//前台调用

var $ = function (args) {

return new Base(args);

}

//基础库

function Base(args) {

//创建一个数组，来保存获取的节点和节点数组

this.elements = [];

if (typeof args == 'string') {

//css模拟

if (args.indexOf(' ') != -1) {

var elements = args.split(' '); //把节点拆开分别保存到数组里

var childElements = []; //存放临时节点对象的数组，解决被覆盖的问题

var node = []; //用来存放父节点用的

for (var i = 0; i < elements.length; i ++) {

if (node.length == 0) node.push(document); //如果默认没有父节点，就把document放入

switch (elements[i].charAt(0)) {

case '#' :

childElements = []; //清理掉临时节点，以便父节点失效，子节点有效

childElements.push(this.getId(elements[i].substring(1)));

node = childElements; //保存父节点，因为childElements要清理，所以需要创建node数组

break;

case '.' :

childElements = [];

for (var j = 0; j < node.length; j ++) {

var temps = this.getClass(elements[i].substring(1), node[j]);

for (var k = 0; k < temps.length; k ++) {

childElements.push(temps[k]);

}

}

node = childElements;

break;

default :

childElements = [];

for (var j = 0; j < node.length; j ++) {

var temps = this.getTagName(elements[i], node[j]);

for (var k = 0; k < temps.length; k ++) {

childElements.push(temps[k]);

}

}

node = childElements;

}

}

this.elements = childElements;

} else {

//find模拟

switch (args.charAt(0)) {

case '#' :

this.elements.push(this.getId(args.substring(1)));

break;

case '.' :

this.elements = this.getClass(args.substring(1));

break;

default :

this.elements = this.getTagName(args);

}

}

} else if (typeof args == 'object') {

if (args != undefined) { //\_this是一个对象，undefined也是一个对象，区别与typeof返回的带单引号的'undefined'

this.elements[0] = args;

}

} else if (typeof args == 'function') {

this.ready(args);

}

}

//addDomLoaded

Base.prototype.ready = function (fn) {

addDomLoaded(fn);

};

//获取ID节点

Base.prototype.getId = function (id) {

return document.getElementById(id)

};

//获取元素节点数组

Base.prototype.getTagName = function (tag, parentNode) {

var node = null;

var temps = [];

if (parentNode != undefined) {

node = parentNode;

} else {

node = document;

}

var tags = node.getElementsByTagName(tag);

for (var i = 0; i < tags.length; i ++) {

temps.push(tags[i]);

}

return temps;

};

//获取CLASS节点数组

Base.prototype.getClass = function (className, parentNode) {

var node = null;

var temps = [];

if (parentNode != undefined) {

node = parentNode;

} else {

node = document;

}

var all = node.getElementsByTagName('\*');

for (var i = 0; i < all.length; i ++) {

if ((new RegExp('(\\s|^)' +className +'(\\s|$)')).test(all[i].className)) {

temps.push(all[i]);

}

}

return temps;

}

//设置CSS选择器子节点

Base.prototype.find = function (str) {

var childElements = [];

for (var i = 0; i < this.elements.length; i ++) {

switch (str.charAt(0)) {

case '#' :

childElements.push(this.getId(str.substring(1)));

break;

case '.' :

var temps = this.getClass(str.substring(1), this.elements[i]);

for (var j = 0; j < temps.length; j ++) {

childElements.push(temps[j]);

}

break;

default :

var temps = this.getTagName(str, this.elements[i]);

for (var j = 0; j < temps.length; j ++) {

childElements.push(temps[j]);

}

}

}

this.elements = childElements;

return this;

}

//获取某一个节点，并返回这个节点对象

Base.prototype.ge = function (num) {

return this.elements[num];

};

//获取首个节点，并返回这个节点对象

Base.prototype.first = function () {

return this.elements[0];

};

//获取末个节点，并返回这个节点对象

Base.prototype.last = function () {

return this.elements[this.elements.length - 1];

};

//获取某组节点的数量

Base.prototype.length = function () {

return this.elements.length;

};

//获取某一个节点的属性

Base.prototype.attr = function (attr, value) {

for (var i = 0; i < this.elements.length; i ++) {

if (arguments.length == 1) {

return this.elements[i].getAttribute(attr);

