2013-05-21至今工作内容：

1. pd2.5部分前端开发（只开发设计页面java开发人员已转成jsp，可以自己修改，不用做交接）
2. pdhelper前端开发（只开发设计页面java开发人员已转成jsp，可以自己修改，不用做交接）
3. pd3.0 公共组件模块、数据中心模块、数据中心和coreplatform公用模块开发

pd3.0详细内容：

目前由我的widget 实例化时默认都会调用initialize方法，

公用模块：

目前实现CommonDialog、CommonGrid、CommonTabs、commonWizard、CommonTree、CommonFormField、CommonWidget.js、GUtil.js

1. CommonDialog主要实现弹出框、loading框、模拟alert、confirm
2. 弹出框可以通过Dialog.create(opts);

opts = {

id: ‘ts\_dialog, ts\_second\_dialog

title: '标题', //标题

contentType: 'html', //传内容类型html 、text、iframe默认html、

url: '地址', //传普通连接地址创建iframe框、传模块类动态创建些模块

widgetOpts: {},//当传模块类时创建模块对象时传的参数对象

height: 320,//显示宽度

width: 640//显示高度

}

1. Dialog.loading(text);
2. Dialog.alert(text)
3. Dialog.confirm(text, function(flag){

if(flag){ //点确认时执行

}else{//点取消时执行

}

})

1. Dialog.close(id)关闭指定dialog、如果创建Dialog没有指定id，关闭时也不用传id

核心代码介绍：

一般调用方式new CommonDialog(opts), 别一种方式CommonDialog.create(opts), 这里create内部调用的new CommonDialog(opts)

CommonDialog.create = function(){

var F = function(){};

F.prototype = CommonDialog.prototype;

F.prototype.constructor = CommonDialog;

return CommonDialog.apply(new F(), arguments);//相当于new CommonDialog(opts)

}

Initialize：function(id, url, title, width, height, opts){

//dialog宽高处理（未传宽高 设置默认值）

dialogList.addDialog(id, that)//把当前dialog放入dialogList容器，方便处理多个dialog

if(opts.contentType === 'html' || /\/([A-Z]\w\*?)$/.test(opts.url)){//创建包含widget dialog

this. appendWidget (opts);

}else if(opts.contentType === 'text'){//传文本

this.appendText(opts);

}else if(opts.contentType === 'confirm'){//模拟window的confirm函数

this.appendConfirm(opts);

}else if(opts.contentType === 'prompt'){ //模拟window的'prompt'函数

this.appendPrompt(opts);

}else if(opts.contentType === 'loading'){ //处理loading

this.appendLoading(opts);

}else{

this.appendIframe(opts);//创建iframe框

}

this.addMask();//添加遮罩

if(opts.animat !== false){//处理渐变显示与隐藏

setTimeout(function(){

that.$dialog.addClass('in');

that.$dialog.next('.modal-backdrop').addClass('in');

that.$dialog.focus();

}, 0);

}else{

that.$dialog.addClass('in');

that.$dialog.next('.modal-backdrop').addClass('in');

that.$dialog.focus();

}

$('body').css({overflow: 'hidden'});

}

appendWidget: function(opts){

var that = this,

$dialog = this.$dialog,

$dialogWidthElem = $dialog.find('.dialog-width'),

$dialogBody = $dialog.find('.modal-body');

$dialogWidthElem.css({'width': opts.width, 'margin-top': opts.top});

$dialogBody.css({height: opts.height, position: 'relative'});

//这里动态引入widget类

require([

opts.url

],function(Constructor){

var widgetOpts = opts.widgetOpts || {};

var widget = new Constructor(widgetOpts);//动态创建widget类

that.innerWidget = widget;

if(typeof that.onLoadCB === 'function' || that.opts.onLoadCB){

that.onLoadCB(widget);

};

widget.closeDialog = function(num){//为widget实例添加壮志关闭Dialog函数，

that.hide(num);

}

$dialogBody.html(widget.rootElement);//把widget模板插入到dialog Body中

if(widget.createCompleteCallback){//调用widget实例完成后回调函数

widget.createCompleteCallback();

}

});

