pd3.0详细内容：

目录

[一、 整体流程介绍、数据中心流程介绍（已完成） 3](#_Toc399691626)

[1.1整体流程： 3](#_Toc399691627)

[1.2数据中心流程 3](#_Toc399691628)

[二、 公共组件核心代码介绍（已完成） 4](#_Toc399691629)

[1.CommonDialog主要实现弹出框、loading框、模拟alert、confirm 4](#_Toc399691630)

[1.1简要介绍 4](#_Toc399691631)

[1.2核心代码介绍： 5](#_Toc399691632)

[2. CommonGrid组件封装了koGrid插件 9](#_Toc399691633)

[2.1简要介绍 9](#_Toc399691634)

[2.2核心代码介绍 10](#_Toc399691635)

[3. CommonTabs选项卡组件 12](#_Toc399691636)

[3.1简要介绍 12](#_Toc399691637)

[3.2核心代码介绍 13](#_Toc399691638)

[4. commonWizard向导组件就是一个容器放N个widget，然后一个一个的显示 13](#_Toc399691639)

[4.1简要介绍 13](#_Toc399691640)

[4.2核心代码介绍： 14](#_Toc399691641)

[5．CommonTree树组件 17](#_Toc399691642)

[5.1简要介绍 17](#_Toc399691643)

[5.2核心代码介绍 18](#_Toc399691644)

[6. CommonFormField表单组件（此组件引用了验证插件。参考：https://github.com/Knockout-Contrib/Knockout-Validation） 20](#_Toc399691645)

[6.1简要介绍 20](#_Toc399691646)

[6.2核心代码介绍： 22](#_Toc399691647)

[7.CommonWidget公共类 24](#_Toc399691648)

[7.1核心代码介绍： 24](#_Toc399691649)

[8. 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 26](#_Toc399691650)

[8.1核心代码介绍： 26](#_Toc399691651)

[三、 数据中心核心代码介绍（完成部分） 29](#_Toc399691652)

[1. DataCenterApp核心代码 29](#_Toc399691653)

[2. DataCenterSection核心代码：（PlatformSection、HostSection、LparSection代码类似） 31](#_Toc399691654)

[3. PlatformList 核心代码： 34](#_Toc399691655)

[4. HostList核心代码： 37](#_Toc399691656)

[5. LparList核心代码和HostList类似参考上面 41](#_Toc399691657)

[6. 注册平台主机分区(RegisterPHLWizard)核心代码 41](#_Toc399691658)

[6.1注册平台(RegisterPlatform)widget 49](#_Toc399691659)

[6.2选择模板(HostTemplate)widget 52](#_Toc399691660)

[6.3注册主机(XcatHostList) widget 54](#_Toc399691661)

[6.4注册主机确认(RegisterHostConfirm) widget 56](#_Toc399691662)

[6.5注册分区 (RegisterLPart) widget 56](#_Toc399691663)

[6.6注册分区确认(RegisterLPartConfirm) widget 57](#_Toc399691664)

[6.7配置分区(ConfigLpar) widget 57](#_Toc399691665)

[6.8配置分区确认(ConfigLparConfirm) widget 57](#_Toc399691666)

[7. 注册主机分区(RegisterXcatHostWizard)核心代码(是RegisterPHLWizard的一部分) 58](#_Toc399691667)

[8. 注册分区(RegisterLpartWizard)核心代码（是RegisterPHLWizard的一部分） 58](#_Toc399691668)

[四、 Coreplatform主要介绍（AppNav、Task模块）（未完成） 58](#_Toc399691669)

[1. AppNav即一个列表 跟PlatformList差不多 58](#_Toc399691670)

[2. Task核心代码 58](#_Toc399691671)

[五、 相关demo（完成部分）(目录：asserts/demo) 61](#_Toc399691672)

# 整体流程介绍、数据中心流程介绍（已完成）

## 1.1整体流程：

打开index.html页面根据requirejs机制默认加载index/main.js

Mani.js时判断如果未登录加载UserLogin widget, 登录加载（coreplatform/index/Index）

Index实例化时加载(任务栏（Taskbar）、应用导航(AppNav)、 任务进度(Task)）

## 1.2数据中心流程

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. DataCenterApp会 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

# 公共组件核心代码介绍（已完成）

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

### 1.1简要介绍

1. 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

### 1.2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

### 2.1简要介绍

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

### 2.2核心代码介绍

：

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)

## 3. CommonTabs选项卡组件

### 3.1简要介绍

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'}

]);

### 3.2核心代码介绍

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;

}

……

},

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

### 4.1简要介绍

var widgets = [{

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

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

loadingText: i18n.getMsg('findLparLoading'),//请求500ms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

});

### 4.2核心代码介绍：

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

},

## 5．CommonTree树组件

### 5.1简要介绍

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)

### 5.2核心代码介绍

：

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

});

}

},

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

### 6.1简要介绍

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){

}

});

}

}

]);

### 6.2核心代码介绍：

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;

},

## 7.CommonWidget公共类

### 7.1核心代码介绍：

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);

},

## 8. 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

### 8.1核心代码介绍：

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;

});

});

# 数据中心核心代码介绍（完成）

## DataCenterApp核心代码

initialize: function(opts){

var that = this;

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

this.container = $(this.rootElement);

var opts = {

url: that.opts.url || '/DataCenterAction.do?method=trees',//请求树地址

iconCB: function(itemData){//根据节点数据type 生成图标className

var iconClsName = itemData.type;

var hostRunningStatusMap = {//主机不同状态显示相应图标

0: 'closed',

1: 'running',

2: 'unknown',

3: 'starting',

4: 'closing',

5: 'restarting'

