# Realspace(Flash)

## 1产品概述

iClient for Realspace是一套基于 SuperMap UGC （Universal GIS Core）底层类库和 OpenGL 三维图形处理库的三维地理信息可视化客户端开发包。整个客户端包括Web三维GIS插件以及ActionScript API开发包。Web三维GIS插件是以ActiveX控件的形式嵌入到网页中，目前可以支持IE浏览器。用户利用iClient for Realspace可以自定义三维可视化场景，能够快速地完成海量数据加载、数据浏览、图层控制，并且支持插件的自动更新等功能，因此开发者通过简洁易用的 ActionScript语言便能轻松打造三维可视化地理信息客户端。该客户端应用程序以 ActiveX 控件形式集成于 Web 网页中。

ActionScript API

JavaScript API

ActiveX控件

Plug-in

Universal GIS Core

封装

JavaScript封装

ActionScript封装

iClient for Realspace技术框架

### 1.1主要特点

#### 1.1.1三维与Web的完美结合，支持Service GIS架构

iClient for Realspace是三维技术与Web技术的完美结合。作为SuperMap系列服务器产品的web客户端，一方面，支持SuperMap iServer提供的强大的二三维一体化的专业GIS功能，可以满足各行业的深入应用；另一方面，客户端嵌入在浏览器中运行，用户只需输入网址就可以体验到三维可视化的效果，极大地方便了用户的使用和软件的分发。

#### 1.1.2高运行效率的 UGC 内核和简易脚本开发环境的结合

iClient for Realspace基于高运行效率的 GIS 基础内核 UGC（Universal GIS Core Class Library），封装成 ActiveX 控件，保证了产品运行的高效性，及产品的跨浏览器。 在此基础上，又用 JavaScript 封装 ActiveX 控件，再用ActionScript为开发者提供了一套简单易用的ActionScriptAPI ，使三维客户端开发变得轻松快捷。

#### 1.1.3二三维一体化的客户端

iClient for Realspace在插件的基础上封装的ActionScriptAPI 与SuperMap的另一个客户端 产品 SuperMap iClient 6R for Ajax具有统一的体系架构，使得iClient for Realspace成为名符其实的二三维一体化客户端。在iClient for Realspace中二维数据可以不经过投影转换直接加载到三维球面上显示，同时二维的GIS分析和处理功能，也可以在三维场景中直接操作和使用。

#### 1.1.4支持丰富的数据服务

iClient for Realspace能够实时获取服务器端提供的服务信息，并且能够流畅的显示来自服务器端或数据库的地形影像数据、KML数据、KMZ数据、模型数据和影像数据。

#### 1.1.5高效的客户端智能缓存技术

iClient for Realspace可以将用户请求的图层数据暂时存储于本地磁盘。高效的客户端缓存技术使得数据加载更加迅速、数据浏览更加平滑。当服务器端的数据更新时，客户端缓存还能及时动态更新数据。

#### 1.1.6高度的数据保密性

iClient for Realspace自动将从服务器端获取的缓存数据保存为SuperMap定义的文件格式，使得客户端获取的是加密后的GIS空间数据，这样从根本上保证了服务提供商花大量精力收集、加工的数据资料，保护了服务提供商的利益。

#### 1.1.7方便的离线访问

iClient for Realspace客户端支持离线访问。iClient for Realspace的缓存技术不仅能够加速浏览速度，提升用户体验，另一方面可以使用户在离线状态下访问存储的缓存数据，给用户带来便利。

#### 1.1.8良好的扩展性

iClient for Realspace具有高内聚、低耦合的架构特性，开发者可根据需要完成部分功能模块的定制与扩展，给用户提供了更多的自由度，以满足不同行业的需求和提供更专业的功能。

#### 1.1.9插件自动更新

iClient for Realspace支持定时检测现有的插件版本信息，能够自动提醒用户进行插件更新工作，保证用户获得最好的体验。

### 1.2主要功能

#### 1.2.1强大的场景浏览功能

iClient for Realspace不仅提供放大、缩小、全幅、漫游等基本的场景浏览功能，同时支持飞行、选择、量算、查询等功能。用户可以方便的浏览二维、三维数据（包括三维专题数据）。

#### 1.2.2提供灵活多样的三维场景

用户可以通过桌面程序或客户端程序自定义三维场景（三维场景元素包括操作罗盘、大气层、星空、比例尺、进度条、雾效等），保存后的配置将作为下一次场景的默认配置，配置文件为 XML 文件；同时还可通过调整场景设置参数管理场景元素。

#### 1.2.3体验优越的二三维一体化功能

iClient for Realspace基于 SuperMap 的二三维一体化的Realspace GIS技术体系，具有功能更强大、性能更优越、效果更逼真的特性。

