UiBot开发者指南（中级）

生成时间：2021-12-30 10:55:03

目录

1 预备知识

通过初级版教程的学习，相信您已经掌握了UiBot最基础的概念和最基本的操作，已经能够编写最简易的流程。本章将开始讲述UiBot中级版教程，您将学习到UB编程语言、数据处理、网络和系统、人工智能、命令扩展等强大而实用的功能。

在初级版的教程中，我们几乎无需接触到UiBot编程语言，也只接触到了几种简单的数据类型。而在中级版的教程中，我们将要接触到一些复合数据类型，例如数组和字典。本章先对这两种数据类型的概念做一个简单介绍，因为后续内容需要经常用到这两种复合数据类型。

1.1 数组

还是使用初级版教程中的例子，如下图所示。这是一张Excel表格，表格中的每一行是一条订单记录，每一列是订单的不同字段，包括订单号、顾客姓名、订单数量和销售额等。

	A	B	C	D
1	订单号	顾客姓名	订单数量	销售额
2	3	李鹏晨	6	261.54
3	6	王勇民	2	6
4	32	姚文文	26	2808.08
5	35	高亮平	30	288.56
6	36	张国华	46	2484.7455
7	65	李丹	32	3812.73
8	66	谢浩谦	41	108.15
9	69	何春梅	42	1186.06

图 1: 虚构的Excel表格

前文已经讲过，可以分别使用不同类型的变量来保存这张Excel表格中的数据，例如：可以用字符串类型的变量来保存顾客姓名、可以用整数类型的变量来保存订单数量等。

那么如何同时保存多个数据呢？比如需要保存100条订单记录的订单号：一种方法是定义多个变量，例如使用No1、No2、No3、.....、No100等100个变量来保存100个订单号，每个订单号使用一个变量来保存，这种方法比较简单直接，但是在数据量大时会非常繁琐；另一种比较聪明的方法，是利用一种叫做**数组**的复合类型。所谓“数组”，指的是可以用来储存多个数据的一组元素，这些元素可以用一个变量来表示。具体使用方法为：使用逗号来分隔每个元素，使用方括号包围起来，这样的一个整体，即构成一个“数组”，可以被放置在一个变量里面（而不需要多个变量）。如下所示：

```
数组变量 = [No1, No2, No3, No4]
```

同一个数组中的多个元素的值可以是任意类型，例如：元素的值是整数，就构成一个整数数组。同一个数组中的多个元素数据类型可以相同，也可以不同，例如：第一个元素是整数，第二个元素是字符串等。甚至，一个数组中的元素也可以是另外一个数组，这样就构成了一般意义上的多维数组。

通常我们只会用到二维数组，三维或者更多维的数组很少用到。下面是一个典型的二维数组，数组中包含了两个元素，其中每个元素又是一个数组，其中包含了六个字符串：

```
二维数组变量 = [[ "刘备", "关羽", "张飞", "赵云", "马超", "黄忠" ],
[ "20K", "18K", "15K", "12K", "10K", "10K" ]]
```

那么如何定位和访问数组中的多个元素呢？这就需要用到**下标**了，所谓下标，指的是用于区分数组的各个元素的数字编号，通俗地说，数组下标就是指数组的第几个元素。不过数组的下标是从0开始编号的，例如数组变量的第1个元素如下：

```
数组变量[0]
```

在这个例子中，数组变量[0]指代的就是No1这个变量的值。如果要引用二维或多维数组的值，则采用多个下标：

```
二维数组变量[0][1]
```

其结果是上面二维数组中的“关羽”这个值。

1.2 字典

除了数组之外，还有一种叫做**字典**的数据类型，也可以实现一个变量保存多个数据。不过，数组的典型应用场景，主要是用来保存多个同样性质、同样类别的数据，例如100个订单号等；而字典的应用场景则更宽泛，主要是用来保存多个有关联、但是数据类型不尽相同的数据，例如一条订单的四个字段等。为了更好地访问这些不同数据类型的字段，字典不仅保存数据的值，还保存数据的名字。

字典类型变量的表示方法为：把多个元素用逗号分隔，然后再使用大括号来包围起来。其中每个元素**必须**包含一个 **名字** 和一个 **值**，名字和值之间用冒号分隔。如下所示：

```
{ 名字1:值1, 名字2:值2, 名字3:值3 }
```

其中 **名字** 只能是字符串，**值** 可以是任意类型的表达式。如果您熟悉JavaScript或者JSON，会发现这种初始化方法和JSON的表示形式高度相似。

上述订单记录就可以这样表示：

```
字典变量 ={ "订单号":"3", "顾客姓名":"李鹏晨", "订单数量":6, "销售额":261.54 }
```

同样地，字典也可以利用下标作为索引来访问其中的元素，只不过，字典索引为**名字**，这是一个字符串。例如，得到上述字典变量的订单号的方法为：

```
字典变量["订单号"]
```

其结果是字典里面，名字为“订单号”的元素的值，也就是字符串"3"。

2 数据获取和处理

数据是信息化发展中的必然产物。对数据进行收集、整理、加工、分析等操作，是RPA流程经常遇到的任务。本章以数据的常见操作为主线，分别介绍数据的获取和处理等方法，涵盖网页数据、应用数据、文件数据等不同数据源的获取，以及JSON、字符串、正则表达式、集合、数组等多种数据的处理方法。

2.1 数据获取方法

2.1.1 数据抓取

在RPA的流程中，经常需要从某个网页、或某个表格中获得一组数据。比如我们在浏览器中打开某个电商网站，并搜索某个商品后，希望把搜到的**每一种**商品的名称和价格都保存下来。这里的商品名称和价格等都是界面元素，可以用UiBot的“界面元素自动化”，逐一去网页中选择界面元素（商品名称、价格等），再用“获取文本”等命令得到每一项的内容。但显然非常繁琐，而且在搜到的商品种类的数量不事先固定的时候，也会比较难以处理。实际上，UiBot提供了“数据抓取”的功能，可以用一条命令，一次性地把多组相关联的数据都读出来，放在一个数组中。我们来看看这个功能如何使用：

进入UiBot的流程块编辑，点击工具栏的“数据抓取”按钮，UiBot将会弹出一个交互引导式的对话框，这个对话框将会引导用户完成网页数据抓取。根据对话框的第一步提示，UiBot目前支持四种程序的数据抓取：桌面程序的表格、Java表格、SAP表格、网页。本文以网页数据抓取为例阐述，其它三种程序的数据抓取在操作上并无显著区别。

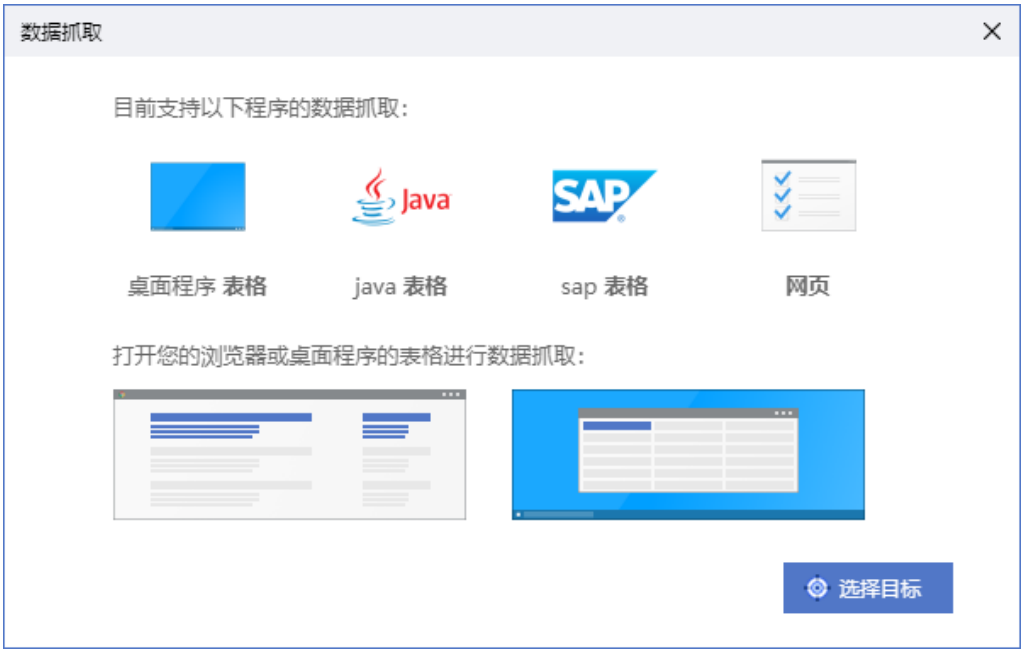

图 2: 开始抓取数据-选择目标

点击“选择目标”按钮，这一按钮与前面我们学习的“界面元素自动化”中的“选择目标”按钮用法一致。需要注意的是：UiBot并不会帮您自动打开想要抓取的网页和页面，因此在数据抓取之前，需要预先打开数

据网页或桌面程序表格。这个工作可以手动完成，也可以通过UiBot其它命令组合完成。例如，这里演示的是抓取某电商网站上的手机商品信息，我们可以使用“浏览器自动化”的“启动新的浏览器”命令打开浏览器并打开该网站，使用“设置元素文本”命令在搜索栏输入“手机”，使用“点击目标”命令点击“搜索”按钮。上述步骤在初级开发者指南中都有阐述，不再展开讲解。

网页准备好后，下一步任务是在网页中定位需要抓取的数据，先抓取商品的名称，仔细选择商品名称的目标（红框蓝底遮罩框）。

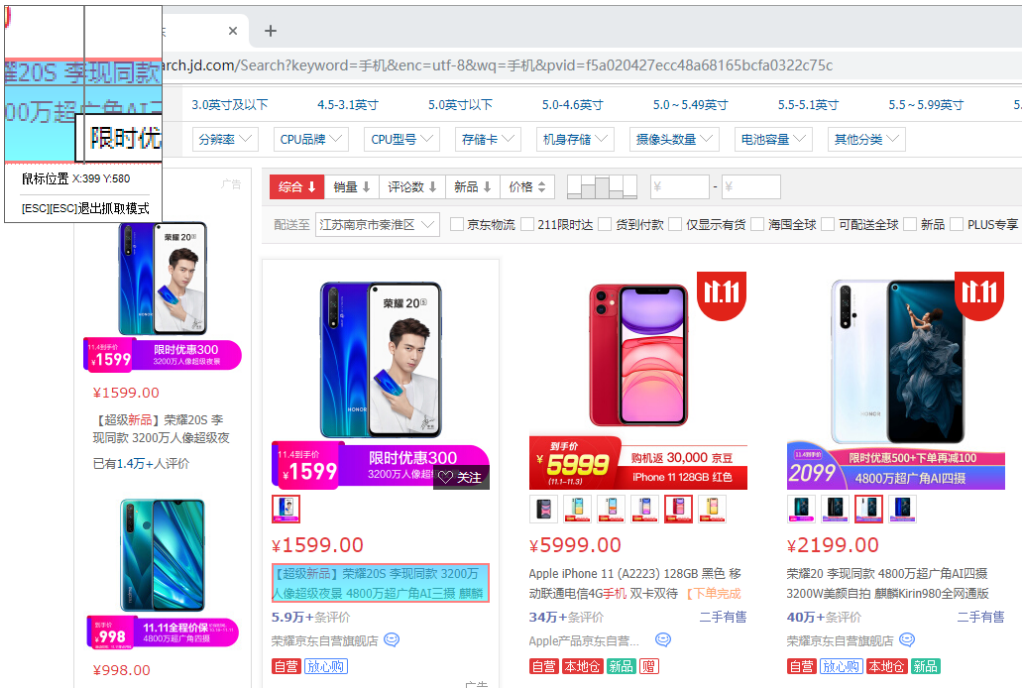

图 3: 选择商品名

此时，UiBot弹出提示框：“请选择层级一样的数据再抓取一次”。您可能会感到疑惑：什么叫层级一样的数据？为什么还要再抓取一次呢？这是因为，我们要抓取的是一组数据，必须找到这些一组数据的共同特征。第一次选取目标后，得到了一个特征，但是仍然不知道哪些特征是所有目标的共性、哪些特征只是第一个目标的特性。只有再选择一个层级一样的数据并抓取一次，UiBot才能保留所有目标的共性，而刨去每个目标各自的特性。就好比在数学中，两个点才能确定一条直线，我们只有选取两个数据，才能确定要抓取哪一列数据。

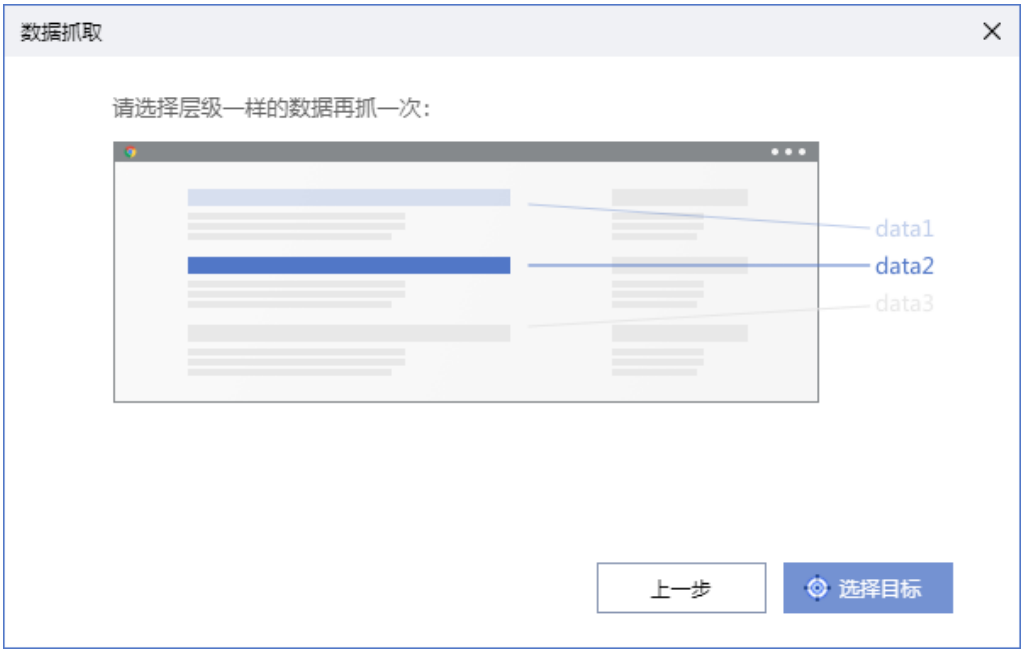

图 4: 提示：选择层级一样的数据再抓取一次

定位需要抓取的数据，这里我们先抓取商品的名称，仔细选择商品名称的目标（红框蓝底遮罩框）。

再次在网页中定位需要抓取的数据，也就是商品的名称，第一次抓取的是第一个商品的名称，这次我们抓取第二个商品的名称。这里一定要仔细选择商品名称的目标，保证第二次和第一次抓取的是同一个层级的目标，因为Web页面的层级有时候特别多，同样一个文本标签嵌套数层目标。当然，强大而贴心的UiBot也会帮您做检查，这里只是先给您提个醒，可不要在UiBot报错的时候惊讶噢！通常都是因为目标选错了导致报错。另外，也可以选择第三个、第四个商品的名称进行抓取，这些都不影响数据的抓取结果，只要是同一层级就可以了。