};

var lparRunningStatusMap = {//lpar不同状态显示相应图标

0: 'unknown',

1: 'running',

2: 'closed',

3: 'suspended',

4: 'unavailable',

5: 'starting',

6: 'closing',

7: 'migrating',

8: 'deleting',

9: 'suspending',

10: 'reconfiguring',

11: 'openfirmware'

};

if(itemData.type === 'xcatPlatformHost' || itemData.type === 'ivmPlatformHost'){

if(itemData.dataset){

iconClsName = 'host';

iconClsName += GUtil.capitalize((hostRunningStatusMap[itemData.dataset.runningStatus] || ''));

}

}

if(itemData.type === 'xcatPlatformHostLpar' || itemData.type === 'ivmPlatformHostLpar'){

if(itemData.dataset){

iconClsName = 'lpar';

iconClsName += GUtil.capitalize((lparRunningStatusMap[itemData.dataset.runningStatus] || ''));

}

}

return iconClsName;

},

selectNodeCB: function(itemData){

var type = itemData.type;

var data = itemData.dataset;

window.itemData = itemData;

if(type === 'dataCenter'){//选择dataCenter节点加载并创建DataCenterSection

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

var dataCenterSection = new DataCenterSection();

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

});

}else if(type === 'xcatPlatform'){//选择xcatPlatform节点加载并创建PlatformSection

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

data.platformName = data.name;

var PlatformSection = new PlatformSection(data);//传当前节点数据

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

});

}else if(type === 'xcatPlatformHost'){//同上

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

data.hostName = data.hostname;

var HostSection = new HostSection(data);

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

});

}else if(type === 'xcatPlatformHostLpart'){//同上

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

var LpartSection = new LpartSection(data);

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

});

}else if(type === 'ivmPlatform'){//同上

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

data.platformName = data.name;

data.projectName = 'ivm';//用于区分xcat 和ivm

var PlatformSection = new PlatformSection(data);

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

});

}else if(type === 'ivmPlatformHost'){//同上

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

data.hostName = data.hostname;

data.projectName = 'ivm';//用于区分xcat 和ivm

var HostSection = new HostSection(data);

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

});

}else if(type === 'ivmPlatformHostLpart'){//同上

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

data.projectName = 'ivm';//用于区分xcat 和ivm

var LpartSection = new LpartSection(data);

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

});

}

}

};

var tree = new CommonTree(opts);

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

that.createCompleteCallback();

tree.placeAt(this.container.find('.tree-container'), 'replace');

window.treePool['dataCenter'] = tree;//放入treePoll里以便任务完成找到此树，并刷新树

});

},

## 2. DataCenterSection核心代码：（PlatformSection、HostSection、LparSection代码类似）

initialize: function(opts){

var that = this;

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

this.container = $(this.rootElement);

this.tabsContainer = this.container.find('.tabs-container');

var dataCenterBaseInfo = null;

var tabContentObj = this.tabContentObj = {};//用于缓存各标签页创建的widget，以防重复创建

//创建tabs widget

var tab = new CommonTabs({clickTab: function(itemData){//点击tab时回调，引入相应widget

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

//根据name创建widget

if(itemData.name === 'DataCenterBaseInfo' && !tabContentObj.dataCenterBaseInfo){

$tabItem.html('');

//创建基本信息

tabContentObj.dataCenterBaseInfo = new CommonBaseInfo({url: '/DataCenterAction.do?method=basicInfo'});

tabContentObj.dataCenterBaseInfo.placeAt($tabItem, 'append');

//创建摘要Actions（按钮）

tabContentObj.DataCenterActions = new DataCenterActions({

clickItemCallback: function(itemData, e){//点击按钮加载相应widget

var name = itemData.name;

if(name === 'RegisterPHLWizard'){

Dialog.create({title: i18n.getMsg('registerXcatPlatform'), url: 'xcat/host/RegisterPHLWizard', widgetOpts:{platformId: '', platformPcid: '', platformType: 3}, width: 1060, height: 450 });

}else if(name == 'RegisterIvmPlatform'){

Dialog.create({title: i18n.getMsg('registerIvmPlatform'), url: 'ivm/host/RegisterIvmWizard', widgetOpts: {platformId: '', platformPcid: '', platformName: '', gridWidget: grid, wizardType: 'platform'}, width: 1060, height: 600 });

}

}

});

tabContentObj.DataCenterActions.placeAt($tabItem, 'append');

}else if(itemData.name === 'PlatformList' && !tabContentObj.platformList){

//创建平台列表

tabContentObj.platformList = new PlatformList();

tabContentObj.platformList.placeAt($tabItem);

}else if(itemData.name === 'HostList' && !tabContentObj.hostList){

//创建主机列表

tabContentObj.hostList = new HostList({url: '/DataCenterAction.do?method=hostList'});

tabContentObj.hostList.placeAt($tabItem);

}else if(itemData.name === 'LparList' && !tabContentObj.lparList){

//创建分区列表

tabContentObj.lparList = new LparList({

url: '/DataCenterAction.do?method=lparList',

type: 'dataCenter'//数据中心下的lparList, 还有platform平台下lparlist, host主机下lparlist, 因为不同xxx下的lparlist的搜索条件是不一样的

//projectName: projectName,//PlatformSection、HostSection还要指定是xcat还是ivm

//params: platformData// PlatformSection下传platformData， HostSection下传HostData

});

tabContentObj.lparList.placeAt($tabItem);