二三维一体化浏览：通过统一的地理坐标实现将其他二维客户端显示结果与 iClient for Realspace三维场景之间进行联动浏览显示。

二三维一体化标绘：在二维窗口中进行的标绘结果，可以在三维场景中的跟踪图层中进行实时显示。

二三维一体化查询：二三维一体化查询可以允许用户将二维中查询结果，同步显示在三维场景中，并高亮显示。

#### 1.2.4支持图层管理功能

iClient for Realspace支持设置图层的可见性、可选择性，以及改变图层顺序等控制管理。

#### 1.2.5强大的地物标绘和编辑功能

用户可在屏幕图层和追踪图层中标绘各种类型的地物，并支持标绘自定义的地物类型；可以对矢量图层、KML 图层、KMZ 图层和三维模型图层中的地物进行实时的编辑。

#### 1.2.6提供多种三维特效

iClient for Realspace提供对场景三维特效的支持，包括动画效果、粒子效果、海洋效果和立体显示效果。

#### 1.2.7任意给定线路的三维飞行

iClient for Realspace可导入预先设计好的线路，进行相关参数的设定（如飞行速度、飞行总时间、是否锁定方位角、是否锁定高程等），然后就可以沿着导入的线路自动飞行，在飞行过程中，还可实时改变各种参数。

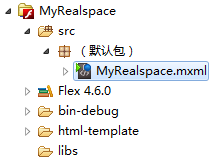
## 2快速入门

### 2.1下载并安装插件

当前iClient for Realspace只支持IE浏览器，首先我们需要打开一个空白网页，如果安装了iserver服务器可以输入地址http://localhost:8090/iserver/iClient/forRealspace/Setup.exe；如果没有安装本地服务器，可以访问我们的官方网站的地址http://support.supermap.com.cn:8090/iserver/iClient/forRealspace/Setup.exe 进行下载安装

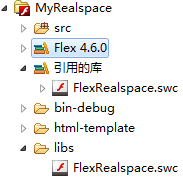
### 2.2创建项目

启动Flash Builder，文件-->新建-->Flex项目，项目结构如下图：

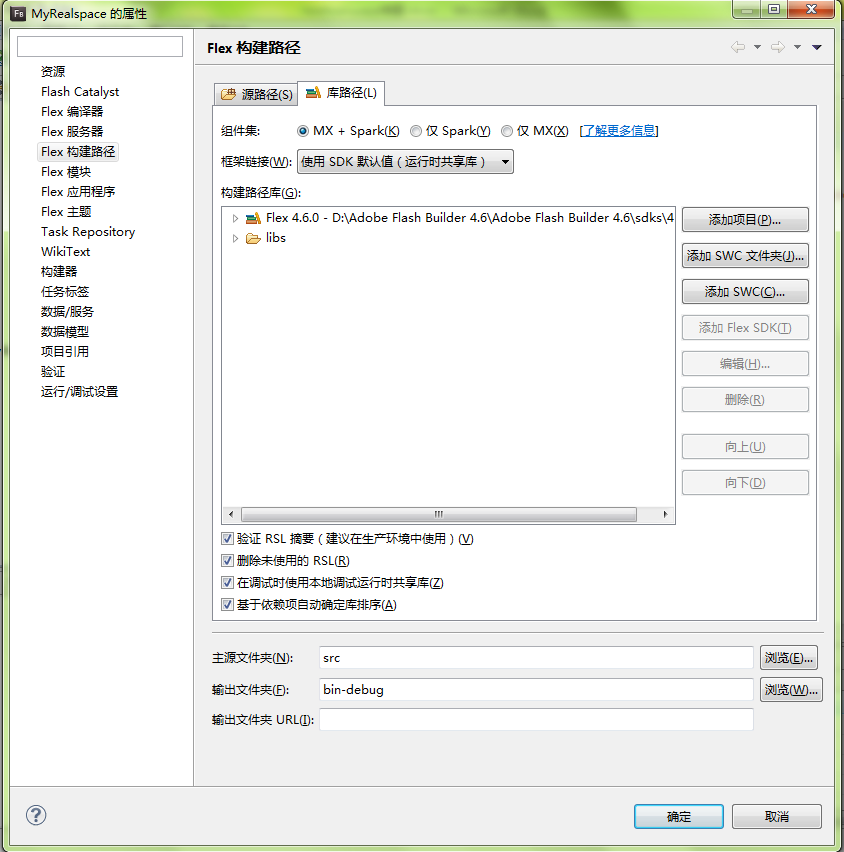


### 2.3添加引用

在<https://github.com/SuperMap/Flex-Realspace>下载的压缩文件... FlexRealspace\bin\下获取到Realspace的开发包FlexRealspace.swc，将其放在libs文件夹下，如下图所示：



Flash Builder会自动将放于libs下的swc文件生成引用，用户可以看见多出了一个“引用的库”，如果放于其他地方，系统不能自动识别，可以通过右键当前项目-->属性-->Flex构建路径-->库路径-->添加SWC来找到自己的FlexRealspace.swc文件，最后点击确定，界面如下：



### 2.4加载场景

在MyRealspace.mxml的设计器上设计界面，从自定义里面拖动SceneDiv控件到设计器上，设置大小、位置，并且设置一个唯一的id。在主程序上注册creationComplete事件。源代码如下：

首先定义两个全局的私有变量：三维场景控件sceneControl；三维场景scene。位置放于代码如下：

//用于装载三维场景

private var sceneControl: SuperMap.Web.UI.Controls.SceneControl;

//一个三维场景控件只有唯一一个三维场景，三维场景只能sceneControl.get\_scene()