},

CommonDialog.close = function(id, time){//关闭弹出框

var $dialog = [];

id = id || 'ts\_dialog';//如果不传id默认为’ts\_dialog’

if(id && id.nodeType == 1){//dialog DOM对象

$dialog = $(id);

id = $dialog.attr('id');

}

if(!$dialog.length){

id = id || 'ts\_dialog';

$dialog = $('#' + id);

}

if(!$dialog.length){

id = id || 'ts\_dialog\_second';

$dialog = $('#' + id);

}

if(!$dialog.length){

id = id || 'ts\_dialog\_third';

$dialog = $('#' + id);

}

if(time){

setTimeout(function(){

CommonDialog.close(id);

// $dialog.remove();

}, time);

return ;

}else{

var dialog = dialogList.getDialog(id);

if(dialog && dialog.opts.animat !== false){//关闭时渐变运来处理

$dialog.removeClass('in');

$dialog.next('.modal-backdrop').removeClass('in');

if(GUtil.support.transition){

$dialog.one(GUtil.support.transition.end, function(){

$dialog.next('.modal-backdrop').remove();

$dialog.remove();

dialogList.removeDialog(id);

var nextDialog = dialogList.getNextDialog();

if(nextDialog){//如果关闭此dialog， dialog容器内还有dialog则容器内最外层dialog遮罩显示出来，

nextDialog.$dialog.next('.modal-backdrop').show();

nextDialog.$dialog.next('.modal-backdrop').addClass('in');

}else{

$('body').css({overflow: ''});

}

});

}else{

$dialog.next('.modal-backdrop').remove();

$dialog.remove();

dialogList.removeDialog(id);

var nextDialog = dialogList.getNextDialog();

if(nextDialog){

nextDialog.$dialog.next('.modal-backdrop').show();

nextDialog.$dialog.next('.modal-backdrop').addClass('in');

}else{

$('body').css({overflow: ''});//弹出框时隐藏body滚动条

}

}

}else{

$dialog.next('.modal-backdrop').remove();

$dialog.remove();

dialogList.removeDialog(id);

var nextDialog = dialogList.getNextDialog();

if(nextDialog){

nextDialog.$dialog.next('.modal-backdrop').show();

nextDialog.$dialog.next('.modal-backdrop').addClass('in');

}else{

$('body').css({overflow: ''});

}

}

}

}

CommonDialog.collectReq = function(reqId, reqUrl){//收集请求

//一个请求包括reqId, reqUrl, reqCB，只有这三个都收集到一个请求才能收集完成

if(loading.reqTmpCollect.reqLoadingFun){//如果已收集reqCB只收集reqId, reqUrl

loading.collectReqId(reqId);// 收集reqId

loading.collectReqUrl(reqUrl) ;// 收集reqUrl

}else{

var text = '';

var time = 50;

var width = 100;

var height = 80;

commonLoading.collectReqLoadingFun(function(){;// 收集reqCB

var opts = {id: 'ts\_common\_loading', text: text, title: 'Loading', contentType: 'loading', width: width, height: height, top: 200, cancelCallback: null, animat: false};

if(time){

return setTimeout(function(){

CommonDialog.create(opts);

}, 50);

}

CommonDialog.create(opts);

});

commonLoading.collectReqId(reqId); // 收集reqId

commonLoading.collectReqUrl(reqUrl); // 收集reqUrl

}

}

2. CommonGrid组件封装了koGrid插件

参考：<http://knockout-contrib.github.io/KoGrid/#/api>

调用方式：var grid = new CommonGrid(gridOpts);

gridOpts = {

url: '', //数据接口

multiSelect: false,//是否多选

footerVisible: false,//是否显示翻页部分

params: {name: 'qqq'},//url后参数

afterSelectionChange: function(itemData, e){//itemData选中行数据

return true;

},//选中后执行函数

gridActions: [//按钮配置

{

text: i18n.getMsg('rename'),//文本

name: 'rename',//

iconClsName: 'glyphicon glyphicon-edit',//样式className

clickCB: function(){//点击按钮回调函数

var grid = this;//grid本身

var data = grid.getData();//获取选中数据

}

}

],

gridSearchCondition: [//搜索条件配置

{

text: i18n.getMsg('platformType'),//显示文本

type: 'select',//类型 input、select、checkbox

name: 'platformType'//name提交数据进的key值

}

],

columnDefs: [//grid列配置

{

field: 'hostName', //列对应是的数据

displayName: i18n.getMsg('lparName'),//页面显示内容

cellFormatter: function(data){//返回值为自定义显示内容，data为当前行数据

return data.hostName + 'sssssssss';