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

//创建日志列表

tabContentObj.log = new LogList({

url: '/LogAction.do?method=list',

entityType: ['PLATFORM', 'HOST', 'LPAR']

});

tabContentObj.log.placeAt($tabItem);

}

}});

tab.loadData([//加载tabs widget Data

{title: i18n.getMsg('summary'), name: 'DataCenterBaseInfo'},

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

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

{title: i18n.getMsg('lparList'), name: 'LparList'},

{title: i18n.getMsg('logList'), name: 'LogList'},

]);

tab.placeAt(this.tabsContainer);

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

that.createCompleteCallback();

});

},

createCompleteCallback: function(){//当前widget创建完毕调用，给树注册回调

var that = this;

var dataCenterTree = GUtil.tree.get('dataCenter');//获取数据中心树

//给根节点注册重reloadData回调，因为操作平台，主机，分区，数据中心摘要信息大多会变化

dataCenterTree.registerCB(0, 'reloadData', function(data){

that.tabContentObj.dataCenterBaseInfo.reloadData();//重新请求数据中心摘要信息

});

// ko.applyBindings(this.vm, this.container[0]);

},

## 3. PlatformList 核心代码：

initialize: function(opts){

var that = this;

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

this.container = $(this.rootElement);

this.projectName = this.opts.projectName || '';

this.packageName = '';

this.interfacePrefix = '';

this.setProject(this.projectName);

var gridOpts = {

url: that.opts.url || '/DataCenterAction.do?method=platformList',

multiSelect: false,//单选

afterSelectionChange: function(itemData, e){//选中行回调

var platformType = '';

var platformTypeMap = {

0: 'IVM',

1: 'HMC',

2: 'XCAT'

};

if(itemData.selected()){

platformType = itemData.entity.type;

//根据IVM和XCAT显示不同按钮

if(platformTypeMap[platformType] == 'IVM'){

grid.hideActions('projectName', 'xcat');

grid.showActions('projectName', 'ivm');

that.setProject('ivm');

}else if(platformTypeMap[platformType] == 'XCAT'){

grid.hideActions('projectName', 'ivm');

grid.showActions('projectName', 'xcat');

that.setProject('xcat');

}

}

return true;

},

gridActions: [//按钮配置

{

text: i18n.getMsg('registerXcatPlatform'),

iconClsName: 'glyphicon glyphicon-plus',

clickCB: function() {

Dialog.create({

title: i18n.getMsg('registerXcatPlatform'),

url: 'xcat/host/RegisterPHLWizard',

widgetOpts: {

platformId: '',

platformPcid: '',

platformType: '',

gridWidget: grid//传当前grid，方便registerXcatPlatform使用

},

width: 1060,

height: 450

});

}

}, {

text: i18n.getMsg('delete'),

iconClsName: 'glyphicon glyphicon-minus-sign',

clickCB: function() {

var grid = this;

var gridData = grid.getData();

if (gridData.length === 0) {

Dialog.alert(i18n.getMsg('please choose to delete records'));

return;

}

Dialog.confirm(i18n.getMsg('unrecoverable deletion, sure to delete?'), function (bool) {

if(bool){

$.post(that.interfacePrefix + 'PlatformAction.do?method=remove', gridData, function(data){

if(data.flag === '1'){

grid.removeData();//删除选中数据

//因删除平台不是任务所以再这里手动删除

GUtil.tree.get('dataCenter').refreshNode(gridData[0].platformPcid);

Dialog.alert(i18n.getMsg('delSucc'));

}

});

}

});

}

}

,{

text: i18n.getMsg('exportPlatformData'),

iconClsName: 'glyphicon glyphicon-export',

clickCB: function() {

var grid = this;

var gridData = grid.getData();

if (gridData.length === 0) {

Dialog.alert(i18n.getMsg('please select to edit the record'));

return;

}

var platformPcid = gridData[0].platformPcid;

var url = that.interfacePrefix + 'PlatformAction.do?method=exportPlatformInfo&platformPcid=' + platformPcid;

var $iframe = $('<iframe src="'+ url +'" style="display:none; visibility:hidden;"></iframe>');

$('body').append($iframe);

setTimeout(function(){

$iframe.remove();//请求后就会下载文件，1秒后删除iframe以防内存泄漏

$iframe = null;

}, 1000);

}

}

],

};

var grid = this.grid = new CommonGrid(gridOpts);

grid.placeAt(this.container, 'append');

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

that.createCompleteCallback();

});

},

createCompleteCallback: function(){

var that = this;

var dataCenterTree = GUtil.tree.get('dataCenter');

// dataCenterTree.removeCB(0, 'reloadData');

dataCenterTree.removeCB(0, 'updateData.platform');//重新创建数据中心下的PlatformList下，删除回调

dataCenterTree.removeCB(0, 'removeData.platform');//同上

// dataCenterTree.registerCB(0, 'reloadData', function(data){

// that.grid.reloadData();

// });

//注册更新平台回调

dataCenterTree.registerCB(0, 'updateData.platform', function(data){

if(!data) return ;

var dataset = data.dataset;

dataset.type = dataset.platformType;

that.grid.updateData(dataset, 'platformPcid');//根据platformPcid更新data，如修改平台名称可以更新相应平台名称

});

//注册删除平台回调, 删除平台时执行

dataCenterTree.registerCB(0, 'removeData.platform', function(data){

if(!data) return ;

// data.platformPcid = data.platformPcid || data.id;

that.grid.removeData(data.dataset);