//获取

private var scene:SuperMap.Web.Realspace.Scene;

creationComplete事件为当界面完成其构建、属性处理、测量、布置和绘制时触发，初始化三维场景控件，在其内部添加如下代码：

sceneControl=new SceneControl(mySceneDiv,succeedCallBack,failedCallBack);

这里的mySceneDiv就是我们的SceneDiv控件的id，我们还需要两个函数succeedCallBack和failedCallBack，分别为三维场景控件初始化后的回调成功函数和失败函数，回调函数代码如下：

private function succeedCallBack(event:CallBackEvent):void

{

//成功初始化三维场景之后的进一步操作，具体代码详见范例

}

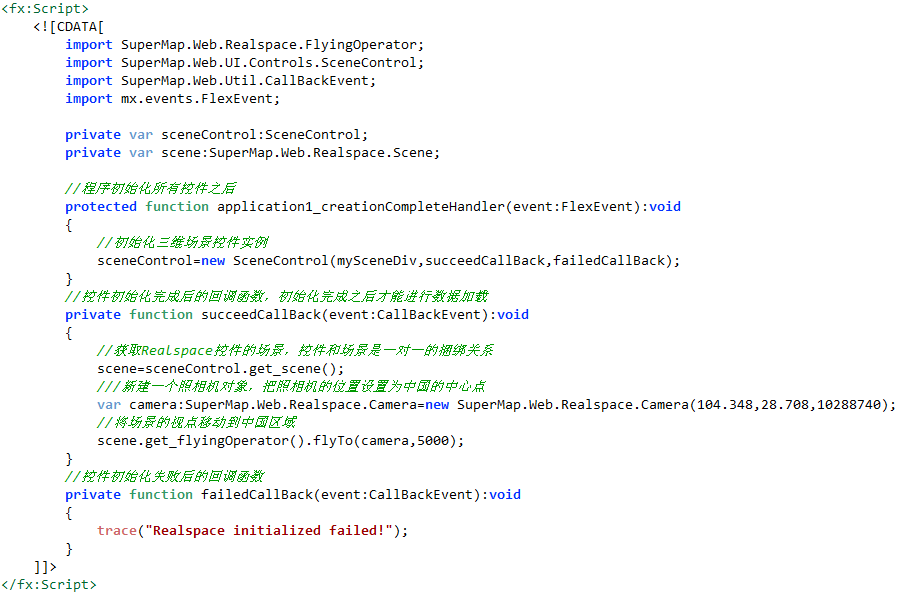
private function failedCallBack(event:CallBackEvent):void

{

//初始化三维场景失败之后的进一步操作，具体代码详见范例

}

最后核心代码如下图：



当前三维场景只支持IE浏览器，运行程序后效果如下图：



## 3范例

从官网可以获取到FlexRealspaceTest（FlexRealspaceSample），里面有当前提供的所有范例程序。总共分为4个部分：场景浏览、空间要素操作、空间查询与分析、二三维一体化。

### 3.1场景浏览

#### 3.1.1三维场景浏览

对应着快速入门的范例，该范例展示如何创建出三维场景，并利用场景的飞行接口飞行到中国区域，源代码详见Sample.SceneView.HelloWorld.mxml。

#### 3.1.2基本操作

该范例展示如何设置场景的Action来操作场景，包括对象的选择以及漫游操作，并且展示了场景的全局显示以及刷新功能，源代码详见Sample.SceneView. BasicAction.mxml。

#### 3.1.3地形拉伸

该范例展示如何使用场景的set\_terrainExaggeration接口来按照给定的拉伸系数拉伸地形，源代码详见Sample.SceneView. TerrainExaggeration.mxml。

#### 3.1.4飞行

该范例展示如何设置地理坐标、高度以及飞行模式进行飞行，飞行本质上是视点的移动，

通过给定飞行时间以及飞行模式，可以模拟现实中的许多移动现象，源代码详见Sample.SceneView.Fly.mxml。

#### 3.1.5沿线飞行

FlyManager 对象负责管理飞行路径的启动、停止、暂停等操作，同时可通过该对象获取飞行路径的状态，时间等。

FlyRoutes 对象是一组飞行路线的集合，通过该对象可获取集合中的任意 FlyRoute 对象，通过 get\_currentRoute() 获取当前飞行路径的对象实例。该对象无法被实例化，只能通过 FlyManager 的 get\_routes() 获取。

通过调用 FlyRoutes 对象上的 fromFile() 方法实现从文件加载三维飞行路径（fpf 文件）。

FlyRoute 对象包含了飞行路径的相关内容，主要包括路径可见性、飞行速度、循环状态等。该对象无法被实例化，只能通过 FlyRoutes 的 get\_currentRoute() 获取。

源代码详见Sample.SceneView. Fly\_Routes.mxml。

#### 3.1.6飞行事件

该范例展示如何注册事件：flownEnd的回调函数，该函数将会在飞行结束后执行，可以用来控制飞行。源代码详见Sample.SceneView. FlyingEvent.mxml。

#### 3.1.7加载图层

该范例展示如何加载通过SuperMap iServer 6R Java发布的影像、模型、kml、kmz、地形图层，源代码详见Sample.SceneView. LoadLayers.mxml。