}

},

],

};

CommonGrid方法：

getData(), //获取选中数据

updateData(data, id), //更新if(itemData[id] === data[\*][id]){itemData = data[\*]}

updateDataByProp (key, val, newVal), //if(itemData[key] === val){itemData[key] = newVal}

addData(data)//data如果是object添加一条数据数据是[]添加多条数据

remveData(data, key)//根据key删除数据

核心代码：

initialize: function(opts){

//代码参考：<http://knockout-contrib.github.io/KoGrid/#/examples>（Server-Side Paging Example）

self.getConditionMap = function(){//搜索条件与模板绑定

var condArr = ko.toJS(self.gridSearchCondition);

var condMap = {};

var key = '';

var val = '';

condArr.forEach(function(cond){

if(cond.type === 'select'){//下拉菜单处理，可以参考ko下拉菜单绑定

if(cond.inputName){

// key = cond[cond.name].val;

key = cond.selectedOption[0];

val = cond[cond.inputName];

if(key == -1){

return

}

}else{

key = cond.name;

val = cond.selectedOption[0];

if(val == -1){

return

}

}

condMap[key] = val;

}else if(cond.type === 'input'){

key = cond.name;

val = cond[cond.aliasName || key];

condMap[key] = val;

}else if(cond.type === 'checkbox'){

key = cond.name;

val = [];

cond.data.forEach(function(item){

if(item.checked){

val.push(item.val);

}

});

condMap[key] = val;

// cond[cond.aliasName || cond.name] = ko.observable();

}

//button

});

return condMap;

};

if(this.opts.gridActions){//grid按钮绑定

this.opts.gridActions.forEach(function(itemData){

itemData.text = ko.observable(itemData.text);

itemData.clsName = ko.observable(itemData.clsName || '');

itemData.iconClsName = ko.observable(itemData.iconClsName);

});

this.vm.gridActions(this.opts.gridActions);

}

}

updateGridLayout: function(){//重绘grid布局

this.container.find('.koGrid').trigger('updateGridLayout');

},

getGridAction(name)//根据name获取按钮

disableActions

enableActions

hideActions

showActions

以上方法处理按钮状态，本质就是切换按钮的className, (主要在主机，lpar列表用到，不同平台显示不同按钮，主机lpar不同运行状态，禁用某些按钮)

1. CommonTabs选项卡组件

var tab = new CommonTabs({

clickTab: function(itemData){ //点击某个选项卡回调函数

var $tabItem = $('#' + itemData.id);

//根据name找到对应的widget,并放到此panel

if(itemData.name === 'PlatformList'){

var platformList = new PlatformList();

platformList.placeAt($tabItem);

}else if(itemData.name === 'HostList'){

}

}

});

tab.loadData([//加载选项卡个数

{title: i18n.getMsg('platformList'), name: 'PlatformList'},

{title: i18n.getMsg('hostList'), name: 'HostList'}

]);

核心代码：

initialize: function(opts){

……

self.tabList= ko.observableArray([]);//几个选项卡

self.clickTab = function(itemData, e){//点击选项卡回调函数（项目中此回调函数主要实现创建widget并放入此选项卡内）

self.tabList().forEach(function(tab){

tab.isActive(false);

});

itemData.isActive(true);

if(opts && opts.clickTab){

opts.clickTab.call(that, ko.mapping.toJS(itemData));

}

}

return self;

}

……

},

1. commonWizard向导组件就是一个容器放N个widget，然后一个一个的显示

var widgets = [{

widgetClass: 'xcat/host/HostLpartTemplate',//模块类

title: i18n.getMsg('selectInventory'),//显示title

loadingText: i18n.getMsg('findLparLoading'),//请求时显示

showWidgetCallback: function(wt){//进入此widget调用

wt.loadData(model.hostBaseInfo);

},

btns: {

// prev: false,//默认按钮、false不显示

next: false,//默认按钮、false不显示

// cancel: false,//默认按钮、false不显示

findHost: {//自定义按钮

text: '发现Lpar',//显示文本

callback: function(wt){

var that = this;

if(wt.validate() === false){//每个widget都不一个验证，false不执行

return ;

}

that.jumpStep('next');//跳转下一个widget

}

}

}

},

...

