"

},

Age:{

id:"age",

valueType:'',

checkTip:function(id){

if (!/\d{2}/.test($(this).val())) return '请输入两位数字';

return '';

}

},

BirthDay:{

id:"birthday",

valueType:"date",

checkTip:function(val,id){

if(val < '1988-02-01') return '年龄太大！'

}

},

}

# 编写通用处理框架

然后我抽取了一个非常强大的方法，实现可顺便校验的查询或提交所需要的字段信息，使用此方法可减少大量的代码和降低字段太多时出错的概率，这个方法会根据valueType和id自动从页面获取对应的值，需要校验时还能自动根据checkTip进行校验，不校验或者校验无误时返回格式为{k:v}，校验错误时返回格式为 {isError:true,errorMsg:'message',{k:v}}，源码如下：

function conditionHandler (model,isCheck,callBack){

var newmodel = {};

//入口参数处理

if(typeof isCheck === 'function'){

callBack = isCheck;

isCheck = undefined;

}

if(isCheck === true){

//需要校验时，一边获取条件一边校验是否满足要求，不满足立即退出

newmodel.isError = false;

newmodel.errorMsg = '';

newmodel.model = {};

muphy.each(model, function(k,v){

var val = getVal(this);

//必须校验的字段checkTip不为空

if(!muphy.isNull(this.checkTip)){

if(typeof this.checkTip === 'function'){

//自定义校验

var res = this.checkTip.call(val,val,'#' + this.id);

if(!muphy.isNull(res)){

newmodel.isError = true;

newmodel.errorMsg = res;

newmodel.model[k] = val;

return false;

}

} else {

//默认判空

if(muphy.isNull(val)){

newmodel.isError = true;

newmodel.errorMsg = this.checkTip;

newmodel.model[k] = val;

return false;

}

}

}

newmodel.model[k] = val;

});

if(!newmodel.isError){

newmodel = newmodel.model;

}

} else {

//不需要校验时，直接获取条件

muphy.each(model, function(k,v)){

newmodel[k] = getVal(this);

}

}

//当传入了回调函数是，可以回调处理结果

if(typeof callBack === 'function'){

callBack.call(newmodel,newmodel);

}

//返回条件参数

return newmodel;

//获取单个条件值

function getVal(obj){

var val;

if(typeof obj === 'string' || typeof obj === 'number' || obj instanceof Date){

val = obj;

} else if(obj.valueType instanceof Array || obj.valueType === 'string'){

val = muphy.nvl($('#' + obj.id).val(), []).join(',');

} else if(typeof obj.valueType === 'function'){

val = obj.valueType.call('#' + obj.id, '#' + obj.id);

} else {

val = muphy.nvl($('#' + obj.id).val());

}

return val;

}

}

# 调用框架处理提交

最后是调用的过程，调用是非常简单的，获取条件的结果可以在回调函数中直接处理，也可以用变量来接收后在处理，功能灵活且强大，调用封装的处理函数：

conditionHandler(model, true, function(con){

if(con.isError){

alert(con.errorMsg);

return;

}

$.post('action', con, function(data){

alert(data);

}, 'json')

})

# 优点

从这里可以看出来，以后需要改动的时候，只需要改动JS模型对影的属性信息就好，其他地方几乎不变。

# 封装的源代码

/\*!

\* muphyjs JavaScript Library v1.0.0

\* http://www.muphy.me/

\*

\* Includes jQuery.js

\* http://jquery.com/

\*

\* Copyright 莫非(muphy)

\*

\* Date: 2018-07-16

\*/

(function(window,$){

var muphy = Object.create({

/\*@description 判断函数是否为空

\*@method isNull

\*@for muphy

\*@examples muphy.isNull(data);

\*@param {object} data 需要判断空的对象

\*@return {bool} 如果为空则返回 true，否则返回 false

\*/

isNull: function(data){

if(data === null || data === undefined){

return true;

}

if( typeof data === "String" && data.trim() ==='' ){

return true;

}

if(data instanceof Array && data.length == 0){

return true;

}

return false;

},

/\*@description 将空对象转换为指定的值

\*@method nvl

\*@for muphy

\*@examples

\*1. muphy.nvl(data);//data为空时返回 ''

\*2. muphy.nvl(data,[]);//data为空时返回 []

\*@param {object} data 需要判断空的对象

\*@param {object} obj 为空时返回的对象

\*@return {object} 如果为空则返回 obj,否则返回 data

\*/

nvl: function(data,obj){

if(obj === 0) return 0;

return data || obj || '';