} else if (arguments.length == 2) {

this.elements[i].setAttribute(attr, value);

}

}

return this;

};

//获取某一个节点在整个节点组中是第几个索引

Base.prototype.index = function () {

var children = this.elements[0].parentNode.children;

for (var i = 0; i < children.length; i ++) {

if (this.elements[0] == children[i]) return i;

}

};

//设置某一个节点的透明度

Base.prototype.opacity = function (num) {

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i].style.opacity = num / 100;

this.elements[i].style.filter = 'alpha(opacity=' + num + ')';

}

return this;

};

//获取某一个节点，并且Base对象

Base.prototype.eq = function (num) {

var element = this.elements[num];

this.elements = [];

this.elements[0] = element;

return this;

};

//获取当前节点的下一个元素节点

Base.prototype.next = function () {

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i] = this.elements[i].nextSibling;

if (this.elements[i] == null) throw new Error('找不到下一个同级元素节点！');

if (this.elements[i].nodeType == 3) this.next();

}

return this;

};

//获取当前节点的上一个元素节点

Base.prototype.prev = function () {

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i] = this.elements[i].previousSibling;

if (this.elements[i] == null) throw new Error('找不到上一个同级元素节点！');

if (this.elements[i].nodeType == 3) this.prev();

}

return this;

};

//设置CSS

Base.prototype.css = function (attr, value) {

for (var i = 0; i < this.elements.length; i ++) {

if (arguments.length == 1) {

return getStyle(this.elements[i], attr);

}

this.elements[i].style[attr] = value;

}

return this;

}

//添加Class

Base.prototype.addClass = function (className) {

for (var i = 0; i < this.elements.length; i ++) {

if (!hasClass(this.elements[i], className)) {

this.elements[i].className += ' ' + className;

}

}

return this;

}

//移除Class

Base.prototype.removeClass = function (className) {

for (var i = 0; i < this.elements.length; i ++) {

if (hasClass(this.elements[i], className)) {

this.elements[i].className = this.elements[i].className.replace(new RegExp('(\\s|^)' +className +'(\\s|$)'), ' ');

}

}

return this;

}

//添加link或style的CSS规则

Base.prototype.addRule = function (num, selectorText, cssText, position) {

var sheet = document.styleSheets[num];

insertRule(sheet, selectorText, cssText, position);

return this;

}

//移除link或style的CSS规则

Base.prototype.removeRule = function (num, index) {

var sheet = document.styleSheets[num];

deleteRule(sheet, index);

return this;

}

//设置表单字段元素

Base.prototype.form = function (name) {

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i] = this.elements[i][name];

}

return this;

};

//设置表单字段内容获取

Base.prototype.value = function (str) {

for (var i = 0; i < this.elements.length; i ++) {

if (arguments.length == 0) {

return this.elements[i].value;

}

this.elements[i].value = str;

}

return this;

}

//设置innerHTML

Base.prototype.html = function (str) {

for (var i = 0; i < this.elements.length; i ++) {

if (arguments.length == 0) {

return this.elements[i].innerHTML;

}

this.elements[i].innerHTML = str;

}

return this;

}

//设置innerText

Base.prototype.text = function (str) {

for (var i = 0; i < this.elements.length; i ++) {

if (arguments.length == 0) {

return getInnerText(this.elements[i]);

}

setInnerText(this.elements[i], text);

}

return this;

}

//设置事件发生器

Base.prototype.bind = function (event, fn) {

for (var i = 0; i < this.elements.length; i ++) {

addEvent(this.elements[i], event, fn);

}

return this;

};

//设置鼠标移入移出方法

Base.prototype.hover = function (over, out) {

for (var i = 0; i < this.elements.length; i ++) {

addEvent(this.elements[i], 'mouseover', over);

addEvent(this.elements[i], 'mouseout', out);

}

return this;

};

//设置点击切换方法

Base.prototype.toggle = function () {

for (var i = 0; i < this.elements.length; i ++) {

(function (element, args) {

var count = 0;

addEvent(element, 'click', function () {

args[count++ % args.length].call(this);

});

})(this.elements[i], arguments);

}

return this;

};