});

//注册添加平台回调， 新增平台时执行

dataCenterTree.registerCB(0, 'addData.platform', function(data){

if(!data) return ;

// data.platformPcid = data.platformPcid || data.id;

dataset.type = dataset.platformType;

that.grid.addData(data.dataset);

});

},

## 4. HostList核心代码：

initialize: function(opts){

var that = this;

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

this.container = $(this.rootElement);

this.projectName = this.opts.projectName || '';

this.packageName = '';

this.interfacePrefix = '';

this.setProject(this.projectName);//设置当前时xcat还是ivm

this.runningStatusMap = {//运行状态

0: 'closed',

1: 'running',

2: 'unknown',

3: 'starting',

4: 'closing',

5: 'restarting',

9: 'deleting'

};

this.runningStatusMapInversion=GUtil.mapInversion(this.runningStatusMap);//key value交换

var gridOpts = {

url: that.opts.url,

params: that.opts.params,

multiSelect: this.projectName ? true : false,

afterSelectionChange: function(itemData, e){

if(itemData.selected()){

var runningStatus = itemData.entity.runningStatus;

var isExistMgrLpar = itemData.entity.isExistMgrLpar;

//根据运行状态和是否存在管理节点设置按钮状态

that.setActionsStatus(grid, runningStatus, isExistMgrLpar);

}

if(itemData.selected()){

var platformType = '';

var platformTypeMap = {

0: 'IVM',

1: 'HMC',

2: 'XCAT'

};

platformType = itemData.entity.platformType;

//根据选择的不同平台切换xcat或ivm

that.changeProject(platformTypeMap[platformType].toLocaleLowerCase());

}

// return true;

},

};

if(opts.type === 'platform'){

// gridOpts.gridActions.splice(1, 1);//删除创建ivm主机， 暂时写法

gridOpts.gridSearchCondition.splice(0, 1);//删除平台类型 暂时写法

}else if(opts.type === 'coreplatform'){

}

var grid = this.grid = new CommonGrid(gridOpts);

grid.placeAt(this.container, 'append');

this.changeProject(this.projectName);

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

that.createCompleteCallback();

});

},

createCompleteCallback: function(){

var that = this;

var id = '0';

var dataCenterTree = GUtil.tree.get('dataCenter');

// if(that.opts.type === 'coreplatform'){

// id = '0';

// }else

if(that.opts.type === 'platform'){

id = that.opts.params.platformPcid;

}

// dataCenterTree.removeCB(0, 'reloadData');

dataCenterTree.removeCB(id, 'updateData.host');//重新XXXSection下的HostList下，删除回调

dataCenterTree.removeCB(id, 'removeData.host');//同上

dataCenterTree.removeCB(id, 'addData.host');//同上

// dataCenterTree.registerCB(id, 'reloadData', function(data){

// that.grid.reloadData();

// });

//注册更新主机回调

dataCenterTree.registerCB(id, 'updateData.host', function(data){

if(!data) return ;

data.hostPcid = data.hostPcid || data.id;

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

that.setActionsStatus(that.grid, data.dataset.runningStatus);

});

//注册删除主机回调，删除主机时执行

dataCenterTree.registerCB(id, 'removeData.host', function(data){

if(!data) return ;

data.hostPcid = data.hostPcid || data.id;

that.grid.removeData(data.dataset);

});

//注册添加主机回调，新增主机时执行

dataCenterTree.registerCB(id, 'addData.host', function(data){

if(!data) return ;

data.hostPcid = data.hostPcid || data.id;

that.grid.addData(data.dataset);

});

//注册更新平台回调，这里主机是更新平台名称时 hostList里的平台名称也要更新

dataCenterTree.registerCB(id, 'updateData.platform', function(data, oldData){

if(!data) return ;

that.grid.updateDataByProp('platformName', oldData.dataset.name, data.dataset.name);

});

},

setActionsStatus: function(widget, runningStatus, isExistMgrLpar){//设置按钮状态

//所以按钮

var allActions = ['rename', 'delete', 'start', 'createLpar', 'createBatchLpar', 'close', 'restart', 'editIP', 'refresh'];

var disableActions = [];//禁用按钮

widget.enableActions('name', allActions)//设置按钮可用

switch(this.runningStatusMap[runningStatus]){//根据运行状态设置禁用用按钮

case 'closed':

widget.disableActions('name', ['createLpar', 'close', 'restart', 'editIP', 'refresh']);

break;

case 'running':

disableActions = ['start'];

if(isExistMgrLpar){

disableActions.push('close', 'restart');

}

widget.disableActions('name', disableActions);

break;

case 'unknown':

widget.disableActions('name', ['createLpar', 'editIP']);

break;

case 'starting':

widget.disableActions('name', ['createLpar', 'start', 'close', 'restart', 'editIP', 'refresh']);

break;

case 'closing':

widget.disableActions('name', ['createLpar', 'start', 'close', 'restart', 'editIP', 'refresh']);

break;

case 'restarting':

widget.disableActions('name', ['createLpar', 'start', 'close', 'restart', 'editIP', 'refresh']);

break;

}

},

## 5. LparList核心代码和HostList类似参考上面

## 6. 注册平台主机分区(RegisterPHLWizard)核心代码

initialize: function(opts){

var that = this;

this.container = $(this.rootElement);

//存储各widget Data

var model = {

platformInfo: {

platformType: '',

platformName: '',

ip: '',

username: '',

userpass: '',

confirmUserpass: ''