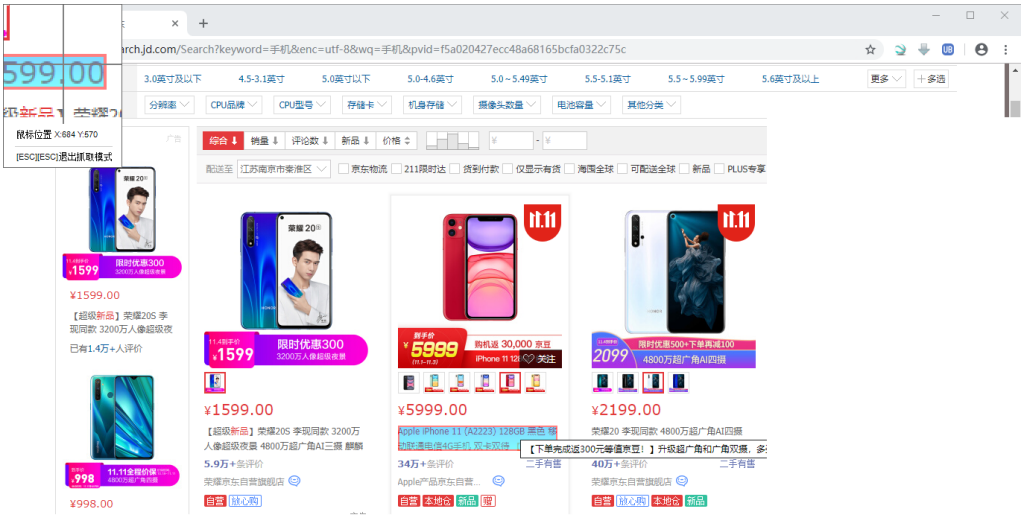

图 5: 再次选择商品名

两次目标都选定完成后，UiBot再次给出引导框，询问“只是抓取文字还是文字链接一起抓取”，按需选择即可。

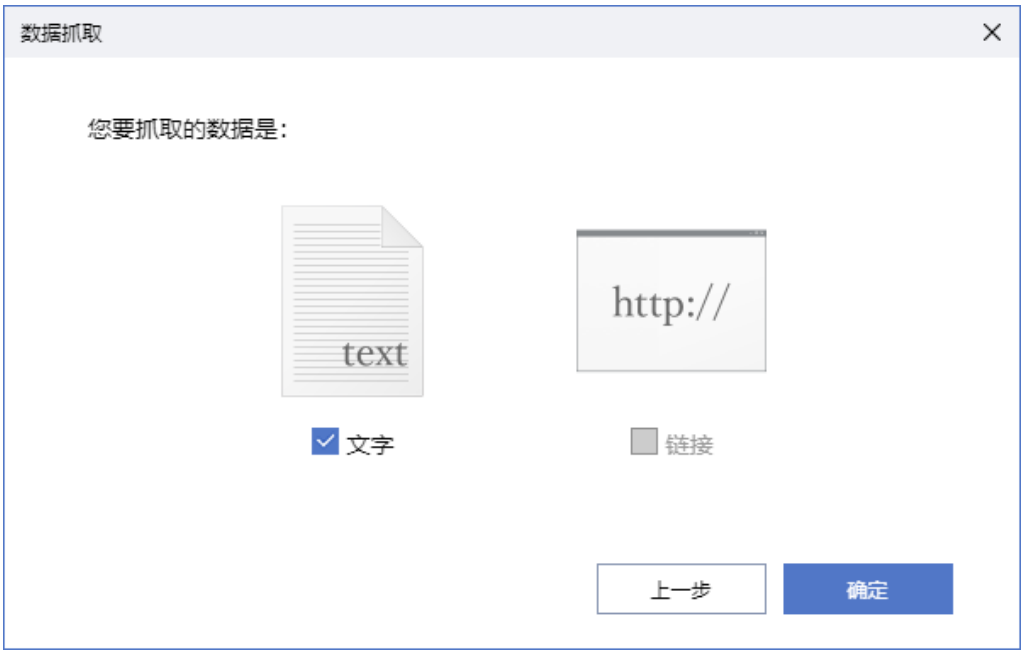

图 6: 提示：您要抓取的数据是

点击“确定”按钮后，UiBot会给出数据抓取结果的预览界面，您可以查看数据抓取结果与您的期望是否一致：如果不一致，可以点击“上一步”按钮重新开始数据抓取；如果一致，且您只想抓取“商品名称”这一组数据，那么点击“下一步”按钮即可；如果您想抓取更多组数据，例如，还想抓取商品的价格，那么可以点击“抓取更多数据”按钮。UiBot会再次弹出选择目标界面。

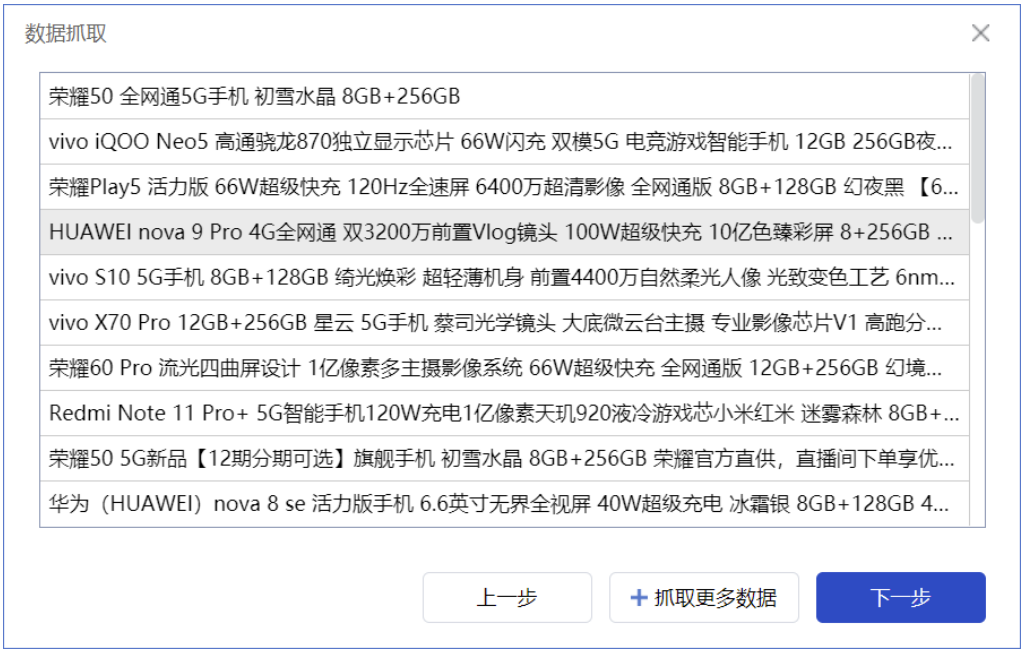

图 7: 预览抓取结果-抓取更多数据

例如，这次我们选择的是商品价格文本标签。

图 8: 选择商品价格

同样经过两次选择目标，再次预览数据抓取结果，可以看到：商品名称和商品价格已成功抓取。而且，这两组数据是一一对应的，第一列的商品名称对应了第二列的价格。

图 9: 再次预览抓取结果

循环使用这个方法，还可以进一步增加多组需要抓取的数据，比如商品的卖家名称、评价数量等。如果

不再需要抓取更多数据了，那么点击“下一步”按钮。此时出现的引导页面询问“是否抓取翻页”，这是什么意思呢？假设把网页数据看成一个二维数据表的话，前面的步骤是增加数据表的列数，例如商品名称、价格等，而抓取翻页，是增加数据表的行数。如果只抓取第一页数据，那么点击“完成”按钮即可；如果需要抓取后面几页的数据，那么点击“抓取翻页”按钮。

图 10: 抓取翻页

点击“抓取翻页”按钮，弹出“目标选择”引导框，选择Web页面中的翻页按钮，这里的翻页按钮为页面中的">"符号按钮。

图 11: 选择翻页按钮目标

当所有步骤完成后，可以看到UiBot插入了一条“数据抓取”命令到命令组装区，且该命令的各个属性都

已通过引导框填写完毕。大部分属性通常都不需要再修改了，个别属性还可以再进一步调整：“抓取页数”属性指的是抓取几页数据；“返回结果数”属性限定每一页最多返回多少结果数，-1表示不限定数量；“翻页间隔”属性指的是每隔多少毫秒翻一次页，有时候网速较慢，需要间隔时间长一些网页才能完全打开。

2.1.2 通用文件处理

除了网页数据抓取，“文件”是另一种非常重要的数据源。UiBot提供了几种格式的文件的读取操作，包括通用文件、INI格式文件、CSV格式文件等，我们先来看一下通用文件。

通用文件处理通常用来读写没有特定格式的文本文件，比如用Windows自带的“记事本”编写的文件，就属于这种类型。除此之外，通用文件处理还包含了判断文件是否存在、判断文件夹里面有哪些文件等功能。这里只介绍读取文本文件的功能。

在命令列表中，找到“文件处理”，并展开“通用文件”一项，找到“读取文件”命令，双击或拖动插入UiBot。该命令的第一个属性是“文件路径”，填写待读取文件的路径即可。可以是绝对路径，也可以是相对路径。如果采用绝对路径，直接点击后面的文件夹形状的图标进行选取即可；如果采用相对路径，建议切换到专业模式下，输入相对路径`@res"test.txt"`，即流程所在文件夹下的res文件夹下的test.txt文件，前面已经学习过这样的相对路径表示方式了。

特别需要注意的是第二项属性，即“字符集编码”。即使是相同内容的文本文件，也会有不同的编码格式，常见的包括ANSI/GBK、UTF-8、Unicode等等。在UiBot中，我们一般都采用UTF-8的编码。如果读取的文本文件是其他编码，只要您在这里正确的选择了编码，UiBot就会自动将其转换为UTF-8，以便后续处理。如果您不了解这些编码的区别，以及您要读取的文件采用了哪种编码，互联网上有大量资料可供参考，本文不再赘述。

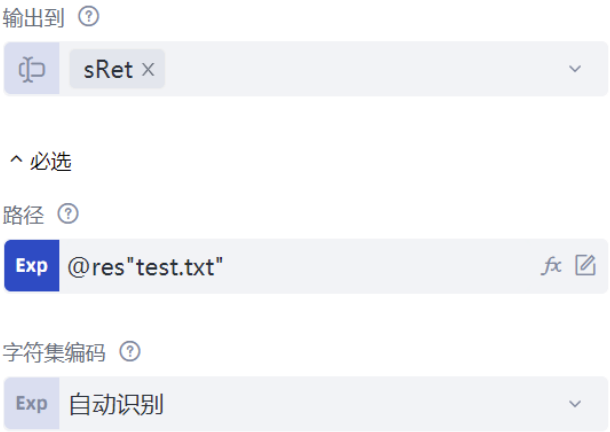

图 12: 读取文件

“读取文件”命令会把文件全部读出来，放在一个字符串类型的变量中。如果需要对有格式的文本文件进行更加细节的操作，可以根据文件类型，选择特定的文件操作命令，例如INI文件、CSV文件等。

2.1.3 INI文件处理

INI文件又叫初始化配置文件，Windows系统程序大多采用这种文件格式，负责管理程序的各项配置信息。INI文件格式比较固定，一般由多个小节组成，每个小节由一些配置项组成，这些配置项就是键值对。

我们来看最经典的INI文件操作：“读取键值”。在命令中心“文件处理”的“INI格式”命令分类下，选择并插入一条“读键值”命令，这条命令可以读取指定INI文件中指定小节下指定键的值。该命令共有五个属性：“配置文件”属性，填写待读取INI文件的路径，这里填写的是相对路径`@res"test.ini"`，说明读取的是流程所在文件夹下的名为`res`的文件夹下的`test.ini`文件，内容如下：

```
[meta]
Name = mlib
Description = Math library
Version = 1.0

[system]
Libs=sysLibs
Cflags=sysCflags

[user]
Libs=userLibs
Cflags=userCflags
```

“小节名”属性填写键值对的查找范围，这里填写的是“user”，说明在`[user]`小节查找键值对；“键名”属性填写待查找的“键”的名称，这里填写的是“Libs”，说明要查找形如“Libs=”后的内容；“默认值”属性指的是，当查找不到键时，返回的默认值；“输出到”属性填写一个字符串变量sRet，sRet将保存查找到的键值。注意在下图中，由于我们在“小节名”和“键名”属性栏里面采用了“普通模式”进行输入，所以直接输入字符串内容即可，不需要加双引号。而如果切换到“专业模式”，就需要加双引号了。

图 13: 读取INI文件

添加一条“输出调试信息”命令，打印出sRet，运行流程后，可以看到sRet的值为“userLibs”。

2.1.4 CSV文件处理

CSV文件以纯文本形式存储表格数据，文件的每一行都是一条数据记录。每条数据记录由一个或多个字段组成，用逗号进行分隔。CSV文件广泛用于不同体系结构的应用程序之间交换数据表格信息，解决不兼容数据格式的互通问题。

在UiBot中，可以使用“打开CSV文件”命令将CSV文件的内容读取到数据表中，然后再基于数据表进行数据处理，数据表的处理方法参见下一节。

先来看“打开CSV文件”命令，这条命令有两个属性：“文件路径”属性填写待读取CSV文件的路径，这里填写的是相对路径`@res"test.csv"`，说明读取的是流程所在文件夹下的`res`文件夹下的`test.csv`文件；“输出到”属性填写一个数据表对象objDataTable，运行命令后，test.csv文件的内容将被读取到数据表objDataTable中，我们可以添加一条“输出调试信息”命令，查看objDataTable数据表的内容。

图 14: 打开CSV文件

再来看“保存CSV文件”命令，这条命令也有两个属性：“数据表对象”属性填写上一步得到的数据表objDataTable；“文件路径”属性填写保存CSV文件的路径，这里填写的是`@res"test2.csv"`，仍然是相对路径，其含义已多次解释，不再赘述。

