SimCube.Automation JavaScript

Version 1.3

编写：谭立方

日期：2015.5.14

目 录

1. 前言

本文档是 SimCube.Automation（仿立方之自动化）中扩展的 JavaScript 使用指南，包含了如何使用 JavaScript 创建模型，也即创建各种仿函、参数、建立参数之间链接的方法；以及各种仿函与参数的 JavaScript 方法。

2. 术语

仿函（SimFunction）：代表一项研发活动、功能、某个应用程序的封装，带有输入、输出参数。

驱动仿函（CompositeSimFunction）：也是一种仿函，但可包含其它子仿函，可控制子仿函的运行逻辑。

参数（Parameter）：数据的封装，为仿函提供输入与输出数据。

链接（Link）：在仿函的参数中建立的表达数据之间的关系。

模型（Model）：由驱动仿函、仿函、参数、链接建立的一种层次化的结构，可解决具体业务问题。

模型元素（ModelElement）：构成模型的仿函与参数对象都称为模型元素。

设计时（Design Time）：当模型不在运行时，那么它处于设计时。

运行时（Run Time）：当模型运行时，那么它处于运行时。

3. 模型概述

Automation 中的模型是一个层次结构，是一颗由各种对象组成的对象树，如图 1 模型的层次结构所示。模型的根节点是一个 Assembly 驱动仿函对象，因此模型自身也是一个仿函对象，其下包含各个子（驱动）仿函对象，仿函带有自身的输入、输出参数对象，参数可通过组参数分类，各个仿函参数的值可通过链接传递，驱动数据在模

型中的流动。

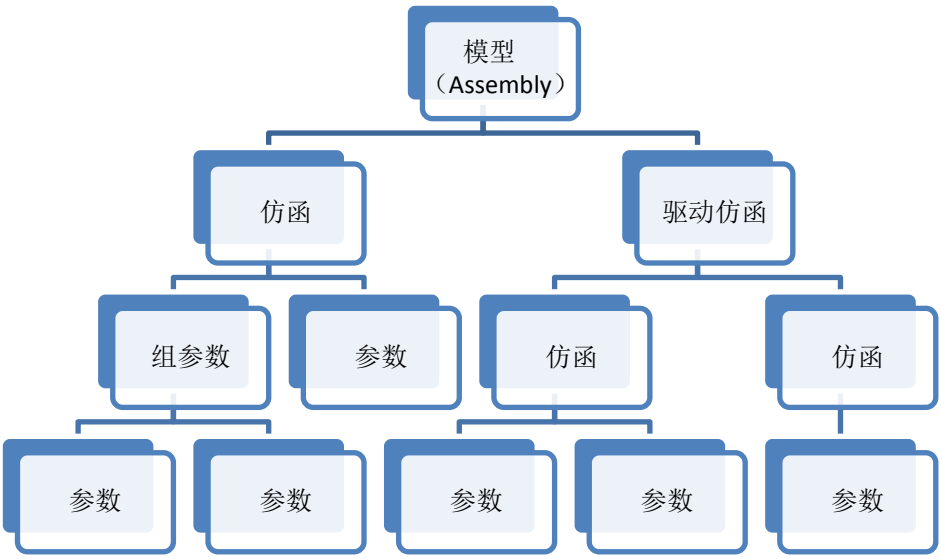

图 1 模型的层次结构

在 Automation 的运行环境中，模型可以处于两个状态：设计期与运行期。在设计期可通过 JavaScript 接口给模型添加仿函、参数和链接对象，当然，一般都是在一个 JavaScript 脚本文件中完成模型的创建的，通过软件打开这个脚本文件即可建立模型；在运行期工作流引擎驱动模型中的各个仿函运行，模型自身也是一个仿函，因此理解仿函的运行模型很重要，如图 2 仿函运行模型所示。

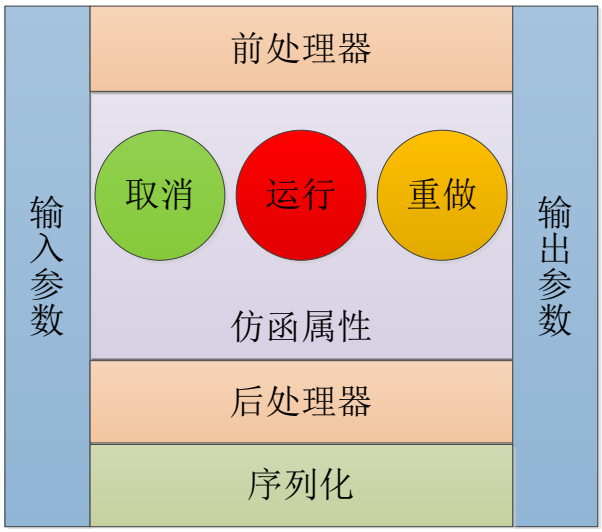

图 2 仿函运行模型

一个仿函对象内部有若干仿函的属性，同时包含若干输入、输出参数，还可设置

它的前、后处理器，分别在仿函运行的前、后两个时刻运行，仿函的运行结果通过序列化保存到数据库中。另外，仿函在运行过程中可通过后处理器设置重做标记，工作流引擎将再次驱动这个仿函运行；也可通过前、后处理器设置取消标记，工作流引擎将取消该仿函的运行，同时也不会把数据序列化至数据库中。

实际上模型在设计期与运行期都可动态的修改模型本身，提供很强的建模能力，在后面的章节中将详述这些功能的接口方法使用。

3.1 仿函

Automation 中提供的仿函分两大类：基础仿函与基础驱动仿函。通过扩展 Automation 还可容纳其它类型的仿函。仿函与驱动仿函的区别是，驱动仿函可包含子仿函，它有驱动子仿函运行的逻辑，每一个驱动仿函的驱动逻辑是不一样的。

仿函是通过 rom 对象中的接口创建的。

3.1.1 基础仿函

在 Automation 中有 6 种基础仿函，如表 1 基础仿函列表所示。

表 1 基础仿函列表

类名	中文名称	描述
SimFunction	仿函	无具体功能，建模时用于占位
AutomationScript	自动化脚本	仿函的功能由编写的 JavaScript 代码决定
SimApp	应用	集成外部可执行软件
ModelDataExtract	模型数据提取	提取其它模型中的数据
DataBase	数据库	与各种关系型数据库系统交互
COM	组件	与 Windows 系统的 COM 组件交互

3.1.2 基础驱动仿函

在 Automation 中提供了 6 种基础驱动仿函，如表 2 基础驱动仿函列表所示。

表 2 基础驱动仿函列表

类名	中文名称	描述
Assembly	装配	模型的根仿函，也可包含其它仿函，按次序执行内部的子仿函
Sequence	序列	按次序执行内部的子仿函
Switch	分支	按条件执行一个分支
Parallel	并行	同时运行多个分支
Bus	总线	以总线调度的方式驱动子仿函执行
Loop	循环	对子仿函迭代运行，直到满足停止条件

3.2 参数

Automation 中提供的参数分两大类：基础单值参数与基础多值参数。通过扩展 Automation 还可容纳其它类型的参数。单值参数与多值参数的区别是，单值参数对象只能包含某种数据类型的一个值，而多值参数是可以包含某种数据类型的多个值。

参数除了有不同的类型之外，还有不同的模式，最典型的有输入、输出、组、链接等等模式，详细说明见下面的章节。

参数是通过 rom 对象中的接口创建的。

3.2.1 基础单值参数

在 Automation 中有 6 种基础单值参数，如表 3 基础单值参数列表所示。

表 3 基础单值参数列表

类名	中文名称	描述
Group	组	用于给参数分组
IntegerParameter	整数	表示整数值
DoubleParameter	双精度	表示双精度的浮点数
BooleanParameter	布尔	表示布尔类型的值
StringParameter	字符串	表示字符串类型的值

FileParameter	文件	表示文件系统中的文件

在 JavaScript 语言中不区分整数与双精度数，统一使用 Number 对象表示，但在 Automation 内部是区分的。Group 参数是给仿函中的参数分类，可容纳多个子参数，但这些子参数也不是组参数的值，因此归入基础单值参数类别中。

3.2.2 基础多值参数

在 Automation 中有 6 种基础多值参数，如表 4 基础多值参数列表所示。

表 4 基础多值参数列表

类名	中文名称	描述
IntegerArrayParameter	整数数组	表示整数数组
DoubleArrayParameter	双精度数组	表示双精度浮点数数组
BooleanArrayParameter	布尔数组	表示布尔值的数组
StringArrayParameter	字符串数组	表示字符串的数组
TableParameter	表	基于列数据的表，每列可有不同的数据类型
CompositeParameter	复合	可组合多个参数的复合参数

3.3 参数链接

在 Automation 中有两种参数链接的方式，一种是等式链接，另一种是表达式链接。等式链接可把一个仿函中的参数等值传递给另一个仿函的参数（一般是输入模式的参数）；表达式链接可以把一个或多个仿函的参数通过一个有效的 JavaScript 表达式描述，表达式的计算结果传递给另一个仿函的参数（一般是输入模式的参数）。

参数链接是通过链接参数所在的仿函对象创建的。具体接口下文详述。

3.4 各种模型元素关系

上述仿函与参数等模型元素在 C++语言层面的类图如图 3 模型元素类图所示。

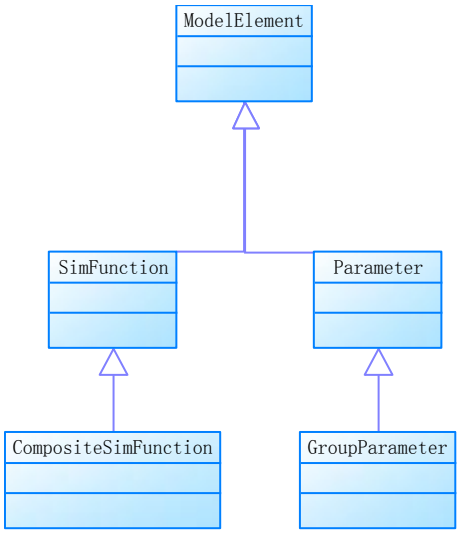

图 3 模型元素类图

仿函与参数都有共同的基类 ModelElement，基础仿函不可带子仿函的仿函从类 SimFunction 中继承，可带子仿函的驱动仿函都从类 CompositeSimFunction 类继承，而 CompositeSimFunction 类又是从类 SimFunction 继承，各种参数类型从类 Parameter 继承。

3.5 建模示例

为了对 Automation 中使用 JavaScript 脚本建模有一个简单直观的理解，这里给出一个简单的建模示例。示例中创建了一个模型，然后在模型中添加了两个自动化脚本仿函，每个仿函创建了三个参数，第二个仿函的两个输入参数都是链接参数，分别使用等式链接与表达式链接创建，每个自动化脚本仿函有一段 JavaScript 脚本用于实现它的功能。下面以行号为标记详细描述各个接口方法的功能。

```
var model = new rom.Assembly('model', null);
rom.model = model;

var script = new rom.AutomationScript('sf1', model);
var p1 = new rom.DoubleParameter('data1', rom.Input, script);
p1.value = 12;
p1.units = '公里';
p1.description = '距离';
var p2 = new rom.DoubleParameter('data2', rom.Output, script);
var p3 = new rom.DoubleParameter('data3', rom.Output, script);

script.scriptFunction = function() {
    self.data2.value = self.data1 + Math.random() * 100;
    self.data3.value = self.data1 * Math.cos(self.data2);
};

var script2 = new rom.AutomationScript('sf2', model);
var p4 = new rom.DoubleParameter('data1', rom.Input, script2);
p4.units = '米';
var p5 = new rom.DoubleParameter('data2', rom.Input, script2);
var p6 = new rom.DoubleParameter('data3', rom.Output, script2);

script2.scriptFunction = function() {
    self.data3.value = (self.data1 + Math.sin(self.data2)) * 100;
};

script2.createLinkByParameters(p5, p2);
script2.createLinkByNames(p4.fullName, p1.fullName + ' * 1000');
```

1：使用 rom 对象中的 Assembly 类创建一个装配驱动仿函对象，该对象的名称是“model”，父对象是 null，为空值，表示模型的根节点，它没有父对象，新建的对象赋值给变量 model。这里要注意：模型的根对象是一个 Assembly 对象，并且它的名称必须是“model”，在 JavaScript 引擎中可使用全局变量 model 引用该模型。

2：把新建的局部变量 model 赋值给 rom 对象的 model 属性，此时真正在 Automation 中创建了一个模型，没有赋值之前 model 对象只是在内存中的一个对象层次树而已。当然，这个赋值动作也可以在这个对象层次树建立完毕之后进行。

4：使用 rom 对象中的 AutomationScript 类创建一个自动化仿函对象，该对象的名

称是“sf1”，父对象是刚创建的 model 对象，也即 sf1 仿函是 model 的子仿函对象，新建的对象赋值给变量 script。

5：使用 rom 对象中的 DoubleParameter 类创建一个双精度参数对象，该对象的名称是“data1”，参数的模式是输入参数，使用 rom.Input 属性表示，参数的父对象是 script，也即表示是仿函 script 对象的参数，新建的对象赋值给变量 p1。

6：通过参数对象 p1 的属性 value 设置参数的值为 12。

7：通过参数对象 p1 的属性 units 设置参数的单位是“公里”。

8：通过参数对象 p1 的属性 description 设置参数的描述信息。

9-10：同样的方式创建 script 仿函的两个参数对象，只不过这里是两个输出模式的参数对象，这通过 rom.Output 属性设置。

12-15：设置自动化脚本仿函 script1 在运行时调用的函数对象，函数中使用的 self 变量代表仿函对象自身，也即代表 script 对象。

17：与第 4 行同样的方式创建第二个自动化脚本仿函对象。

18：与第 5 行同样的方式创建 script2 的参数对象 p4。

19：设置参数 p4 的单位为“米”。

20-21：创建两个参数对象。

23-25：设置自动化脚本仿函 script2 在运行时调用的函数对象。

27：通过仿函的 createLinkByParameters 方法在自动化脚本仿函 script2 中把参数 p5 创建为一个等式链接参数对象，链接了仿函 script1 中的 p2 输出参数。

28：通过仿函的 createLinkByNames 方法在自动化脚本仿函 script2 中把参数 p4 创建为一个表达式链接参数对象，链接了仿函 script1 中的 p1 参数，并在该参数值的基础上乘以 1000，达到转化单位制的目的。

这里只是以 AutomationScript 仿函为例创建了一个简单的模型，实际上，在利用

其它仿函和参数创建模型时与上面的代码结构是一致的，只不过每个仿函或参数有其特定的方法而已。

4. rom 对象

在 Automation 软件中，rom 对象是一个核心对象，从上面创建的模型示例可见，模型中的仿函与参数对象都是通过该对象的属性类创建的，rom 对象为创建模型，操作模型提供了大量的便利设施，下面详细描述该对象拥有的常量、属性与方法。