},

hostBaseInfo: {

platformId: '',

platformPcid: '',

platformName: '',

file: ''

},

hostListInfo: [],

RegisterHostConfirm: [],

RegisterLPart: [{

lpars:[]

}],

RegisterLPartConfirm: []

};

//注册xcat平台,如果不传值会下拉菜单可选平台类型

model.platformInfo.platformId = opts.platformId;//因前期平台id设计不合理，platformId、platformPcid都存在，这样不管用哪个都可以

model.platformInfo.platformPcid = opts.platformPcid;

model.platformInfo.platformType = opts.platformType;

var widgets = this.widgets = [//widget容器

{widgetClass: 'xcat/host/RegisterPlatform',//widget类路径，widget类， widget类实例

widgetOpts: {},// new RegisterPlatform时传此参数

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

loadingText: i18n.getMsg('registerPlatformLoading'),//请求超过500ms时显示

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

wt.loadData(model.platformInfo);//为当前widget加载数据

},

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: i18n.getMsg('register'),//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 ;

}

model.platformInfo = wt.getData(Object.keys(model.platformInfo));//获取注册平台数据

//注册平台，成功后跳转到下一个widget

this.req('xcat/XCATPlatformAction.do?method=registePlatform', model.platformInfo, function(data){

model.hostBaseInfo.platformId = data.platformId;

model.hostBaseInfo.platformPcid = data.platformPcid || data.platformId;

model.hostBaseInfo.platformName = data.platformName;

that.jumpStep('next');//跳转下一个widget

//注册成功刷新树，因注册platform是任务，所以任务完成刷新树

// GUtil.tree.get('dataCenter').refreshNode(data.platformPcid, '0');

});

}

}

}

},

{widgetClass: 'xcat/host/HostTemplate',

title: i18n.getMsg('selectInventory'),

loadingText: i18n.getMsg('findHostLoading'),

showWidgetCallback: function(wt){

wt.loadData(model.hostBaseInfo);

},

btns: {

next: false,

findHost: {

text: i18n.getMsg('findHost'),

callback: function(wt){

var that = this;

if(GUtil.isRealInterface()){//是真实接口

if(wt.validate() === false){

return ;

}

$.extend(model.hostBaseInfo, wt.getData(Object.keys(model.hostBaseInfo)));

this.showLoading();

wt.req('xcat/XCATHostAction.do?method=registerXCATHostUploadBaseInfo', model.hostBaseInfo, function(data){

Dialog.closeLoading();

if(data.flag === '0'){

Dialog.alert(data.msg);

return ;

}

model.hostListInfo = data;

that.jumpStep('next');

}, 'json');

}else{//假接口，不用真实提交

$.extend(model.hostBaseInfo, wt.getData(Object.keys(model.hostBaseInfo)));

this.req('xcat/XCATHostAction.do?method=registerXCATHostUploadBaseInfo', model.hostBaseInfo, function(data){

model.hostListInfo = data;

that.jumpStep('next');

}, 'json');

}

}

}

}

},

{widgetClass: 'xcat/host/XcatHostList',

title: i18n.getMsg('selectHost'),

loadingText: i18n.getMsg('registerHostLoading'),

showWidgetCallback: function(wt){

wt.loadData(model.hostListInfo.hostList);

},

btns: {

next: false,

register: {

text: i18n.getMsg('register'),

callback: function(wt){

var that = this;

var data = {};

if(wt.validate() === false){

return ;

}

model.hostListInfo.hostList = wt.getData();

data.platformId = model.hostBaseInfo.platformId;

data.platformPcid = model.hostBaseInfo.platformPcid;

data.hostInfos = model.hostListInfo.hostList.filter(function(o){

if(o.checked === true){

return true;

}

});

// console.log(data)

this.req('xcat/XCATHostAction.do?method=registerXCATSaveInfo', data, function(data){

model.RegisterHostConfirm = data;

that.jumpStep('next');

//

// GUtil.tree.get('dataCenter').refreshNode(data[0].hostPcid, model.hostBaseInfo.platformPcid);

}, 'json');

}

}

}

},

{widgetClass: 'xcat/host/RegisterHostConfirm',

title: i18n.getMsg('registerHostConfirm'),

loadingText: i18n.getMsg('reqRegisterLparLoading'),

showWidgetCallback: function(wt){

wt.loadData(model.RegisterHostConfirm);

},

btns: {

prev: false,

next: false,

complete: {

text: i18n.getMsg('complete'),

callback: function(wt){

that.closeDialog();

}

},

registerLpart: {

text: i18n.getMsg('registerLpar'),

callback: function(wt){

var that = this;

var data = {};

if(wt.validate() === false){

return ;

}

model.RegisterHostConfirm = wt.getData();

data.platformId = model.hostBaseInfo.platformId;

data.platformPcid = model.hostBaseInfo.platformPcid;

data.hostList = model.RegisterHostConfirm;

this.req('xcat/XCATLparAction.do?method=registerXCATLparBaseInfo', data, function(data){

model.RegisterLPart = data.hostinfo;

that.jumpStep('next');

}, 'json');

}

},

}

},

{widgetClass: 'xcat/host/RegisterLPart',

title: i18n.getMsg('registerLpar'),

loadingText: i18n.getMsg('registerLparLoading'),

showWidgetCallback: function(wt){

wt.loadData(model.RegisterLPart);

},

btns: {

prev: false,

next: false,

register: {

text: i18n.getMsg('register'),

callback: function(wt){

var that = this;

if(wt.validate() === false){

return ;