2.1.5 PDF文件处理

在办公场景中，PDF格式文件是Office格式文件之外最常用的文件格式，因此对PDF文件的处理也显得非常重要。从UiBot Creator 5.0版本开始，UiBot提供了对PDF文件处理的支持。所支持的命令如下所示：

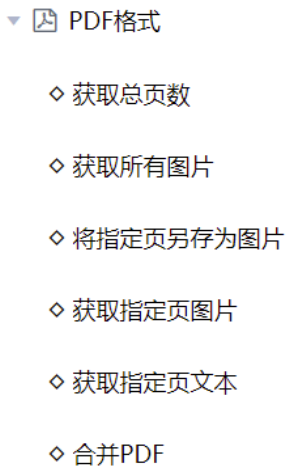

图 15: PDF命令列表

在命令列表中，“文件处理”的“PDF格式”分类下，选择并插入一条“获取总页数”命令，这条命令可以得到指定PDF文件的页数。该命令共有三个属性：“文件路径”属性，填写待读取PDF文件的路径，这里填写的是`@res"PDF.pdf"`，说明读取的是流程文件夹下的`res`文件夹下的`PDF.pdf`文件；“密码”属性，填写的是`PDF.pdf`文件的打开密码，如果无密码，那么保持默认值即可。运行该命令后，“输出到”属性中填写的变量名，将会保存PDF文件的页数。

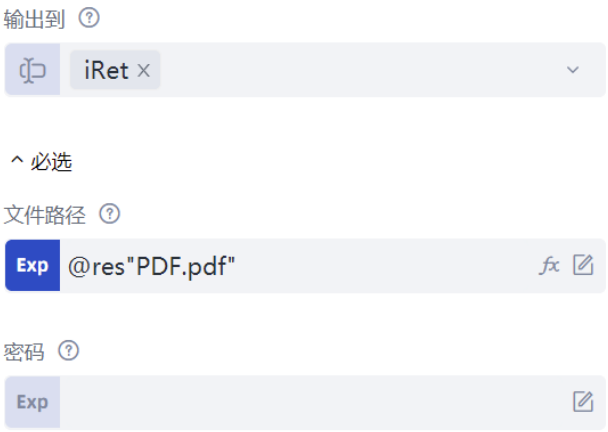

图 16: 获取PDF文件总页数

UiBot还可以将PDF的单页转换成图片文件，选择并插入一条“将指定页另存为图片”命令，该命令共有五个属性：“文件路径”属性和“密码”属性的含义同“获取总页数”命令；“开始页码”和“结束页码”属性指定PDF文件的开始和结束页码，这里填写1和2，表示转换第1页到第2页；“保存目录”属性填写转换后图片的保存路径，这里填写的是`@res""`，说明转换后图片保存到流程所在文件夹下的`res`文件夹下。运行后，会自动生成两个文件：PDF_1.png和PDF_2.png，它们都是图片文件，分别是第1页和第2页的内容。

图 17: 将指定页另存为图片

除了处理单个PDF文件，UiBot还能将多个PDF文件合并成一个PDF文件，选择并插入一条“合并PDF”命令，该命令共有两个属性：“文件路径”属性填写需要合并的多个PDF文件路径。既然是合并文件，自然需要有不止一个文件才具有合并的意义。因此，必须输入多个PDF文件的路径，也就是需要填写一个数组，这里填写的是`[@res"PDF.pdf",@res"PDF1.pdf"]`，表示合并流程所在文件夹下的`res`文件夹下的`PDF.pdf`文件和`PDF1.pdf`文件；“保存路径”属性填写合并后的PDF文件路径，这里填写的仍然是相对路径`@res"PDF2.pdf"`。运行后，两个PDF文件即可合二为一，且在合并后的文件中，原先的两个PDF文件的排列顺序和它们在数组中的顺序是一致的。

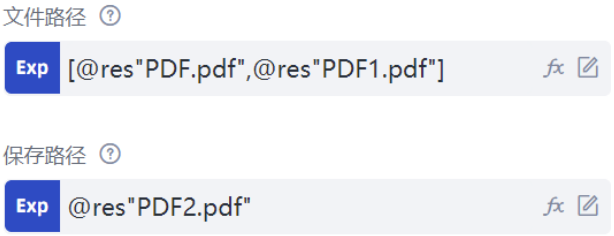

图 18: 合并PDF

2.2 数据处理方法

当数据完成读取后，接下来就要对数据进行处理。根据数据格式的不同，UiBot提供了不同的数据处理方法和命令，包括数据表、字符串、集合、数组、时间或者正则表达式等。下面分别介绍这些方法。

2.2.1 数组

我们在前文中已经学习了采用数组来存储多个数据的方式。当构造好一个数组之后，UiBot还提供了一系列命令，对其进行各种处理。包括数组编辑（添加元素、删除元素、截取合并数组）、数组信息获取（长度、下标等）等等。例如，可以在UiBot的命令列表中，找到“数据处理”的“数组”分类，选择并插入一条“在数组尾部添加元素”命令，该命令可在数组的末尾添加一个元素。该命令有三个属性：“目标数组”属性，填写添加元素前的数组，这里填写`["1", "2"]`；“添加元素”属性填写待添加的元素，这里填写`"3"`；“输出到”属性保存添加后的数组变量。添加完成后，用“输出调试信息”显示该变量的内容，预期输出结果为`["1", "2", "3"]`。

图 19: 在数组尾部添加元素

再来看“过滤数组数据”命令，这条命令可以快速对数组中的元素进行筛选，留下或者剔除满足条件的元素。该命令有四个属性：“目标数组”属性，填写待过滤的数组，这里填写的是`["12", "23", "34"]`；“过滤内容”属性填写按照什么条件过滤数组，这里填写`"2"`，表示数组元素只要是字符串，并且包含了`"2"`就满足条件；“保留过滤文字”属性有两个选项，“是”表示满足条件的数组元素将会保留，剔除不满足条件的元素；“否”表示满足条件的数组元素将会剔除，保留不满足条件的元素。

可以尝试一下，对“保留过滤文字”的属性选择“是”，并输出过滤后的数组变量arrRet，输出结果为`[ "12", "23" ]`，可见包含了字符串`"2"`的数组元素都被保留；而如果“保留过滤文字”属性选择“否”，过滤后的数组变量arrRet的输出结果为`[ "34" ]`，包含字符串`"2"`的数组元素都被剔除了。

图 20: 过滤数组数据

2.2.2 集合

集合可视作一种特殊的一维数组，它和数组的不同之处主要有两点：

1. 数组中的元素可以重复，而集合中的元素不允许重复。例如，`[1，2，2，3]`是一个普通的数组，但如果将其转换为一个集合的话，就会剔除掉一个2，只保留1，2，3这三个元素。
2. 数组中的元素是有序的，而集合中的元素是无序的。例如，往一个数组中依次添加元素1，2，3，往另一个数组中添加3，2，1，得到的会是两个不同的数组。而如果往两个集合中分别依次添加元素1，2，3和3，2，1，这两个集合仍然是等价的。

我们首先尝试创建一个集合。在命令列表中，找到“数据处理”下面的“集合”分类，选择并插入一条“创建集合”命令。该命令只有一个“输出到”属性，它会创建一个空集合，并将此集合置入ObjSet变量中。如果您熟悉UiBot的源代码视图，这里还有一个技巧：可以切换到源代码视图，把一个数组当作`Set.Create`命令的输入，可以直接把这个数组转换为集合。如果您还不熟悉源代码视图也没关系，UiBot的后续版本会允许在可视化视图中把数组转换为集合。

当创建一个集合后，还可以继续往这个集合中插入元素。使用“添加元素到集合”命令，该命令有两个属性：“集合”属性填写上一步创建的集合ObjSet；“添加元素”属性填写集合元素，可以是数字、字符串等，也可以是变量。

同一个集合中，能否既有数字元素，又有字符串元素呢？答案是肯定的！我们可以调用两次“添加元素到集合”命令，一次插入1，一次插入“2”，再输出调试信息，可以看到两个元素都成功的插入集合。

如果创建了多个集合，还可以计算它们的交集、并集（这些概念在初中数学课本中有阐述，如果您还不熟悉，可以忽略这段内容）。以取集合的并集为例。通过插入元素构建出两个集合，一个为`{1，"2"}`，另一个为`{"1"，"2"}`。添加一条“取并集”命令。该命令有三个属性：“集合”属性和“比对集合”分别填写需要合并的两个集合；“输出到”属性填写合并之后的集合变量。输出调试信息，可以看到合并之后集合

变为{1, "1", "2"}，这说明并集剔除了重复元素"2"，1和"1"一个是数值，一个是字符串，不属于重复元素，因此同时选入并集。

以上内容的关键源代码如下：

```
ObjSet=Set.Create()
Set.Add(ObjSet,1)
Set.Add(ObjSet,"2")
TracePrint(objSet)

ObjSet2=Set.Create()
Set.Add(ObjSet2,"1")
Set.Add(ObjSet2,"2")

objSetRet = Set.Union(ObjSet,ObjSet2)
TracePrint(objSetRet)
```

创建好一个集合，或者计算出交集、并集之后，还可以把集合转换为普通的数组，UiBot提供了一条命令实现此功能，请读者自行练习。

2.2.3 数据表

我们在前文中学习了二维数组的概念。数据表可以看作是一种特殊的二维数组，但比普通的二维数组增加了很多功能，例如可以包含表头，可以进行排序、过滤等实用的操作。

通过一个例子来看如何构建数据表。在命令列表中找到“数据处理”的“数据表”命令分类，选择并插入一条“构建数据表”命令。这条命令可以通过表头和构建数据，来生成一个数据表，该命令共有三个属性：“表格列头”属性，用于表示数据表的表头，可填写一个一维数组，我们这里填写的是["姓名", "科目", "分数"]；接下来是“构建数据”属性，可以填写一个二维数组，表示数据表中的初始数据，这里填写的是[["张三", "语文", "78"],["张三", "英语", "81"],["张三", "数学", "75"],["李四", "语文", "88"],["李四", "英语", "84"],["李四", "数学", "65"]]。您也可以选择不要表头，或者不要初始数据，在相应的属性里输入null即可。

图 21: 构建数据表

这样，数据表就构建好了，并且存储到了“输出到”属性中填写的变量objDatatable中。这个数据表实际上表示的是如下的一个表格，类似的表格在办公领域中经常遇到，可以举一反三：

姓名	科目	分数
张三	语文	78
张三	英语	81
张三	数学	75
李四	语文	88
李四	英语	84
李四	数学	65

数据表构建完成后，可以基于数据表进行读取、排序、过滤等各种数据操作。先来看数据的排序操作。插入一条“数据表排序”命令，这条命令共有四个属性：“数据表”属性填写待排序的数据表，这里填写上一步获得的数据表对象objDatatable；“排序列”属性表示按哪一列进行排序，这里填写的是“科目”；“升序排序”属性指的排序方法，“是”表示升序，“否”表示降序。

图 22: 数据表排序

“输出到”属性填写排序之后的数据表对象，这里仍然填写objDatatable。使用“输出调试信息”命令查看排序后的数据表，如下所示：

序号	姓名	科目	分数
2	张三	数学	75
5	李四	数学	65
1	张三	英语	81
4	李四	英语	84
0	张三	语文	78
3	李四	语文	88

再来看数据的筛选。插入一条“数据筛选”命令，这条命令共有四个属性：“数据表”属性填写待筛选的数据表，这里填写上一步获得的数据表对象objDatatable；“筛选条件”属性指的是筛选出哪些满足条件的数据，点击属性栏右边的“更多”按钮，会弹出“筛选条件”输入框。筛选条件包括为“列”、“条件”、“值”的组合，例如"科目 等于 '语文'"，表示筛选出科目为“语文”的所有数据。我们可以增加筛选条件，多个筛选条件可以是“且”的关系，也可以是“或”的关系。

图 23: 数据筛选

图 24: 数据筛选条件

使用“输出调试信息”命令查看筛选后的数据表，如下所示：

序号	姓名	科目	分数
0	张三	语文	78
3	李四	语文	88

数据表还可以转换为二维数组，UiBot提供了一条命令实现此功能，请读者自行练习。

2.2.4 字符串

字符串是我们最常用的数据类型，字符串操作也是最常见的数据操作。熟练掌握字符串操作，后续开发将会受益良多。先来看一条最经典的命令：“查找字符串”。这条命令将会查找字符串内是否存在指定的字符，该命令有五个属性：“目标字符串”属性填写被查找字符串，这里填写的是`"abcdefghijklmn"`；“查找内容”属性填写待查找的指定字符，这里填写的是`"cd"`；“开始查找位置”属性指的是从哪个位置开始查找，起始位置为1；“区分大小写”属性指的是查找时是否区分大小写，默认为“否”；“输出到”属性填写一个变量iRet，该变量保存查找到的字符位置。运行命令，显示变量iRet的值，输出结果为3，表

明"cd"出现在"abcdefghijklmn"的第3位。如果要查找的字符串不存在，输出的结果将会是0。

图 25: 查找字符串

再来看一条常见的字符串操作：“分割字符串”命令。这条命令使用特定分隔符，将字符串分割为数组。比如可以用这条命令来处理前面提到的CSV格式文件，因为CSV格式文件中是有明确的分隔符的。该命令有三个属性：“目标字符串”属性填写待分割的字符串，这里填写"zhangsan|lisi|wangwu"；“分隔符”属性填写用以分割字符串的符号，这里填写的是"|"；“输出到”属性保存分割后的字符串数组到arrRet。为了查看结果，我们再来添加一条“输出调试信息”命令，输出变量arrRet的值，可以看到结果为["zhangsan", "lisi", "wangwu"]，表明字符串"zhangsan|lisi|wangwu"通过分隔符"|"，被成功地分割为字符串数组["zhangsan", "lisi", "wangwu"]。

图 26: 分割字符串

2.2.5 正则表达式

在编写字符串处理流程时，经常会需要查找和测试某个字符串是否符合某些特定的复杂规则，正则表达式就是用于描述这些复杂规则的工具，它不仅可以很方便地对单个字符串数据进行查找和测试，也可以很好地处理大量数据（如：数据采集、网络爬虫等）。

先来看“正则表达式查找测试”命令，这条命令尝试使用正则表达式查找字符串，能够找到则结果为真（True），找不到则结果为假（False），该命令可用于判断一个字符串是否满足某个条件。该命令有三个属性：“目标字符串”属性填写待测试的字符串；“正则表达式”属性填写正则表达式；“输出到”属性保存测试结果。举个例子，网站判断用户输入的注册用户名是否合法，首先将合法用户名的判断条件写成正则表达式，然后使用正则表达式去测试用户输入的字符串是否满足条件。具体来看，“正则表达式”属性填入`"^[a-zA-Z0-9_-]{4,16}$"`，表示注册名为4到16位，字符可以是大小写字母、数字、下划线、横线；“目标字符串”如果填入`"abc_def"`，测试结果为`True`，说明`"abc_def"`符合正则表达式。“目标字符串”如果填入`"abc"`或`"abcde@"`，测试结果为`False`，因为`"abc"`的长度为3，`"abcde@"`中含有字符`"@"`，都不符合正则表达式规则。