4.1 常量

为了提供在 Automation 中使用 JavaScript 编码建模的便捷性，在 rom 对象中设置了多个常量，用于标记参数的模式与仿函的工作路径。如表 5 rom 对象常数所示。

表 5 rom 对象常数

常数	值	说明
参数模式		
rom.Input	1	输入参数
rom.Output	2	输出参数
rom.Group	4	组参数
rom.Linked	8	链接参数
rom.RefInput	33	输入引用参数
rom.RefOutput	34	输出引用参数
rom.ComInput	17	输入复合参数
rom.ComOutput	18	输出复合参数
rom.InputFile	65	输入文件参数
rom.OutputFile	66	输出文件参数
工作路径		
rom.BaseDirectory	1	存放仿函的输入、输出文件路径

rom.InputDirectory	2	存放仿函的输入文件路径
rom.OuputDirectory	4	存放仿函的输出文件路径

如果不使用上述常数，那么在建模过程中书写将繁琐，例如创建一个输入模式的整数参数场景，使用上述常数将是如下代码：

var p = new rom.IntegerParameter(‘p’, rom.Input, simfun);

如果不使用则代码如下：

var p = new rom.IntegerParameter(‘p’, rom.IntegerParameter.Input, simfun);

将导致多输入不少字符，因此在建模时尽量使用这些便捷常数，使得代码更加简洁。

4.2 属性

4.2.1 currentModelId

该属性是一个可读写属性，用于获取或设置当前模型的标识号，可获取的模型号都是已经保存至 rom 文件中的，在一个 rom 文件中可保存多个模型，也即存在多个模型标识号。若还未保存模型至 rom 文件，那么该属性值是-1，模型标识号从 1 开始计数。若设置的模型标识号还未在 rom 文件中保存，那么该设置无效。*版本 1.2 中实现*。

4.2.2 currentResultId

该属性是一个可读写属性,用于获取或设置当前模型标识号下的当前结果标识号，可获取的结果号都是已经保存至 rom 文件中的,对应一个模型标识号可保存多个结果，也即存在多个结果标识号。若当前模型还从未执行，那么该属性值是-1，结果标识号从 1 开始计数。若设置的结果标识号还未在 rom 文件中保存，那么该设置无效。*版本 1.2 中实现*。

4.2.3 model

该属性是一个可读可写属性，用于获取和设置模型对象，该对象实际上是一个 Assembly 对象，其父对象为 null 值，代表了模型的根节点。此属性与全局空间下的 model 实际上是指向同一个对象。

4.2.4 modelFile

该属性是一个只读属性，获取模型保存后的 rom 文件绝对路径字符串。

4.3 类

在 rom 对象中关联了 Automation 中所有用于建模的仿函与参数类型，使用 new 操作符可相应创建这些模型元素。在模型概述中通过列表的形式整理了可创建的仿函与参数类型，具体的创建方法放在相应仿函与参数的章节中描述。

4.4 方法

4.4.1 clearModel()

清除当前模型，将删除当前模型中所有的仿函，参数对象，以及链接，并新建一个 Assembly 对象，仿函名称是“model”，也即模型的根对象。

4.4.2 clearRetrievedResultData(resultId)

清除从结果标识号是 resultId 中的结果中提取的所有参数数据。*版本 1.3 中实现。*

4.4.3 cloneParameterByJSON(json)

字符串 json 是一个参数对象的 JSON 表示，通过它复制一个参数对象并返回。

4.4.4 createLocalInputTextFile(inputPath, templateFilePar)

根据模板文件参数（FileParameter）对象 templateFilePar 创建输入文件，并把它放

入路径 inputPath 文件夹中，如果成功则返回创建的本地文件路径，否则返回空字符串。

该方法主要用于对文本文件中的数据进行参数化，模板文件参数对象表示的文本文件是经过标记的文件，文件中标记的行被提取为一个参数对象，在模型运行时，通过获取这些标记并结合当前模型中对应标记的参数的值创建一个新的文本文件。

4.4.5 createParametersFromFileParameter(filePar)

解析文件参数对象 filePar 引用文本文件中的参数标记并创建参数对象，这些创建的参数对象都是文件参数对象 filePar 的子参数。当 filePar 关联的文件不存在，或是非文本文件，无注释字符串，都将不会解析出参数，返回-1。同时把文件参数 filePar 设置为在图形界面下不可删除，其下创建的子参数也不可删除。返回从 filePar 引用文件中解析出的参数个数。

4.4.6 deleteModelElement(modelElement)

删除模型元素 modelElement，如果成功则返回 true，否则返回 false。

4.4.7 enterBatchProcessStatus(inBatchProcess)

在建模的过程中，有时希望把一系列建模操作，如新建多个模型元素的操作作为一个过程，在图形界面的重做与撤销时作为一个动作。此时可在多个建模操作之前传入 true 参数调用此方法，完成多个建模操作之后传入 false 参数调用此方法。

实际上，用户很少有调用此方法的场景，该方法主要被 Automation 内部使用。

4.4.8 loadModel(fileName, [modelId])

从文件路径 fileName 表示的数据库文件中加载模型标识号为 modelId 的模型，如果未指定 modelId，那么将加载 rom 文件中最新的模型，也即 rom 文件中最大的模型标识号表示的模型。成功加载返回 true，否则返回 false。

4.4.9 loadResult(resultId)

加载数据库中结果标识号为 resultId 的结果至内存中，成功加载返回 true，否则返回 false。

4.4.10 loadScriptFile(scriptFileName)

加载 JavaScript 文件 scriptFileName，成功加载返回 true，否则返回 false。

4.4.11 maxResultId()

返回当前模型中最大的结果标识号。

4.4.12 parameterFromFullName(fullName)

返回参数名称是 fullName 的参数对象，若无对应名称的参数对象则返回 null。

4.4.13 parseLocalOutputTextFile(outputPath, templateFilePar)

根据输出文件模板中的标记，解析输出文本文件，并更新相关输出参数的值。参数 outputPath 是输出文件所在的目录，参数 templateFilePar 是输出文件的模板文件参数对象。成功解析返回 true，否则返回 false。

4.4.14 placeParameterFromRetrieve(targetPar, sourcePar, resultId, index)

该方法用于从数据库中提取迭代计算过程产生的数据放入参数 targetPar 中，参数 resultId 指定了模型的计算结果标识号，参数 index 指定了迭代的索引，参数对象 sourcePar 是保存在数据库中的参数数据，把该参数在第 index 次计算的结果提取出来，赋值给 targetPar 参数对象。要注意的是，targetPar 参数对象可以不在模型中，只是一个用于存放提取数据的参数而已。成功设置参数数据则返回 true，否则返回 false。

该方法一般用于从计算结果数据库中提取模型迭代过程产生的数据。调用该方法前必须先调用了 retrieveParameter(par, resultId)，把相关数据预先从数据库文件中提取放入内存中，否则将有可能出错。

4.4.15 resultList(par, resultId)

返回从计算结果数据库中提取的模型迭代过程中所有的结果数据数组，par 表示参数对象，参数 resultId 指定了模型的计算结果标识号。

该方法一般用于从计算结果数据库中提取模型迭代过程产生的数据。调用该方法前必须先调用了 retrieveParameter(par, resultId)，把相关数据预先从数据库文件中提取放入内存中，否则将有可能出错。

4.4.16 resultListCount(par, resultId)

获取参数 par 在当前计算结果标识号下迭代计算过程中的数据个数。如果该参数还未从数据库中提取，那么返回-1。参数 resultId 指定了模型的计算结果标识号。

该方法一般用于从计算结果数据库中提取模型迭代过程产生的数据。调用该方法前必须先调用了 retrieveParameter (par, resultId)，把相关数据预先从数据库文件中提取放入内存中，否则将有可能出错。

4.4.17 retrieveParameter(par, resultId)

提取结果标识号为 resultId 中的参数对象 par 的参数数据至内存中。

该方法一般用于从计算结果数据库中提取模型迭代过程产生的数据。

4.4.18 runModel()

运行模型。调用此方法时如果模型未保存为 rom 文件，则调用操作无效。

4.4.19 runModelSync()

同步运行模型。调用此方法时如果模型未保存为 rom 文件，则调用操作无效。该方法用于在控制台程序 AutomationShell 中编写 JavaScript 脚本时使用，调用该方法可执行模型，并等待模型执行完毕才返回。*版本 1.2 中实现。*

4.4.20 runSimfun(sf)

单独运行仿函对象 sf。调用此方法时如果模型未保存为 rom 文件，则调用操作无效。

4.4.21 isRunning()

判断模型是否在运行中，若是则返回 true，否则返回 false。

4.4.22 saveModel(fileName, [modelId])

保存模型至文件名是 fileName 的数据库中，如果没有指定 modelId 参数，那么将首先在数据库中创建一个 modelId，然后再保存模型数据至 modelId 标识的数据库表中。保存成功返回 true，否则返回 false。

4.4.23 stop()

发出停止模型运行的命令，模型将等待正在运行的仿函完毕，然后停止。

4.4.24 waitForRetrieveFinished(par, resultId, [msecs])

等待参数对象 par 从数据库中提取完毕，等待时间是 msecs 毫秒。msecs 值为-1 则永久等待，不指定 msecs 的值则默认等待 30 秒。参数 resultId 指定了模型的计算结果标识号。

从数据库中提取参数数据是在工作线程中进行的，因此在编写提取数据的脚本文件中为了同步的需要，一般需调用此方法等待提取参数数据的动作结束。

该方法一般用于从计算结果数据库中提取模型迭代过程产生的数据。调用该方法前必须先调用了 retrieveParameter(par, resultId)，把相关数据预先从数据库文件中提取放入内存中，否则将有可能出错。

5. 模型元素对象表示

在阐述各个模型元素对象的接口之前，这里先介绍对模型元素对象在 Automation 的 JavaScript 引擎中是如何表示以及如何访问的。

模型中各个模型元素在逻辑上是一个层次结构，在表示与访问方式上也体现了这种层次结构。通过全局对象 model，通过它可按层次一级一级地访问模型中任意一个模型元素对象。

5.1 model 对象

在 Automation 软件中的 JavaScript 引擎的全局空间有一个 model 对象，实际上此对象是 rom.model 的引用，在 JavaScript 解释器下使用 model==rom.model 表达式的求值结果为 true，它们都引用了模型的根节点对象，也即一个 Assembly 仿函对象。

前面已描述了模型是一个层次结构，通过全局 model 对象可访问模型的根节点，model 是 Assembly 驱动仿函对象，它包含子仿函对象，子仿函对象再包含参数对象，形成对象的层次结构。

5.2 链式访问法则

为了能在 JavaScript 引擎中方便地引用各个模型元素对象，Automation 中设计了依据层次结构关系的链式访问对象法则，通过模型元素的名称引用对应的模型元素对象，在 JavaScript 引擎中模型元素的名称也即它的变量名，这也解释了模型的根节点取名为“model”以及各个模型元素名称必须是一个有效的 JavaScript 变量标识符的原因。例如通过 model.sf1 可引用模型中的仿函对象 sf1，而 model.sf1.par1 可引用模型中仿函对象 sf1 中的参数对象 par1。通过这种链式的访问方式可引用模型中的任一个模型元素对象。

6. process 对象

Automation 中的 process 对象添加了一些特别的属性与方法。版本 1.1 中实现。

6.1 属性

6.1.1 mainWindow

该属性获得 Automation 软件的主窗口对象。

6.2 方法

6.2.1 scriptPath()

该方法返回 Automation 软件中脚本扩展插件的路径字符串对象。

7. 模型元素

模型是由模型元素构成的，模型元素包括仿函与参数两种类型，这里将阐述适用于仿函与参数对象共有的属性与方法，本质上，模型元素（ModelElement 类）是仿函（SimFunction 类）和参数（Parameter 类）的基类。

7.1 属性

7.1.1 actionListCode 与 actionListFunction

字符串，但 actionListFunction 可设置为一个函数对象。这是两个用于获取或设置模型元素的动作列表源代码的属性，一般在脚本中不使用 actionListCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 actionListFunction 属性以函数体的方式设置。

模型元素的动作列表在 Automation 的图形界面下将被解析为各个动作列表的菜单项，并能在该属性中设置响应此菜单的脚本代码。下面以一个示例说明如何设置这

个属性。

```
modelElement.actionListFunction = function() {
    [{
        menuItem: '菜单一',
        statusTip: '菜单一描述',
        constraint: 1,
        icon: '$$/menuitem1.png',
        actor: function(self) {
            log('响应菜单一');
        }
    }, {
        menuItem: 'seperator'
    }, {
        menuItem: '菜单二',
        defaultActor: true,
        statusTip: '菜单二描述',
        icon: '$$/menuitem2.png',
        actor: function(self) {
            log('响应菜单二');
        }
        }];
};
```

示例中 modelElement 是一个模型元素对象，可以是一个仿函或参数对象，它通过属性 actionListFunction 设置一个匿名无参函数。函数内部只能定义一个数组对象，这是动作列表属性的要求，必须遵守。数组对象的每一个元素描述一个动作项，体现在图形界面上也即一个菜单项。

数组的元素是一个对象，该对象的键值对如表 6 动作列表中数组元素对象所示。

表 6 动作列表中数组元素对象

键	值类型	值描述
menuItem	字符串	动作名称，图形界面上是菜单的名称，当该值是“seperator”时，则表示菜单上的一个分隔条，此时忽略其它键值。此键

		值不可缺。
statusTip	字符串	可选项，当鼠标移动到该菜单时，在窗口状态栏中显示的动作描述信息，在右键弹出菜单下无效，只在下拉菜单中有效。
icon	字符串	可选项，指定菜单的图标文件，可以是绝对路径，也可以使用$$代表 Automation 应用程序所在目录的 images 子目录。
actor	函数	带一个参数 self，参数名称不一定非是 self，也可以用其它有效变量名，但用 self 是一个好习惯，它表示当前模型元素对象。函数内部可访问软件中的 JavaScript 各种接口。
defaultActor	布尔	可选项，为 true 表示该动作在菜单上是缺省的，点击工具栏上的按钮即可执行该动作，无需打开菜单。一个动作列表中只能有一个缺省菜单项，设置多个 true 时，将覆盖前面的设置项。
constraint	整数	可选项，指定菜单项应用的约束，默认值为 0，表示在模型设计时与运行时都可应用;当为 1 是只能在模型设计时应用;当为 2 时只能在模型运行时应用。*版本 1.3 中实现*。