}

var data = {};

model.RegisterLPart = wt.getData();

data.hostId = model.hostBaseInfo.hostId;

data.hostPcid = model.hostBaseInfo.hostPcid;

data.hostList = model.RegisterLPart;

this.req('xcat/XCATLparAction.do?method=registerXCATLparSaveInfo', data, function(data){

model.RegisterLPartConfirm = data.hostinfo;

that.jumpStep('next');

});

}

}

}

},

{widgetClass: 'xcat/host/RegisterLPartConfirm',

title: i18n.getMsg('registerLparConfirm'),

showWidgetCallback: function(wt){

wt.loadData(model.RegisterLPartConfirm);

},

btns: {

prev: false,

next: false,

configLpar: {

text: i18n.getMsg('configLpar'),

callback: function(){

this.jumpStep('next');

}

}

}

},

{widgetClass: 'xcat/host/ConfigLpar',

title: i18n.getMsg('configLpar'),

loadingText: i18n.getMsg('confiLparLoading'),

showWidgetCallback: function(wt){

// var configLpars = model.RegisterLPartConfirm.filter(function(item){

// return item.registerSuccess;

// });

// wt.loadData(configLpars);

wt.loadData(model.RegisterLPartConfirm);

},

btns: {

prev: false,

next: false,

config: {

text: i18n.getMsg('config'),

callback: function(wt){

var that = this;

if(wt.validate() === false){

return ;

}

var data = {};

model.ConfigLpar = wt.getData();

data.hostId = model.hostBaseInfo.hostId;

data.hostPcid = model.hostBaseInfo.hostPcid;

data.hostList = model.ConfigLpar;

this.req('xcat/XCATHostAction.do?method=registerXCATLparConfigIp', data, function(data){

model.ConfigLparConfirm = data.hostinfo;

that.jumpStep('next');

});

}

}

}

},

{widgetClass: 'xcat/host/ConfigLparConfirm',

title: i18n.getMsg('configLparConfirm'),

showWidgetCallback: function(wt){

wt.loadData(model.ConfigLparConfirm);

},

btns: {

prev: false,

next: false,

cancel: false,

close: {

text: i18n.getMsg('close'),

callback: function(){

that.closeDialog();

}

}

}

}

];

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

if(!opts.cancelCB){

opts.cancelCB = function(){

that.closeDialog();

};

}

//一般调用

var steps = new CommonWizard({

widgets: widgets,

cancelCB: function(){

that.closeDialog();

}

});

//Dialog传参调用

this.container.append(steps.container);

}

### 6.1注册平台(RegisterPlatform)widget

initialize: function(opts){

var that = this,

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

$listItems = null;

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

//验证配置参考验证插件

ko.validation.configure({

decorateInputElement: true,

registerExtenders: true,

messagesOnModified: true,

insertMessages: false,

parseInputAttributes: true,

errorClass: 'has-error'

});

ko.validation.configuration.insertMessages=false;

function ViewModel(){

var self = this;

self.platformType = ko.observable(2);

self.platformTypeEnable = ko.observable(true);

//对字段值进行验证

self.platformName = ko.observable('xcat').extend({

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: '^[A-Za-z0-9\_-]{3,16}$'

},

validation: {//自定义验证

async: true,//是否异步验证

validator: function (val, params, callback) {

$.post('xcat/XCATPlatformAction.do?method=checkPlatformName', {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')

}

});

self.ip = ko.observable('172.30.126.76').extend({

required: {

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

params: true

},

pattern: {

message: i18n.getMsg('please enter the correct IP format'),

params: '((2[0-4]\\d|25[0-5]|[01]?\\d\\d?)\\.){3}(2[0-4]\\d|25[0-5]|[01]?\\d\\d?)'

},

validation: {

async: true,

validator: function (val, params, callback) {

$.post('xcat/XCATPlatformAction.do?method=checkPlatformRegisteredByIp', {ip: val}, function(data){

var isExsit = false;

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

if(data.flag === true){

isExsit = true;

msg = data.msg || i18n.getMsg('this IP has been used');

}

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

});

},

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

}

});

self.username = ko.observable('root').extend({

required: {

message: i18n.getMsg('please enter your user name'),

params: true

},

pattern: {

message: i18n.getMsg('please enter the correct user name format'),

params: '^[a-z0-9\_-]{3,16}$'

}

});

self.userpass = ko.observable('root1234').extend({

required: {

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

params: true

}

});

// self.confirmUserpass = ko.observable('123456').extend({

// required: {

// message: "请输入密码",

// params: true

// },

// validation: {

// validator: function (val, other) {

// return val == other;

// },

// message: '两次输入密码不一致',

// params: self.userpass

// }});

//单独打开时调用（即非Steps方式调用）

self.save = function(){

that.req(ko.toJSON(self));

}

}

this.vm = new ViewModel();

window.vm = this.vm;

this.vm.errors = ko.validation.group(this.vm);

},

loadData: function(data){//widget向导显示此widget时调用，加载平台相关数据

if(data.platformType){

this.vm.platformType(data.platformType);

this.vm.platformTypeEnable(false);

}

},

getData: function(keysObj){//注册平台时获取输入的表单数据

var data = JSON.parse(ko.toJSON(this.vm, keysObj));

data.userpass = $.jCryption.encryptKey(data.userpass);//加密

return data;

},

createCompleteCallback: function(){

ko.applyBindings(this.vm, this.container[0]);

},

validate: function(){//widget向导点击下一步时调用，返回true跳转到下一widget

if (this.vm.errors().length == 0) {

//可以提交数据了.