//设置显示

Base.prototype.show = function () {

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i].style.display = 'block';

}

return this;

};

//设置隐藏

Base.prototype.hide = function () {

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i].style.display = 'none';

}

return this;

};

//设置物体居中

Base.prototype.center = function (width, height) {

var top = (getInner().height - height) / 2 + getScroll().top;

var left = (getInner().width - width) / 2 + getScroll().left;

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i].style.top = top + 'px';

this.elements[i].style.left = left + 'px';

}

return this;

};

//锁屏功能

Base.prototype.lock = function () {

for (var i = 0; i < this.elements.length; i ++) {

fixedScroll.top = getScroll().top;

fixedScroll.left = getScroll().left;

this.elements[i].style.width = getInner().width + getScroll().left + 'px';

this.elements[i].style.height = getInner().height + getScroll().top + 'px';

this.elements[i].style.display = 'block';

parseFloat(sys.firefox) < 4 ? document.body.style.overflow = 'hidden' : document.documentElement.style.overflow = 'hidden';

addEvent(this.elements[i], 'mousedown', predef);

addEvent(this.elements[i], 'mouseup', predef);

addEvent(this.elements[i], 'selectstart', predef);

addEvent(window, 'scroll', fixedScroll);

}

return this;

};

Base.prototype.unlock = function () {

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i].style.display = 'none';

parseFloat(sys.firefox) < 4 ? document.body.style.overflow = 'auto' : document.documentElement.style.overflow = 'auto';

removeEvent(this.elements[i], 'mousedown', predef);

removeEvent(this.elements[i], 'mouseup', predef);

removeEvent(this.elements[i], 'selectstart', predef);

removeEvent(window, 'scroll', fixedScroll);

}

return this;

};

//触发点击事件

Base.prototype.click = function (fn) {

for (var i = 0; i < this.elements.length; i ++) {

this.elements[i].onclick = fn;

}

return this;

};

//触发浏览器窗口事件

Base.prototype.resize = function (fn) {

for (var i = 0; i < this.elements.length; i ++) {

var element = this.elements[i];

addEvent(window, 'resize', function () {

fn();

if (element.offsetLeft > getInner().width + getScroll().left - element.offsetWidth) {

element.style.left = getInner().width + getScroll().left - element.offsetWidth + 'px';

if (element.offsetLeft <= 0 + getScroll().left) {

element.style.left = 0 + getScroll().left + 'px';

}

}

if(element.offsetTop > getInner().height + getScroll().top - element.offsetHeight) {

element.style.top = getInner().height + getScroll().top - element.offsetHeight + 'px';

if (element.offsetTop <= 0 + getScroll().top) {

element.style.top = 0 + getScroll().top + 'px';

}

}

});

}

return this;

};

//设置动画

Base.prototype.animate = function (obj) {

for (var i = 0; i < this.elements.length; i ++) {

var element = this.elements[i];

var attr = obj['attr'] == 'x' ? 'left' : obj['attr'] == 'y' ? 'top' :

obj['attr'] == 'w' ? 'width' : obj['attr'] == 'h' ? 'height' :

obj['attr'] == 'o' ? 'opacity' : obj['attr'] != undefined ? obj['attr'] : 'left';

var start = obj['start'] != undefined ? obj['start'] :

attr == 'opacity' ? parseFloat(getStyle(element, attr)) \* 100 :

parseInt(getStyle(element, attr));

var t = obj['t'] != undefined ? obj['t'] : 10; //可选，默认10毫秒执行一次

var step = obj['step'] != undefined ? obj['step'] : 20; //可选，每次运行10像素

var alter = obj['alter'];

var target = obj['target'];

var mul = obj['mul'];

var speed = obj['speed'] != undefined ? obj['speed'] : 6; //可选，默认缓冲速度为6

var type = obj['type'] == 0 ? 'constant' : obj['type'] == 1 ? 'buffer' : 'buffer'; //可选，0表示匀速，1表示缓冲，默认缓冲

if (alter != undefined && target == undefined) {

target = alter + start;

} else if (alter == undefined && target == undefined && mul == undefined) {

throw new Error('alter增量或target目标量必须传一个！');