7.1.2 canDelete

布尔值。设置或获取模型元素是否可被删除，可删除则为 true，否则为 false，默认是 true。有的模型元素是不可被删除的，例如 Loop 仿函中 For 循环类型的 From，To，Increment 等自动创建的参数对象，它们是与 Loop 对象依存的，这些参数必须设置为不可删除。

7.1.3 description

字符串。设置或获取模型元素的描述字符串。

7.1.4 fullName

字符串。只读属性，获取模型元素在模型中的全名称，全名称是以 model 开始的以“.”分割的多段字符串，每一段字符串表示一个模型元素，左侧的模型元素是右侧

模型元素的父对象。

7.1.5 name

字符串。设置或获取模型元素的名称，该名称必须是一个有效的 JavaScript 变量标识符，在 JavaScript 环境中可通过该名称表示的变量获取模型元素对象。

7.1.6 respondChangedCode 与 respondChangedFunction

字符串，但 respondChangedFunction 可设置为一个函数对象。这是两个用于获取或设置响应模型元素对象变化的源代码属性，一般在脚本中不使用 respondChangedCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串的字面量编码使得脚本代码不易读，容易出错，应该使用 respondChangedFunction 属性以函数体的方式设置。

respondChangedFunction 设置的函数对象是一个无参函数，函数将在该模型元素发生变化（例如参数的值改变了）时被调用，可模型的设计期与运行期都可被调用。

7.1.7 saveToDatabase

布尔值。设置或获取模型元素在模型运行期是否需要保存到数据库中。若为 true，表示需要保存，否则不保存。当仿函对象设置为 false 时，则其包含的子仿函或参数对象都将不会保存至数据库中。

7.1.8 self

模型元素对象。该属性是在特定时刻动态创建的，代表模型元素对象本身，使用完毕后会被自动删除。一般在为模型元素添加的各类脚本代码中使用，可以便捷地访问当前模型元素对象。

随 self 属性动态创建的还有一个方法 dynamicModifyModel()，用于动态修改模型。

7.1.9 valid

布尔值。设置或获取模型元素是否有效，若为 true 表示有效，否则无效。当仿函对象的一个输入参数发生变化时，该仿函以及它的所有参数，包括依赖于此输入参数的仿函或参数都将失效。

7.2 方法

7.2.1 append(modelElem) / prepend(modelElem) / insert(index, modelElem)

在当前模型元素对象上的一个容器尾部追加(或前部或 index 索引处)插入模型元素 modelElem，若成功返回 true，否则返回 false。该方法是否成功取决与当前模型元素对象与追加或插入的模型元素对象的类型。如表 7 模型元素对象插入模型元素对象所示，第一列表示当前模型元素对象的类型，第一行表示待追加或插入的模型元素对象类型，True 表示该配合有意义，该方法有可能返回 true；False 表示无意义，该方法一定返回 false。

表 7 模型元素对象插入模型元素对象

	CompositeSimFunction	SimFunction	GroupParameter CompositeParameter FileParameter	其它 Parameter
CompositeSimFunction	True	True	True	True
SimFunction	False	False	True	True
GroupParameter CompositeParameter FileParameter	False	False	True	True
其它 Parameter	False	False	False	False

7.2.2 dynamicModifyModel(code)

该方法与 self 属性一样是在特定时刻动态创建的，它在创建后是模型元素对象的方法，使用完毕后会被自动删除。一般在为模型元素添加的各类脚本代码中使用，通

过 self. dynamicModifyModel(code)的方式调用。用于动态的修改模型，如给模型添加新的仿函或参数，甚至可以在模型运行期间修改模型。

参数 code 是用于修改模型的 JavaScript 代码。如果 code 代码中的最后一条语句是创建一个模型元素对象，那么将返回该对象。

7.2.3 inModel()

如果当前模型元素在模型中，那么返回 true，否则返回 false。

7.2.4 remove(modelElem)

在当前模型元素对象上移除模型元素 modelElem，若成功返回被移除的模型元素对象，否则返回 null。

对于仿函对象只可移除它内部的参数对象；对于驱动仿函对象可移除它的子仿函对象以及参数对象；对于组参数、复合参数、文件参数对象可移除它包含的参数对象。

7.2.5 removeAllParameter()

在当前模型元素对象上移除它所有的参数对象，返回被移除参数对象的数组。

7.2.6 removeAllSimfun()

在当前模型元素对象上移除它所有的仿函对象，返回被移除仿函对象的数组。

7.2.7 removeParameterAt(index)

在当前模型元素对象上移除索引是 index 处的参数对象，返回被移除的参数对象。

7.2.8 removeSimfunAt(index)

在当前模型元素对象上移除索引是 index 处的仿函对象，返回被移除的仿函对象。

7.2.9 replace(index, newModelElem)

在当前模型元素对象上替换索引是 index 处的模型元素为新的模型元素对象

newModelElem，若成功返回被替换的模型元素对象，否则返回 null。

对于仿函对象只可替换它内部的参数对象；驱动仿函可替换它内部的子仿函对象或参数对象；对于组参数、复合参数、文件参数对象可替换它内部的参数对象。

7.2.10 toJSON()

返回模型元素对象的 JSON 字符串表示。如果要对该 JSON 字符串解析，那么这里要注意返回的 JSON 字符串需在左侧添加字符“{”，右侧添加字符“}”确保其有效。

7.2.11 toString()

返回模型元素的字符串表示。

7.2.12 tooltip()

以字符串数组的形式返回模型元素的工具提示信息，用于在图形界面上显示模型元素的信息。

8. 基础仿函

基础仿函是不能包含子仿函的仿函，所有基础仿函的基类是 SimFunction，该类提供的属性与方法可被其它基础仿函，包括基础驱动仿函访问。下面首先阐述基础仿函 SimFunction 的属性与方法。

8.1 SimFunction

仿函 SimFunction 不提供任何功能，它只是一个占位仿函，一般在建模初期可使用它表示这里应该有一个仿函。虽然它未提供运行期的具体功能，但是它提供了一些常量、大量属性与方法，这些接口都可以在其它仿函对象中直接访问。

底层 C++实现中 SimFunction 从 ModelElement 类中派生，因此它可访问 ModelElement 的接口，也即上述的模型元素的接口。

8.1.1 常量

8.1.1.1 仿函工作目录

当仿函对象在运行的过程中需要输入文件或者创建了输出文件时，此时一般需设置该仿函的工作目录，工作目录的设定可通过几个常数确定，如所示。

常数	值	描述
rom.SimFunction.None	0	仿函运行过程中不需要工作目录，默认是此设置。
rom.SimFunction.BaseDirectory	1	缺省的工作目录，仿函运行过程中自动确定目录名称，该目录是建立在模型 rom 文件所在的目录下，一般地，如果不区分输入、输出文件目录，那么只需设置此常数即可。
rom.SimFunction.InputDirectory	2	仿函运行时希望把输入文件单独放在一个目录下，则设置此常数，将在仿函文件目录下再创建一个“input”子目录。
rom.SimFunction.OutputDirectory	4	仿函运行时希望把输出文件单独放在一个目录下，则设置此常数，将在仿函文件目录下再创建一个“output”子目录。

上面几个常数可通过“|”操作符组合起来，如 rom.SimFunction.InputDirectory | rom.SimFunction.OutputDirectory 将创建“input”与“output”两个子目录。注意到使用这些常数的记号很冗长，为此，在 rom 对象下也定义了上表中的后三个常数，这样上面的表达式可简化为 rom.InputDirectory | rom.OutputDirectory。

8.1.2 属性

8.1.2.1 additionalIconFilePath

字符串。设置或获取仿函对象附加的图标文件路径，默认不设置该路径。使用字符串"$$"代表可执行程序所在目录下的 images 子目录。当一个仿函对象设置了附加的

图标文件时，在图形界面的模型视图以及树形视图中将以该图标表示此仿函对象。

8.1.2.2 cancel

布尔值。设置或获取仿函对象运行时是否可取消的属性，若为 true 则模型运行至此仿函时将取消该仿函的运行。该属性可在仿函的前、后处理器中设置。

8.1.2.3 derivedName

字符串。设置或获取仿函对象的派生名称属性。默认为空。用于通过已有仿函类编写 JavaScript 脚本实现派生仿函类。版本 1.1 中实现。

8.1.2.4 links

链接集合对象。只读属性，获取仿函对象链接参数的所有链接对象。链接参数的名称是返回链接集合对象中的键的名称，键的值是该参数的链接对象（类 ParameterLink）。

8.1.2.5 next

仿函对象。只读属性，获取仿函对象所在父仿函中的后一个仿函对象，若无则返回 null。

8.1.2.6 parameters

参数集合对象。只读属性，获取仿函对象的所有参数集合，参数的名称是返回的参数集合对象中的键的名称，键的值是该参数对象。

8.1.2.7 parentSimfun

仿函对象。只读属性，获取仿函对象的父对象，是一个驱动仿函对象，可以包含子仿函。

8.1.2.8 postProcessingCode 与 postProcessingFunction

字符串，但 postProcessingFunction 可设置为一个函数对象。这是两个用于获取或设置仿函对象后处理器源代码的属性，一般在脚本中不使用 postProcessingCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 postProcessingFunction 属性以函数体的方式设置。

后处理中设置的代码将在仿函主体功能运行完毕后执行，因此在这里可对仿函运行的结果进行评估，根据业务逻辑可设置仿函进行重做，再运行一次，或取消等操作。

8.1.2.9 preProcessingCode 与 preProcessingFunction

字符串，但 preProcessingFunction 可设置为一个函数对象。这是两个用于获取或设置仿函对象前处理器源代码的属性，一般在脚本中不使用 preProcessingCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 preProcessingFunction 属性以函数体的方式设置。

前处理中设置的代码将在仿函主体功能运行前执行，因此在这里可对仿函运行的条件进行评估，根据业务逻辑可取消仿函的运行，或者根据当前模型运行上下文环境预先准备仿函的运行条件。

8.1.2.10 previous

仿函对象。只读属性，获取仿函对象所在父仿函中的前一个仿函对象，若无则返回 null。

8.1.2.11 redo

布尔值。设置或获取仿函对象运行时是否重做的属性，若为 true 则该仿函将再次被工作流引擎调度运行。该属性一般在仿函的后处理器中设置，可根据当前运行结果

进行判断。

8.1.2.12 simfuns

仿函对象数组。只读属性，获取仿函对象的子仿函对象数组，若无子仿函对象，则返回空数组。

8.1.2.13 workDirectory

数值。设置或获取仿函对象工作目录的设置常数。

8.1.3 方法

8.1.3.1 rom.SimFunction(name, [parent=null])

创建仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问仿函对象，也即此名称是仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

8.1.3.2 allParameterNames()

返回仿函对象包含的所有参数对象的全名称数组对象。

8.1.3.3 check()

检查仿函对象是否符合运行条件，如果符合则返回空字符串，否则返回错误信息字符串。

8.1.3.4 containSimfun(name)

判断在当前仿函对象中是否包含名称为 name 的仿函对象，若包含则返回 true，否则返回 false。

8.1.3.5 containParameter(name)

判断在当前仿函对象中是否包含名称为 name 的参数对象，若包含则返回 true，否则返回 false。

8.1.3.6 createLinkByNames(leftParName, rhs)

创建链接参数。leftParName 是将要创建为链接参数的参数全名或名称，该参数必须是当前仿函对象中的参数，rhs 是链接参数的右侧表达式字符串，通过在表达式中使用其它仿函对象中的参数全名链接这些参数。若成功创建了链接参数返回创建的链接参数对象（ParameterLink），否则返回 null。

该方法主要用于创建复杂的表达式链接，等式链接使用 createLinkByParameters 方法。

8.1.3.7 createLinkByParameters(leftPar, rightPar)

创建链接参数。leftPar 是将要创建为链接参数的参数对象，该参数必须是当前仿函对象中的参数，rightPar 是其它仿函对象中的参数对象。若成功创建了链接参数返回创建的链接参数对象（ParameterLink），否则返回 null。

该方法主要用于创建参数之间的等式链接，表达式链接可使用 createLinkByNames 方法。

8.1.3.8 encodeSimfunNumber()

返回仿函对象在模型中位置的编码，模型中的仿函对象都有唯一编码。编码后的字符串可用于创建一个文件目录，用于放置仿函对象在运行时产生的中间文件，或者输入、输出文件。

8.1.3.9 parameterByName(name, [fullName=false])

按参数名称 name 获取仿函对象中的参数对象，名称可以是参数的全名，也可以

只是参数的名称，同时可包含组参数的名称。fullName 指定参数名称是否为全名，缺省不是全名。若仿函对象中存在对应名称的参数对象，则返回该参数对象，否则返回 null。

此方法不会遍历它的子仿函中的参数对象。

8.1.3.10 simfunByName(name, [fullName=false])

按仿函名称 name 获取仿函对象中的子仿函对象，名称可以是仿函的全名，也可以只是仿函的名称。fullName 指定仿函名称是否为全名，缺省不是全名。若仿函对象中存在对应名称的子仿函对象，则返回该子仿函对象，否则返回 null。

8.1.3.11 simfunCount()

返回仿函对象及其所有子仿函对象的总个数，此方法遍历子仿函对象的子仿函对象。

8.1.3.12 workDirectoryByFlag(wd)

根据工作目录的常量 wd 返回仿函对象的一个工作目录字符串，参数 wd 可以是工作目录常量之一。

仿函的工作目录是根据设置的常量在仿函运行之前的那一刻自动创建的，如果仿函还未调度运行，那么通过该方法获取的工作目录是一个空字符串。

8.2 AutomationScript

自动化脚本仿函（AutomationScript）通过编写 JavaScript 代码实现仿函的功能，可访问 Automation 软件中 JavaScript 引擎所有的接口。

底层 C++实现中 AutomationScript 从 SimFunction 类中派生，因此它可访问 SimFunction 的接口。

8.2.1 属性

8.2.1.1 scriptCode 与 scriptFunction

字符串，但 scriptFunction 可设置为一个函数对象。这是两个用于获取或设置自动化脚本仿函对象的 JavaScript 源代码的属性，一般在脚本中不使用 scriptCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 scriptFunction 属性以函数体的方式设置。

在脚本代码中可使用 self 属性引用当前自动化脚本仿函对象。

8.2.2 方法

8.2.2.1 rom.AutomationScript(name, [parent])

创建自动化脚本仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问自动化脚本仿函对象，也即此名称是自动化脚本仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

8.3 SimApp

应用仿函（SimApp）用于集成外部的可执行程序，当 Automation 的工作流引擎调度 SimApp 运行时，SimApp 将根据给定的命令行参数调用设置的外部可执行程序。