];

var wizard = new CommonWizard({

widgets: widgets

});

核心代码：

initialize: function(opts){

var that = this,

$container = this.container = $(this.rootElement),

$listItems = null;

this.stepsContainer = this.container.find('.steps');

this.stepsContentContainer = this.container.find('.step-content');

this.btnsContainer = this.container.find('.step-btns');

this.opts = $.extend({}, opts);

this.widgets = this.opts.widgets;

var widgets = this.widgets;

this.buildTab();//根据widget个数创建wizard

this.jumpStep(0);//默认显示第一个widget

this.container.on('click', '.complete', function(){

// that.currStep = $(this).index();

// that.jumpStep(that.currStep);

});

//点上一步按钮跳转到上一个widget

this.container.on('click', '.prev-btn', function(){

var currStep = that.currStep;

var widget = that.widgets[currStep];

if(widget.btns && widget.btns.prev){

//回调函数返回true才跳转到下步页面

if(widget.btns.prev(widget.widgetObj, currStep, that.widgets) === false){

return ;

}

}

that.jumpStep('prev');

});

//点下一步按钮跳转到下一个widget

this.container.on('click', '.next-btn', function(){

var currStep = that.currStep;

var widget = that.widgets[currStep];

//调用验证

if(that.validate(widget.widgetObj) === false){

console.log('验证未通过！');

return ;

}

if(widget.btns && widget.btns.next){

//回调函数返回true才跳转到下步页面

if(widget.btns.next.call(that, widget.widgetObj, currStep, that.widgets) === false){

return ;

}

}

that.jumpStep('next');

});

//点取消关闭此向导（目前向导都是嵌套在dialog内的，所以点取消一般是关闭dialog）

this.container.on('click', '.cancel-btn', function(){

var currStep = that.currStep;

var widget = that.widgets[currStep];

if(widget.btns && widget.btns.cancel){

//回调函数返回true才跳转到下步页面

if(widget.btns.next.call(that, widget.widgetObj, currStep, that.widgets) === false){

return ;

}

}

if(that.opts.cancelCB){

that.opts.cancelCB();

}

});

},

jumpStep: function(num){

if(num === 'prev'){

num = --this.currStep;

}

if(num === 'next'){

num = ++this.currStep;

this.hideLoading();

}

if(num < 0){

num = 0;

this.currStep = 0;

}

if(num > (this.widgets.length - 1)){

num = this.widgets.length - 1;

this.currStep = this.widgets.length - 1;

}

var that = this;

var widgets = this.widgets;

var widget = widgets[num];

var ConstructorPath = widget.widgetClass;

var widgetOpts = widget.widgetOpts || {};

this.stepsContentContainer.find('.step-pane').removeClass('active');

this.stepsContentContainer.find('.step-pane').eq(this.currStep).addClass('active');

if(!widget.widgetObj){//如果widget对象已存在不再重复创建

var type = $.type(ConstructorPath);

if(type === 'function'){//如果widgetClass是一个widget类

widget.widgetObj = new ConstructorPath(widgetOpts);

that.widgetObjInit(widget.widgetObj);

}else if(type === 'object'){//如果widgetClass是一个widget实例

widget.widgetObj = ConstructorPath;

that.widgetObjInit(widget.widgetObj);

}else if(type === 'string'){//如果widgetClass是一个widget类路径，重新加载并创建此widget

require([

ConstructorPath

],function(Constructor){

var obj = new Constructor(widgetOpts);

that.widgetObjInit(obj);

});

}

}else{

if(widget.showWidgetCallback){

widget.showWidgetCallback.call(that, widget.widgetObj, that.currStep, widgets);