}

if (start > target) step = -step;

if (attr == 'opacity') {

element.style.opacity = parseInt(start) / 100;

element.style.filter = 'alpha(opacity=' + parseInt(start) +')';

} else {

//element.style[attr] = start + 'px';

}

if (mul == undefined) {

mul = {};

mul[attr] = target;

}

clearInterval(element.timer);

element.timer = setInterval(function () {

/\*

问题1：多个动画执行了多个列队动画，我们要求不管多少个动画只执行一个列队动画

问题2：多个动画数值差别太大，导致动画无法执行到目标值，原因是定时器提前清理掉了

解决1：不管多少个动画，只提供一次列队动画的机会

解决2：多个动画按最后一个分动画执行完毕后再清理即可

\*/

//创建一个布尔值，这个值可以了解多个动画是否全部执行完毕

var flag = true; //表示都执行完毕了

for (var i in mul) {

attr = i == 'x' ? 'left' : i == 'y' ? 'top' : i == 'w' ? 'width' : i == 'h' ? 'height' : i == 'o' ? 'opacity' : i != undefined ? i : 'left';

target = mul[i];

if (type == 'buffer') {

step = attr == 'opacity' ? (target - parseFloat(getStyle(element, attr)) \* 100) / speed :

(target - parseInt(getStyle(element, attr))) / speed;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

}

if (attr == 'opacity') {

if (step == 0) {

setOpacity();

} else if (step > 0 && Math.abs(parseFloat(getStyle(element, attr)) \* 100 - target) <= step) {

setOpacity();

} else if (step < 0 && (parseFloat(getStyle(element, attr)) \* 100 - target) <= Math.abs(step)) {

setOpacity();

} else {

var temp = parseFloat(getStyle(element, attr)) \* 100;

element.style.opacity = parseInt(temp + step) / 100;

element.style.filter = 'alpha(opacity=' + parseInt(temp + step) + ')';

}

if (parseInt(target) != parseInt(parseFloat(getStyle(element, attr)) \* 100)) flag = false;

} else {

if (step == 0) {

setTarget();

} else if (step > 0 && Math.abs(parseInt(getStyle(element, attr)) - target) <= step) {

setTarget();

} else if (step < 0 && (parseInt(getStyle(element, attr)) - target) <= Math.abs(step)) {

setTarget();

} else {

element.style[attr] = parseInt(getStyle(element, attr)) + step + 'px';

}

if (parseInt(target) != parseInt(getStyle(element, attr))) flag = false;

}

//document.getElementById('test').innerHTML += i + '--' + parseInt(target) + '--' + parseInt(getStyle(element, attr)) + '--' + flag + '<br />';

}

if (flag) {

clearInterval(element.timer);

if (obj.fn != undefined) obj.fn();

}

}, t);

function setTarget() {

element.style[attr] = target + 'px';

}

function setOpacity() {

element.style.opacity = parseInt(target) / 100;

element.style.filter = 'alpha(opacity=' + parseInt(target) + ')';

}

}

return this;

};

//插件入口

Base.prototype.extend = function (name, fn) {

Base.prototype[name] = fn;

};

TOOL

//浏览器检测

(function () {

window.sys = {};

var ua = navigator.userAgent.toLowerCase();

var s;

(s = ua.match(/msie ([\d.]+)/)) ? sys.ie = s[1] :

(s = ua.match(/firefox\/([\d.]+)/)) ? sys.firefox = s[1] :

(s = ua.match(/chrome\/([\d.]+)/)) ? sys.chrome = s[1] :

(s = ua.match(/opera\/.\*version\/([\d.]+)/)) ? sys.opera = s[1] :

(s = ua.match(/version\/([\d.]+).\*safari/)) ? sys.safari = s[1] : 0;

if (/webkit/.test(ua)) sys.webkit = ua.match(/webkit\/([\d.]+)/)[1];

})();

//DOM加载

function addDomLoaded(fn) {

var isReady = false;

var timer = null;

function doReady() {

if (timer) clearInterval(timer);

if (isReady) return;

isReady = true;

fn();

}

if ((sys.opera && sys.opera < 9) || (sys.firefox && sys.firefox < 3) || (sys.webkit && sys.webkit < 525)) {