底层 C++实现中 SimApp 从 SimFunction 类中派生，因此它可访问 SimFunction 的接口。

8.3.1 属性

8.3.1.1 appPath

字符串。设置或获取外部可执行程序的文件路径。

8.3.1.2 args

字符串数组。设置或获取外部可执行程序的命令行参数，数组元素的设置应该与调用外部可执行程序的命令行参数顺序一致。

8.3.2 方法

8.3.2.1 rom.SimApp(name, [parent])

创建应用仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问应用仿函对象，也即此名称是应用仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

8.4 ModelDataExtract

模型数据提取仿函（ModelDataExtract）用于提取已运行的另一个模型的结果数据，提取的方式是通过编写一段 JavaScript 脚本，访问另一个模型结果数据的 rom 文件，访问该文件提取数据（主要是参数结果数据）的方法定义在 rom 对象中，可参看 rom 对象中的有关方法。

底层 C++实现中 ModelDataExtract 从 SimFunction 类中派生，因此它可访问 SimFunction 的接口。

8.4.1 属性

8.4.1.1 extractCode 与 extractFunction

字符串，但 extractFunction 可设置为一个函数对象。这是两个用于获取或设置模型数据提取仿函对象的用于提取数据的 JavaScript 源代码属性，一般在脚本中不使用 extractCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 extractFunction 属性以函数体

的方式设置。

8.4.1.2 modelId

整数。设置或获取待提取数据模型中的模型标识号。缺省值是 1。

8.4.1.3 resultId

整数。设置或获取待提取数据模型中的结果标识号。缺省值是 1。

8.4.1.4 romFile

字符串。设置或获取待提取数据的模型 rom 文件。

8.4.2 方法

8.4.2.1 rom. ModelDataExtract(name, [parent])

创建模型数据提取仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问模型数据提取仿函对象，也即此名称是模型数据提取仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

8.5 Database

数据库仿函（Database）用于与关系型数据库系统交互，交互的方式是编写一段 JavaScript 代码，与数据库直接交互的 SQL 代码可以在 JavaScript 代码段中编写。

底层 C++实现中 Database 从 SimFunction 类中派生，因此它可访问 SimFunction 的接口。

8.5.1 属性

8.5.1.1 connectionName

字符串。设置或获取数据库的连接名称。

8.5.1.2 databaseName

字符串。设置或获取数据库的名称。对于 sqlite 数据库即文件名称。

8.5.1.3 driverName

字符串。设置或获取数据库驱动器名称。目前支持的关系型数据库系统有 Sqlite，MySQL，PostgreSQL。可分别使用字符串“QSQLITE”，“QMYSQL”，与“QPSQL”设置。

8.5.1.4 hostName

字符串。设置或获取数据库所在的主机名称，也可是主机所在的 IP 地址。Sqlite 数据库不需设置此属性。

8.5.1.5 interactCode 与 interactFunction

字符串，但 interactFunction 可设置为一个函数对象。这是两个用于获取或设置数据库仿函对象的与数据库系统交互的 JavaScript 源代码属性，一般在脚本中不使用 interactCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 interactFunction 属性以函数体的方式设置。

Automation 的工作流引擎调度数据库仿函对象运行时，与之关联的数据库系统连接已经打开，在与数据库交互的代码中可直接访问已打开的数据库连接对象。该对象可通过 self. openedDatabase 访问，对象的类型是 DatabaseObject，它可使用的接口可参看此类的文档。

8.5.1.6 openedDatabase

DatabaseObject 对象。此对象是在数据库仿函运行的时刻动态创建的，代表打开的数据库对象，可在 interactCode 交互代码中使用 self. openedDatabase 访问打开的数

据库对象。

8.5.1.7 password

字符串。设置或获取数据库的密码，Sqlite 数据库无需设置。

8.5.1.8 port

整数。设置或获取数据库的端口号，Sqlite 数据库无需设置。

8.5.1.9 stayConnected

布尔值。设置或获取保持数据库连接。当此属性值为 true 时，在数据库仿函运行完毕后并不关闭已打开的连接，当为 false 时将会关闭。当数据库仿函在一个迭代过程中时，如果设置为 false，那么会反复打开与关闭数据库连接，影响效率，此时可以把该属性设置为 true。

8.5.1.10 userName

字符串。设置或获取数据库的用户名，Sqlite 数据库无需设置。

8.5.2 方法

8.5.2.1 rom.Database(name, [parent])

创建数据库仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问数据库仿函对象，也即此名称是数据库仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

8.5.2.2 closeDatabase()

如果数据库仿函对象打开了一个数据库连接，则此方法可关闭该连接。

8.6 COM

COM 组件仿函（COM）用于与微软的 COM 组件对象交互，交互的方式是编写一段 JavaScript 代码。

底层 C++实现中 COM 从 SimFunction 类中派生，因此它可访问 SimFunction 的接口。

8.6.1 属性

8.6.1.1 scriptCode 与 scriptFunction

字符串，但 scriptFunction 可设置为一个函数对象。这是两个用于获取或设置 COM 组件仿函对象的与微软 COM 组件交互的 JavaScript 源代码属性，一般在脚本中不使用 scriptCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 scriptFunction 属性以函数体的方式设置。

8.6.2 方法

8.6.2.1 rom.COM(name, [parent])

创建 COM 组件仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问 COM 组件仿函对象，也即此名称是 COM 组件仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

8.6.2.2 callComFunction(func, args)

调用加载的 COM 组件实例中的方法 func，func 是 COM 组件中的方法原型，以字符串形式传入，并把组件方法的参数以 args 数组传递。返回的值是 COM 组件中的方法 func 的返回结果，如果该方法存在输出参数，那么也在此返回值中，返回值将是

一个数组对象。

8.6.2.3 callComFunctionWithOutputParameter(func, args)

调用加载的 COM 组件实例中的方法 func，func 是 COM 组件中的方法原型，以字符串形式传入，并把组件方法的参数以 args 数组传递。返回的值是 COM 组件中的方法 func 的返回结果。

8.6.2.4 comProperty(name)

获取加载的 COM 组件实例中的属性 name 的值。

8.6.2.5 generateDocument()

获取加载的 COM 组件实例中的文档，文档内容以 HTML 格式输出。

8.6.2.6 isNull()

判断 COM 仿函中的 COM 组件是否已经加载，返回 true 表示已加载，否则返回 false。

8.6.2.7 loadCOMObject(comObject)

加载以 comObject 标识的 COM 组件。参数 comObject 可以是 COM 组件注册的 UUID 值，例如"{8E27C92B-1264-101C-8A2F-040224009C02}"，这是最高效的加载 COM 组件的方法；也可以是 COM 组件的类名（可带可不带版本号），例如 "MSCal.Calendar"，这是第二高效的加载方法；最后，也可以使用 COM 组件的全名，例如"Calendar Control 9.0"，这是最低效的加载方式。如果该方法返回 true 表示加载成功，返回 false 表示加载失败。

8.6.2.8 setComProperty(name, value)

设置已加载的 COM 组件实例的属性名称是 name 的属性值为 value。

9. 基础驱动仿函

基础驱动仿函包括了装配（Assembly）、序列（Sequence）、分支（Switch）、并行（Parallel），循环（Loop），总线（Bus）共两个仿函。底层 C++实现中是从类 CompositeSimFunction 派生的，而后者是从 SimFunction 派生的，因此它们都可以访问 SimFunction 的接口。类 CompositeSimFunction 没有对外暴露 JavaScript 接口。

9.1 Sequence

序列仿函（Sequence）是驱动仿函，继承于 CompositeSimFunction 仿函，它可包含子仿函，它在运行时将调度所有的子仿函按顺序依次运行，也即首先是它的第一个子仿函被调度运行，在它运行完毕后，再调度第二个子仿函运行，以此类推，当最后一个子仿函运行完毕后，序列仿函才结束它的运行。

序列仿函没有比 SimFunction 更多的属性和方法，这里给出创建它的方法说明。

9.1.1 rom.Sequence(name, [parent])

创建序列仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问序列仿函对象，也即此名称是序列仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

9.2 Assembly

装配仿函（Assembly）是驱动仿函，它继承于 Sequence，同样可以包含子仿函，对子仿函的调度逻辑与 Sequence 一致，这里不再赘述，请参考 Sequence 的说明。

创建 Assembly 对象的方法也与 Sequence 的类似，rom.Assembly(name, [parent])。在前面模型概述中阐述了模型是一个层次结构，它的根节点是 Assembly 对象。另外，顾名思义，装配仿函可封装其它仿函组成子模型，然后子模型可嵌入到不同的模型中

去，达到对子模型重用的效果。

9.3 Parallel

并行仿函（Parallel）是驱动仿函，继承于 CompositeSimFunction 仿函，它可包含子仿函，它在运行时将调度所有的子仿函同时运行，当所有的子仿函运行完毕后，并行仿函才结束它的运行。

此仿函没有提供更多的接口。创建它的方式与其它仿函一致：rom.Parallel(name, [parent])。

9.4 Switch

分支仿函（Switch）是驱动仿函，继承于 CompositeSimFunction 仿函，它可包含子仿函，包含的子仿函是对应的分支，每个分支对应一个可设置的条件，它使用 JavaScript 表达式描述，求值后的结果是布尔值，求值的顺序是对应分支的子仿函列表顺序。

分支仿函有两种运行模式：排他模式与非排他模式。在排他模式中，依序对分支条件求值，当第一次遇到某个分支条件求值为 true 时，此时将忽略对其它分支条件的求值，同时调度求值为 true 的分支运行；在非排他模式中，将会对所有分支条件求值，求值结果为 true 的所有分支都将被调度运行，运行的顺序按其在分支仿函中的索引序号进行。

另外，还可以设置缺省分支，可设置为缺省分支的子仿函是最后一个子仿函。当所有的分支条件求值都为 false 时，若存在缺省分支，那么缺省分支将被调度运行。

被调度运行的子仿函分支运行完毕后，分支仿函才结束它的运行。

9.4.1 属性

9.4.1.1 defaultSwitch

布尔值。设置或获取是否设置了缺省分支。默认值是 false，表示不设置缺省分支。

9.4.1.2 exclusive

布尔值。设置或获取是否设置了排他运行模式。默认值是 true，表示以排他模式运行。

9.4.2 方法

9.4.2.1 rom.Switch(name, [parent])

创建分支仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问分支仿函对象，也即此名称是分支仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

9.4.2.2 setBranchCondition(branchSimfun, condition)

设置分支仿函 branchSimfun 的运行条件是 condition，若设置成功则返回 true，否则返回 false。

9.5 Loop

循环仿函（Loop）是驱动仿函，继承于 CompositeSimFunction 仿函，它可包含子仿函，可调度包含的子仿函迭代运行，子仿函运行的顺序按其在循环仿函中的索引序号进行。循环仿函有三种循环模式，分别是：For，While 与 RepeatUntil。

For 循环与一般程序语言中的 for 语句类似，需要设置一个循环变量，给定循环变量一个初值进行迭代，如果未达到终值，那么循环变量按步长累加，直到达到终值，停止循环。

While 循环也与一般程序语言中的 while 语句类似，需要设置一个循环条件，当该条件求值为真时将进行循环。RepeatUntil 与 While 类似，只不过它是当循环条件求值为假时将进行循环。在这两种类型的循环中有初始值（initials），输入值（inputs），反馈值（feedbacks），复数表示它们可以有多个，但数量必须相等。当循环开始运行时，首先把初始值赋值给输入值，如果没有初始值，那么就使用输入值的当前值；迭代一次后判断循环条件，如果还可以迭代，再把反馈值赋值给输入值，进行第二次迭代。初始值可以是一个常量，或者另一个参数，或者一个表达式，也可以为空，此时当循环开始迭代时将使用输入值的当前值。反馈值一般是仿函的输出参数，它在每次迭代计算后会改变，也可以是一个表达式。

9.5.1 常量

循环仿函中有一个标识循环类型的常量 LoopType。

常数	值	描述
rom.Loop.For	0	For 循环
rom. Loop.While	1	While 循环
rom. Loop.RepeatUntil	2	RepeatUntil 循环

9.5.2 属性

9.5.2.1 loopType

整数，LoopType 类型。设置或获取循环仿函的循环类型，循环仿函默认是 For 循环类型。

9.5.2.2 loopParameter

引用参数，ReferenceParameter。设置或获取 For 循环类型的循环参数，它是一个引用参数类型。

9.5.2.3 from

引用参数，ReferenceParameter。设置或获取 For 循环类型的初值，它是一个引用参数类型。

9.5.2.4 to

引用参数，ReferenceParameter。设置或获取 For 循环类型的终值，它是一个引用参数类型。

9.5.2.5 increment

引用参数，ReferenceParameter。设置或获取 For 循环类型的步长值，它是一个引用参数类型。

9.5.2.6 condition

引用参数，ReferenceParameter。设置或获取 While/RepeatUntil 循环类型的条件，它是一个引用参数类型。

9.5.2.7 initials

引用数组参数，ReferenceArrayParameter。设置或获取 While/RepeatUntil 循环类型中的初始值，初始值可以有多个，类似于 For 循环中的初值，它是一个引用数组参数类型。

9.5.2.8 inputs

引用数组参数，ReferenceArrayParameter。设置或获取 While/RepeatUntil 循环类型中的输入值，输入值可以有多个，类似于 For 循环中的循环参数，它是一个引用数组参数类型。

9.5.2.9 feedbacks

引用数组参数，ReferenceArrayParameter。设置或获取 While/RepeatUntil 循环类型中的反馈值，反馈值可以有多个，它是一个引用数组参数类型。

9.5.3 方法

9.5.3.1 rom.Loop(name, [parent])

创建循环仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问循环仿函对象，也即此名称是循环仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

9.5.3.2 setLoopParameterByGeneralParameter(par)

为 For 循环模式设置循环参数 par，这里是以参数对象的方式直接设置，没有通过引用参数封装。

9.5.3.3 setFrom(val)

为 For 循环模式设置循环的初值，是一个字面量，在 JavaScript 中可以是 Number 类型的变量。

9.5.3.4 setTo(val)

为 For 循环模式设置循环的终值，是一个字面量，在 JavaScript 中可以是 Number 类型的变量。

9.5.3.5 setIncrement(val)

为 For 循环模式设置循环的步长，是一个字面量，在 JavaScript 中可以是 Number 类型的变量。

9.5.3.6 setCondition(conditon)

为 While/RepeatUntil 循环模式设置循环的条件为 condition，是一个字符串，包含一个表达式描述循环条件。

9.6 Bus

总线仿函（Bus）是驱动仿函，继承于 CompositeSimFunction 仿函，它可包含子仿函，它在调度其子仿函运行时与其它驱动仿函不同，不是单一的规则确定哪个子仿函被调度运行。