return true;

} else {

this.vm.errors.showAllMessages();

return false;

}

},

### 6.2选择模板(HostTemplate)widget

initialize: 方法和注册平台类似

req: function(url, fromObj, CB){//ajax提交form表单，和上传文件

var fd = new FormData();

var xhr = new XMLHttpRequest();

var $file = this.container.find('.template-file');

var file = $file[0].files[0];

fd.append("file", file);

// fd.append("platformId", fromObj.platformId);

url = url + '&platformId=' + fromObj.platformId + '&platformPcid=' + fromObj.platformPcid;

xhr.upload.addEventListener("progress", function(){

// alert('上传完成10%！')

}, false);

xhr.addEventListener("load", function(data){

// Dialog.alert('上传成功！')

// CB(data);//上传成功跳转到下一widget

}, false);

xhr.addEventListener("error", function(){

Dialog.alert(i18n.getMsg('uploadFail'))

}, false);

xhr.addEventListener("abort", function(){

Dialog.alert(i18n.getMsg('uploadSuspend'))

}, false);

// xhr.addEventListener("onreadystate", function(data){

//// alert('返回数据：' + data);'

// if((xhr.readyState==4 && xhr.status==200)){

// CB(data);

// }

// }, false);

xhr.onreadystatechange = function(data){

// alert('返回数据：' + data);'

if((xhr.readyState==4 && xhr.status==200)){

if(xhr.responseText){

CB(JSON.parse(xhr.responseText));

}

}

};

xhr.open("POST", GUtil.getRootPath() + url);

xhr.send(fd);

},

### 6.3注册主机(XcatHostList) widget

initialize: function(opts){

var that = this,

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

$listItems = null,

vm = null;

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

function ViewModel(){

var self = this;

self.checkedAllBtn = ko.observable(false);

self.hostList = ko.observableArray([]);

self.errorDetail = function(itemData){//显示错误详情

Dialog.alert(itemData.errorMsg);

}

self.checkedAll = function(vm, e){//点全选框

var $currentTarget = $(e.currentTarget);

var isChecked = $currentTarget.find('input').prop('checked');

that.container.find('tbody input').not('[disabled]').prop('checked', !isChecked);

checkedAll(!isChecked);

};

self.checkedItem = function(itemData, e){//点单选框

var $currentTarget = $(e.currentTarget);

var isChecked = $currentTarget.find('input').prop('checked');

itemData.checked = !isChecked;

if(isCheckedAll()){

self.checkedAllBtn(true);

}else{

self.checkedAllBtn(false);

}

};

self.isEnable = function(itemData){//当前host是否可注册

return itemData.registered === '0' && itemData.rackRepeat === '0' && itemData.licIsValid === '1' && itemData.env === '1';

};

self.tipInfo = function(itemData){//不可注册时相关提示

var btnStr = '<a class="btn btn-default error-detail">详情</a>';

var tipStr = '<span class="label label-success">'+ i18n.getMsg('unregistered') +'</span>';

if(itemData.registered === '1'){//已注册

tipStr = '<span class="label label-warning">'+ i18n.getMsg('registered') +'</span>';

return tipStr;

}

if(itemData.rackRepeat === '1'){//机架重复

tipStr = '<span class="label label-warning">'+ i18n.getMsg('frameRepeat') +'</span>';

return tipStr;

}

if(itemData.env === '0'){//环境不存在

tipStr = '<span class="label label-warning">'+ i18n.getMsg('environmentNoExist') +'</span>';

return tipStr;

}

if(itemData.licIsValid === '0'){//license无效

tipStr = '<span class="label label-warning">'+ i18n.getMsg('licenseInvalid') +'</span>';

return tipStr;

}

return tipStr;

}

self.save = function(obj, e){

alert(ko.toJSON(this))

};

function checkedAll(flag){

$.each(self.hostList(), function(){

if(self.isEnable(this)){

this.checked = flag;

}

});

}

function isCheckedAll(flag){

var result = true;

var hostList = self.hostList();

$.each(self.hostList(), function(){

if(self.isEnable(this)){

if(!this.checked){

result = false

return false;

}

}

});

return result;

}

}

this.vm = vm = new ViewModel();

},

### 6.4注册主机确认(RegisterHostConfirm) widget

和注册主机差不多，把输入框改为文本框

### 6.5注册分区 (RegisterLPart) widget

Initialize：和注册主机类似

validate: function(){

var result = false;

var reg = /((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)/;

ko.mapping.toJS(this.vm).hostList.forEach(function(host){

if(host.checked === true){

result = true;

return false;

}

});

if(result === false){//是否选中lpar

// alert('请要注册的Lpart');

Dialog.alert(i18n.getMsg('please choose to register Lpar'));

return false;

}

//ip验证

this.container.find('input.ip').each(function(){

var $this = $(this);

if($this.prop('disabled')) return ;

if(!reg.test(this.value)){

result = false;

$this.addClass('has-error');

}else{

$this.removeClass('has-error');

}

});

if(result === false){//格式不正确提示

Dialog.alert(i18n.getMsg('please enter the correct format'));

return false;

}

return result;