}

}

this.refreshTabs();//切换当前widget状态

this.refreshBtns();//切换当前widget操作按钮

},

4．CommonTree树组件

var tree = new CommonTree(opts);

var opts = {

url: '/DataCenterAction.do?method=trees',//请求地址

iconCB: function(itemData){//定义节点图标

var iconClsName = itemData.type;

return iconClsName;

},

selectNodeCB: function(itemData){//选中节点时调用

var type = itemData.type;

if(type === 'dataCenter'){//显示数据中心模块

require(['ts/common/DataCenterSection'], function(DataCenterSection){

var dataCenterSection = new DataCenterSection();

dataCenterSection.placeAt($('.content-main'));

});

}else if(type === 'xcatPlatform'){//显示xcat模块

}

}

};

//lpar节点数据有变化时调用

tree.registerCB('节点id', 'updateData.lpar', function(data){

that.grid.updateData(data.dataset, 'lparPcid');

});

//删除lpar节点时调用

tree.registerCB('节点id', 'removeData.lpar', function(data){

that.grid.removeData(data.dataset, 'lparPcid');

});

//添加lpar节点时调用

tree.registerCB('节点id', 'addData.lpar', function(data){

that.grid.addData(data.dataset);

});

Tree.refreshNode(data.platformPcid || data.platformId)

核心代码：

只要请求数据都会调用此函数

updateData: function(node, newData){//newData为新节点

var that = this;

var currData = node.children();//当前所以节点

var removeDataArr = [];//要删除的节点

var addDataArr = [];//要新增的节点

var updateDataArr = [];//要更新的节点

var currDataMap = this.transformToMap(currData);

var newDataMap = this.transformToMap(newData);

var noUpdateProp = ['open', 'children'];

var typeNameSpace = '';

that.triggerCB(0, 'reloadData', node);//根节点都会触发（执行注册到根节点上的reloadData回调函数）

// that.triggerCB(node.id, 'reloadData', node);//触发重新加载数据

if(!currData.length){//如果currData,newData全部为新增的节点

node.children(newData);

addDataArr = newData;

addDataArr.forEach(function(item){

item.open(true);

});

addDataArr.forEach(function(item){

that.triggerCB(item.id, 'addData', item);//触发添加

});

}else{

currData.forEach(function(item){//获取要删除的节点

if(!newDataMap[item.id]){

removeDataArr.push(item);

}

});

newData.forEach(function(item){

if(!currDataMap[item.id]){//获取新增的节点

addDataArr.push(item);

}else{//其它为要删除的节点

updateDataArr.push(item);

}

});

//删除节点并执行注册到此节点上的删除回调

removeDataArr.forEach(function(item){

GUtil.arrayRemoveItem(node.children(), 'id', item.id);

that.triggerCB(item.id, 'removeData', item);//触发删除

});

//新添加的默认展开，让用户看到新添加了

addDataArr.forEach(function(item){

item.open(true);

});

//添加节点并执行注册到此节点上的添加回调

node.children.push.apply(node.children, addDataArr);

addDataArr.forEach(function(item){

that.triggerCB(item.id, 'addData', item);//触发添加

});

//更新节点并执行注册到此节点上的更新回调

updateDataArr.forEach(function(item){

var currNode = currDataMap[item.id];

var oldNode = $.extend(true, {}, currNode);

var val = '';

for(var prop in currNode){

if(GUtil.array.indexOf(noUpdateProp, prop) >= 0) continue;

// if(prop === 'open') continue;

val = currNode[prop];

if(typeof currNode[prop] === 'function'){

if(typeof item[prop] === 'function'){

val(item[prop]());

}else{

val(item[prop]);

}

}else{

if(typeof item[prop] === 'function'){

currNode[prop] = item[prop]();

}else{

currNode[prop] = item[prop];

}

}

}

that.triggerCB(item.id, 'updateData', item, oldNode);//触发更新

that.triggerCB(item.id, 'reloadData', item);//触发重新加载数据

});

}

},

1. CommonFormField表单组件（此组件引用了验证插件。参考：https://github.com/Knockout-Contrib/Knockout-Validation）

实现只需配置即可创建表单

var formField = new CommonFormField();