总线仿函首先必须指定它的一个子仿函用于当它运行时调度的第一个子仿函，同时每一个子仿函需设置一个运行条件，当一个子仿函运行完毕后，所有子仿函的运行条件都将被求值。它也有类似迭代运行的机制，可设置一个停止条件，当停止条件为真时总线仿函运行完毕，否则将再次调度它的某个符合运行条件的子仿函运行，若无符合条件的子仿函，则总线仿函结束运行。当有多个子仿函的运行条件求值为真时，总线仿函对每个子仿函还可设置优先级，若设置了优先级，那么符合条件且优先级高的被调度运行，若没有设置优先级，或运行条件为真的多个子仿函优先级相同，那么将调度子仿函在总线仿函中排序靠前的运行。最后，每一个子仿函还可设置一个当自己运行完毕后接下来运行的子仿函，这种调度运行是无条件的，无需经过对此子仿函运行条件求值。以上是总线仿函运行过程的机理，用流程图表示则如图 4 总线仿函运行过程所示。

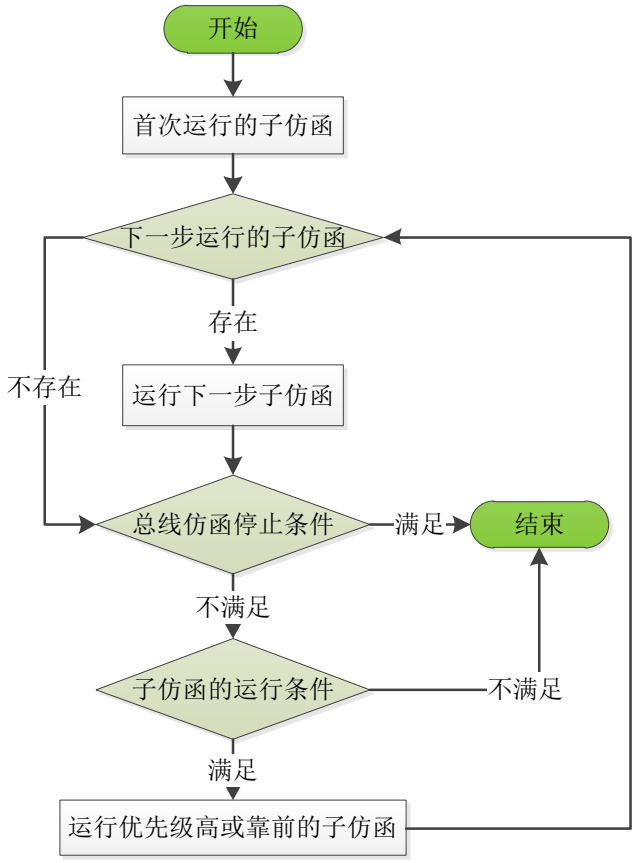

图 4 总线仿函运行过程

9.6.1 属性

9.6.1.1 firstRunIndex

整数。设置或获取首次运行子仿函的索引，驱动仿函中子仿函的索引号基于 0，若设置的索引号不在有效范围内，则不作任何操作。

9.6.1.2 nextRuns

整数数组。设置或获取各个子仿函下一步运行的子仿函索引，数组中第一个元素代表第一个子仿函运行完毕后无条件运行的下一步子仿函的索引号，以此类推。默认值都是-1，表示无下一步运行的子仿函。

9.6.1.3 priorities

整数数组。设置或获取各个子仿函运行的优先级，数值越大表示优先级越高，数

组中第一个元素代表第一个子仿函的优先级，以此类推。默认值都是 0，表示无优先级。

9.6.1.4 runConditionsCode

字符串数组。设置或获取各个子仿函运行的条件，每个条件是一段 JavaScript 代码，求值后返回最后一条语句的结果，若为真则此子仿函具备运行的条件。数组中第一个元素代表第一个子仿函的运行条件代码，以此类推。默认值都是 false，表示都不可运行。

当运行条件代码超过一行时，建议使用方法 setRunConditionFunction()设置，这个方法可直接传递一个函数对象，便于书写代码。

9.6.1.5 stopConditionCode 与 stopConditionFunction

字符串。但 stopConditionFunction 可设置为一个函数对象。这是两个用于获取或设置总线仿函停止条件源代码的属性，一般在脚本中不使用 stopConditionCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 stopConditionFunction 属性以函数体的方式设置。

9.6.2 方法

9.6.2.1 rom.Bus(name, [parent])

创建总线仿函对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问总线仿函对象，也即此名称是总线仿函对象的变量名称，parent 参数指定仿函对象的父对象，一般是驱动仿函对象，也可以为 null，表示无父对象。

9.6.2.2 setRunConditionFunction(index, func)

设置第 index 个子仿函的运行条件为函数对象 func 中的代码，将对 func 中的语句

求值，若最后一条语句求值为真则表示子仿函运行条件为真。

10. 优化仿函

用于优化计算的仿函是驱动仿函，一般都要驱动它的子仿函迭代计算，从中寻优。同时优化计算中一般涉及设计变量、约束与目标三个内容，这些基本内容将在下节描述。

10.1 优化基础

10.1.1 常量

优化计算的设计变量或约束一般有上下限，寻优目标可以是目标函数的最小、大值，也可以是一个目标值。关于这些设定，在名字空间 rom.Optimization 中提供了 5 个常量值，见下表所示。

常数	值	描述
rom.Optimization.Lower	1	设计变量或约束下限
rom.Optimization.Upper	2	设计变量或约束上限
rom.Optimization.LowerUpper	3	设计变量或约束上、下限
rom.Optimization.Target	4	目标值
rom.Optimization.Minimize	8	最小值
rom.Optimization.Maximize	16	最大值

10.1.2 属性

10.1.2.1 designVariables

数组。获取或设置设计变量。优化计算中可设置一个或多个设计变量，每一个设计变量是数组中的一个元素，元素本身也是一个数组，该数组有四个元素：

1. 第一个元素是一个字符串，是代表设计变量的参数全名；

2. 第二个元素是一个整数，指定设计变量的上、下限模式，由上表的常数确定；
3. 第三个元素是一个浮点数，指定设计变量的下限；
4. 第四个元素是一个浮点数，指定设计变量的上限。

10.1.2.2 constraints

数组。获取或设置约束。优化计算中可设置零个或多个约束，每一个约束是数组中的一个元素，元素本身也是一个数组，该数组有五个元素：

1. 第一个元素是一个字符串，是代表约束的参数全名；
2. 第二个元素是一个整数，指定约束的上、下限模式，由上表的常数确定；
3. 第三个元素是一个浮点数，指定约束的下限；
4. 第四个元素是一个浮点数，指定约束的上限；
5. 第五个原始是一个浮点数，指定约束的目标值。

10.1.2.3 Objective

数组。获取或设置优化目标。优化计算中可设置 1 个或多个优化目标，每一个目标是数组中的一个元素，元素本身也是一个数组，该数组有三个元素：

1. 第一个元素是一个字符串，是代表目标的参数全名；
2. 第二个元素是一个浮点数，指定目标值；
3. 第三个元素是一个整数，指定目标是最大值还是最小值模式，由上表的常数确定。

10.1.3 方法

10.1.3.1 appendConstraint(variableName, mode, target, lowBound, upperBound)

追加一个约束。方法中的五个参数分别对应约束属性中数组的五个元素。

10.1.3.2 appendConstraintByParameter(par, mode, target, [useParBound=true], [lowBound=0.], [upperBound=0.])

通过指定一个参数对象 par 追加一个约束。参数 mode 指定约束的模式，target 指定约束目标值，可选布尔参数 useParBound 指定是否使用参数本身的上下限，默认值是使用，可选参数 lowBound 与 upperBound 分别指定约束的上下限值，默认值是 0。

方法添加约束成功返回 true，否则返回 false。

10.1.3.3 appendDesignVariable(variableName, mode, lowBound, upperBound)

追加一个设计变量。方法中的四个参数分别对应设计变量属性中数组的四个元素。

10.1.3.4 appendDesignVariableByParameter(par, mode, [useParBound=true], [lowBound=0.], [upperBound = 0.])

通过指定一个参数对象 par 追加一个设计变量。参数 mode 指定设计变量的模式，可选布尔参数 useParBound 指定是否使用参数本身的上下限，默认值是使用，可选参数 lowBound 与 upperBound 分别指定设计变量的上下限值，默认值是 0。

方法添加设计变量成功返回 true，否则返回 false。

10.1.3.5 appendObjective(variableName, mode, [target=0.])

追加一个优化目标。方法中的三个参数分别对应优化目标属性中数组的三个元素。

10.1.3.6 appendObjectiveByParameter(par, mode, [target=0.])

通过指定一个参数对象 par 追加一个优化目标。参数 mode 指定优化目标的模式，可选参数 target 指定优化目标的目标值，默认值是 0。

方法添加优化目标成功返回 true，否则返回 false。

10.2 PSO

粒子群优化（PSO）是驱动仿函，继承于 CompositeSimFunction 仿函，它可包含

子仿函，可调度包含的子仿函迭代运行，子仿函运行的顺序按其在 PSO 仿函中的索引序号进行。目前只实现了单目标约束功能。*版本 1.2 中实现*。

10.2.1 属性

10.2.1.1 c1

数值。获取或设置粒子群优化算法的局部认知权重值，默认值是 1.49445。

10.2.1.2 c2

数值。获取或设置粒子群优化算法的全局认知权重值，默认值是 1.49445。

10.2.1.3 iterationCount

数值。获取或设置粒子群优化算法的各个粒子的最大迭代步数，默认值是 50。

10.2.1.4 particlesCount

数值。获取或设置粒子群优化算法的粒子数量，默认值是 10。

10.2.1.5 w

数值。获取或设置粒子群优化算法的惯性权重值，默认值是 0.729。

11. 功能模型单元仿函

功能模型单元仿函实现了与功能模型单元（FMU，Functional Mock-up Units）交互的功能，FMU 是基于功能模型接口（FMI，Functional Mock-up Interface）的实现，它本质上是一个动态链接库，以 fmu 为其文件名后缀。

11.1 CoSimulation

联合仿真（CoSimulation）是驱动仿函，继承于 CompositeSimFunction 仿函，它可包含子仿函，可调度包含的子仿函迭代运行，子仿函运行的顺序按其在 CoSimulation

仿函中的索引序号进行。本质上，它与 Loop 仿函类似，但它可以驱动功能模型单元代理仿函（FMUProxy）。目前实现了 FMI2.0 中的联合仿真。*版本 1.3 中实现*。

11.1.1 属性

11.1.1.1 currentTime

数值。获取或设置联合仿真的当前时间。

11.1.1.2 startTime

数值。获取或设置联合仿真的开始时间。

11.1.1.3 stepTime

数值。获取或设置联合仿真的迭代时间步长。

11.1.1.4 stopConditionCode 与 stopConditionFunction

字符串，但 stopConditionFunction 可设置为一个函数对象。这是两个用于获取或设置联合仿真迭代运行时停止条件 JavaScript 源代码的属性，一般在脚本中不使用 stopConditionCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 stopConditionFunction 属性以函数体的方式设置。

在脚本代码中可使用 self 属性引用当前联合仿真的仿函对象。

11.1.1.5 stopTime

数值。获取或设置联合仿真的终止时间。

11.1.1.6 stopTimeDefined

布尔值。获取或设置联合仿真是否设置了终止时间，若没有设置终止时间，该属性为 false，此时联合仿真仿函将以 stopConditionCode 代码确定何时退出迭代。

11.2 FMUProxy

功能模型单元代理仿函（FMUProxy）是实现了驱动功能模型单元（也即实现了FMI 接口）的仿函，它的父仿函只能是联合仿真（CoSimulation）。

底层 C++实现中 FMUProxy 从 SimFunction 类中派生，因此它可访问 SimFunction 的接口。

11.2.1 属性

11.2.1.1 fmuFilePath

字符串。获取或设置 fmu 文件的路径。功能模型单元的 fmu 文件通过 Automation 软件导入后，也即纳入 Automation 软件的管理之中，在设置 fmu 文件的路径时，无需设置全路径，只要设置 fmu 的文件名称即可。

12. 基础单值参数

基础单值参数是一个参数中只包含一个数据类型值的参数。在底层实现 C++中，所有的参数，包括基础多值参数，都直接或间接从类 Parameter 中派生，而类 Parameter 不可实例化，它只为派生类提供属性和方法。类 Parameter 是从模型元素类 ModelElement 中派生，因此模型元素中所有的接口在 Parameter 中都可访问。

12.1 Parameter

Parameter 是所有参数的基类，本身不可实例化，只为其它派生类参数提供接口。

12.1.1 常量

12.1.1.1 参数模式

参数模式指定了一个参数是否输入、输出、组参数、链接参数或者文件参数等等。参数模式是在 Parameter 类中定义的，但为了代码编写的便利，在 rom 对象中也作了

等同的定义。如参数模式中的输入模式在 Parameter 中可以这样访问：rom.Parameter.Input，而在 rom 对象中可直接使用 rom.Input 访问。请参考 rom 对象中的常数定义，这里不再赘述。

12.1.2 属性

12.1.2.1 mode

整数。设置或获取参数模式。

12.1.2.2 owningSimfun

仿函对象。只读属性，获取参数所属的仿函对象，若参数无所属的仿函对象则返回 null。

12.1.3 方法

12.1.3.1 fromVariant(value)

从值 value 中重构一个参数，成功返回 true，否则返回 false。

12.1.3.2 toVariant()

把参数对象作为一个值返回。

12.1.3.3 type()

返回参数类型的字符串表示。

12.1.3.4 unit()

返回参数的单位制，若无则返回空字符串。

12.1.3.5 isValidate()

判断参数是否有效，有效返回 true，否则返回 false。

12.2 GroupParameter

组参数（GroupParameter）不封装任何数据类型，它用于为其它参数（也即组参数的子参数）分类，相当于其它参数的一个容器。

底层 C++实现中 GroupParameter 从 Parameter 类中派生，因此它可访问 Parameter 的接口，也即上述参数的接口。

12.2.1 属性

12.2.1.1 parameters

子参数集合对象。只读属性，获取组参数对象的所有子参数集合，子参数的名称是返回的子参数集合对象中的键的名称，键的值是该子参数对象。

12.2.2 方法

12.2.2.1 rom.GroupParameter(name, [parent=null])

创建组参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问组参数对象，也即此名称是组参数对象的变量名称，parent 参数指定组参数对象的仿函父对象，也可以为 null，表示无父对象。

12.2.2.2 rom.GroupParameter(name, parentGroupPar)

与上一个方法类似，但这里的第二个参数也是一个组参数对象，也即组参数的父对象也是一个组参数对象。在图形界面禁止在组参数下再创建组参数。

12.2.2.3 containParameter(name)

判断组参数对象中是否包含名称是 name 的子参数，包含则返回 true，否则返回 false。

12.2.2.4 parameterByName(name, [fullName=false])