注意：在上面的正则表达式中，开头的`"^"`符号和结尾的`"$"`符号代表匹配到字符串的开头或者结尾，有了这两个符号，待测试的字符串必须**全部匹配**正则表达式，才会得到`True`的结果。如果没有这两个符号，则待测试的字符串中只要包含了能匹配的**子串**，就会得到`True`的结果。

图 27: 正则表达式查找测试

再来看“正则表达式查找全部”命令，这条命令使用正则表达式查找字符串，并找出所有满足条件的字符串。该命令也有三个属性：“目标字符串”属性填写待查找的字符串；“正则表达式”属性填写正则表达式；“输出到”属性则以数组的形式，保存所有找到的子串。举个例子，“目标字符串”属性填写网络爬虫爬回来的一段网页，如下所示：

```
<p/>
<img src="https://avatar.csdn.net/A/4/C/3.jpg"/>
<p/>
<img src="https://g.csdnimg.cn/static/1x/11.png"/>
<p/>
```

“正则表达式”属性填写 `https?://[-A-Za-z0-9+&@#/%?=~_|!:,.;]+[-A-Za-z0-9+&@#/%=~_|]`，这段正则表达式看起来很复杂，其实它不是我写出来的，是我在互联网上找到的一段可以用来匹配URL的正则表达式。通过这个正则表达式，就可以把爬回来的网页中的所有链接全部抽取出来。

配置好属性之后，选中这一行命令，点击右边的三角形，单独运行这一行命令，即可自动输出结果。可以看到结果为：

```
arrRet = [
    "https://avatar.csdn.net/A/4/C/3.jpg",
    "https://g.csdnimg.cn/static/1x/11.png"
]
```

两个URL都被成功地抽取出来了！

正则表达式命令的使用本身不难，难点在于正则表达式如何书写。对于这方面内容，互联网上已经有大量的教程了，本文不再赘述。

2.2.6 JSON

JSON是一种轻量级的数据交换格式，用于存储和交换文本信息。JSON易于被人阅读和编写，同时也易于机器解析和生成。JSON在用途上类似XML，但比XML更小、更快，更易解析。

UiBot共有两条JSON命令，一条是“JSON字符串转换为数据”，一条是“数据转换为JSON字符串”。这里的数据，指的是UiBot中数组、字典格式的数据。也就是说，通过这两条命令，可以把JSON字符串和UiBot中的数组、字典进行双向转换。其中UiBot中的数组对应于JSON中的Array（中文通常也称为数组），UiBot中的字典对应于JSON中的Object（中文通常称为对象）。

先来看“JSON字符串转换为数据”命令，这条命令可以将JSON形式的字符串转换为UiBot中的数组或者字典。该命令有两个属性：“转换对象”属性，填写JSON字符串，这里填写的是 `{ "姓名": "张三", "年龄": "26" }`。“输出到”属性，填写转换后的UiBot变量，例如我们这里填写objJSON。

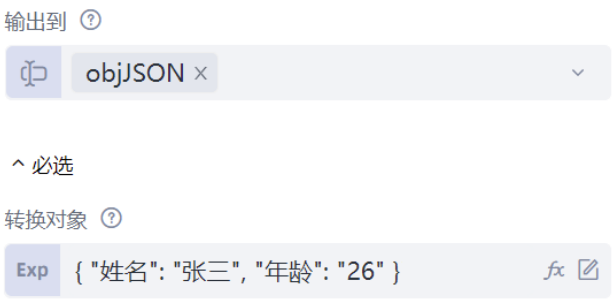

图 28: JSON字符串转换为数据

使用“输出调试信息”命令，可以输出结果：`{ "姓名": "张三", "年龄": "26" }`。有的读者可能会觉得疑惑，这输入和输出看起来没啥区别？其实不然！注意在上图中，我们在填写JSON字符串的时候，

是采用普通模式输入的（左边的Exp按钮为灰色），实际上UiBot会进行必要的转义后，将其变成一个字符串。切换到专业模式（点击Exp按钮，使其变为蓝色），就可以看到这个字符串的细节。而输出则是一个UiBot中的字典。输入字符串，输出字典，两者在使用时差别很大。例如，UiBot中的字典可以采用objJSON["姓名"]的方式，来使用其中相应元素的值，其结果为"张三"，而字符串就不能使用这种方式来操作了。

切换到源代码视图来尝试一下：

```
TracePrint(objJSON["姓名"])
```

既然能访问，应该也能修改，添加一条赋值语句，该语句将objJSON的"年龄"修改为30。

```
objJSON["年龄"]="30"
```

最后，再通过“数据转换为JSON字符串”命令，将修改后的JSON对象转换为字符串。这条命令共有两个属性：“转换对象”属性，填写待转换的UiBot数组或字典，也就是前面一直使用的objJSON；“输出到”属性填写一个字符串变量，该变量将会保存转换后的JSON字符串。使用“输出调试信息”命令查看转换后的JSON字符串："{ "姓名" : "张三", "年龄" : "30" } "，可以看到，其内容修改成功。

下面来做一个综合的实验：读取一个JSON格式的文件，将读到的数据修改后再写回。首先，我们新建一个流程，并在流程所在文件夹下的res文件夹下新建一个person.json文件，其内容为：

```
{ "姓名" : "张三", "年龄" : "30" }
```

打开流程块，切换到源代码模式，把下面的源代码粘贴进去。

```
Dim obj
File.Read(@res"person.json","auto")
obj=JSON.Parse($PrevResult)
obj['性别']='男'
JSON.Stringify(obj)
File.WriteFile(@res"person.json", $PrevResult)
```

这个简单的流程综合使用了本章所述的文件读写和JSON数据处理的命令，运行后，再次打开person.json文件并进行查看，可以看到，在其内容中增加了一个字段“性别”，取值是“男”。

这段流程并不复杂，即使不加注释，相信读者也不难读懂。稍微值得注意的一点是：这段流程里面两次使用了系统变量$PrevResult，来指代“上一条命令的结果”。如果某条命令的输出仅作为下一条命令的输入，而不会用在其他地方，那么采用这个系统变量，可以减少对变量的定义和使用。并且，把流程切换到可视化视图的时候，也会更加简洁易读。

2.2.7 时间

时间操作命令主要包括时间数据和字符串的互相转换，以及对时间数据的各种操作。首先来看如何获取当前时间，在命令中心“数据处理”的“时间”命令分类下，选择并插入一条“获取时间”命令。这条命令可以获取从1900年1月1日起到现在经过的天数，该命令只有一个“输出到”属性，保存当前时间，这里填

写dTime。在可视化视图下，点击这一行命令右边的三角形，运行这一行命令，会自动输出得到的结果：43771.843969907，说明从1900年1月1日到现在，已经过去了43771.843969907天，后面的小数位代表了一天当中的时、分、秒，大家可以大致估算一下是否正确。

用这种方式来保存时间，有时候会更容易处理，比如要获得100天以后的时间，只需要把上面的值加100就可以了，而不用考虑年、月、日的进位。当然，这种表示时间的方式毕竟不符合我们的日常习惯，不适合给人阅读。如果需要把时间展示给人去看，则可以通过“格式化时间”命令，将时间数据转换成各种格式的字符串。“格式化时间”命令有三个属性：“时间”属性填写刚刚得到的时间数据dTime；“格式”属性填写时间格式，其中年(yyyy)占4位、月(mm)、日(dd)、24小时(hh)、分(mm)、秒(ss)都占2位，例如`"yyyy-mm-dd hh:mm:ss"`填写时间后转换为：`"2019-11-02 20:29:58"`；“输出到”属性保存格式化时间的字符串。细心的读者可能已经发现了：用这种格式，“月”和“分”的格式都是`mm`，这是因为这两个词在英语中分别是“month”和“minute”，都是以字母`m`开头的。所以格式都是`mm`，而UiBot会根据上下文来判断`mm`到底是指代“月”还是“分”。如果您希望更加明确地指定，而不要让UiBot去判断，也可以用大写的`MM`来作为“月”的格式，而用`nn`来作为“分”的格式，效果是一样的。

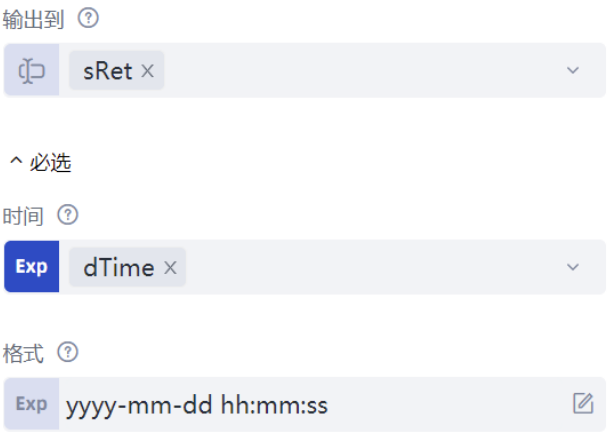

图 29: 格式化时间

除了将时间数据转换成各种格式的字符串，也可以直接获取时间数据的某一分项。例如可以使用“获取月份”命令获取时间数据dTime中的月份，以此类推，其它命令类似。

3 网络和系统操作

至此为止，已经讲述了UiBot中大部分常用的命令。实际上，UiBot中的“网络”和“系统操作”这两类命令虽然使用频率稍微低一些，但应用得当，也能发挥很大作用。我们在本章内容中介绍这两类命令。

3.1 网络操作

3.1.1 HTTP操作

在网络时代，我们经常会听到 `HTTP` 这个词，因为几乎所有网址都需要加一个前缀 `http://` 或者 `https://` 。其实，`HTTP` 是 Hyper Text Transfer Protocol（超文本传输协议）的缩写，指的是本地浏览器与网络服务器进行超文本传输和通信协议。也就是说，我们使用浏览器访问网站时，本质上就是通过HTTP协议与远程的网络服务器打交道。

其实不止是浏览器，只要遵循HTTP协议的标准，其它应用程序同样也可以与网络服务器进行通信。这就厉害了！这说明，前面章节中介绍的某些网页自动化操作，其实可以不通过浏览器，而是直接通过HTTP协议进行操作。这就为网络操作自动化，开辟了一条新的思路。

应用得比较多的HTTP请求，主要有两大类：**HTTP GET**和**HTTP POST**。我们来看看如何在UiBot中实现这两类请求，首先是HTTP GET请求。在命令中心“网络”的“HTTP”目录下，选择并插入一条“Get获取数据”命令，该命令将创建一个指向特定网址的HTTP GET请求。该命令有四个属性，如下图所示。

图 30: Get获取数据

“链接地址”属性表示此次HTTP请求的网址；“表单数据”属性是一个JSON格式的字符串，表示此次HTTP请求需要发送的数据，这里一共有两组键值对，分别为“user”和“username”、“password”和“12345678”，如下所示。

```
{
    "user": "username",
    "password": "12345678"
}
```

“超时时间”属性指定了此次HTTP请求的超时时间，如果超过这个时间仍没有数据返回，则认为此次HTTP请求失败。“输出到”属性填写一个变量名，这个变量将会以字符串的形式，保存此次HTTP请求返回的结果。不同的网址，返回的结果格式也会有很大不同。一般来说，如果是可以被人浏览的网址（例如 `https://www.laiye.com`），那么返回的往往是一个HTML格式的字符串；如果是一个在线服务，那么返回的经常是一个XML或者JSON格式的字符串。在本例中，我们测试的网址 `http://httpbin.org/get` 是一个在线服务，它的返回结果也是一个JSON格式的字符串，如下所示。如果需要对这个返回结果进行进一步的处理，那么可以根据我们前文所学的内容，解析这个JSON字符串。

```
{
    "args": {
        "password": "12345678",
        "user": "username"
    },
    "headers": {
        "Accept": "*/*",
        "Accept-Encoding": "gzip, deflate",
        "Host": "httpbin.org",
        "User-Agent": "python-requests/2.21.0"
    },
    "origin": "49.94.16.174, 49.94.16.174",
    "url": "https://httpbin.org/get?user=username&password=12345678"
}
```

HTTP GET请求除了能够执行普通的GET请求之外，还能够下载文件，这个功能通过“Get下载文件”命令实现。这个命令有五个属性，其中“链接地址”、“表单数据”、“超时时间”、“输出到”四个属性的含义，与上一条“Get获取数据”命令的四个属性一致，另外一个“文件路径”属性填写下载文件的保存路径。

图 31: Get下载文件

我们再来看看如何在UiBot中实现HTTP POST请求。在命令中心“网络”的“HTTP”目录下，选择并插入一条“Post提交表单”命令，该命令将创建一个指向特定网址的HTTP POST请求。该命令有四个属性，如下图所示。

图 32: Post提交表单

“链接地址”、“表单数据”、“超时时间”、“输出到”属性的含义与“Get获取数据”命令的属性含义相同。需要注意的是：HTTP POST请求发送的“表单数据”与请求的网址相关，需要跟相关的业务人员仔细核对，

“表单数据”的键值对仔细填写正确。同样的，HTTP POST请求返回的结果也是一个字符串，但具体的格式会随着不同的网址而不同。可能是JSON字符串，也可能是HTML字符串，或者其他字符串。

3.1.2 电子邮件协议

在企业业务流程自动化中，电子邮件的自动化收发是非常重要的一环，而且非常基础和高频。除了自动化操作邮件客户端或者通过浏览器自动登录邮箱，UiBot还提供基于电子邮件协议直接收发邮件的命令。常用的电子邮件协议有SMTP、POP3、IMAP等，它们都隶属于TCP/IP协议簇，默认状态下，分别通过TCP端口25、110和143建立连接。在使用命令(SMTP、POP、IMAP)之前，通常需要登录邮箱做一些设置，下面以QQ邮箱为例进行说明。

打开浏览器，登录QQ邮箱，点击“设置”按钮，再点击“账户”标签页，找到“POP3/IMAP/SMTP/Exchange/CardDAV/务”，开启“POP3/SMTP服务”和”IMAP/SMTP服务“（服务默认是关闭的），这个时候QQ邮箱系统会生成一串授权码，将这串授权码保存好（只会显示1次），后续的邮件收取和邮件发送操作都使用这串授权码，而不是使用您邮箱的原始密码。

图 33: QQ邮箱开启POP和SMTP

然后点击“如何使用 Foxmail 等软件收发邮件？”，这篇帮助文档中有QQ邮箱的关键设置信息，如下图所示：POP服务器地址为pop.qq.com，端口号为995，SSL为“是”；SMTP服务器地址为smtp.qq.com，端口号为465，SSL为“是”。这些信息在后续的UiBot邮件命令中都会用到。

图 34: QQ邮箱设置信息

接着点击“什么是IMAP，它又是如何设置？”，这篇帮助文档中同样有QQ邮箱的关键设置信息，如下图所示：接收邮件服务器：imap.qq.com，使用SSL，端口号993。同样这些信息在后续的UiBot邮件命令中也会用到。

如何设置IMAP服务的SSL加密方式?