#### 3.1.8添加WMS图层

该范例展示如何向三维场景中添加WMS图层，源代码详见Sample.SceneView. AddWMS.mxml。

#### 3.1.9添加WMTS图层

该范例展示如何向三维场景中添加WMTS图层，源代码详见Sample.SceneView. AddWMTS.mxml。

### 3.2空间要素操作

#### 3.2.1自定要素动画

该范例展示如何控制模型对象按照既定路径的移动，该示例代码包含两部分功能：

首先加载KML图层，读取运动路径，然后加载模型，并控制模型按既定路线运动。

，源代码详见Sample.Feature. FeatureAnimation.mxml。

### 3.3空间查询与分析

#### 3.3.1量算

该范例展示如何进行距离、面积、高程等地理信息的量算，并将量算结果实时显示。源代码详见Sample.Query. Measure.mxml。

#### 3.3.2属性字段查询

范例展示如何获取矢量图层选择集中元素的属性字段信息。源代码详见Sample.Query. FieldQuery.mxml。

#### 3.3.3查询服务

示范如何使用 SQL 查询服务进行三维数据集查询。源代码详见Sample.Query. QueryService.mxml。

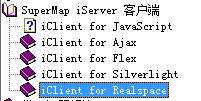
### 3.4二三维一体化

暂未开放

## 4扩展开发

扩展开发前我们需要做一些准备，拥有一定权限的用户可以在我们的官网上直接下载到FlexRealspace.swc的源代码，没有源代码也可以进行扩展开发，直接使用FlexRealspace.swc进行扩展，两种方式一样。

扩展开发前我们还需要对SuperMapiServerJava7C.chm帮助文档有一定的了解，尤其是里面的iClient for Realspace部分，如下图：



Realspace本身是使用Ajax做的，开出的API为JavaScript，没有ActionScript，我们通过两种语言的通信使用ActionScript控制JavaScript来使用三维，JavaScript API已经比较全面，我们的扩展主要是对ActionScript API的扩展，所以需要参照JavaScript API，为方便使用，开出的API尽量统一。

### 4.1扩展类

扩展类主要有两种类，细心的用户可以从JavaScript API里发现有一些类可以用new初始化，有一些类只能通过其他对象的方法获取。暂且将他们称作可初始化类和不可初始化类。

这里的初始化只是对使用者而言的，其实所有类都可以初始化。

#### 4.1.1不可初始化类

在JavaScript API的命名空间SuperMap.Web.Realspace下我们可以找到类Layer3DServicesList，下面我们对它进行扩展。

查看JavaScript API我们知道它是不能使用new来初始化的类，在JavaScript API里面没有其它的类是它的基类（如果有父类，需要继承父类），所以继承于SuperMap.Web.Util.ClassBase，完整代码如下：

package SuperMap.Web.Realspace

{

import SuperMap.Web.Util.ClassBase;

/\*\*

\* 图层服务列表信息类。

\* 该类主要用于获取服务端发布的全部三维图层集合服务及其三维图层集合中的各个

\* 图层的信息。

\* 需要注意的是该类只获取了有哪些三维图层服务，而不负责具体数据的加载。

\* 该类不可创建，只能通过 SceneControl.get\_layer3DServiceInfo获取。

\*/

public class Layer3DServicesList extends ClassBase

{

/\*\*

\* 用于确认对象的类型

\*/

public var CLASSNAME:String="Layer3DServicesList";

//存储此类的全称，需要和js端名称一样

private var REALSPACE\_CLASSNAME:String="SuperMap.Web.Realspace.Layer3DServicesList";

public function Layer3DServicesList(object:Object=null)

{

var key:Number=object["key"] as Number;

var isNew:Boolean=object["isNew"] as Boolean;

super(key,isNew,null);

}

}

}

首先包（命名空间）可以自己定义，不过在此建议用户参照JavaScript API里面的命名空间，最好一致。

import SuperMap.Web.Util.ClassBase;为继承基类ClassBase需要添加的引用。

一个公开的字段CLASSNAME可写可不写，作用是为了方便用户在使用对象的时候跟方便的知道当前对象的类型，建议按照同样的格式，公开、字符串形式。

一个私有字段REALSOACE\_CLASSNAME必须写，必须私有，字段名称固定，不能改变，都是字符串形式，此字符串的内容必须是此类在JavaScript API里面的全称。此字段用于在js端初始化类的操作。

一个构造函数，此类对用户不能使用new初始化，但是需要一个object类型的参数，默认值必须设置为null。方便基类初始化。

var key:Number=object["key"] as Number;此句必写，从初始化参数里面获取此类需要的key值，用于唯一确定一个对象，其实object是一个哈希表。

var isNew:Boolean=object["isNew"] as Boolean;此句必写，从初始化参数里面获取是否需要向js发送消息。

super(key,isNew,null);此句必写，将整理好的参数传递给基类。

至此一个不可初始化类就构建好了。

#### 4.1.2可初始化类

在JavaScript API的命名空间SuperMap.Web.Realspace下我们可以找到类TerrainLayer，下面我们对它进行扩展。

查看JavaScript API我们知道它可以使用new来初始化的类，在JavaScript API里面没有其它的类是它的基类（如果有父类，需要继承父类），所以继承于SuperMap.Web.Util.ClassBase，完整代码如下：

package SuperMap.Web.Realspace

{

import SuperMap.Web.Util.ClassBase;

public class TerrainLayer extends ClassBase

{

//提供给用户识别对象类型的字段

public var CLASSNAME:String="TerrainLayer";