返回组参数对象中名称是 name 的子参数对象，名称可以是参数的全名，也可以只是参数的名称，同时可包含组参数的名称。fullName 指定参数名称是否为全名，缺省不是全名。若组参数对象中存在对应名称的参数对象，则返回该参数对象，否则返回 null。

12.3 IntegerParameter

整数参数（IntegerParameter）拥有一个 64 位长度的整数值，实际上在 JavaScript 中是不区分整数与浮点数的，它统一使用 Number 类型表示。在底层 C++层面是使用的整数类型。它从 Parameter 类中派生，因此它可访问 Parameter 的接口。

12.3.1 属性

12.3.1.1 lowerLimitEnabled

布尔值。设置或获取整数参数的下限是否启用，设置 true 表示启用，默认是 false，表示不启用。

12.3.1.2 upperLimitEnabled

布尔值。设置或获取整数参数的上限是否启用，设置 true 表示启用，默认是 false，表示不启用。

12.3.1.3 lessEqual

布尔值。设置或获取整数参数上限设置模式的开闭区间，设置 true 表示闭区间，表示参数值可小于等于上限，默认是 true；设置 false 表示开区间，表示参数值可小于上限。

12.3.1.4 greaterEqual

布尔值。设置或获取整数参数下限设置模式的开闭区间，设置 true 表示闭区间，表示参数值可大于等于下限，默认是 true；设置 false 表示开区间，表示参数值可大于下限。

12.3.1.5 units

字符串。设置或获取整数参数的单位制。若无单位制则返回空字符串。

12.3.1.6 lowerLimit

整数。设置或获取整数参数的下限值。

12.3.1.7 upperLimit

整数。设置或获取整数参数的上限值。

12.3.1.8 customFormatEnabled

布尔值。设置或获取整数参数是否启用自定义格式。默认为 false，以十进制表示整数，支持 2~36 进制。

12.3.1.9 base

整数。设置或获取整数参数启用自定义格式后的进制。默认是 10，表示十进制，支持 2~36 进制。

12.3.1.10 enumEnabled

布尔值。设置或获取整数参数是否启用枚举机制。默认是 false，不启用。若启用了则参数只能在设置的各个枚举值中取值。

12.3.1.11 enumIndex

整数。设置或获取枚举值的索引，基于 0。

12.3.1.12 enumValues

整数数组。设置或获取枚举值集合。

12.3.1.13 enumNames

字符串数组。设置或获取与枚举值一一对应的枚举名称集合，确保首先设置了枚举值集合，否则设置的枚举名称集合无效。

12.3.1.14 value

整数。设置或获取整数参数的值。

12.3.2 方法

12.3.2.1 rom.IntegerParameter(name, mode, [parent=null])

创建整数参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

12.3.2.2 rom.IntegerParameter(name, mode, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

12.3.2.3 valueOf()

返回参数的值。

12.4 DoubleParameter

双进度参数（DoubleParameter）底层的 C++数据类型是 double，实际上在 JavaScript 中是不区分整数与浮点数的，它统一使用 Number 类型表示。它从 Parameter 类中派生，

因此它可访问 Parameter 的接口。

12.4.1 属性

12.4.1.1 lowerLimitEnabled

布尔值。设置或获取双精度参数的下限是否启用，设置 true 表示启用，默认是 false，表示不启用。

12.4.1.2 upperLimitEnabled

布尔值。设置或获取双精度参数的上限是否启用，设置 true 表示启用，默认是 false，表示不启用。

12.4.1.3 lessEqual

布尔值。设置或获取双精度参数上限设置模式的开闭区间，设置 true 表示闭区间，表示参数值可小于等于上限，默认是 true；设置 false 表示开区间，表示参数值可小于上限。

12.4.1.4 greaterEqual

布尔值。设置或获取双精度参数下限设置模式的开闭区间，设置 true 表示闭区间，表示参数值可大于等于下限，默认是 true；设置 false 表示开区间，表示参数值可大于下限。

12.4.1.5 units

字符串。设置或获取双精度参数的单位制。若无单位制则返回空字符串。

12.4.1.6 lowerLimit

数值。设置或获取双精度参数的下限值。

12.4.1.7 upperLimit

数值。设置或获取双精度参数的上限值。

12.4.1.8 customFormatEnabled

布尔值。设置或获取双精度参数是否启用自定义格式。默认为 false，在图形界面上以小数点后 6 位的精度显示一个浮点数。

12.4.1.9 precision

整数。设置或获取双精度参数小数点后的显示精度。默认是小数点后 6 位数字，支持 0~20 位。

12.4.1.10 format

字符串。设置或获取双精度参数的表示风格。它有五种表示方法，如下表所示。

格式	描述
e	显示为这样的格式[-]9.9e[+|-]999
E	显示为这样的格式[-]9.9E[+|-]999
f	显示为这样的格式[-]9.9
g	选择 e 或 f 格式中表示更紧凑的
G	选择 E 或 f 格式中表示更紧凑的

12.4.1.11 enumEnabled

布尔值。设置或获取双精度参数是否启用枚举机制。默认是 false，不启用。若启用了则参数只能在设置的各个枚举值中取值。

12.4.1.12 enumIndex

整数。设置或获取枚举值的索引，基于 0。

12.4.1.13 enumValues

数值数组。设置或获取枚举值集合。

12.4.1.14 enumNames

字符串数组。设置或获取与枚举值一一对应的枚举名称集合，确保首先设置了枚举值集合，否则设置的枚举名称集合无效。

12.4.1.15 value

数值。设置或获取双精度参数的值。

12.4.2 方法

12.4.2.1 rom.DoubleParameter(name, mode, [parent=null])

创建双精度参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

12.4.2.2 rom.DoubleParameter(name, mode, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

12.4.2.3 valueOf()

返回参数的值。

12.5 BooleanParameter

布尔参数（BooleanParameter）底层的 C++数据类型是 bool。它从 Parameter 类中派生，因此它可访问 Parameter 的接口。

12.5.1 属性

12.5.1.1 value

数值。设置或获取布尔参数的值。

12.5.2 方法

12.5.2.1 rom.BooleanParameter(name, mode, [parent=null])

创建布尔参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

12.5.2.2 rom. BooleanParameter (name, mode, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

12.5.2.3 valueOf()

返回参数的值。

12.6 StringParameter

字符串参数（StringParameter）表示一段字符串。它从 Parameter 类中派生，因此它可访问 Parameter 的接口。

12.6.1 属性

12.6.1.1 enumEnabled

布尔值。设置或获取字符串参数是否启用枚举机制。默认是 false，不启用。若启用了则参数只能在设置的各个枚举值中取值。

12.6.1.2 enumIndex

整数。设置或获取枚举值的索引，基于 0。

12.6.1.3 enumValues

数值数组。设置或获取枚举值集合。

12.6.1.4 enumNames

字符串数组。设置或获取与枚举值一一对应的枚举名称集合，确保首先设置了枚举值集合，否则设置的枚举名称集合无效。

12.6.1.5 value

字符串。设置或获取字符串参数的值。

12.6.2 方法

12.6.2.1 rom.StringParameter(name, mode, [parent=null])

创建字符串参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

12.6.2.2 rom. StringParameter (name, mode, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

12.6.2.3 valueOf()

返回字符串参数的值。

12.7 FileParameter

文件参数（FileParameter）表示文件系统中的一个文件。它从 GroupParameter 类中派生，因此可访问 GroupParameter 的接口，也获得了可包含多个子参数的功能，如果能从文件参数绑定的文件中提取参数，那么提取的参数将成为文件参数的子参数。

12.7.1 属性

12.7.1.1 comment

字符串。设置或获取文本文件的行注释字符串，例如 Python 语言的行注释字符串是“#”，JavaScript 语言的行注释字符串是“//”。如果是普通的文本文件，而不是编程语言的代码文件，那么可以设置一个特殊的字符（例如该文本文件不曾使用的字符）用作行注释字符串。

12.7.1.2 externalOpenCode 与 externalOpenFunction

字符串。但 externalOpenFunction 可设置为一个函数对象。这是两个用于获取或设置文件参数外部打开操作源代码的属性，可使用 JavaScript 代码编写操作被绑定文件的功能，简单的如使用一个合适的外部软件打开它。Automation 图形界面上文件参数的右键菜单提供一个菜单项使用该功能。该属性默认是空字符串，使用被绑定文件在操作系统上默认打开的软件。

一般在脚本中不使用 externalOpenCode 属性设置源代码，尤其是当源代码超出一行时，把源代码作为一个字符串字面量编码使得脚本代码不易读，容易出错，应该使用 externalOpenFunction 属性以函数体的方式设置。

12.7.1.3 fileFilter

字符串。设置或获取文件参数绑定文件的过滤字符串，用于在文件对话框中过滤文件。默认是空字符串，表示不过滤。在字符串中可设置一个或多个过滤器，例如：

“Images (*.png *.xpm *.jpg);;Text files (*.txt);;XML files (*.xml)”，中间使用“;;”分隔。

12.7.1.4 fileName

字符串。设置或获取文件参数绑定的文件名称。

12.7.1.5 isTemplate

布尔值。设置或获取文件参数绑定文件是否是模板文件。

模板文件一般都是文本文件（没限制一定是文本文件），可从中利用编写特殊的标记提取参数，这些提取的参数作为文件参数的子参数。输入文件中提取输入参数，输出文件中提取输出参数。文件参数所在仿函运行前将从图形界面获得提取的输入参数的新值，依据模板文件新建一个相同的文本文件，同时把新值更新到这个文件中；仿函运行后将从输出的文本文件中提取新值更新到图形界面中对应的输出参数。

12.7.1.6 isText

布尔值。设置或获取绑定的文件是否文本文件。

12.7.1.7 localFileName

字符串。设置或获取文件参数绑定文件的本地文件名称。

该属性主要用于模板文件的情况下，因为模板文件（输入文件）的内容在仿函运行时是不可更改的，所以需要设定一个依据模板文件生成的本地文件的名称，如果与模板文件不在同一个目录下，那么最好设置一样的文件名，该文件将在仿函运行前创建，然后可以作为仿函的输入。

为了便于在图形界面上查看创建的本地文件，当使用此属性设置了一个本地文件名称后，将同时在文件参数下自动创建另一个子文件参数。子文件参数的名称是按“local_文件参数名称”的规则命名的。

若设置本地文件名称的字符串为空，则删除已创建的子文件参数对象。若成功设

置了文件参数的本地文件名称，则自动把当前文件设置为模板文件。

12.7.2 方法

12.7.2.1 rom.FileParameter(name, mode, [parent=null])

创建文件参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

12.7.2.2 rom.FileParameter (name, mode, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

12.7.2.3 localFileObject()

返回文件参数下的本地文件子参数对象，若无则返回 null。

12.8 XlsxFileParameter[1]

Xlsx 文件参数（XlsxFileParameter）表示文件系统中的一个 xlsx 文件。它从 FileParameter 类中派生，因此可访问 FileParameter 的接口。目前它未实现其他更多的接口。

13. 基础多值参数

基础多值参数是一个参数中包含多个数据类型值的参数。在底层实现 C++中，所有的参数，包括基础单值参数，都直接或间接从类 Parameter 中派生，而类 Parameter 不可实例化，它只为派生类提供属性和方法。类 Parameter 是从模型元素类

[1] 版本 1.1 中引入

ModelElement 中派生，因此模型元素中所有的接口在 Parameter 中都可访问。

13.1 ArrayParameter

ArrayParameter 是所有数组参数的基类，本身不可实例化，只为其它派生类参数提供接口。所有数组参数对象内部都使用一维数组管理数据，数据是按行优先的方式存储的，例如一个 2 行 3 列的二维整数数组对象，设置它的数组数据是：

integerArrPar.array = [1, 2, 3, 4, 5, 6];

那么二维数组实际的存储是这样的：[[1, 2, 3], [4, 5, 6]]。

13.1.1 属性

13.1.1.1 dimensions

整数数组。设置或获取数组的维数，例如数组[2, 3]表示一个 2 行 3 列的数组。

13.1.1.2 dimensionNumber

整数。获取数组的维数。

13.1.1.3 fixed

布尔值。设置或获取数组大小是否固定，若为 true 表示固定，否则不固定，可调节数组大小。默认为 false 不固定。

13.1.1.4 size

整数。获取数组元素的总个数。

13.2 IntegerArrayParameter

整数数组参数（IntegerArrayParameter）内部数组元素拥有一个 64 位长度的整数值，实际上在 JavaScript 中是不区分整数与浮点数的，它统一使用 Number 类型表示。在底层 C++层面是使用的整数类型。它从 ArrayParameter 类中派生，因此它可访问

ArrayParameter 的接口。

13.2.1 属性

13.2.1.1 array

整数数组。设置或获取整数数组参数内部包含的整数数组。以一维数组的形式表示，内部按行优先的方式安排位置。

13.2.1.2 lowerLimitEnabled

布尔值。设置或获取整数数组参数的下限是否启用，设置 true 表示启用，默认是 false，表示不启用。

13.2.1.3 upperLimitEnabled

布尔值。设置或获取整数数组参数的上限是否启用，设置 true 表示启用，默认是 false，表示不启用。

13.2.1.4 lessEqual

布尔值。设置或获取整数数组参数上限设置模式的开闭区间，设置 true 表示闭区间，表示参数值可小于等于上限，默认是 true；设置 false 表示开区间，表示参数值可小于上限。

13.2.1.5 greaterEqual

布尔值。设置或获取整数数组参数下限设置模式的开闭区间，设置 true 表示闭区间，表示参数值可大于等于下限，默认是 true；设置 false 表示开区间，表示参数值可大于下限。

13.2.1.6 units

字符串。设置或获取整数数组参数的单位制。若无单位制则返回空字符串。

13.2.1.7 lowerLimit

整数。设置或获取整数数组参数的下限值。它将对数组中每一个元素生效。

13.2.1.8 upperLimit

整数。设置或获取整数数组参数的上限值。它将对数组中每一个元素生效。

13.2.2 方法

13.2.2.1 rom.IntegerArrayParameter(name, mode,dims, [parent=null])

创建整数数组参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，dims 是一个整数数组，表示数组参数各个维数的大小，如[2,3]表示一个 2 行 3 列的数组，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

13.2.2.2 rom.IntegerParameter(name, mode, dims, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

13.2.2.3 value(index)

返回数组参数中索引 index 处的元素，index 是一个整数数组，与数组参数的维度相对应。

13.2.2.4 setValue(index, value)

设置数组参数中索引 index 处的元素值为 value，index 是一个整数数组，与数组参数的维度相对应。

13.3 DoubleArrayParameter

双精度数组参数（DoubleArrayParameter）内部数组元素拥有一个双精度的数值，实际上在 JavaScript 中是不区分整数与浮点数的，它统一使用 Number 类型表示。在底层 C++层面是使用的 double 类型。它从 ArrayParameter 类中派生，因此它可访问 ArrayParameter 的接口。