这里的formField和grid里的搜索条件类似，

formField.loadData([{

\_type: 'input',//创建表单字段类型,有text, select, textarea, password, checkbox, radio, button, input默认为input

\_name: 'username',//字段name属性值

\_val: '张三',//字段name属性值

\_text: '用户名',//字段前的描述

\_valid: {//对字段值进行验证

required: {//必填项及提示

message: i18n.getMsg('please enter the platform name'),

params: true

},

pattern: {//正则匹配及提示

message: i18n.getMsg('please type in the name of the platform the correct format'),

params: '^[\\w-]{3,}$'

},

validation: {//自定义验证

async: true,//是否异步验证

validator: function (val, params, callback) {

$.post( that.opts.checkNameUrl, {name: val}, function(data){

var isExsit = false;

var msg = i18n.getMsg('congratulations you can use this name');

if(data.flag === true){

isExsit = true;

msg = data.msg || i18n.getMsg('this name is already used');

}

//这里是核心传isVallid是验证是否成功，message验证后提示

callback({ isValid: !isExsit, message: msg});

});

},

message: i18n.getMsg('congratulations you can use this name')

}

}

}, {

\_isSubmitGroup: true,//是否在提交按钮组，默认false

\_type: 'button',

\_name: 'save',

\_text: i18n.getMsg('submit'),

\_clickCB: function(){//点击按钮回调

if(!this.validate()){//验证表单

return ;

}

var currData = formField.getData(originData);//获取表单数据

$.post(that.opts.url, currData, function(data){

if(data.flag){

}

});

}

}

]);

核心代码：

initialize: function(opts){

var that = this;

this.opts = $.extend(this.opts, opts);

var url = this.opts.url;

this.container = $(this.rootElement);

//配置验证

ko.validation.configure({

decorateInputElement: true,

registerExtenders: true,

messagesOnModified: true,

insertMessages: false,

parseInputAttributes: true,

errorClass: 'has-error'

});

ko.validation.configuration.insertMessages=false;

function FormFieldViewModel(){

var self = this;

self.fields = ko.observableArray([]);//form表单字段

self.submitGroup = ko.observableArray([]);//提交组按钮

self.cancelBtnIsShow = ko.observable(true);//是否显示取消按钮

}

this.vm = new FormFieldViewModel();

this.vm.errors = ko.validation.group(this.vm.fields);//表单验证错误容器

this.addEventListener("DOMNodeInserted",function(){

that.createCompleteCallback();

});

},

processData: function(data){

var that = this;

data.\_type = data.\_type || 'input';

switch(data.\_type){

case 'text'://文本框

data[data.\_name] = ko.observable(data.\_val);

break;

case 'select': //下拉菜单

data.\_options = ko.observableArray(data.\_options);

data[data.\_name] = ko.observable(data.\_selectedOption);

break;

case 'textarea': //多行文本框

data[data.\_name] = ko.observable(data.\_val);

break;

case 'password': //密码框

data[data.\_name] = ko.observable(data.\_val);

break;

case 'checkbox': //复选框

if(data.\_data){

data.\_data.forEach(function(o){

o.isChecked = ko.observable(o.isChecked || false);

});

}else{

data[data.\_name] = ko.observable();

}

break;

case 'radio': //单选

if(data.\_data){

data.\_checked = ko.observable(data.\_checked || '0');

}else{

data[data.\_name] = ko.observable();

}

break;

case 'button': //按钮

if(data.\_isSubmitGroup){//是否在提交按钮组

var tmpclickCB = data.\_clickCB || function(){};

if(tmpclickCB){

data.\_clickCB = function(){

tmpclickCB.apply(that, arguments);

}

}

data['\_clsName'] = ko.observable(data['\_clsName'] || 'btn-primary');

// }

break;

case 'submit':

break;

case 'input': //单行文本框，默认单行文本框

default:

data[data.\_name] = ko.observable(data.\_val);

if(data.\_valid){

data[data.\_name].extend(data.\_valid);

}

}

return data;

},

5.CommonWidget公共类

统一处理template和国际化、实现placeAt()