//无论采用哪种，基本上用不着了

/\*timer = setInterval(function () {

if (/loaded|complete/.test(document.readyState)) { //loaded是部分加载，有可能只是DOM加载完毕，complete是完全加载，类似于onload

doReady();

}

}, 1);\*/

timer = setInterval(function () {

if (document && document.getElementById && document.getElementsByTagName && document.body) {

doReady();

}

}, 1);

} else if (document.addEventListener) {//W3C

addEvent(document, 'DOMContentLoaded', function () {

fn();

removeEvent(document, 'DOMContentLoaded', arguments.callee);

});

} else if (sys.ie && sys.ie < 9){

var timer = null;

timer = setInterval(function () {

try {

document.documentElement.doScroll('left');

doReady();

} catch (e) {};

}, 1);

}

}

//跨浏览器添加事件绑定

function addEvent(obj, type, fn) {

if (typeof obj.addEventListener != 'undefined') {

obj.addEventListener(type, fn, false);

} else {

//创建一个存放事件的哈希表(散列表)

if (!obj.events) obj.events = {};

//第一次执行时执行

if (!obj.events[type]) {

//创建一个存放事件处理函数的数组

obj.events[type] = [];

//把第一次的事件处理函数先储存到第一个位置上

if (obj['on' + type]) obj.events[type][0] = fn;

} else {

//同一个注册函数进行屏蔽，不添加到计数器中

if (addEvent.equal(obj.events[type], fn)) return false;

}

//从第二次开始我们用事件计数器来存储

obj.events[type][addEvent.ID++] = fn;

//执行事件处理函数

obj['on' + type] = addEvent.exec;

}

}

//为每个事件分配一个计数器

addEvent.ID = 1;

//执行事件处理函数

addEvent.exec = function (event) {

var e = event || addEvent.fixEvent(window.event);

var es = this.events[e.type];

for (var i in es) {

es[i].call(this, e);

}

};

//同一个注册函数进行屏蔽

addEvent.equal = function (es, fn) {

for (var i in es) {

if (es[i] == fn) return true;

}

return false;

}

//把IE常用的Event对象配对到W3C中去

addEvent.fixEvent = function (event) {

event.preventDefault = addEvent.fixEvent.preventDefault;

event.stopPropagation = addEvent.fixEvent.stopPropagation;

event.target = event.srcElement;

return event;

};

//IE阻止默认行为

addEvent.fixEvent.preventDefault = function () {

this.returnValue = false;

};

//IE取消冒泡

addEvent.fixEvent.stopPropagation = function () {

this.cancelBubble = true;

};

//跨浏览器删除事件

function removeEvent(obj, type, fn) {

if (typeof obj.removeEventListener != 'undefined') {

obj.removeEventListener(type, fn, false);

} else {

if (obj.events) {

for (var i in obj.events[type]) {

if (obj.events[type][i] == fn) {

delete obj.events[type][i];

}

}

}

}

}

//跨浏览器获取视口大小

function getInner() {

if (typeof window.innerWidth != 'undefined') {

return {

width : window.innerWidth,

height : window.innerHeight

}

} else {

return {

width : document.documentElement.clientWidth,

height : document.documentElement.clientHeight

}

}

}

//跨浏览器获取滚动条位置

function getScroll() {

return {

top : document.documentElement.scrollTop || document.body.scrollTop,

left : document.documentElement.scrollLeft || document.body.scrollLeft

}

}

//跨浏览器获取Style

function getStyle(element, attr) {

var value;

if (typeof window.getComputedStyle != 'undefined') {//W3C

value = window.getComputedStyle(element, null)[attr];

} else if (typeof element.currentStyle != 'undeinfed') {//IE

value = element.currentStyle[attr];

}

return value;

}

//判断class是否存在

function hasClass(element, className) {

return element.className.match(new RegExp('(\\s|^)' +className +'(\\s|$)'));

}

//跨浏览器添加link规则

function insertRule(sheet, selectorText, cssText, position) {

if (typeof sheet.insertRule != 'undefined') {//W3C

sheet.insertRule(selectorText + '{' + cssText + '}', position);

} else if (typeof sheet.addRule != 'undefined') {//IE

sheet.addRule(selectorText, cssText, position);