13.3.1 属性

13.3.1.1 array

双精度数组。设置或获取双精度数组参数内部包含的双精度数组。以一维数组的形式表示，内部按行优先的方式安排位置。

13.3.1.2 lowerLimitEnabled

布尔值。设置或获取双精度数组参数的下限是否启用，设置 true 表示启用，默认是 false，表示不启用。

13.3.1.3 upperLimitEnabled

布尔值。设置或获取双精度数组参数的上限是否启用，设置 true 表示启用，默认是 false，表示不启用。

13.3.1.4 lessEqual

布尔值。设置或获取双精度数组参数上限设置模式的开闭区间，设置 true 表示闭区间，表示参数值可小于等于上限，默认是 true；设置 false 表示开区间，表示参数值可小于上限。

13.3.1.5 greaterEqual

布尔值。设置或获取双精度数组参数下限设置模式的开闭区间，设置 true 表示闭

区间，表示参数值可大于等于下限，默认是 true；设置 false 表示开区间，表示参数值可大于下限。

13.3.1.6 units

字符串。设置或获取双精度数组参数的单位制。若无单位制则返回空字符串。

13.3.1.7 lowerLimit

双精度数值。设置或获取双精度数组参数的下限值。它将对数组中每一个元素生效。

13.3.1.8 upperLimit

双精度数值。设置或获取双精度数组参数的上限值。它将对数组中每一个元素生效。

13.3.2 方法

13.3.2.1 rom.DoubleArrayParameter(name, mode,dims, [parent=null])

创建双精度数组参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，dims 是一个整数数组，表示数组参数各个维数的大小，如[2,3]表示一个 2 行 3 列的数组，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

13.3.2.2 rom.DoubleParameter(name, mode, dims, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

13.3.2.3 value(index)

返回数组参数中索引 index 处的元素，index 是一个整数数组，与数组参数的维度相对应。

13.3.2.4 setValue(index, value)

设置数组参数中索引 index 处的元素值为 value，index 是一个整数数组，与数组参数的维度相对应。

13.4 BooleanArrayParameter

布尔数组参数（BooleanArrayParameter）内部数组元素是一个布尔值。它从 ArrayParameter 类中派生，因此它可访问 ArrayParameter 的接口。

13.4.1 属性

13.4.1.1 array

布尔数组。设置或获取布尔数组参数内部包含的布尔数组。以一维数组的形式表示，内部按行优先的方式安排位置。

13.4.2 方法

13.4.2.1 rom. BooleanArrayParameter (name, mode,dims, [parent=null])

创建布尔数组参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，dims 是一个整数数组，表示数组参数各个维数的大小，如[2,3]表示一个 2 行 3 列的数组，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

13.4.2.2 rom. BooleanArrayParameter (name, mode, dims, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

13.4.2.3 value(index)

返回数组参数中索引 index 处的元素，index 是一个整数数组，与数组参数的维度相对应。

13.4.2.4 setValue(index, value)

设置数组参数中索引 index 处的元素值为 value，index 是一个整数数组，与数组参数的维度相对应。

13.5 StringArrayParameter

字符串数组参数（StringArrayParameter）内部数组元素是一个字符串。它从 ArrayParameter 类中派生，因此它可访问 ArrayParameter 的接口。

13.5.1 属性

13.5.1.1 array

字符串数组。设置或获取字符串数组参数内部包含的字符串数组。以一维数组的形式表示，内部按行优先的方式安排位置。

13.5.2 方法

13.5.2.1 rom. BooleanArrayParameter (name, mode,dims, [parent=null])

创建字符串数组参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，dims 是一个整数数组，

表示数组参数各个维数的大小，如[2,3]表示一个 2 行 3 列的数组，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

13.5.2.2 rom. BooleanArrayParameter (name, mode, dims, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

13.5.2.3 value(index)

返回数组参数中索引 index 处的元素，index 是一个整数数组，与数组参数的维度相对应。

13.5.2.4 setValue(index, value)

设置数组参数中索引 index 处的元素值为 value，index 是一个整数数组，与数组参数的维度相对应。

13.6 TableParameter

表参数（TableParameter）可包含多列数据，是列数据的容器，每一列可设置不同的数据类型。每一列也可设置它的物理意义，用于在可视化这些列数据时可自动确定如数据列是否是 X、Y 或 Z 轴的值。它从 Parameter 类中派生，因此它可访问 Parameter 的接口。

13.6.1 常量

列类型

常数	值	说明
TableParameter.Integer	0	整数列
TableParameter.Float	1	单精度浮点数列
TableParameter.Double	2	双精度浮点数列

TableParameter.FloatComplex	3	单精度复数列
TableParameter.DoubleComplex	4	双精度复数列
TableParameter.String	5	字符串列
TableParameter.DateTime	6	日期时间列

列的物理意义

常数	值	说明
TableParameter.X	0	X 轴
TableParameter.Y	1	Y 轴
TableParameter.Z	2	Z 轴
TableParameter.V	3	值
TableParameter.VX	4	X 轴上的分量值
TableParameter.VY	5	Y 轴上的分量值
TableParameter.VZ	6	Z 轴上的分量值
TableParameter.XError	7	X 轴上的误差值
TableParameter.YError	8	Y 轴上的误差值
TableParameter.Label	9	标签

13.6.2 方法

13.6.2.1 rom.TableParameter (name, mode, [parent=null])

创建表参数对象，它的名称由 name 参数指定，必须是一个有效的 JavaScript 变量标识符，在 Automation 的 JavaScript 解释器中将以该名称访问参数对象，也即此名称是参数对象的变量名称，mode 指定参数的模式，parent 参数指定参数对象的仿函父对象，也可以为 null，表示无父对象。

13.6.2.2 rom.TableParameter (name, mode, dims, parentGroupPar)

与上一个方法类似，但这里的第三个参数是一个组参数对象，表示新建的参数对象的父对象是一个组参数。

13.6.2.3 appendColumn(colType, [size])

向表参数最后一列中追加列类型是 colType，长度为 size 的列，size 默认值是 0。

13.6.2.4 insertColumn(index, colType, [size])

向表参数中的 index 位置处插入一列类型是 colType，长度是 size 的列，size 默认值是 0。若 index 小于或等于 0，将插入表参数的 0 位置，若其大于或等于表当前的列数目，则将追加在表参数的最后。

13.6.2.5 removeColumnAt(index)

从表参数中移除索引是 index 处的列，若 index 不在表参数列的有效索引范围内，则此方法调用后不做任何操作。在移除了列之后，该列的数据将被删除。

13.6.2.6 setColumnNameAt(index, name)

设置列索引是 index 处的列名为 name，若 index 不在表参数列的有效索引范围内，则此方法调用后不做任何操作。

13.6.2.7 setColumnLongNameAt(index, longName)

设置列索引是 index 处的列长名为 longName，若 index 不在表参数列的有效索引范围内，则此方法调用后不做任何操作。

13.6.2.8 setColumnUnitAt(index, unit)

设置列索引是 index 处的列单位制为 unit，若 index 不在表参数列的有效索引范围内，则此方法调用后不做任何操作。

13.6.2.9 setColumnCommentAt(index, comment)

设置列索引是 index 处的列注释为 comment，若 index 不在表参数列的有效索引范围内，则此方法调用后不做任何操作。

13.6.2.10 setColumnExpressionAt(index, expr)

设置列索引是 index 处的列用户函数表达式为 expr，该表达式必须是一个标准 JavaScript 解释器可解析的。若 index 不在表参数列的有效索引范围内，则此方法调用后不做任何操作。

13.6.2.11 setColumnSignificanceAt(index, sig)

设置列索引是 index 处的列物理意义为 sig，若 index 不在表参数列的有效索引范围内，则此方法调用后不做任何操作。

13.6.2.12 columnNameAt(index)

获取列索引是 index 处的列名。若 index 不在表参数列的有效索引范围内，则返回空字符串。

13.6.2.13 columnLongNameAt(index)

获取列索引是 index 处的列长名。若 index 不在表参数列的有效索引范围内，则返回空字符串。

13.6.2.14 columnUnitAt(index)

获取列索引是 index 处的列单位制。若 index 不在表参数列的有效索引范围内，则返回空字符串。

13.6.2.15 columnCommentAt(index)

获取列索引是 index 处的列注释。若 index 不在表参数列的有效索引范围内，则返回空字符串。

13.6.2.16 columnExpressionAt(index)

获取列索引是 index 处的列表达式。若 index 不在表参数列的有效索引范围内，则

返回空字符串。

13.6.2.17 columnSignificanceAt(index)

获取列索引是 index 处的列的物理意义常量。若 index 不在表参数列的有效索引范围内，则返回 TableParameter.NoneSig。

13.6.2.18 size()

返回表参数的列数目。

13.6.2.19 columnType(index)

获取列索引是 index 处的列的类型常量。若 index 不在表参数列的有效索引范围内，则返回 TableParameter.NoneType。

13.6.2.20 fillColumnAt(colType, index, values)

在表参数的列索引 index 处填充类型是 colType 列的元素内容是 values。若成功返回 true，否则返回 false。参数 values 是一个数组，若要填充复数列，那么数组的元素也是一个两个元素的数组对象，分别表示实部与虚部。

13.6.2.21 appendRowAt(index, value)

在表参数列索引 index 处的列中追加一个元素 value，若成功返回 true，否则返回 false。若要追加复数列，则 value 使用一个两元素的数组，分别表示实部与虚部。

13.6.2.22 insertRowAt(col, row, value)

在表参数列索引 col 处的列中的 row 行插入一个元素 value，若成功返回 true，否则返回 false。若要插入复数列，则 value 使用一个两元素的数组，分别表示实部与虚部。

13.6.2.23 integerColumnAt(index)

返回表参数列索引 index 处的整数列，以数组形式返回。

13.6.2.24 floatColumnAt(index)

返回表参数列索引 index 处的浮点数列，以数组形式返回。

13.6.2.25 floatComplexColumnAt(index)

返回表参数列索引 index 处的浮点复数列，以数组形式返回。

13.6.2.26 doubleColumnAt(index)

返回表参数列索引 index 处的双精度列，以数组形式返回。

13.6.2.27 doubleComplexColumnAt(index)

返回表参数列索引 index 处的双精度复数列，以数组形式返回。

13.6.2.28 stringColumnAt(index)

返回表参数列索引 index 处的字符串列，以数组形式返回。

13.6.2.29 dateTimeColumnAt(index)

返回表参数列索引 index 处的日期时间列，以数组形式返回。

13.6.2.30 swap(i, j)

交换表参数中列索引是 i 与 j 的两列，若 i 或 j 无效或交换失败，则返回 false，否则返回 true。

13.7 CompositeParameter

CompositeParameter 是从 GroupParameter 中继承的，目前还无更多的接口。

14. 链接对象

仿函之间参数数据的传递由链接对象表示，每个仿函对象都有一个 links 的属性，该属性返回仿函的所有链接对象，每个链接对象都有如下所述的属性与方法。

14.1 属性

14.1.1 leftHandSide

字符串，获取链接对象的左侧等式字符串，也即链接参数的全名字符串。*版本 1.3 中实现。*

14.1.2 owningSimfun

该属性是一个只读属性，返回链接对象所属的仿函对象。

14.1.3 rightHandSide

字符串，获取或设置链接对象的右侧等式。*版本 1.3 中实现。*

14.1.4 shortLeftHandSide

字符串，获取链接对象的左侧等式短字符串，也即链接参数的名字字符串。*版本 1.3 中实现。*

14.1.5 suspend

布尔值，获取或设置链接对象的链接功能是否暂停，若为 true 暂停链接参数的功能，否则启用链接功能。

14.2 方法

14.2.1 isValid()

判断参数链接的有效性，如果有效返回 true，否则返回 false。

14.2.2 toString()

返回参数链接的字符串表示，以一个等式表示。

15. 文本文件标记语法

应用程序的集成一般需管理其输入文件与输出文件，集成应用程序时为了能够参数化，需要从应用程序的输入、输出文件中提取参数。从输入文件中提取的参数都是输入模式的参数，用户可以在 Automation 软件图形界面上修改这些参数；在应用程序运行完毕产生了输出文件，从输出文件中提取的参数的值相应得到了更新，用户可以在图形界面上查看这些变化的参数值。这里阐述当输入、输出文件是文本文件时如何通过在文件中添加标记的方法提取相关参数。

15.1 模板文件

在 Automation 中添加了标记的文本文件称之为模板文件，对于输入与输出模板文件的标记语法是一致的。添加标记的动作实际上是在待提取参数的文本行上添加一行特别的文本，以此文本信息提取参数。为了使得添加的文本不会影响原来的文本，需使用特别的记号，在 Automation 中采用了设置行注释的方式，如果当前文本文件是某种编程语言的代码，那么利用该编程语言的行注释字符，例如对于 Python 代码，它的行注释是字符#，那么标记提取参数的文本行以三个字符“#?#”开头，同理对于 Ansys 中的 APDL 代码，它的行注释符是!，因此使用“!?!”进行标记。如果文本文件只是普通的数据文件，那么可以设置一个该文件中不会使用到的字符作为行注释符。

集成的应用程序的输入文件将结合模板文件与当前输入参数的值生成；Automation 将利用输出文件的模板文件并解析集成的应用程序的输出文件更新输出参数的值。

15.2 标记语法

在标记提取参数的字符后面添加一个使用 JSON 表示的字符串，所有内容需放置在同一行。下面以一个实例说明 JSON 表示中的标记语法。带背景颜色的是标记字符串，在实际文件中是放置在同一行，这里切分成多行了。未带背景颜色的行是将被提取的参数行，这里要把 Expression=“15”中的数值 15 提取为一个双精度的参数。被标记的文本文件是 Python 代码，因此标记行的开头是#?#三个字符， 接下来的字符串包裹在{}中形成一个有效的 JSON 对象，这个对象中可包含九个键值对，见表 8 JSON 对象键值说明所示。

#?#{"id": "P3", "group": "静域网格参数", "desc": "径向网格节点 1", "type": "DoubleParameter", "value": 15, "regexp": "(Expression=\")([0-9.]+)(\")", "replace": "\\1%1\\3", "action": "(function(value) { var result = 'parameter1 = Parameters.GetParameter(Name=\"P31\")\\n'; result += 'designPoint1.SetParameterExpression(Parameter=parameter1,Expression=\"' + value + '\")'; return result;})"}

designPoint1.SetParameterExpression(Parameter=parameter1,Expression="15")