使用SSL的通用配置如下:

接收邮件服务器: imap.qq.com，使用SSL，端口号993

发送邮件服务器: smtp.qq.com，使用SSL，端口号465或587

账户名: 您的QQ邮箱账户名（如果您是VIP帐号或Foxmail帐号，账户名需要填写完整的邮件地址）

密码: 您的QQ邮箱密码

电子邮件地址: 您的QQ邮箱的完整邮件地址

图 35: QQ邮箱IMAP设置

3.1.2.1 SMTP

SMTP的全称是“Simple Mail Transfer Protocol”，即简单邮件传输协议。它是一组用于从源地址到目的地址传输邮件的规范，通过它来控制邮件的中转方式。在命令区域“网络”的“SMTP/POP”分类下，选择并插入一条“发送邮件”命令，该命令将使用SMTP协议给指定邮箱发送一封邮件。

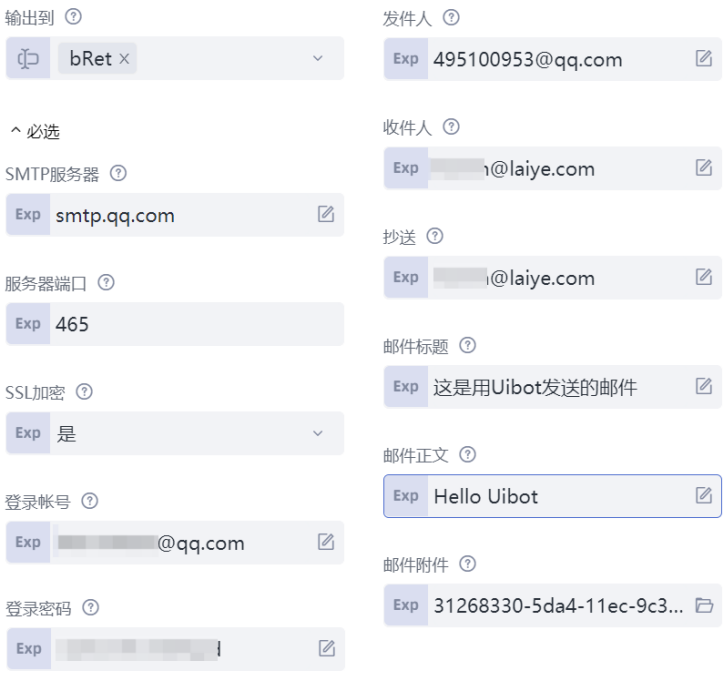

图 36: 发送邮件属性区域

该命令有几个属性：“SMTP服务器”属性填写邮箱的SMTP服务器地址，“服务器端口”属性填写SMTP协议端口号，“SSL加密”属性选择“是”，这三个信息在上一步QQ邮箱的设置中已经得到，其中服务器地址为smtp.qq.com，端口号为465；“邮箱帐号”属性填写需要登录的邮箱帐号（邮箱账户与发送邮箱信息一致）；“登录密码”属性填写生成的授权码（在QQ邮箱的设置中生成授权码，不同的邮箱可能设置也会不同）；“收信邮箱”属性填写对方的邮箱帐号；“邮件标题”属性填写待发送邮件的标题；“邮件正文”属性填写待发送邮件的正文（仅文本格式）；“邮件附件”属性填写待发送邮件的附件文件地址；最后，“输出到”一栏会把此次邮件发送操作是否成功置入指定的变量中，成功则置入True，失败则置入False。如下图即为已发送成功的邮件：

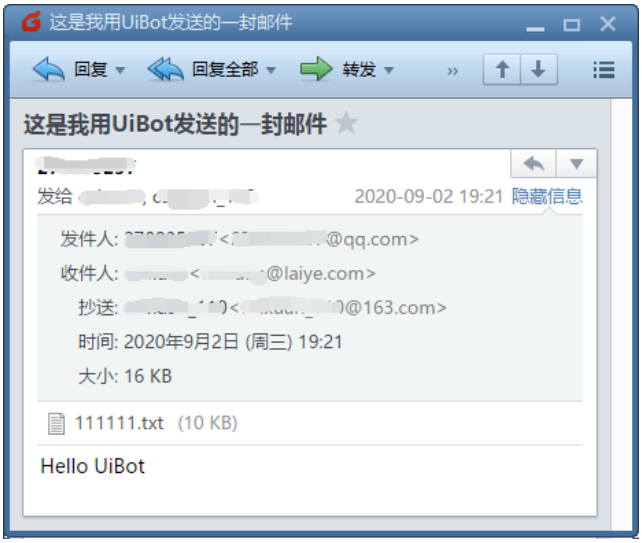

图 37: 收到使用SMTP发送的邮件

SMTP协议的使用比较简单，一条命令就够了，但它只能发邮件。如果要收邮件，需要用到POP协议。

3.1.2.2 POP

POP的含义是邮局协议，它负责从邮件服务器中检索电子邮件，POP3(Post Office Protocol 3)即邮局协议的第3个版本，是因特网电子邮件的第一个离线协议标准。POP协议在今天的互联网上早已广泛使用。

用POP协议可以收邮件，但会稍微复杂一点儿。一条命令是不够的，需要多条命令配合才行。

在命令区域“网络”的“SMTP/POP”分类下，选择并插入一条“连接邮箱”命令，该命令将使用POP协议连接上指定邮箱，并支持后续收取邮件的操作。

图 38: POP连接邮箱

该命令有如下几个属性：“服务器地址”属性填写邮箱的POP服务器地址，“服务器端口”属性填写POP协

议端口号，“SSL加密”属性选择“是”，这三个信息在上一步QQ邮箱的设置中已经得到，其中服务器地址为pop.qq.com，端口号为995；“邮箱帐号”属性填写需要收取邮件的邮箱帐号；“登录密码”属性填写邮箱的授权码（和SMTP一致）；“使用协议”属性默认填写“POP3”；“输出到”属性填写变量名objMail，这个变量会保存连接邮箱后得到的邮箱对象，后续获取邮件列表、删除邮件、下载附件等命令，都要使用这个邮箱对象。

成功连接邮箱后，接下来可以收取邮件了。

在命令区域“网络”的“SMTP/POP”分类下，选择并插入一条“获取邮件列表”命令，该命令将输出指定数量的邮件信息清单（结果是一个字典类型的变量，可以获取长度、遍历其中内容等。另外，邮箱服务器一般限制了只收取最近30天的邮件，可自行更改此限制）。该命令有几个属性：“操作对象”属性填写刚刚在“连接邮箱”命令中得到的邮箱对象objMail；“邮件数量”填写我们需要收取多少封邮件，这里填写的是10，表示收取10封邮件。如果不希望限定数量，可以填写0；“输出到”属性填写一个变量名，这个变量会保存“获取邮件列表”命令的执行结果：10封邮件信息。

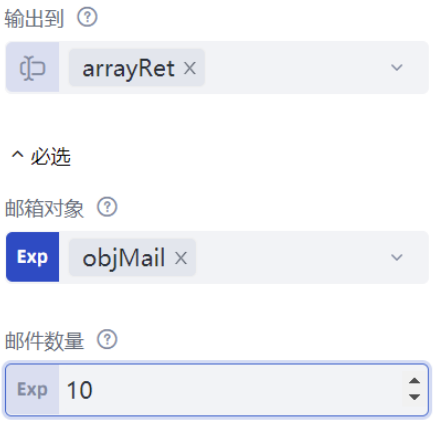

图 39: POP获取邮件列表

有两个命令——“下载附件”、“删除邮件”，不仅需要结合“连接邮箱”命令返回的邮箱对象，而且还要结合”获取邮件列表“的返回结果（即从邮件列表中获取一个元素，称为邮件对象，邮件对象实际上是一个字典，包含一封邮件的标题、正文、附件、发送人、日期、大小等字段信息），才可以使用。

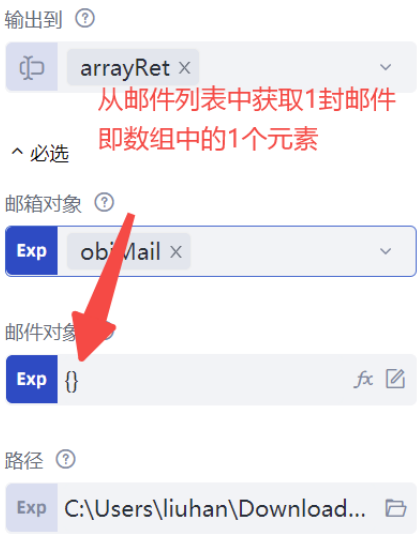

图 40: POP下载附件

另外指定邮件对象删除对应邮件，在使用该命令删除邮件后，必须调用”断开邮箱连接“命令，才能真正删除成功。如果邮件服务器设置了“禁止收信软件删除邮件”，则依然无法删除。

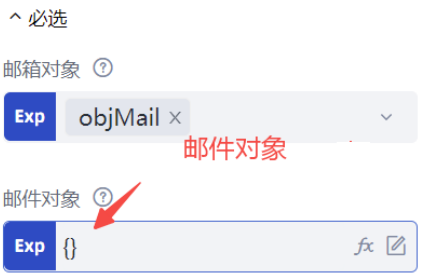

图 41: POP删除邮件

3.1.2.3 IMAP

互联网信息访问协议（IMAP）是一种优于POP的新协议。和POP一样，IMAP也能下载邮件、从服务器中删除邮件或询问是否有新邮件，但IMAP克服了POP的一些缺点。例如，它可以决定客户机请求邮件服务器提交所收到邮件的方式，请求邮件服务器只下载所选中的邮件而不是全部邮件。客户机可先阅读邮件信息的标题和发送者的名字再决定是否下载这个邮件。通过用户的客户机电子邮件程序，IMAP可让用户在服务器上创建并管理邮件文件夹或邮箱、删除邮件、查询某封信的一部分或全部内容，完成所有这些工作时都不需要把邮件从服务器下载到用户的个人计算机上。

在命令区域“网络”的“IMAP”目录下，选择并插入一条“连接邮箱”命令，该命令将使用IMAP协议连接上指定邮箱，并支持后续收取邮件的操作。

图 42: IMAP连接邮箱

该命令有如下几个属性：“服务器地址”属性填写邮箱的IMAP服务器地址，“服务器端口”属性填写IMAP协议端口号，“SSL加密”属性选择“是”，这三个信息在上一步QQ邮箱的设置中已经得到，其中服务器地址为imap.qq.com，端口号为993；“邮箱帐号”属性填写需要收取邮件的邮箱登录帐号；“登录密码”属性填写邮箱授权码；“邮箱地址”属性默认填写需要收取邮件的邮箱地址；“输出到”属性填写变量名objIMAP，这个变量会保存连接邮箱后得到的邮箱对象，后续获取邮件列表、删除邮件、移动邮件、查找邮件、下载附件等命令，都要使用这个邮箱对象。

另外，需要注意“登录账号”、“邮箱地址”这两个属性。对于普通的QQ邮箱，这两个的值是一样的，而VIP的QQ邮箱，这两个值分别是“登录账号”对应QQ号，“邮箱地址”对应VIP邮箱地址，如下图所示：

图 43: IMAP连接QQVip邮箱

“登录账户”、“邮箱地址”是2个属性，这一点要非常注意，在很多域计算机环境下，“登录账户”对应的可能是域用户。

对邮件进行获取、删除、移动、查找操作时，有一个属性“邮箱文件夹”必须是知晓的，怎么知晓呢？有人说，直接登录邮箱进去看一眼就知道了，如下图以QQ邮箱举例所示：

图 44: QQ邮箱文件夹列表

如上图所展示的“收件箱”、“草稿箱”、“已发送”、“已删除”、“垃圾箱”、“我的文件夹”、“test”等，是不是就是我们要填写的“邮箱文件夹”属性呢？让我们先来使用一个命令——获取邮箱文件夹列表(主要是辅助作用，为其他命令的邮箱文件夹属性提供可用值)，看看输出结果是什么？

```
"其他文件夹",
"INBOX",
"Sent Messages",
"Drafts",
"Deleted Messages",
"Junk",
"其他文件夹/QQ邮件订阅",
"其他文件夹/DATA"
```

图 45: IMAP获取QQ邮箱文件夹列表

看起来似乎不对，“我的文件夹”好像没有了。实际上，“我的文件夹”在这里对应的是“其他文件夹”，“收件箱”实际对应“INBOX”，“已发送”实际对应“Sent Messages”。基于电子邮件协议，拿到的报文信息与实际在界面上显示看到的会有所不同(Web、邮件客户端在显示时会做优化处理)，若调试“获取邮件列表”命令，在“邮箱文件夹”属性处输入“我的文件夹”，会看到报错信息“Folder not exist”，若改成输入“其他文件夹”，就不会报错了。这里只是以QQ邮箱举例，其实不同的邮箱服务器会有各自的访问要求和报文返回格式，很难统一。因此，请多使用“获取邮箱文件夹列表”命令，获得具体的邮箱文件夹列表信息，并以此为准。属性“原始报文”若选择为“是”,则输出的是原始报文信息，选择为“否”，则输出的是基于一定的提取规则进行提取后的文件夹列表信息(有的邮箱不支持此提取规则)。如下图，获取QQ邮箱文件夹

列表的原始报文信息：

```
"(\\NoSelect \\HasChildren) \"/\" \"其他文件夹\"",
"(\\HasNoChildren) \"/\" \"INBOX\"",
"(\\HasNoChildren) \"/\" \"Sent Messages\"",
"(\\HasNoChildren) \"/\" \"Drafts\"",
"(\\HasNoChildren) \"/\" \"Deleted Messages\"",
"(\\HasNoChildren) \"/\" \"Junk\"",
"(\\HasNoChildren) \"/\" \"其他文件夹/QQ邮件订阅\"",
"(\\HasNoChildren) \"/\" \"其他文件夹/DATA\""
```