}

}

//跨浏览器移出link规则

function deleteRule(sheet, index) {

if (typeof sheet.deleteRule != 'undefined') {//W3C

sheet.deleteRule(index);

} else if (typeof sheet.removeRule != 'undefined') {//IE

sheet.removeRule(index);

}

}

//跨浏览器获取innerText

function getInnerText(element) {

return (typeof element.textContent == 'string') ? element.textContent : element.innerText;

}

//跨浏览器设置innerText

function setInnerText(elememt, text) {

if (typeof element.textContent == 'string') {

element.textContent = text;

} else {

element.innerText = text;

}

}

//获取某一个元素到最外层顶点的位置

function offsetTop(element) {

var top = element.offsetTop;

var parent = element.offsetParent;

while (parent != null) {

top += parent.offsetTop;

parent = parent.offsetParent;

}

return top;

}

//删除左后空格

function trim(str) {

return str.replace(/(^\s\*)|(\s\*$)/g, '');

}

//某一个值是否存在某一个数组中

function inArray(array, value) {

for (var i in array) {

if (array[i] === value) return true;

}

return false;

}

//获取某一个节点的上一个节点的索引

function prevIndex(current, parent) {

var length = parent.children.length;

if (current == 0) return length - 1;

return parseInt(current) - 1;

}

//获取某一个节点的下一个节点的索引

function nextIndex(current, parent) {

var length = parent.children.length;

if (current == length - 1) return 0;

return parseInt(current) + 1;

}

//滚动条固定

function fixedScroll() {

window.scrollTo(fixedScroll.left, fixedScroll.top);

}

//阻止默认行为

function predef(e) {

e.preventDefault();

}

//创建cookie

function setCookie(name, value, expires, path, domain, secure) {

var cookieText = encodeURIComponent(name) + '=' + encodeURIComponent(value);

if (expires instanceof Date) {

cookieText += '; expires=' + expires;

}

if (path) {

cookieText += '; expires=' + expires;

}

if (domain) {

cookieText += '; domain=' + domain;

}

if (secure) {

cookieText += '; secure';

}

document.cookie = cookieText;

}

//获取cookie

function getCookie(name) {

var cookieName = encodeURIComponent(name) + '=';

var cookieStart = document.cookie.indexOf(cookieName);

var cookieValue = null;

if (cookieStart > -1) {

var cookieEnd = document.cookie.indexOf(';', cookieStart);

if (cookieEnd == -1) {

cookieEnd = document.cookie.length;

}

cookieValue = decodeURIComponent(document.cookie.substring(cookieStart + cookieName.length, cookieEnd));

}

return cookieValue;

}

//删除cookie

function unsetCookie(name) {

document.cookie = name + "= ; expires=" + new Date(0);

}

**//封装ajax**

function ajax(obj) {

var xhr = (function () {

if (typeof XMLHttpRequest != 'undefined') {

return new XMLHttpRequest();

} else if (typeof ActiveXObject != 'undefined') {

var version = [

'MSXML2.XMLHttp.6.0',

'MSXML2.XMLHttp.3.0',

'MSXML2.XMLHttp'

];

for (var i = 0; version.length; i ++) {

try {

return new ActiveXObject(version[i]);

} catch (e) {

//跳过

}

}

} else {

throw new Error('您的系统或浏览器不支持XHR对象！');

}

})();

obj.url = obj.url + '?rand=' + Math.random();

obj.data = (function (data) {

var arr = [];

for (var i in data) {

arr.push(encodeURIComponent(i) + '=' + encodeURIComponent(data[i]));

}

return arr.join('&');

})(obj.data);

if (obj.method === 'get') obj.url += obj.url.indexOf('?') == -1 ? '?' + obj.data : '&' + obj.data;

if (obj.async === true) {

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

callback();

}

};

}

xhr.open(obj.method, obj.url, obj.async);

if (obj.method === 'post') {

xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

xhr.send(obj.data);

} else {

xhr.send(null);

}

if (obj.async === false) {

callback();

}

function callback() {

if (xhr.status == 200) {

obj.success(xhr.responseText); //回调传递参数

} else {

alert('获取数据错误！错误代号：' + xhr.status + '，错误信息：' + xhr.statusText);

}

}

}