},

/\*@description 遍历处理数组、集合或者对象，同jQuery.each

\*@method each

\*@for muphy

\*@examples

\*1. muphy.each(arr, function(){console.log(this);});//遍历打印数组的值

\*2. muphy.each(obj,function(k,v){console.log(k + ':' + v);});//遍历打印对象键值对

\*@param {object} obj 需要遍历的数组、集合或者对象

\*@param {function} fun 回调函数，处理每项结果

\*/

each: function(obj,fun){

for (var key in obj) {

if(fun.call(obj[key], key, obj[key]) === false){

break;

}

}

},

/\*@description 为元素添加事件处理函数

\*@method addEvent

\*@for muphy

\*@param {Elenent} ele 需要绑定事件的元素

\*@param {string} type 事件类型

\*@param {function} handler 事件处理程序

\*/

addEvent: function(ele,type,handler){

if(ele.addEventListener){

ele.addEventListener(type,handler,false);

} else if(ele.attachEvent){

ele.attachEvent("on" + type, handler);

} else {

ele["on" + type] = handler;

}

},

/\*@description 移除元素事件处理函数

\*@method removeEvent

\*@for muphy

\*@param {Elenent} ele 需要移除事件的元素

\*@param {string} type 事件类型

\*@param {function} handler 事件处理程序

\*/

removeEvent: function(ele,type,handler){

if(ele.removeEventListener){

ele.removeEventListener(type,handler,false);

} else if(ele.detachEvent){

ele.detachEvent("on" + type, handler);

} else {

ele["on" + type] = null;

}

},

/\*@description 为元素添加拖动事件

\*@method \_dragEvent

\*@for muphy

\*@param {Elenent} ele 需要拖动事件的元素，必须绝对定位，并且添加类名 class='draggable'

\*/

\_dragEvent: function(ele){

var drag = null,

diffx = 0,

diffy = 0,

$e = null;

if($.isNull(ele)) return;

if(!($e = $(ele))) return;

$(document).bind("mousedown",handleEvent);

$(document).bind("mousemove",handleEvent);

//$(document).bind("mouseup",handleEvent);

$.muphy.addEevent(document,"mouseup",handleEvent);

function handleEvent(event){

event = event || window.event;

var target = $e[0];

switch(event.type){

case "mousedown":

if($m.nvl(target.className).indexOf("draggable") > -1){

drag = target;

diffx = event.clientX - target.offsetLeft;

diffy = event.clientY - target.offsetTop;

}

break;

case "mousemove":

if(drag !== null){

drag.style.left = (event.clientX - diffx) + "px";

drag.style.top = (event.clientY - diffy) + "px";

}

break;

case "mouseup":

drag = null;

break;

}

}

},

/\*@description 为元素添加拖动事件，并且新增自定义事件：dragstart、drag、dragend

\*@method dragEvent

\*@for muphy

\*/

dragEvent: function(){

var drag = null,

ce = new muphy.costomEvent();

diffx = 0,

diffy = 0;

$(document).unbind("mousedown",handleEvent);

$(document).unbind("mousemove",handleEvent);

$(document).unbind("mouseup",handleEvent);

$(document).bind("mousedown",handleEvent);

$(document).bind("mousemove",handleEvent);

$(document).bind("mouseup",handleEvent);

function handleEvent(event){

event = event || window.event;

var target = event.target || event.srcElement;

switch(event.type){

case "mousedown":

if(muphy.nvl(target.className).indexOf("draggable") > -1){

drag = target;

diffx = event.clientX - target.offsetLeft;

diffy = event.clientY - target.offsetTop;

event.type = "dragstart";

ce.fire(event)

}

break;

case "mousemove":

if(drag !== null){

drag.style.left = (event.clientX - diffx) + "px";

drag.style.top = (event.clientY - diffy) + "px";

event.type = "drag";

ce.fire(event)

}

break;

case "mouseup":

if(drag != null){

drag = null;

event.type = "dragend";

ce.fire(event)

}

break;

}

}

return ce;

},

/\*@description 自定义事件

\*@method \_costomEvent

\*@for muphy

\*/

\_costomEvent: function(){

var handlers = {};

// 添加自定义事件的函数

this.addEvent = function(type, handler){

if(typeof handlers[type] === 'undefined'){

handlers[type] = [];

}

handlers[type].push(handler);

}

// 移除自定义事件的函数

this.removeEvent = function(type, handler){

if(handlers[type] instanceof Array){

muphy.each(handlers[type], function(i){

if(this === handler){

handlers[type].splice(i,1);

return false;