图 46: QQ邮箱文件夹列表原始报文

已经能够知晓到邮箱文件夹信息，再来重点讲解“获取邮件列表”、“查找邮件”、“移动邮件”这3条命令的用法，也是与POP协议的关键区别：

图 47: IMAP获取邮件列表

如上图为“获取邮件列表”命令的属性区域，支持“仅限未读消息”、“标记为已读”设置，比POP协议更加灵活，输出的数组中元素的字段结构与POP也有稍微区别，即POP的邮件对象与IMAP的邮件对象存在区别。

图 48: IMAP查找邮件

如上图为“查找邮件”命令的属性区域，可以指定邮箱文件夹和查找关键字（检索邮件头信息：主题、发件人、收件人、抄送人）进行邮件查找，返回的结果为数组，若没有检索到邮件则返回空数组，检索到1封邮件，则数组里只有1个元素，若多封邮件则多个元素。“字符集”属性默认为“gb2312”,遇到邮箱服务器的编码不一致才需要调整。

图 49: IMAP移动邮件

如上图“移动邮件”命令的属性区域，依赖“邮箱对象”和“邮件对象”(与“下载附件”命令相同)，指定目标邮

箱文件夹后，就可把指定的邮件移动到指定的邮箱文件夹下，而POP协议是缺失此功能的。

3.2 系统操作

3.2.1 系统命令

UiBot提供了一些实用的命令，来调用操作系统的相关功能，例如播放声音、读取和设置环境变量、执行命令行和PowerShell、获取系统和用户的文件夹路径等。这些操作系统相关的命令（简称“系统命令”）与其它命令搭配使用，能够让您的自动化流程如虎添翼。

我们先尝试找到并插入一条“播放声音”命令，该命令可以播放指定文件路径的声音文件。该命令的属性只有一个：“文件路径”属性，这个属性填写待播放声音的完整文件路径。目前，UiBot暂时只支持wav格式的声音播放。

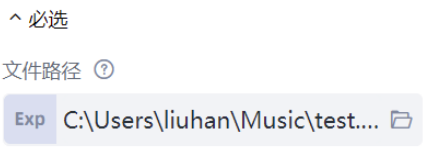

图 50: 播放声音

在命令中心“系统操作”的“系统”目录下，选择并插入一条“读取环境变量”命令，该命令可以读取Windows操作系统的环境变量。该命令的属性有两个：“环境变量”属性，填写环境变量的名称；“输出到”属性填写一个变量名，这个变量将会保存读取出来环境变量的值。需要注意的是：“读取环境变量”命令一次只能读取一个变量，如果需要读取多个环境变量的值，可以多次调用该命令。另外，读取出来的环境变量的文本格式与操作系统中环境变量的文本格式完全一致，如果需要进一步解析环境变量文本，可采用上一章学习的数据处理的相关命令。“设置环境变量”命令的用法与“读取环境变量”命令类似，这里也不再展开讲解。

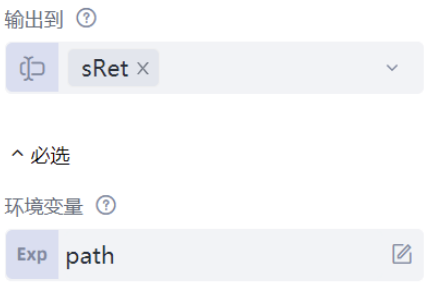

图 51: 读取环境变量

在命令中心“系统操作”的“系统”目录下，选择并插入一条“执行命令行”命令，该命令可以执行Windows脚本，该命令的属性有两个：“命令行”属性，填写待执行的Windows脚本；“输出到”属性，填写一个变量，这个变量保存脚本的执行结果。Windows脚本的写法，请查阅相关文档。

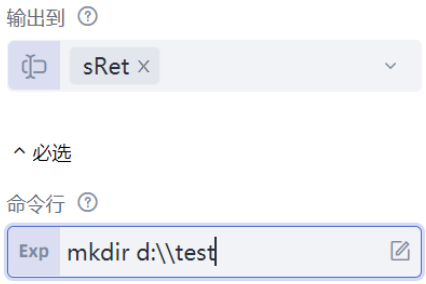

图 52: 执行命令行

在命令中心“系统操作”的“系统”目录下，选择并插入一条“获取系统文件夹路径”命令，该命令可以获取各个系统文件夹的路径，该命令的属性有两个：“获取目录”属性，目前UiBot支持如下路径的获取：系统目录、Windows目录、桌面目录、软件安装目录、开始菜单目录，用户可以根据需要自行选择；“输出到”属性，填写一个变量，这个变量保存“获取系统文件夹路径”命令的执行结果，即最后获取到的路径。

图 53: 获取系统文件夹路径

“获取临时文件夹路径”、“获取用户文件夹路径”与“获取系统文件夹路径”命令用法类似，不再赘述。

3.2.2 应用命令

在企业业务流程中，可能会需要启动、关闭应用程序。我们固然可以通过界面模拟，来实现这些功能。但实际上，UiBot提供了更简单易用的“应用命令”，可以直接管理应用程序的生命周期，包括启动应用、关闭应用、获取应用程序的状态等。先插入一条“启动应用程序”命令，该命令的属性如下：“文件路径”属性，填写待的启动应用程序的文件路径；“等待方式”属性指的是系统与该应用程序的等待关系，UiBot提供三种等待方式：“不等待”、“等待应用程序准备好”、“等待应用程序执行到退出”，“不等待”指

的是目标程序立即启动，命令即完成；“等待应用程序准备好”指的是只有应用程序准备好了，命令才算完成，否则会一直等待；“等待应用程序执行到退出”指的是启动应用程序后，UiBot会一直等待，直到应用程序退出，该命令才算执行完成。“显示样式”属性指的是当应用程序启动时，以何种方式显示，UiBot支持的显示样式有：“默认”、“最大化”、“最小化”、“隐藏”。“默认”的指的是应用程序以默认显示方式启动，“最大化”指的是启动时应用程序窗口最大化，“最小化”指的是启动时应用程序窗口最小化，“隐藏”指的是启动时不显示应用程序窗口。最后，还有“输出到”属性，填写一个变量，这个变量保存启动后应用程序的PID（即进程ID），这个PID在后续的命令中还会用到。

图 54: 启动应用程序

“启动应用程序”命令打开的是一个应用程序，但是有些场景不仅仅需要打开应用程序，还需要应用程序打开一个文件或网址，这个时候可以用到“打开文件或网址”命令。该命令的几个属性与“启动应用程序”命令大致相同，唯一不同的是“文件路径”属性，这里填写需要打开的文件或网址。

图 55: 打开文件或网址

通过“启动应用程序”命令或“打开文件或网址”命令启动应用程序后，在流程运行的过程中，我们可以随时查看某个特定应用是否仍在运行。插入一条“获取应用运行状态”命令，该命令可以根据进程名或PID判断一个进程是否存活。该命令的属性有两个：“进程名或PID”属性，填写应用的进程名或PID，如果填写PID，即是上一条“启动应用程序”命令“输出到”属性得到的值，如果填写进程名，则填写应用程序的完整文件名，例如上一条“启动应用程序”命令的进程名为“notepad.exe”；“输出到”属性保存命令的执行结果，True表示进程仍然存活，False表示进程已经关闭。

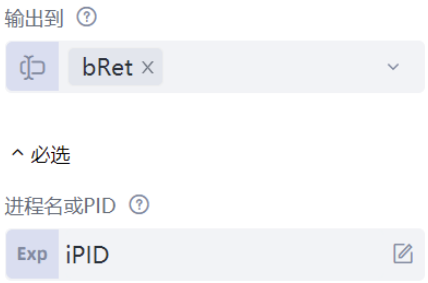

图 56: 获取应用运行状态

最后，别忘了关闭应用程序。选择并插入一条“关闭应用”命令，该命令可以根据进程名或PID关闭一个进程。该命令的属性只有一个：“进程名或PID”属性，填写待关闭应用的进程名或PID，含义与“获取应用运行状态”命令的同名属性是一样的。

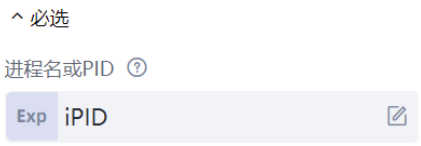

图 57: 关闭应用程序

3.2.3 对话框

RPA流程运行的过程中，一般来说不需要人工干预。但是某些场景下，流程需要与人进行双向的信息沟通，一方面将流程的关键信息通知给人，另一方面获取人的控制和决策信息。为此，UiBot提供了对话框机制，它既可以将流程关键信息在一个对话框里显示出来，也可以弹出对话框，让用户进行选择和输入，从而实现了流程与人的双向信息沟通。

在命令中心“系统操作”的“对话框”目录下，选择并插入一条“消息框”命令，该命令将在流程运行的过程中在屏幕的中间弹出一个消息框。

图 58: 消息框

该条命令有如下属性：“消息内容”属性填写消息内容主体；“对话框标题”属性指的是弹出对话框的标题栏；“按钮样式”属性指的是这个对话框中显示哪几个按钮，“按钮样式”属性选项有：“只显示确定”、“显示是和否按钮”、“显示放弃、重试和跳过按钮”、“显示是、否和取消按钮”、“显示重试和取消按钮”、“显示确认和取消按钮”，大家可以根据业务场景的需求，合理地选择“按钮样式”属性；

图 59: 消息框设置

“图标样式”属性指定了对话框显示什么图标，选项有：“不显示图标”、“显示消息图标”、“显示询问图标”、“显示警告图标”、“显示出错图标”，大家也可以根据业务场景合理选择图标；“超时时间”属性指的是多少毫秒以后，该对话框强制关闭，例如填写5000，表示5000毫秒后，即使用户未点击对话框的任何按钮，该对话框也会强制关闭。如果“超时时间”属性填0，UiBot就不考虑超时时间了，即用户必须点击对话框，对话框才会消失；“输出到”属性填写了一个变量，该变量在用户点击了对话框按钮后，会记录用户点击了哪个按钮。接下来的流程运行逻辑，可以根据用户点击了哪个按钮，选择不同的流程走向。各按钮的值如下：

按钮的值	按钮的含义
1	确定
2	取消
3	放弃
4	重试
5	跳过
6	是
7	否

“消息框”命令是一条非常强大的命令，用户可以根据实际需要填写消息框的内容、标题、按钮样式、图标样式等。但是有时候不需要这么复杂的对话框，也不需要用户来确认，只要弹出一条简单的消息通知即可，这时可以用“消息通知”命令。“消息通知”命令将会以气泡的方式弹出一条通知消息，即使用户不进行任何操作，也会在几秒钟后自动消失。

“消息通知”命令有几个属性：“消息内容”、“对话框标题”、“对话框图标”的含义与“消息框”命令相同。需要注意的是：“消息框”命令一般会停留几秒才消失，但是如果流程结束运行的话，流程中弹出的消息框也会随之消失，可以在“消息框”命令后面接一个“延时”命令，延时几秒钟时间，这样可以让用户查看消息框的时间更加充裕一些。

图 60: 消息通知设置

“消息框”命令和“消息通知”命令实现流程向用户传递消息，“消息框”命令虽然能够得到“用户点击哪个按钮”，完成用户向流程传递消息的部分功能，但是非常有限，如果用户要向流程传递更多的信息，该怎么办呢？UiBot提供了“输入对话框”命令，这条命令可以弹出一个对话框，用户可以在对话框中输入需要传递的信息，这个信息将会被传递到流程中。

我们来看看“输入对话框”命令的具体用法。“消息内容”属性和“对话框标题”属性的含义与“消息框”命令相同；“默认内容”属性是给用户的提示信息，如果用户不修改，那么“默认内容”属性中的内容将会传递给流程；“仅支持数字”属性是布尔值，“是”表示用户只能输入数字，“否”表示用户输入不受限制，可以输入任意字符；“输出到”属性填写一个变量，当执行完命令后，用户的输入信息将会保存到这个变量中，后续流程可以使用这个变量，这样就打通了用户输入到流程的通道。

图 61: 输入对话框

除了支持用户输入文字外，UiBot还支持用户输入文件。插入一条“打开文件对话框”命令，这个命令可以在流程运行的过程中，弹出一个“打开文件对话框”，用户选择文件，并将文件路径传递给流程。该命令有几个属性：“初始目录”属性指的是弹出对话框时，默认打开哪个目录，我们可以点击“初始目录”属性右边的文件夹按钮选择初始目录，如果没有选择初始目录（即该属性默认值为""）或者选择的初始目录不存在，UiBot会打开一个默认目录作为初始目录。“文件类型过滤”属性是一个字符串，指明了该对话框可以打开哪些类型的文件，具体的写法请参阅相关文档；“对话框标题”属性含义与其它对话框命令相同。“输出到”属性填写一个变量，当执行完命令后，用户选择文件的完整路径将会保存到这个变量中，也就是说，这个“打开文件对话框”并**“不会真正”**打开文件，而是得到文件路径，后续流程可以使用这个变量，这样就打通了用户输入到流程的通道。

图 62: 打开文件对话框

“打开文件对话框[多选]”、“保存文件对话框”与“打开文件对话框”命令用法类似，“打开文件对话框[多选]”命令可以选择多个文件，这条命令返回的是一个文件路径数组，用户可以遍历数组依次处理每个文件，这里不再展开讲解。

3.2.4 剪贴板

剪贴板的操作是我们在日常工作中非常常用的，它可以把文字或图像从一个应用程序迁移到另一个应用程序。我们固然可以通过界面模拟剪贴板的复制、粘贴等操作，更便捷的是，UiBot还可以直接读取或设置剪贴板中的文本或图像，让剪贴板真正成为业务流程的一部分。我们尝试插入一条“设置剪贴板文本”命令，该命令将一段文字置入剪贴板中。该命令只有一个“剪贴板内容”属性，显然，表示的是将要设置到剪贴板中的文字内容。

图 63: 设置剪贴板文本

我们可以接着插入一条“读取剪贴板文本”命令，查看上一条“设置剪贴板文本”命令是否成功。“读取剪贴板文本”命令同样只有一个属性——“输出到”属性，运行该命令后，属性中填写的变量sRet将会保存剪贴板中的文本。添加一条“输出调试信息”命令，将sRet输出，即可查看设置和读取剪贴板文本是否执行成功。

除了通过剪贴板传输文字，剪贴板同样也可以传输图像。在命令中心“系统操作”的“剪贴板”目录下，选择并插入一条“图片设置到剪贴板”命令，该命令将一个图像文件设置到剪贴板中。该命令只有一个“文件路径”属性，这个属性填写将要设置到剪贴板的图像文件的路径。