//提供给js端进行解析的对象类型的全称，必写

private var REALSPACE\_CLASSNAME:String="SuperMap.Web.Realspace.TerrainLayer";

//从JavaScript API上面可以看到此类有三个参数，应该都是String类型的，每

//一个的作用参考JavaScript API，这里主要解释一下为什么第一个参数需要写成

//Object形式，此类用户使用的时候第一个参数按照JavaScript API里面提供的

//形式使用，在内部进行了判定，在此使用Object也是因为在基类调用的时候初 //始化所有类只能统一成一个参数，所以就传递哈希表，就得是Object形式。

public function TerrainLayer(strServerRootUrl:Object=null, strLayerName:String=null, strDataName:String=null)

{

//定义了三个局部变量，这三个参数是需要传递给基类的，对应基类的参数。

var key:Number;

var isNew:Boolean;

var array:Object;

//此句判定的作用主要是区分是用户使用new来初始化类还是基类在初始化

//类，从这里看出用户使用类必须按照JavaScript API上面来使用，这里如果

//为String类型，就确定是用户在使用new初始化类。

if(strServerRootUrl is String)

{

//用户初始化类的key必须设置为0，在基类里会重新创建一个唯一的

//key

key=0;

//必须为true，表示需要向js端发送消息

isNew=true;

//这是一个数组，里面的参数可以看出就是构造函数的参数，并且先后顺

//序需要一致，都需要转换为字符串的形式

var realArgument:Array=[

//第一个参数是字符串，后面必须添加上"$String"（数字为

//"$Number"布尔值为”$Boolean”，对象比较特殊，由于js和

//as之间不能传递对象，所以我们传递对象的key值，后缀统一

//为”$Object”

strServerRootUrl.toString()+"$String",

strLayerName.toString()+"$String",

strDataName.toString()+"$String"

];

//哈希表，基类的构造函数第三个参数需要此哈希表

array={

//此类的全称，必写

className:this.REALSPACE\_CLASSNAME,

//初始化类的参数数组，必写

realArgument:realArgument

};

}

//此种情况是基类在调用，不需要向js端发送消息，并且调用此类初始化

//的时候只有第一个参数，并且是哈希表，哈希表里面会有两个参数

else

{

//获取哈希表中的key赋予此对象

key=strServerRootUrl["key"] as Number;

//获取哈希表中的isNew赋予此对象，代表初始化此类是否需要向js

//端发送信息，一般都是false

isNew=strServerRootUrl["isNew"] as Boolean;

}

//将整理好的参数传递给基类进行类的初始化

super(key,isNew,array);

}

}

}

### 4.2扩展方法

扩展方法的方式比较简单，首先需要知道扩展那个类的基本方法，在JavaScript API里面先确定好，再按照如下四种情况扩展。

#### 4.2.1无参数无返回值

在JavaScript API的命名空间SuperMap.Web.Realspace下我们可以找到类Scene，它有一个方法叫refresh，用于刷新三维场景。此方法无参数，无返回值，下面我们对它进行扩展。代码如下：

//如下是as语法定义公开方法的形式，无参数，无返回值

public function refresh():void

{

//此处为一个哈希表，此哈希表记录当前方法的所有信息，用于传递给js端解析

//后进行正确的操作

var array:Object={

//该标签代表行为，用action表示，此次行为是方法，所以使用字符串

//”FUNCTION”来标示，在js端会通过action来识别，然后会按

//照”FUNCTION”方法的方式来解析，必设标签

action:"FUNCTION",

//该标签代表是否向js端发送消息，然后进行此次行为，必设标签，这里默

//认都设置为true，设置为false就是代表不向js端传递消息，也就是此方法

//作废

isNew:true,

//该标签代表此方法所属对象的key，在js端通过key来找到相应的对象进

//行此操作，必设标签

key:this.KEY,

//该标签代表此次方法的名称，必设标签，必须为字符串形式，在js端会自

//动解析

functionName:"refresh",

//该标签代表此方法是否有返回值，必设标签

isReturn:false

};

//最后将这个哈希表传递给基类的此方法，此方法会将此信息解析后传递给js端

//进相应的操作

this.flexToJsCall(array);