}

});

}

}

// 触发自定义事件

this.fire = function(event){

if(!event.target){

event.target = this;

}

if(handlers[event.type] instanceof Array){

muphy.each(handlers[event.type],function(){

this(event);

})

}

}

},

/\*@description 自定义事件

\*@method \_costomEvent

\*@for muphy

\*/

costomEvent: function(){

},

/\*@description 实现可顺便校验的查询或提交所需要的字段信息，使用此方法可减少大量的代码和降低字段太多时出错的概率

\*@method conditionHandler

\*@for muphy

\*@param {object} model 自定义在页面和后台模型之间建立联系的model

\*@examples

1.model:{

id: 1,

number: {id:"number", valueType:function(id){return parseInt($(id).val())}, checkTip:"工号不能为空"},

Name:{id:"name", valueType:"string", checkTip:"姓名不能为空"},

Age:{id:"age", valueType:'', checkTip:function(id){if (!/\d{2}/.test($(this).val())) return '请输入两位数字'; return '';}},

BirthDay:{id:"birthday", valueType:"date", checkTip:function(val,id){if(val < '1988-02-01') return '年龄太大！'}},

}

2.校验： muphy.conditionHandler(model, true, fun)

3.不校验： muphy.conditionHandler(model, fun);

\*@param {bool} isCheck 是否需要校验，需要校验必须为true,不需要时可不传

\*@param {\_function} callBack 回调函数，获取完成传入后台的参数后可以直接在回调函数中处理

\*@return {object} newmodel 返回获取的参数，校验无误是返回格式为{k:v},校验错误时返回格式为 {isError:true,errorMsg:'message',{k:v}}

\*/

conditionHandler: function(model,isCheck,callBack){

var newmodel = {};

//入口参数处理

if(typeof isCheck === 'function'){

callBack = isCheck;

isCheck = undefined;

}

if(isCheck === true){

//需要校验时，一边获取条件一边校验条件是否满足要求，不满足立即退出

newmodel.isError = false;

newmodel.errorMsg = '';

newmodel.model = {};

muphy.each(model, function(k,v){

var val = getVal(this);

//必须校验的字段checkTip不为空

if(!muphy.isNull(this.checkTip)){

if(typeof this.checkTip === 'function'){

//自定义校验

var res = this.checkTip.call(val,val,'#' + this.id);

if(!muphy.isNull(res)){

newmodel.isError = true;

newmodel.errorMsg = res;

newmodel.model[k] = val;

return false;

}

} else {

//默认判空

if(muphy.isNull(val)){

newmodel.isError = true;

newmodel.errorMsg = this.checkTip;

newmodel.model[k] = val;

return false;

}

}

}

newmodel.model[k] = val;

});

if(!newmodel.isError){

newmodel = newmodel.model;

}

} else {

//不需要校验时，直接获取条件

muphy.each(model, function(k,v)){

newmodel[k] = getVal(this);

}

}

//当传入了回调函数是，可以回调处理结果

if(typeof callBack === 'function'){

callBack.call(newmodel,newmodel);

}

//返回条件参数

return newmodel;

//获取单个条件值

function getVal(obj){

var val;

if(typeof obj === 'string' || typeof obj === 'number' || obj instanceof Date){

val = obj;

} else if(obj.valueType instanceof Array || obj.valueType === 'string'){

val = muphy.nvl($('#' + obj.id).val(), []).join(',');

} else if(typeof obj.valueType === 'function'){

val = obj.valueType.call('#' + obj.id, '#' + obj.id);

} else {

val = muphy.nvl($('#' + obj.id).val());

}

return val;

}

}

});

/\*@description 自定义事件添加原型函数

\*@method costomEvent

\*@static costomEvent的原型对象

\*@for muphy

\*/

(function(muphy){

muphy.costomEvent.prototype = {

constructor: muphy.costomEvent1,

handlers:{},

addEvent: function(type, handler){

if(typeof this.handlers[type] === 'undefined'){

this.handlers[type] = [];

}

this.handlers[type].push(handler);

},

removeEvent: function(type, handler){

if(this.handlers[type] instanceof Array){

var handlers = this.handlers[type];

muphy.each(handlers, function(i){

if(this === handler){

handlers.splice(i,1);

return false;

}

});

}

},

fire: function(event){

if(!event.target){

event.target = this;

}

if(this.handlers[event.type] instanceof Array){

muphy.each(this.handlers[event.type],function(){

this(event);

})

}

}

}

})(muphy);

window.$m = muphy;

})(window,jQuery);