核心代码：

function CommonWidget(){

var that = this;

var opts = arguments[0] || {};

this.name = this.constructor.name || '';

this.container = null;

this.events = {};

if(opts && typeof opts === 'object'){

if(opts.json){

//this.i18n = new MessageBundle(opts.json);

this.i18n = Object.getPrototypeOf(this).i18n.createBranch(opts.json);

}

if(opts.html){

this.template = opts.html;

}

}

//上面取代每个widget里的代码

///SetProperties(EngineForm.prototype,DONT\_ENUM,[

// "template",htm,

// "i18n",i18n

// ]);

//每个widget里调用 CommonWidget.call(this, {html: html, css: css, json: null});

\_\_super\_\_.constructor.call(this);

this.initial.apply(this, arguments);

if(this.name){

var name = 'A\_' + this.name;//A\_可以在输入window时头部显示

window[name] = this;

}

}

SetProperties(EngineForm.prototype,DONT\_ENUM,[

"template",htm,

"i18n",i18n

]);

placeAt: function(elem, pos, CB){//重写父类placeAt以更方便使用

var $elem = $(elem);

var $container = this.container;

pos = pos || 'replaceInner';

pos = pos.toLocaleLowerCase();

var posMap = {//兼容父类placeAt方法

before: 'before',

beforebegin: 'before',

after: 'after',

afterend: 'after',

append: 'append',

afterbegin: 'prepend',

prepend: 'prepend',

beforeend: 'append',

replaceinner: 'replaceInner',

replace: 'replace',

};

if(pos === 'replace'){

$elem.replaceWith($container);

}else if(posMap[pos] === 'before'){

$elem.before($container);

}else if(posMap[pos] === 'after'){

$elem.after($container);

}else if(posMap[pos] === 'append'){

$elem.append($container);

}else if(posMap[pos] === 'prepend'){

$elem.prepend($container);

}else if(posMap[pos] === 'replaceInner'){

$elem.html('');

$elem.append($container);

}else if(posMap[pos] === 'replace'){

$elem.replaceWith($container);

}

var that = this;

setTimeout(function(){

that.trigger();

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

CB.call(that);

}

}, 0);

},

6.GUtil公共类重写jquery $.get和$.post方法 拦截错误、拦截未登录、模拟接口真实接口映射、请求时间超过500毫秒自己显示loading

重写jquery $.get和$.post方法

jQuery.each( [ "get", "post" ], function( i, method ) {

jQuery[ method ] = function( url, data, callback, type ) {

if ( jQuery.isFunction( data ) ) {

type = type || callback;

callback = data;

data = {};

}

return jQuery.ajax({

url: url,

type: method,

dataType: type,

data: data,

success: callback,

error: function(xhr, textStatus, errorThrown){//统一拦截错误

// var callback = this.success;

Dialog.removeCollectReq(callback.reqId);//删除请求

that.failIntercept(xhr, textStatus, errorThrown, callback.reqBeforeUrl, callback.url);//拦截错误

}

});

};

});

$.get({success: function(){ssss}})

Xxxx(data);

Success(data);

jQuery.each(['get', 'post'], function(i, method) {

//$.get 和$.post之前执行

that.AOP.before(jQuery, method, function(url, data, callback, type){

url = window.GUtil.getRootPath() + url;

if(jQuery.isFunction(data)){

type = type || callback;

callback = data;

data = {};

}

callback = callback || function(){};

var reqId = window.GUtil.getUUID();

var reqBeforeUrl = url;

var reqBeforeData = data;

var success = callback;

if(that.repeatReqIntercept(url, data)){//拦截重复请求

return false;

}

$.get(url, {data:ddd})

reqBeforeData = data;

{data: }

if(data){

data = that.buildJsonData(data);//创建json数据

}

url = that.interfaceIntercept(url);//拦截是用真实接口还是模拟接口

that.printReqData(url, data, 'before');//请求前打印

Dialog.collectReq(reqId, url);

success.reqId = reqId;

success.reqBeforeUrl = reqBeforeUrl;

success.url = url;

var o = {success: success};

that.AOP.before(o, 'success', function(data){//ajax成功回调之前执行

Dialog.removeCollectReq(success.reqId, url);//删除请求

if(that.errorIntercept(data)){//自定义错误拦截

if(callback.errorCB){

callback.errorCB(data);