表 8 JSON 对象键值说明

键	值类型	描述
id	字符串	必需。参数的名称，是一个有效的 JavaScript 变量标识符，确保在同一个名字作用域中的唯一性。
group	字符串	可选。参数的分组，是一个有效的 JavaScript 变量标识符，确保在同一个名字作用域中的唯一性。
desc	字符串	可选。参数的描述信息。
type	字符串	必需。参数的类型，目前支持 IntegerParameter，DoubleParameter，BooleanParameter， StringParameter 以及 FileParameter 参数。

value	与参数类型相对应	可选。如设置了则是提取参数的默认值。对于 IntegerParameter，DoubleParameter 参数是一个数值，对于 BooleanParameter 参数是一个布尔值，对于 StringParameter 参数是一个字符串，对于 FileParameter 参数是一个字符串，表示文件的路径。
properties	子对象	可选。在这个子对象中可设置数值类参数的若干属性：如单位制与上下限等，详情见表 9 properties JSON 对象键值说明所示。
regexp	字符串	必需。提取参数的正则表达式。表达式的编写根据待提取参数处文本的实际情况而定，因此这里是很灵活的。一般地，在这个正则表达式中可区分三个捕获组，第一个是提取参数值的左侧固定字符串，第二个则是参数值本身，第三个是提取参数的右侧固定字符串，如上面实例中所示就是此种情况。对于简单的情况则无需分组，直接利用正则表达式定位参数值即可。
replace	字符串	如果正则表达式使用了分组，那么需提供这个键值对。如对于上例，可以使用**错误!超链接引用无效。**表示，其中\\1 与\\3 表示引用了正则表达式中的第一、三两个捕获的内容，而%1 则表示替换新的参数值。
action	字符串	可选。JavaScript 代码，用在输入文本文件中。在某些场景下，输入文本文件中的多行文本有相关性，当在利用图形界面上的参数值生成新的输入文本文件时，可利用这段 JavaScript 代码完成额外动作。这段代码需包裹在()之内，表示是一个表达式，在这个括号之中是一个匿名函数的定义，形如(function(value){…})，value 是当前提取参数的值，在函数调用时将被自动传入，同时函数内部必须返回一个字符串值，这段字符串将被插入到生成的输入文本文件中，它的位置在提取参数行文本之后。

properties 的子对象进一步设置了数值类参数的属性，如表 9 properties JSON 对象键值说明所示。

表 9 properties JSON 对象键值说明

键	值类型	描述

units	字符串	可选。参数的单位制。
upperEnabled	布尔值	可选。为 true 表示参数有上限，默认是 false。
lessEqual	布尔值	可选。为 true 表示参数可以等于上限，默认是 true。
upper	数值	上限值。若 upperEnabled 设置了，则也需设置。
lowerEnabled	布尔值	可选。为 true 表示参数有下限，默认是 false。
greaterEqual	布尔值	可选。为 true 表示参数可以等于下限，默认是 true。
lower	数值	下限值。若 lowerEnabled 设置了，则也需设置。

为了可在一行文本中提取多个参数，标记文本使用 JSON 中的数组表示，也即以#?#[{…}, {…}, {…}]方式标记，可同时提取三个参数，花括号中的语法与上面的一致。*版本 1.3 中实现。*

16. 使用 JavaScript 创建仿函[2]

基础仿函与基础驱动仿函中描述的仿函是使用 C++语言通过编写 Qt 插件的方式实现的，为了扩展的方便性与灵活性，实际上也可在这些已有仿函的基础上通过编写 JavaScript 脚本创建新的仿函类型，既可以是仿函，也可以是包含子仿函的驱动仿函，它们同样也可按自定义类别纳入仿函库中。这些 JavaScript 脚本代码实际上也是一种插件，Automation 运行时动态加载它们。

下面首先描述如何使用 JavaScript 制作仿函库的插件，然后再分两部分描述如何使用 JavaScript 在仿函库中创建仿函与驱动仿函。

16.1 创建仿函库插件

首先需要为使用 JavaScript 创建的仿函库插件新建一个子目录，该子目录以可执行文件 Automation 所在目录为基准，在其子目录 plugins/automation/script 下创建。以联合仿真仿函库为例，可以在该目录下创建一个 hysim 子目录，子目录命名需满足可

[2]在版本 1.1 中引入

以充当 JavaScript 变量的要求，一般使用不带空格的英文字母即可。

在新建的 hysim 目录中创建一个字符编码是 UTF-8，文件名为“__init__.js”的文本文件，在此文件中编写 JavaScript 代码实现各个自定义的仿函。与自定义仿函相关的文件也可放入 hysim 目录下，如标识仿函的图片文件可在 hysim 目录下新建一个 image 子目录。

创建仿函库的主要逻辑是在脚本文件__init__.js 中编写，该文件在 Automation 运行时作为插件被动态加载，文件内容基本包含三部分，在文件开头首先使用 log()函数输出一些插件加载的信息：

```
log("导入 " + __extension__ + " 插件...");
```

变量__extension__是一个字符串，内容是插件的所在的目录名，这里将是 hysim。

第二部分编写一个函数注册自定义的仿函：

```
function registerSimfuns() { … }
```

在该函数中定义插件中的各个仿函，并实现自定义仿函的功能。该函数的编写放在接下来的两节中描述。

最后一部分是定义一个初始化的匿名函数：

```
__postInit__ = function() {
log('导入联合仿真中的仿函。');
log('注册联合仿真中的仿函。');
    registerSimfuns();
};
```

在匿名函数中输出一些信息，并调用第二部分定义的函数。这里要注意，匿名函数作为一个对象赋值给__postInit__变量。

16.2 创建仿函

在 registerSimfuns()函数中定义一个仿函分三部分：

1. 检查仿函是否已被注册；
2. 定义仿函的构造函数，实现仿函的具体功能；
3. 把定义的仿函注册到仿函库中。

使用 JavaScript 创建仿函一般使用 AutomationScript 仿函即可，这里以给 hysim 插件创建一个名为 Fluent 的仿函为例说明各个步骤。

首先在 registerSimfuns()函数中编写如下语句检查 Fluent 仿函是否已被注册：

```
if ('Fluent' in rom.library.simfuns)
    throw new Error('Fluent has been in existence.');
```

所有已被注册的仿函都在 rom.library.simfuns 对象的属性中，如果“Fluent”已被注册则抛出异常。

第二步定义自定义仿函的构造函数，按 Automation 中的规定，所有的仿函构造函数对象都是 rom 对象的一个属性。如下代码所示：

```
rom.Fluent = function(name, sfParent) {
    var sf = new rom.AutomationScript(name, sfParent);
    sf.derivedName = 'Fluent';
    var data1 = new rom.DoubleParameter('data1', rom.Input, sf);
    var data2 = new rom.DoubleParameter('data2', rom.Output, sf);
    data1.canDelete = false; // 指定参数不可删除
    data2.canDelete = false;
    // 提供预置功能
    sf.scriptFunction = function() {
        log('测试流体计算');
        self.data2.value = self.data1 + Math.random();
    };
        return sf;
    };
```

在检查仿函库中没有 Fluent 之后，即可为 rom.Fluent 定义一个匿名函数，用作仿函的构造函数，构造函数有两个约定的参数，第一个参数是仿函对象的名称，第二

个参数是仿函对象的父仿函对象。在匿名函数体中，首先创建一个 AutomationScript 对象 sf；然后必须设置该仿函对象的派生名称，以示与使用 C++编写的仿函的区别，该名称与自定义仿函的名称必须一致；接下来实现该仿函特定的功能；最后返回函数中创建的仿函对象。

最后一步把自定义的仿函注册到仿函库中，如下所示：

```
rom.library.simfuns['Fluent'] = {
    name : '流体计算',
    extension : __extension__, // 插件名称，也即目录名称
    composite : false,
    tools : [ // 定制仿函的工具菜单
        {menuItem : '流体计算菜单', icon : 'plotter16.png', actor :
function(sf) {
            log('流体计算菜单测试: ', sf.fullName);}},
        {menuItem : 'seperator'},
        {menuItem : '流体计算菜单 2', defaultActor : true, actor :
function(sf) {
            log('流体计算菜单 2 测试: ', sf.fullName);}}
    ],
    graphicsItem : 'Item',
    version : '1.0',
    icon : 'fluent.svg', // 设定仿函的图片，image 目录下须有 16,24 结尾
的图片文件
    group : 'HySim', // 仿函的分组
    author : '安世亚太科技股份有限公司',
    description : '用于流体计算。',
    os : ['windows', 'linux']
    };
```

所有的仿函都注册在 rom.library.simfuns 对象中，成为该对象的一个属性，属性名称也即仿函的名称，该属性也是一个对象，对象中包含多个预定义的键值对，见表 10 仿函注册的属性说明所示：

表 10 仿函注册的属性说明

属性名称	描述
name	字符串，用于仿函在图形界面中显示的名称
extension	插件名称，也即目录名称，其值为__extension__
composite	布尔值，确定是否驱动仿函，若是驱动仿函其值为 true，否则为 false
tools	数组，定义仿函的工具菜单。与 7.1.1 中的 actionListCode 中的定义只有两点不一致，第一个区别是对菜单项中图标 icon 路径定位的区别，这里默认是插件所在的 image 目录；第二个区别是 actor 中响应菜单项动作的匿名函数，该函数有一个参数 sf，表示仿函对象
graphicsItem	字符串，标识仿函的图形项，设置为'Item'即可
version	字符串，设置仿函的版本号
icon	字符串，设置标识仿函的图标，可以是矢量图 svg 格式，尺寸是 24*24，也可以设置两个 png 格式的图片，图片有 16*16 与 24*24 两种大小，文件名分别是 filename16.png 与 filename24.png，但在这里指定 png 图片是可以忽略尺寸，如 icon: 'filename.png'，所有的图片都放入插件所在的 image 目录中
group	字符串，仿函的分组，将在图形界面中以此分组显示各类仿函
author	字符串，仿函的作者
description	字符串，仿函功能的描述信息
os	数组，表示仿函可在哪些操作系统上运行，目前还无实际用处

16.3 创建驱动仿函

在 registerSimfuns()函数中定义一个驱动仿函也分三部分：

4. 检查驱动仿函是否已被注册；
5. 定义驱动仿函的构造函数，实现驱动仿函的具体功能；
6. 把定义的驱动仿函注册到仿函库中。

使用 JavaScript 创建驱动仿函一般使用 Assembly 驱动仿函即可，这里以给 hysim 插件创建一个名为 Dummy 的驱动仿函为例说明各个步骤。

首先在 registerSimfuns()函数中编写如下语句检查 Dummy 驱动仿函是否已被注册：

```
if ('Dummy' in rom.library.simfuns)
    throw new Error('Dummy has been in existence.');
```

所有已被注册的驱动仿函都在 rom.library.simfuns 对象的属性中，如果“Dummy”已被注册则抛出异常。

第二步定义自定义驱动仿函的构造函数，按 Automation 中的规定，所有的驱动仿函构造函数对象都是 rom 对象的一个属性。如下代码所示：

```
rom.Dummy = function(name, sfParent) {
    var assembly = new rom.Assembly(name, sfParent);
    assembly.derivedName = 'Dummy';
    var sf1 = new rom.Fluent('sf1', assembly);
    sf1.scriptFunction = function() {
        self.data2.value = self.data1 + 100 * Math.random();
    };
    var sf2 = new rom.AutomationScript('sf2', assembly);
    var data1 = new rom.DoubleParameter('data1', rom.Input, sf2);
    var data2 = new rom.DoubleParameter('data2', rom.Output, sf2);
    sf2.scriptFunction = function() {
        self.data2.value = self.data1 + 100 * Math.random();
    };
    sf2.createLinkByParameters(data1, sf1.parameters.data2);
    return assembly;
    };
```

在检查仿函库中没有 Dummy 之后，即可为 rom. Dummy 定义一个匿名函数，用作驱动仿函的构造函数，构造函数有两个约定的参数，第一个参数是仿函对象的名称，第二个参数是仿函对象的父仿函对象。在匿名函数体中，首先创建一个 Assembly 对象 assembly；然后必须设置该仿函对象的派生名称，以示与使用 C++编写的仿函的区别，

该名称与自定义仿函的名称必须一致;接下来创建驱动仿函对象 assembly 的各个子仿函，参数，以及参数之间的链接，实现该仿函特定的功能；最后返回函数中创建的仿函对象 assembly。

最后一步把自定义的驱动仿函注册到仿函库中，如下所示：

```
rom.library.simfuns['Dummy'] = {
    name : 'Dummy 驱动仿函',
    extension : __extension__,
    composite : true,
    graphicsItem : 'SequenceItem',
    version : '1.0',
    tools : [
        {menuItem : '驱动仿函菜单', icon : 'plotter16.png', actor :
function(sf) {
            log('驱动仿函菜单测试: ', sf.fullName);
        }},
        {menuItem : 'seperator'},
        {menuItem : '驱动仿函菜单 2', defaultActor : true, actor :
function(sf) {
            log('驱动仿函菜单 2 测试: ', sf.fullName);}}
    ],
    icon : 'plotter.png',
    group : 'HySim',
    author : '安世亚太科技股份有限公司',
    description : '用于流体计算。',
    os : ['windows', 'linux']
    };
```

所有的仿函都注册在 rom.library.simfuns 对象中，成为该对象的一个属性，属性名称也即仿函的名称，该属性也是一个对象，对象中包含多个预定义的键值对，见表 11 驱动仿函注册的属性说明所示：

表 11 驱动仿函注册的属性说明

属性名称	描述

name	字符串，用于仿函在图形界面中显示的名称
extension	插件名称，也即目录名称，其值为__extension__
composite	布尔值，确定是否驱动仿函，若是驱动仿函其值为 true，否则为 false
tools	数组，定义仿函的工具菜单。与 7.1.1 中的 actionListCode 中的定义只有两点区别，第一个区别是对菜单项中图标 icon 路径定位的区别，这里默认是插件所在的 image 目录；第二个区别是 actor 中响应菜单项动作的匿名函数，该函数有一个参数 sf，表示仿函对象
graphicsItem	字符串，标识仿函的图形项，设置为'SequenceItem'即可
version	字符串，设置仿函的版本号
icon	字符串，设置标识仿函的图标，可以是矢量图 svg 格式，尺寸是 24*24，也可以设置两个 png 格式的图片，图片有 16*16 与 24*24 两种大小，文件名分别是 filename16.png 与 filename24.png，但在这里指定 png 图片是可以忽略尺寸，如 icon: 'filename.png'，所有的图片都放入插件所在的 image 目录中
group	字符串，仿函的分组，将在图形界面中以此分组显示各类仿函
author	字符串，仿函的作者
description	字符串，仿函功能的描述信息
os	数组，表示仿函可在哪些操作系统上运行，目前还无实际用处