}

#### 4.2.2无参数有返回值

在JavaScript API的命名空间SuperMap.Web.Realspace下我们可以找到类Camera，它有一个方法叫isVaild，用于判断当前相机设置是否有效。此方法无参数，有一个类型为布尔值的返回值，下面我们对它进行扩展。代码如下：

public function isVaild():Boolean

{

//需要创建的哈希表，用于传递给基类向js端发送消息，此数组部分信息基类会使

//用，部分信息会在js端被解析

var array:Object={

//代表行为，需要action表示，此次行为是方法，所以使用字符串”FUNCTION”，

//在js端会通过action来识别，然后会按照方法的方式来解析，必设

action:”FUNCTION”,

//是否向js发送消息，在基类会通过isNew来识别，肯定设置为true

isNew:true,

//此对象的KEY，this.KEY是基类的一个字段，需要传递给js，两种语言的对象

//的识别是通过key来完成的

key:this.KEY,

//执行方法的名称，需要使用字符串，对应词方法的名称

functionName:”isVaild”,

//是否有返回值

isReturn:true

};

//需要将创建的哈希表array传递给基类的方法flexToJsCall，并且将返回值传唤为//Boolean值

return this.flexToJsCall(array) as Boolean;

}

#### 4.2.3有参数无返回值

在JavaScript API的命名空间SuperMap.Web.Realspace下我们可以找到类FlyingOperator，它有一个方法叫flyTo，用于判断当前相机设置是否有效。此方法无参数，有一个类型为布尔值的返回值，下面我们对它进行扩展。代码如下：

/\*\*

\* 在指定的时间内，以指定的方式从当前场景飞行至指定的相机位置。

\* camera： 相机类，设置场景将要飞至的位置。

\* nMillSec：飞行时间，单位为毫秒。可选参数，默认值为-1，将以一个默认速度匀速飞行到\*指定地点。

\* flyingMode：飞行方式。默认值为 SuperMap.Web.Realspace.FlyingMode.CAMERA\_NORMAL。\*（也就是6）

\*/

public function flyTo(camera:Camera,nMillSec:Number=-1,flyingMode:Number=6):void

{

//方法的参数以一个数组形式传递，数组内部的参数需要和方法的参数顺序一致

//此数组内部所有参数最终都必须是字符串形式

var realArgument:Array=[

//第一个参数是一个对象，as和js之间的传递时不能传递对象的，所以这里只传//递KEY来唯一识别，后面配上类型，对象都写成$Object

camera.KEY.toString()+"$Object",

//第二个参数是数字，形式如下

nMillSec.toString()+"$Number",

//同上

flyingMode.toString()+"$Number"

];

//此哈希表同无参数有返回值

var array:Object={

action:"FUNCTION",

isNew:true,

key:this.KEY,

functionName:"flyTo",

isReturn:false,

//此处多了一个标示realArgument，代表方法的参数，放没有参数就不需要写这//个标示

realArgument:realArgument

};

//此处无返回值，所以只需要将哈希表传给基类去执行就行了

this.flexToJsCall(array);

}

#### 4.2.4有参数有返回值

在JavaScript API的命名空间SuperMap.Web.Realspace下我们可以找到类Scene，它有一个方法叫findSelection3Ds，用于查找当前三维场景中图层的选择集。此方法有参数，有一个数组类型的返回值，下面我们对它进行扩展。代码如下：

/\*\*

\* 查找当前三维场景中图层的选择集。该方法返回的是 Selection3D 类对象的数组，

\* 每一个 Selection3D 类对象对应三维场景中一个图层中所有处于选中状态的对象集合，若

\*没有对象处于选中状态，则数组为空。

\* hasObjectSelected：指定返回的选择集是所有图层的选择集，还是选择集对象不为空的图

\*层选择集。当其为 true 时，返回选择集对象不为空的图层选择集，否则返回所有图层的

\*选择集。

\*/

public function findSelection3Ds(hasObjectSelected:Boolean=false):Array

{

var realArgument:Array=[hasObjectSelected.toString()+"$Boolean"];

var array:Object={

action:"FUNCTION",

isNew:true,

key:this.KEY,

functionName:"findSelection3Ds",

isReturn:true,

realArgument:realArgument

};

return this.flexToJsCall(array) as Array;

}

不做过多解释，方式就是前面两种的组合。

### 4.3扩展属性

仔细查看JavaScript API里面的属性，使用方式还是形如方法，只是改成了set\_和get\_加属性名称，方式和方法差不多，下面将详细介绍如何扩展。

#### 4.3.1get\_获取

在JavaScript API的命名空间SuperMap.Web.Realspace下我们可以找到类Camera，它有一个属性叫altitude，获取或设置当前的相机高度。我们想获取，使用get对它进行扩展。代码如下：

/\*\*

\* 获取相机高度

\*/

public function get\_altitude():Number

{

var array:Object={

//属性的行为其实和方法是一样的，所以这里也使用"FUNCTION"

action:"FUNCTION",

isNew:true,

key:this.KEY,

functionName:"get\_altitude",

isReturn:true

};

return this.flexToJsCall(array) as Number;

}

其实注意属性的使用时以get\_或者set\_开头，其他和方法几乎是一样的，这样的开头主要是保持和JavaScript API一样。

#### 4.3.2set\_设置

我们现在对上面的altitude属性进行设置，代码如下：

/\*\*

\* 设置相机高度

\* value：相机高度

\*/

public function set\_altitude(value:Number):void

{

var realArgument:Array=[value.toString()+"$Number"];

var array:Object={

action:"FUNCTION",

isNew:true,

key:this.KEY,

functionName:"set\_altitude",

isReturn:false,

realArgument:realArgument

};

this.flexToJsCall(array);