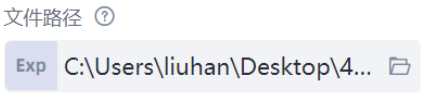

图 64: 图片设置到剪贴板

我们可以插入一条“保存剪贴板图像”命令，查看上一条“图片设置到剪贴板”命令是否成功。“保存剪贴板图像”命令只有一个“保存路径”属性，这个属性填写剪贴板中图像文件的保存路径。

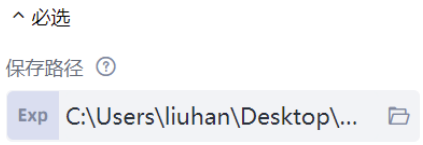

图 65: 保存剪贴板图像

3.2.5 文字写屏

文字写屏是一个非常有趣的功能。它可以在流程运行的过程中，把一段文字输出到屏幕上，不管屏幕正在显示什么内容，都会覆盖到显示内容之上。同时，文字的空隙区域又是透明的，仍然能通过这些空隙区域看到屏幕原有的内容。通过文字写屏，可以用非常醒目的方式，让用户看到必要的提示信息，而对流程的运行几乎没有影响。下图是UiBot的测试工程师在运行自测的实景图，UiBot的每个版本在发布前都会经过复杂的自测，通过这些写屏的红色文字，让测试工程师一眼就能看到自测仍然在进行中。

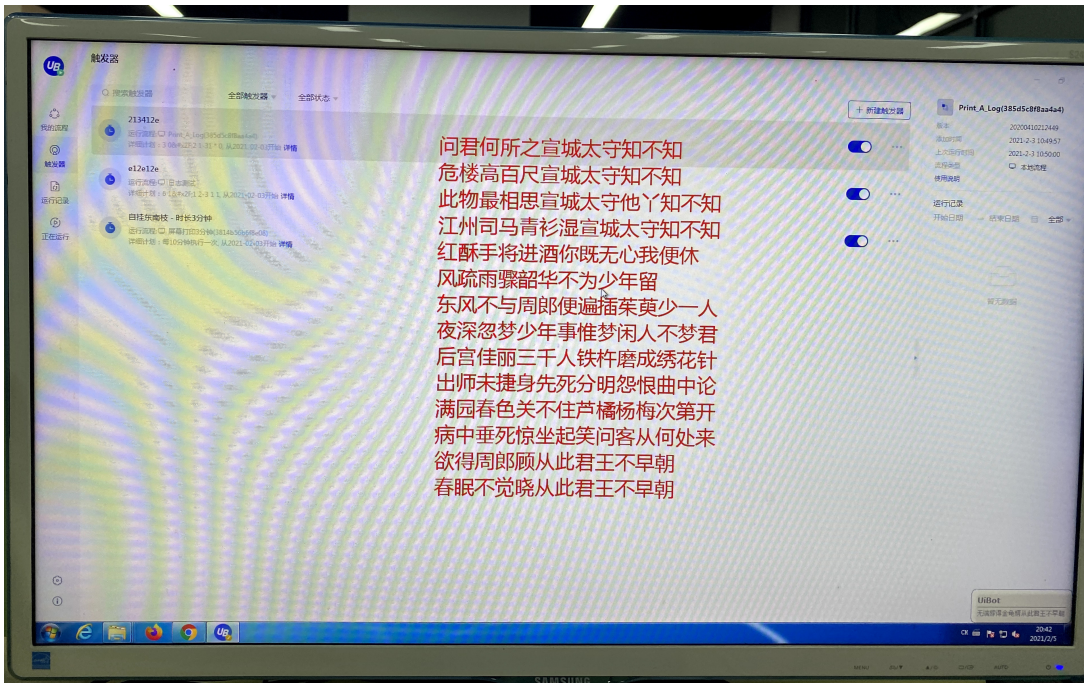

图 66: “文字写屏”的实景图

如此炫酷的效果，实现起来需要不止一条命令，但其实仍然很简单。在命令中心“系统操作”的“文字写屏”分类下，选择并插入一条“创建写屏对象”命令，后续的写屏操作都针对这个对象来进行。

图 67: 创建写屏对象

该条命令有三个属性：“写屏区域”是一个字典类型的值，表示文字写在屏幕的哪块区域。您暂时看不懂也没关系，点击写屏区域右边的范围选择按钮，此时界面蒙上了一层幕布，按住鼠标左键开始勾选，当觉得选择好以后，松开鼠标左键，UiBot会自动选择好写屏区域，例如刚才我们选择的区域，基于1920×1080分辨率，起始点坐标（左上角）为(312,174)，写屏区域的高度为346，宽度为598，单位都是像素。“自适应”属性是一个布尔值，表明写屏区域是否随着分辨率的变化而自动适应，True表示“写屏区域”属性会随着分辨率的变化而自动适应，False表示严格按照“写屏区域”属性的值进行写屏；输出到”属性填写写屏对象objWindow，后续操作将会用到这个写屏对象。

图 68: 设置写屏区域

创建写屏对象后，就可以利用这个写屏对象绘制文字了，选择并插入一条“绘制文字”命令，该命令将往屏幕上写一段文字。这条命令有四个属性：“写屏窗口对象”属性填写写屏对象objWindow；“显示内容”属性填写所要显示的文字内容；“文字大小”填写文字的字号；“文字颜色”一栏中，用[255,0,0]代表红色，[0,0,255]代表蓝色，[255,255,0]代表黄色等，关于RGB颜色的相关知识，请参见相关教程。

图 69: 绘制文字

有时候我们需要多次写屏，写屏对象objWindow可以反复使用。在上一条“绘制文字”命令后面，再加入一条“绘制文字”命令，两条“绘制文字”命令只有“显示内容”属性不同。运行后发现上一条“绘制文字”命令显示的文本并没有消失。原来，UiBot并不会默认擦除上一条“绘制文字”命令的写屏内容，如果要擦除写屏文字，需要用到“清除文字”命令。“清除文字”命令只有一个“写屏窗口对象”属性，这个属性与“绘制文字”命令的“写屏窗口对象”属性含义相同，填入objWindow即可。

再次运行流程，这次上一条文字的内容倒是擦除了，不过文字显示的时间太短了。原来UiBot的命令都是实时执行的，如果需要文字在屏幕中停留一段时间，在“绘制文字”命令和“清除文字”命令中间插入一条“延时”命令即可。

最后切记，在所有的写屏操作完成后，一定要添加一条“关闭窗口”命令，这条命令和“创建写屏对象”命令一一对应，将“创建写屏对象”命令创建的objWindow对象释放。

3.3 锁屏与解锁

当我们使用UiBot，在普通的桌面计算机上运行流程（在有的资料中，把这种场合也称为“桌面自动化”或者“有人值守自动化”）的过程中，经常遇到这样的场合：临时有事需要暂时离开办公桌，又不想让别人乱动这台计算机，破坏流程的执行，或者不想让别人看见当前的流程。我们希望能够锁住屏幕，比如在Windows系统中按“Win+L”键。但是采用这种锁屏方法后，UiBot的流程还能正常运行吗？下面用一个具体的例子来实验一下，这个例子只有一个流程块，执行一条“屏幕OCR”命令，获得桌面上“回收站”图标的文字并显示出来，为了预留按“Win+L”键的时间，我们再加入一条“延时”命令，延时5秒钟。

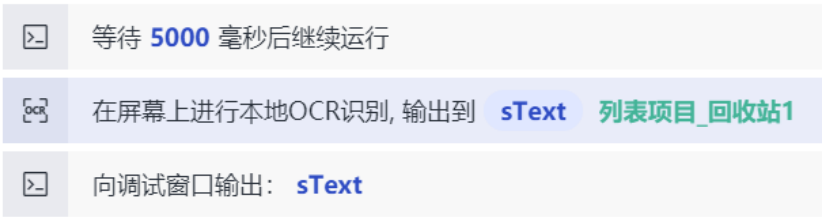

图 70: 一个具体流程：识别桌面上的回收站图标

先正常执行流程，能够正确识别出文字“回收站”：

```
[INFO] 选中的内容 第3行："回收站"
[INFO] 选中的内容 运行已结束
```

图 71: 正常执行后，正确识别出文字

再次运行流程，马上按“Win+L”键锁住屏幕，然后等待流程执行完成，输入解锁密码，此时UiBot并未识别出文字“回收站”：

```
[INFO] 选中的内容 第3行：""
[INFO] 选中的内容 运行已结束
```

图 72: 锁屏执行后，无法正确识别出文字

这是因为“回收站”的图标和文字实际上都是Windows“画”在桌面上的， UiBot的OCR命令需要获得Windows“画”的那些内容， 才能识别。 但Windows也会“偷懒”， 当屏幕锁定的时候， 它就不会再“画”这些内容了。所以UiBot的OCR命令自然也不会获得预想的效果。

那有没有别的办法呢？UiBot从5.0版本开始，提供了一种名叫“屏幕锁屏”的命令，可以锁住屏幕，却不影响流程的正常运行，我们来看具体的用法：

在UiBot的命令列表中，找到“系统操作”的“锁屏解锁”分类，选择并插入一条“屏幕锁屏”命令，该命令会将屏幕锁住，但又不影响后续流程的执行。这条命令有三个属性：“输出到”属性返回锁屏命令是否执行成功，这个属性暂时用不到；“用户或账户”属性填写Windows系统的用户名；“密码”属性填写该用户名的密码，如果不想让别人看见输入框和源代码中的密码原文，可以点击密码输入框右边的“切换使用密文或明文”按钮，再次输入时即为密文。

图 73: 锁屏命令的属性

类似的，插入一条“屏幕解锁”命令，即可将屏幕解锁。该命令的三个属性含义同“屏幕锁屏”命令。在“屏幕锁屏”和“屏幕解锁”命令之间，填入原来的命令（OCR识别屏幕命令、输出调试信息命令等）。运行流程，此时屏幕会被自动锁住，且流程仍然在正常执行，执行完成后，屏幕自动解锁，并正确识别出文字“回收站”。

是不是很神奇？通过锁屏和解锁命令，既可以将屏幕锁住，又可以正确运行流程，一举两得，完美！别着急，为了支持这些命令，Windows系统需要预先进行一些设置：

第一、 Windows系统需要支持远程桌面连接， 这个是先决条件， 因为这两条命令实际上是使用了Windows的远程桌面协议 （Remote Desktop Protocol， 简称RDP）。 一般来说， 家庭版或者教育版的Windows系统不支持RDP，而企业版、旗舰版等都支持RDP。

第二、Windows系统需要启用远程桌面连接。启用方法参考如下：

1、右键“此电脑”或“我的电脑”，点击“属性”，在“高级系统设置”里，点击“远程桌面”或者“远程”选项卡。

2、在“远程桌面”或者“远程”选项卡下，选择“允许远程连接到此计算机”。

4 多流程协作

在前面的学习中，我们编写的流程可能包含了若干个流程块，复杂一些的流程可能还包含了多个条件判断，但它们仍然只是一个流程，因为它们只有一个“开始”块。在本章的内容中，我们会学习到如何针对一个任务，编写多个流程（每个流程都有自己的“开始”块），通过多个流程之间的协作来完成这个任务。

在UiBot中，既支持多个流程之间并行地运行（多个流程同时运行），也支持多个流程之间串行地运行（先运行一个，再运行另一个）。前者称之为“**辅助流程**”，后者称之为“**子流程**”，两者各有不同的用途。下文将会逐一讲解。

4.1 辅助流程

如果您使用的是UiBot Creator 5.1.0或更新的版本，您可能会注意到：流程图上有且仅有一个称为“主流程开始”的组件，这是流程的起点，可以把多个流程块用箭头连接到“主流程开始”的后面，让这些流程块依次运行，这就构成了一个流程。如下图所示：

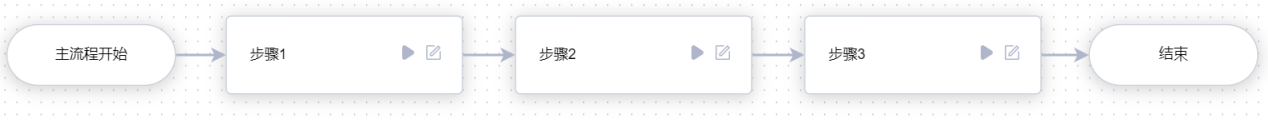

图 74: 主流程示意

顾名思义，既然有“主流程”，自然就会有“副流程”。在UiBot中，我们将其命名为“辅助流程”，而不是“副流程”。之所以这样命名，是因为它的用途通常是帮助主流程完成一些额外的任务，起到“辅助”的作用。具体的用法，后文会举例说明。我们先来看如何创建一个辅助流程。

在流程图左侧的组件面板中，可以看到有一种名为“辅助流程开始”的组件，可以拖动到流程图中来。这个组件的形状很像“主流程开始”，但除了文字上有差异之外，还多了一圈虚线，如下图中红框所示：

图 75: 辅助流程开始

在一个流程图中，可以没有，也可以有一个或多个“辅助流程开始”的组件。我们在流程图中创建了多少个“辅助流程开始”组件，在运行的时候就产生多少个辅助流程。当然，只创建“辅助流程开始”还不够，还需要创建若干个流程块，并且用箭头将其依次连接到“辅助流程开始”的后面，就像创建主流程一样。如下图所示，我们创建了一个主流程和一个辅助流程，它们各自又连接了三个流程块。

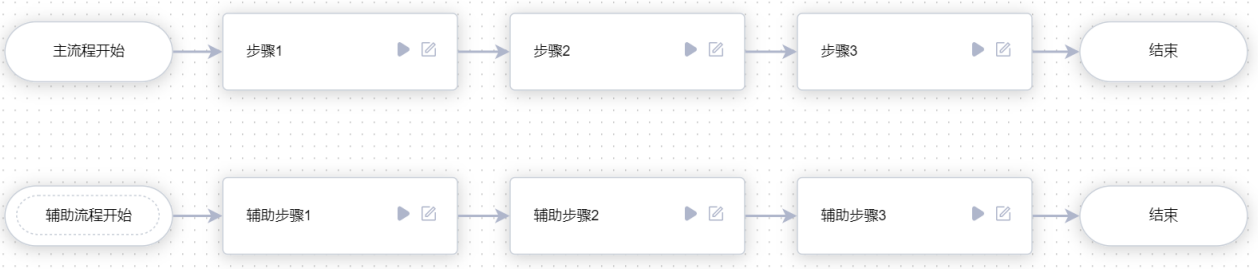

图 76: 一个主流程和一个辅助流程

值得注意的是：在流程图中可能有多个流程块，每个流程块要么隶属于主流程，要么隶属于某个辅助流程，而不能同时隶属于主流程和辅助流程，也不能同时隶属于多个辅助流程。因此，在UiBot Creator的流程图界面中，一个流程块一旦被连接到了“主流程”后面，您就无法再将其连接到“辅助流程”后面了。