},

### 6.6注册分区确认(RegisterLPartConfirm) widget

和注册lpar差不多，把输入框改为文本框

### 6.7配置分区(ConfigLpar) widget

配置分区是从注册分区里拆出来的。

注册lpar隐藏ip 网关、网卡 ， 配置分区时显示ip 网关、网卡

### 6.8配置分区确认(ConfigLparConfirm) widget

配置分区是从注册分区里拆出来的。

注册lpar隐藏ip 网关、网卡 ， 配置分区时显示ip 网关、网卡

## 7. 注册主机分区(RegisterXcatHostWizard)核心代码(是RegisterPHLWizard的一部分)

## 8. 注册分区(RegisterLpartWizard)核心代码（是RegisterPHLWizard的一部分）

# Coreplatform主要介绍（AppNav、Task模块）（完成）

## AppNav即一个列表 跟PlatformList差不多

## Task核心代码

initialize: function(opts){

var that = this;

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

this.container = $(this.rootElement);

function ViewModel(){

var self = this;

self.list = ko.observableArray([]);//任务列表

self.runTime=function(itemData){//运行时间

if(itemData.timer){//如果定时器存在删除

clearInterval(itemData.timer);

}

itemData.timer=setInterval(function(){

itemData.runTime(Format.toDurationString(Date.now()+ServerDate.millisOffset-itemData.beginTime));

},1000);

};

self.process = function(itemData){//任务进度

if(itemData.status===TaskStatus.WAITING){

return "-";

}

return Format.toPercentString(itemData.process/100);

};

self.showTaskStatus = function(itemData){//任务状态

return TaskStatus.nameOf(itemData.status);

};

self.showChildTasks = function(itemData, e){//子任务

var isExtend = !(itemData.\_extend() || false);

itemData.\_extend(isExtend);

};

return self;

}

this.vm = new ViewModel();

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

that.createCompleteCallback();

});

},

updateData: function(data){//推送任务更新task列表

var that = this;

var taskList = ko.toJS(this.vm.list);

var originDataMap = this.rebuildMap(taskList, 'taskId');

var errorCount = 0;

var runCount = 0;

var successCount = 0;

data.forEach(function(item,index){

if(index===0){console.log(JSON.stringify(item,null,"\t"));}

if(item===null){return;}

var task = originDataMap[item.taskId];

if(task){

$.extend(true, task, item);

}else{

task=item;

taskList.push(task);

//第一次创建刷新树节点

that.createRefreshNode(task);

}

if(+task.status===TaskStatus.SUCCESS){

//TMP 若成功，清空描述字段的值 Thu, 07 Aug 2014 02:26:04 GMT

task.errorDesc="";

that.removeCompleteTask(task);

successCount ++;

//刷新树节点

that.completeRefreshNode(task);

}else if(+task.status===TaskStatus.ERROR){

that.removeCompleteTask(task);

errorCount ++;

//刷新树节点

that.completeRefreshNode(task);

}

});

that.opts.setTaskSuccessShowNum && that.opts.setTaskSuccessShowNum(successCount || '')//显示成功任务个数

that.opts.setTaskErrorShowNum && that.opts.setTaskErrorShowNum(errorCount || '');//显示失败任务个数

that.opts.setTaskShowNum && that.opts.setTaskShowNum(that.vm.list().length);//显示正在运行任务个数

this.setObservableProp(taskList);

this.vm.list(taskList);

this.vm.list().forEach(function(item){

that.vm.runTime(item);

});

var e=new Event("updateddata");

TSWidget.prototype.dispatchEvent.call(this,e);

},

createRefreshNode: function(data){//第一次创建任务刷新

//目前只能数据中心tree刷新

var typeMap = {

'HOST': true,

// 'IMAGE': true,

'PLATFORM': true,

'LPAR': true

};

var dataCenterTree = GUtil.tree.get('dataCenter');

if(!dataCenterTree) return ;

if(typeMap[data.entityType]){

dataCenterTree.refreshNode(data.entityId);//刷新树节点，触发相应回调

}

},

completeRefreshNode: function(data){//任务完成是刷新树节点

var dataCenterTree = GUtil.tree.get('dataCenter');

if(!dataCenterTree) return ;

if(data.entityType === 'PLATFORM'){

dataCenterTree.refreshNode(data.platformPcid || data.platformId);

}else if(data.entityType === 'HOST'){

dataCenterTree.refreshNode('', data.platformPcid || data.platformId, 'parent');//刷新父点，即主机所在的平台节点

}else if(data.entityType === 'LPAR'){//刷新每个lpar的父点，即每个lpar所在的主机节点

data.hostPcid = data.hostPcid || data.hostId;

var hostPcidArr = data.hostPcid.split(',');

hostPcidArr && hostPcidArr.forEach(function(item){

dataCenterTree.refreshNode('', item, 'parent');

});

}

//完成、失败刷新

// dataCenterTree.refreshNode(data.entityId);

},

req: function(){

var host = location.host || '172.24.18.157:8080';

var protocol = location.protocol !== 'file:' ? location.protocol : 'http:';

var interfaceMap = {

webSocket: 'ws://' + host + '/ws/task',

eventSource: protocol + '//' + host+ '/es/task',

longConnect: '/longConnect/task',

polling: '/polling/task'

};

if(!request.onLine){return false;}

if(this.checkUseType(interfaceMap.webSocket, 'webSocket')){//webSocket

this.useWebSocket(interfaceMap.webSocket);

}else if(this.checkUseType(interfaceMap.eventSource, 'eventSource')){//eventSource

this.useEventSource(interfaceMap.eventSource);

}else if(this.checkUseType(interfaceMap.longConnect, 'longConnect')){//长连接

this.useLongConnect(interfaceMap.longConnect);

}else{//轮询

this.usePolling(interfaceMap.polling);

}

},

# 相关demo（完成部分）(目录：asserts/demo)