}

### 4.4扩展字段

仔细查看JavaScript API里面的字段的使用，这和我们习惯使用的方式是一样的，字段一般也是用于获取或者设置，在此我们将字段按照属性的形式封装，使用方式也和属性一样了。

#### 4.4.1get\_获取

在JavaScript API的命名空间SuperMap.Web.Core下我们可以找到类Rectangle2D，它有一个字段叫leftBottom，地理坐标系下矩形对象的左下角点坐标值。现在获取，使用get对它进行扩展。代码如下：

/\*\*

\* 获取地理坐标系下矩形对象的左下角点坐标值。该点对象坐标值 x、y 的默认值为 \*NaN。

\*/

public function get\_leftBottom():Point2D

{

var array:Object={

//注意行为变成了"FIELD"，由于在js端的字段的使用方式不是按照方法那样

//使用的，所以在js端解析的时候的方式会不同，而在js端的方法和属性以

//及后面的事件都是按照方法的形式使用的，所以可以归为一类

action:"FIELD",

isNew:true,

key:this.KEY,

functionName:"get\_leftBottom",

isReturn:true

};

return this.flexToJsCall(array) as Point2D;

}

其他形式和方法几乎一样。

#### 3.4.2set\_设置

我们现在对上面的leftBottom字段进行设置，代码如下：

/\*\*

\* 设置地理坐标系下矩形对象的左下角点坐标值。该点对象坐标值 x、y 的默认值为

\* NaN。

\*/

public function set\_leftBottom(point2D:Point2D):void

{

var realArgument:Array=[

point2D.KEY.toString()+"$Object"

];

var array:Object={

action:"FIELD",

isNew:true,

key:this.KEY,

functionName:"set\_leftBottom",

isReturn:false,

realArgument:realArgument

};

this.flexToJsCall(array);

}

### 4.5扩展事件

仔细查看JavaScript API的事件，我们发现总共有两种情况：

一种是使用类似于方法addEvent来注册，使用removeEvent来移除事件，这种方式一般有两个参数，第一个为字符串，代表事件的类型，第二个为Function，触发后调用的函数，这种方式一般可以注册对象所提供的任意事件；

第二种是以add\_开头的方法来注册固定的某种事件，以remove\_开头的方法来移除固定的某种事件，此种方式只能针对某一种事件的注册和移除。

#### 4.5.1不固定事件注册

在JavaScript API的命名空间SuperMap.Web.UI.Controls下我们可以找到类SceneControl，它有一个方法叫addEvent，用于注册事件的。下面我们将对它进行扩展，从addEvent里面可以查看此方法可以注册很多个事件。

首先我们需要在SceneControl类上标记我们提供的事件类型，位置和引用包的位置一样，代码如下：

// Event为事件的一种标记，name为我们提供的事件名称，type代表事件的类型，事

//件类型都默认为SuperMap.Web.Util.CallBackEvent

[Event(name="objectSelected", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="measureDistance", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="measureArea", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="measureHeight", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="measureDistanceFinished", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="measureAreaFinished", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="measureHeightFinished", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="flownEnd", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="addExcavationRegion", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="bubbleInitialize", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="bubbleResize", type="SuperMap.Web.Util.CallBackEvent")]

[Event(name="bubbleClose", type="SuperMap.Web.Util.CallBackEvent")]

事件已经申明好了，现在开始封装addEvent方法，代码如下：

/\*\*

\* 在地图控件中添加事件，将事件与回调函数 handler 绑定。

\* 即当 eventName 指定的事件发生时，完成 handler 回调函数中的操作。

\*/

public function addEvent(eventName:String,handler:Function):Boolean

{

//此处的eventName就是我们申明的事件name中的一个，只有申明了的事件才

//会有效，此方法是基类的方法，用于在flex端注册监听器

addEventListener(eventName,handler);

//此处第一个参数和普通的方法一样，但是第二个参数有所不同，js和as通信也

//不能传递函数，这里虽然后面标签是$Function，但是前面需要时事件的名称，

//这样回调的时候会通过它进行绑定

var realArgument:Array=[eventName.toString()+"$String",eventName+"$Function"];

var array:Object={

action:"FUNCTION",

isNew:true,

key:this.KEY,

functionName:"addEvent",

isReturn:true,

realArgument:realArgument

};

return this.flexToJsCall(array) as Boolean;

}

现在只是从felx端注册了事件，并且发送消息到js端，在js也注册了事件，当js端监听到事件的触发后会传递到flex端的基类，现在还需要在子类里面进行接收，所以有事件回调的类需要重写基类的方法callback，代码如下：

/\*\*

\* 重写基类的回调函数

\* 此方法主要作用在于按照传回来的参数分发给此对象里面应该来处理的外部方法，

\* 并且将需要的数据传给此方法

\* event：事件类型，用于回调后绑定外部处理函数

\* array：回调的数据，提供给外部函数使用

\*/

override protected function callback(event:String,array:Object):void

{

//将事件调度到事件流中，也就是触发event事件，这里需要初始化一个

//CallBackEvent，中间两个参数默认false，最后一个参数就是回调回来的数据，

//需要传递给外部使用

dispatchEvent(new CallBackEvent(event,false,false,array));