}

return ;

}

//登录拦截

if(that.loginIntercept(url, reqBeforeUrl, reqBeforeData, callback, type, data)) return ;

that.printReqData(url, data, 'after'); //请求后打印

});

o.success.reqId = reqId;

o.success.reqBeforeUrl = reqBeforeUrl;

o.success.url = url;

arguments[0] = url;

arguments[1] = data;

arguments[2] = o.success;

var args = Array.prototype.slice.call(arguments, 0);

args[1] = args[1] || data;//可能不会传此参数

args[2] = args[2] || o.success;

args[3] = args[3] || 'json';

return args;

});

});

数据中心模块：

平台管理（ 注册xcat平台、注册ivm平台 、注册主机 、注册Lpar 、重命名、 删除 ）

主机管理（ 注册lpar、新建Lpar、 批量创建Lpar 、重命名 、编辑IP、 资源分配情况、 移除 、启动 、关闭、 重启 、配置主机间SSH信任、刷新）

分区管理（重命名、 编辑IP 、删除、 查看配置、 启动 、关闭、 重启、 安装系统 、关联系统映像、 配置分区间SSH信任 ）

这里主要看

1.注册平台主机分区xcat/host/RegisterPHLWizard

2.通用的form表单

3.注册刷新树和刷新树时主机或分区状态变化

4.摘要界面的基本信息

5.和coreplatform相关的主要就是task完成树刷新

数据中心入口

1. 双击数据中心app
2. 加载DataCenterApp，并实例化，生成数据中心页面
3. 加载并实例化CommonTree（加载数据中心、平台、主机、分区节点）
4. 点击数据中心节点，加载DataCenterSection并实例化，
   1. 加载数据中心摘要（CommonBaseInfo、DataCenterActions）、平台列表（PlatformList）、主机列表（HostList）、分区列表（LparList）、日志列表（LogList）
5. 点击平台节点加载PlatformSection并实例化
   1. 加载平台摘要（CommonBaseInfo、CommonActions）、主机列表（HostList）、分区列表（LparList）、日志列表（LogList）

6. 点击主机节点加载HostSection并实例化

加载主机摘要（CommonBaseInfo、CommonActions）、资源分配情况（CommonBaseInfo）、分区列表（LparList）、日志列表（LogList）

7. 点击分区节点加载LparSection并实例化

加载分区摘要（CommonBaseInfo、CommonActions）网卡列表（NetworkCardList

）、存储列表（StorageList）、日志列表（LogList）

8. 平台列表（PlatformList）

加载CommonGrid并实例化，CommonGrid配置的gridActions包括（注册xcat平台（RegisterPHLWizard）、注册ivm平台（RegisterIvmWizard） 、注册主机（RegisterXcatHostWizard） 、注册Lpar（PlatformRegisterLpartWizard） 、重命名（EditPlatform）、 删除(删除grid一条数据) ），根据相应的按钮可以找到相应widget类

9. 主机列表（HostList）

（ 注册lpar(RegisterLpartWizard)、新建Lpar(CreateLparWizard)、 批量创建Lpar(CreateBatchLpar)、重命名 (EditHost)、编辑IP(EditHostIP)、 资源分配情况(ResAssignInfo)、 移除(无widget类) 、启动(无widget类) 、关闭(无widget类)、 重启 (无widget类)、配置主机间SSH信任(SetupHostTrust)、刷新(无widget类)

10. 分区列表（LparList）

（重命名(EditLpar)、 编辑IP (EditLparIP)、删除()、 查看配置(LparConfigInfo)、 启动 ()、关闭()、 重启()、 安装系统 (InstallOSWizard)、关联系统映像(RelevanceOSImage)、 配置分区间SSH信任(SetupLparTrust) ）

11. 注册xcat平台（RegisterPHLWizard）

注册平台(RegisterPlatform)、选择清单(HostTemplate)、选择主机(XcatHostList)、主机注册确认(RegisterHostConfirm)、注册分区(RegisterLPart)、分区确认(RegisterLPartConfirm)、配置分区(ConfigLpar)、配置分区确认(ConfigLparConfirm)

向导页面找子widget 只要再配置里找widgetClass就OK