}

这样，当触发该事件的时候所有处理就会自动传递给用户在外面的回调函数。

#### 4.5.2不固定事件移除

对应的有一个removeEvent的方法是用来移除事件的，移除事件很简单，下面我们来封装此方法，代码如下：

/\*\*

\* 移除与事件绑定的指定回调函数 handler。

\*/

public function removeEvent(eventName:String,handler:Function):Boolean

{

//基类提供的移除flex端的监听事件，这里只需要断开flex端的事件，不需要再

//次断开js端的事件

removeEventListener(eventName,handler);

// removeEventListener方法无返回值，这里提供给用户移除成功

return true;

}

#### 4.5.3固定事件注册

在JavaScript API的命名空间SuperMap.Web.UI.Action3Ds下我们可以找到类SceneAction，它有一个事件叫actionCompleted。下面我们将对它进行封装，首先还是需要申明我们的事件，代码如下：

[Event(name="actionCompleted", type="SuperMap.Web.Util.CallBackEvent")]

然后需要封装方法add\_actionCompleted来注册js端的事件，代码如下：

/\*\*

\* 添加actionCompleted事件

\*/

public function add\_actionCompleted(handler:Function):void

{

//了解as事件机制的应该知道这里的actionCompleted需要和申明的名称一致

addEventListener("actionCompleted",handler);

var realArgument:Array=["actionCompleted$Function"];

var array:Object={

action:"FUNCTION",

isNew:true,

key:this.KEY,

functionName:"add\_actionCompleted",

isReturn:false,

realArgument:realArgument

};

this.flexToJsCall(array);

}

/\*\*

\* 重写基类的回调函数

\* 此方法主要作用在于按照传回来的参数分发给此对象里面应该来处理的外部方法，

\* 并且将需要的数据传给此方法

\* event：事件类型，用于回调后绑定外部处理函数

\* array：回调的数据，提供给外部函数使用

\*/

override protected function callback(event:String,array:Object):void

{

dispatchEvent(new CallBackEvent(event,false,false,array));

}

方法和不固定事件很相似，注意观察不通之处和相同之处。

#### 4.5.4固定事件移除

移除事件也只需要移除flex端的事件即可，方法remove\_actionCompleted的封装如下：

/\*\*

\* 移除actionCompleted事件

\*/

public function remove\_actionCompleted(handler:Function):void

{

removeEventListener("actionCompleted",handler);

}

### 4.6扩展枚举

仔细查看JavaScript API里面的枚举，严格来说其实并不是形如java或者C#里面的枚举，在js以及as里面还不存在枚举的概念，仔细观察其实就是静态字段，并且是Number类型的静态字段，而且还是不可变的，也就是常量，由此封装就很简单了。

在JavaScript API的命名空间SuperMap.Web.Realspace下我们可以找到枚举AltitudeMode，下面我们将对它进行封装代码如下：

package SuperMap.Web.Realspace

{

import SuperMap.Web.Util.Enum;

/\*\*

\* 高度模式常量。

\* 枚举类，用来设置相机或三维数据的海拔高度值。

\*/

public class AltitudeMode extends Enum

{

/\*\*

\* 绝对高度模式

\*/

public static const ABSOLUTE:Number=2;

/\*\*

\* 地表层高度模式

\*/

public static const CLAMP\_TO\_GROUND:Number=0;

/\*\*

\* 距地相对高度模式

\*/

public static const RELATIVE\_TO\_GROUND:Number=1;

/\*\*

\* 相对于地下

\*/

public static const RELATIVE\_TO\_UNDERGROUND:Number=3;

public function AltitudeMode()

{

super();

}

}

}

所有枚举需要继承父类Enum，统一管理，所做的就是封装静态常量字段。

### 4.7参数详解

我们在初始化类、调用方法，设置属性、使用字段、注册事件等时候都可能遇到需要传递参数给js端，由于在js端解析方式限制，传递给js的数据都只能是字符串，为了更好的识别字写字符串的区别，我们在所有字符串后面都添加了形如”$...”的标签，这里规定：

字符串：本身为字符串的后面添加”$String”；

数字：调用toString()方法后添加”$Number”；

布尔值：调用toString()方法后添加”$Boolean”；

时间：也就是Date，调用time获取毫秒形式的时间再转换为字符串后添加”$Date”；

数组：也就是Array，调用基类的方法objectToString()转化为特殊的字符串，后面不需要添加别的标签；

对象：也就是我们封装的类的对象，调用它们的KEY，转化为字符串后添加”$Object”；

场景Div：也就是我们的自定义控件SceneDiv，比较特殊，有兴趣的可以查看源代码，它通过KEY转换为字符串后添加的是”$Element”；

方法：也就是Function，我们在有回调函数或者注册事件的地方会遇到这种参数，只要出现这种参数，都需要按照事件的形式处理，需要添加形如[Event(name="objectSelected", type="SuperMap.Web.Util.CallBackEvent")]

的标签，并且在方法里面需要调用基类的addEventListener(eventName,handler);方法注册事件，详见扩展事件，这里的参数后缀名为”$Function”。