辅助流程是如何工作的呢？当流程开始运行的时候，主流程和所有的辅助流程都会同时开始，同时从“主流程开始”和每个“辅助流程开始”的组件处，根据箭头指向，依次运行每个流程块中的内容。您可以把这样的运行过程想象成为田径运动中的接力赛跑。发令枪响起之后，会有多组运动员在多个赛道上同时起

跑，遇到了赛道中的队友，就会把接力棒交给队友，由队友接着往下跑。

图 77: 接力赛跑

正如同图中赛跑的多组运动员都在各自的跑道上，而不会互相冲突一样。主流程和辅助流程所使用的变量也都是各自私有的，不会产生冲突。比如在主流程中有一个命名为a的变量，它的值为1，那么在辅助流程中完全有可能有一个同名的变量，它的值为2。

当主流程中的某个流程块运行完毕后，它的后面没有连接其他流程块，或者连接了“结束”组件，主流程就会结束。辅助流程也是类似的，如果没有后续的流程块或者遇到“结束”组件就会结束。两者的差异在于：如果主流程结束了，会自动通知每个辅助流程，要求它们也结束。而辅助流程结束后，则不会影响到主流程或者其他辅助流程。

有的读者可能会质疑：既然辅助流程和主流程是同时运行的，且辅助流程没有数量限制。那么，如果创建很多个辅助流程，每个辅助流程都去做一个独立的任务，岂不是相当于同时运行了很多个软件机器人，可以让它们在固定的时间内做更多的事情？实际上并非如此，UiBot的辅助流程是采用操作系统的多线程机制实现的，每个辅助流程都是一个独立的线程，如果读者熟悉计算机系统原理，就会知道线程的数量增加并不一定会带来工作效率的提升，甚至线程数量过多，反而会带来效率的下降。如果读者不熟悉线程的基本原理也没关系，只要记住辅助流程的数量不宜过多即可。

实际上，UiBot设计辅助流程机制的初衷，并不是让我们同时运行多个软件机器人，去做不同的任务。因为UiBot经常需要模拟界面操作，如果多个流程都在同一套界面上进行操作，实际上很难协调，让它们能够有条不紊的做不同的操作，就像两个人各拿一个鼠标，去操作同一台计算机一样，稍有不慎就会产生冲突。比如一个人正在文本框里输入，另一个人却点击了某个按钮，导致输入焦点丢失，等等。辅助流程顾名思义，只是主流程的“助手”，帮助主流程做一些杂事儿。

下面的例子可以说明辅助流程的一般用法：

在某个电信运营商的业务系统中，需要不停地处理客户发来的业务请求（如修改手机话费套餐等）。处理每个请求都需要在业务系统中进行一系列复杂的操作，虽然复杂，但由于有特定的规则，所以特别适合用UiBot来自动化操作，只要把操作分为多个步骤，每个步骤都按规则点击特定的界面元素即可。但是，在这个业务系统中，可能会反复弹出“通知”对话框，“通知”对话框是异步产生的，也就是说，在进行任何操作的时候，随时都可能弹出，没有规律可循。而一旦弹出了这个对话框，就不得不点击对话框中的“确认”按钮，把这个对话框关掉，才能继续操作。

如下图所示，当我们正在填写业务系统中的某个表单的时候，系统却弹出了一个无关紧要的对话框。如果是人在操作，很简单，关掉这个对话框就可以继续了。但如果是软件机器人操作，就会变得非常麻烦。因为我们在设计流程的时候，无法预测这个对话框什么时候会弹出，所以不得不在操作表单中的每个界面元素之前，都先检测一下对话框是否弹出。否则，万一哪个步骤中没有检测，而对话框又偏偏弹出来了，就会造成流程运行的错误。

图 78: 弹出对话框示例

如果采用UiBot的辅助流程，这个问题就很好解决了。我们只需要在主流程中照常处理表单，而不必考虑对话框是否弹出。另外再创建一个辅助流程，其作用是随时检测对话框是否弹出，一旦弹出，立即将其关闭，以免影响到主流程。这个辅助流程只有一个流程块，其内容大致如下图所示：

图 79: 检测并关闭对话框的辅助流程

这样一来，主流程和辅助流程各司其职，职责更加明确，流程编写更加简单。辅助流程就像是给主流程“保驾护航”一样，当主流程结束后，辅助流程也会自动结束。

4.2 子流程

当我们在开发一个稍微复杂一些的流程时，为了能够提高效率，确保流程尽快完成，常见的方法是把流程划分为多个步骤，由不同的人去编写不同的步骤，最后再把这些步骤组装在一起。

用UiBot如何做到这一点呢？如果读者有实践经验，可能会把每个步骤用一个流程块来实现。然后再使用UiBot的“导入”功能，把这些流程块逐一导入到同一张流程图中。

这种方式并非不可以，但存在以下的问题：

- 如果流程块的内容有更新，需要每次都在流程图中，把旧的流程块删掉，在重新导入新的流程块，非常麻烦；
- 如果一个步骤比较复杂，用一个流程块描述起来不太方便，必须要用多个流程块才更清晰，这种方式就不支持了；
- 我们在流程图中定义了一个变量，在每个流程块中都可以使用这个变量。但如果两个人之间缺乏协调，互相不知道彼此是如何使用这个变量的，会造成数据的相互覆盖等冲突（比如我们定义了一个流程图变量a，一个流程块将其值设为1，另一个流程块将其值设为2，如果两个流程块不是同一个人编写的，第一个流程块的编写者并不知道这个值已经被修改了，仍然按照1去处理，就会出现问题）。

从UiBot Creator 5.2版本开始，支持一种新的机制：子流程。正确地使用子流程，能够避免前面的问题，更有助于多人协作。下面详细介绍这种机制。

4.2.1 子流程的概念

举例来说，假设我们有一个流程，其中包含一个前置步骤、两个业务步骤、一个后置步骤。我们希望把这两个业务步骤交给其他人来编写，其中每个业务步骤都是一个“子流程”，每个子流程里面又可以包含

多个流程块，以及条件判断等。从我们的视角来看，这个流程图大致是如下的状态（不妨把这个顶级的流程称之为“总流程”，下文将延续这种称呼）：

图 80: 总流程示意

其中，“前置步骤”和“后置步骤”这两个流程块由我们来编写，“子流程1”和“子流程2”这两个步骤由其他同事来编写。细心的读者可能注意到了，“子流程1”和“子流程2”这两个步骤并不是普通的流程块，因为图标两侧各有一条竖线，和流程块的图标不一样。实际上，“子流程1”和“子流程2”分别都是一张完整的流程图，里面还可以包含其他流程块。但在我们的视角中，只是把这两位同事编写的“子流程1”和“子流程2”当作两个单独的组件，并不关心其中的细节。

例如，对于“子流程1”来说，其中可能包含了两个流程块，如下图：

图 81: 子流程1示意

而对于“子流程2”来说，其中可能包含了一个流程块和一个条件判断，如下图：

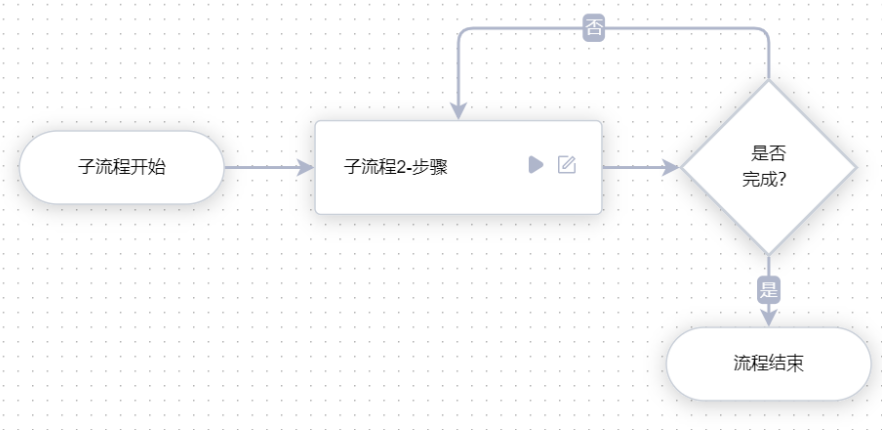

图 82: 子流程2示意

我们作为“总包方”，并不关心子流程1和子流程2的这些细节，只要写好前置步骤和后置步骤，再等两位

同事分别完成子流程1和子流程2，即可运行这个流程。当我们点击了“运行”按钮之后，会先运行前置步骤，然后运行子流程1里面包含的流程块，当遇到子流程1中的“结束”块时，并不会结束整个流程，只是跳出子流程1，然后马上运行接下来的子流程2。类似地，当遇到子流程2中的“结束”块时，也不会结束，而是跳回到我们的流程图中，接着运行后续的“后置步骤”。直到在我们的流程图中遇到了“结束”块，才会真正结束。

也就是说，实际运行的路径是：

开始 ⇒ 前置步骤 ⇒ 子流程1-步骤1 ⇒ 子流程1-步骤2 ⇒ 子流程2-步骤 ⇒ 是否完成（在子流程2中，假设这里的判断条件为“是”）⇒ 后置步骤 ⇒ 结束

值得注意的是，对于总流程、子流程1、子流程2，它们的变量都是彼此隔离的。也就是说，如果在总流程和子流程里面都定义一个变量a，无论a是流程图变量，还是流程块变量，它们在总流程或者子流程里面各有各的取值，不会互相影响。这样的话，我们在编写流程或子流程的时候，即使是由不同的人完成，也不用担心大家凑巧给变量起了相同的名字而导致冲突。

创建这样一个包含子流程的流程其实非常容易，下面介绍其操作方法。

4.2.2 创建子流程

使用UiBot，您可以先创建子流程（包含其中的流程块），然后再在总流程中去引用子流程。也可以在总流程编写的过程中，创建一个子流程并引用它。而且，子流程中还可以再创建其他子流程。创建顺序没有限定。如下图中，总流程包含了3个子流程，其中有的子流程又包含了其他子流程（不妨称之为“孙流程”）。那么可以先逐一创建孙流程，再创建子流程，最后创建总流程；也可以先创建总流程，再逐一创建子流程，再在每个子流程中逐一创建孙流程。

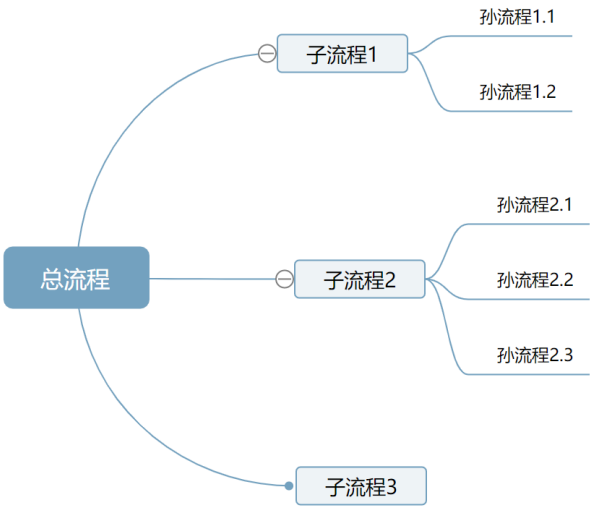

图 83: 多级子流程示意

我们先考虑最简单的情况：在一个总流程里引用一个子流程。掌握了这种操作以后，再考虑多级子流程，也就可以举一反三了。

假设我们当前正在编写总流程，并且需要引用一个子流程。在左边的组件面板中，可以看到一个如图所示的图标，其下方的文字表明，用这个组件可以引入“子流程”。如下图：

图 84: “子流程”组件图标

把这个组件拖入流程图，UiBot Creator会立即弹出一个对话框，询问我们是希望引用已有的子流程，还是创建一个子流程并引用它，如下图。如果之前已经建好了子流程，显然这里应该选择“打开”，来选择已有的子流程；否则，可以选择“空白创建⇒流程”或者“从模板创建⇒企业级流程模板”来创建一个子流程。

图 85: 创建或打开子流程

值得注意的是：在UiBot Creator中，会使用一个Windows文件夹来放置一个流程，这个流程包含的所有文件都在这个文件夹下面。所以，如果是选择已有的子流程，UiBot Creator会立即弹出一个“选择文件夹”的对话框，其含义是确定被引入的子流程在哪个文件夹下面。如果是新建子流程，同样是在“位置”一栏指定流程所在路径，并点击“创建”即可。无论是选择已有的子流程还是创建子流程，完成操作后，流

程图中就多了一个“子流程”的图标。在流程沿着箭头运行的过程中，每当遇到这个图标，即开始运行这个子流程，直到子流程运行完毕，再回到这一张流程图。

在流程图中加入一个子流程之后，UiBot并没有把子流程中的所有内容都复制到我们的流程图中，而是采取了“引用”的方式，只记录子流程的路径信息。也就是说，创建子流程之后，如果子流程又发生了修改，我们不需要做任何额外动作，再次运行的时候，子流程已经是修改之后的内容。

您可能会注意到，在流程图中加入一个子流程之后，在表示子流程的方块上，也会有一个显示为纸和笔的按钮（如下图中红圈中所示）。点击这个按钮后，UiBot Creator会自动打开子流程的流程图，让我们有机会对子流程进行修改。

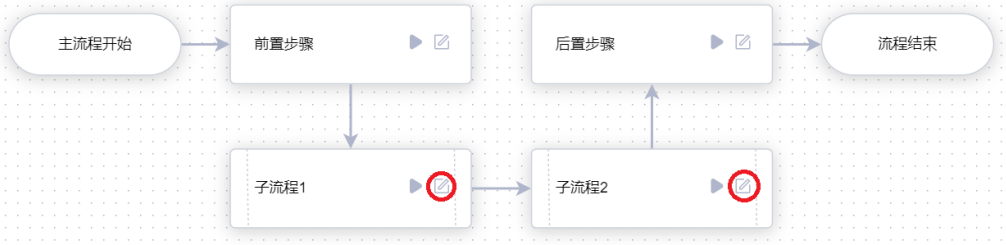

图 86: 修改子流程的按钮

有的读者可能会担心：子流程里面又有孙流程，如果嵌套层数太多，“子子孙孙无穷匮也”，会不会忘记了子流程的层级结构。不必担心，UiBot Creator都帮您记得清清楚楚。无论是在编辑流程图还是流程块的时候，左侧都有一个名为“流程”的选项卡（可以通过拖动，放置到其他位置，但默认位置在左侧）。切换到这个选项卡，里面显示了一个树形结构，从总流程到子流程，再到每个子流程里面的流程块，都清清楚楚。这个选项卡有两个主要作用：

1. 显示整个流程的结构，并且以加粗的字体来表示当前正在编辑的流程图或流程块在整个流程里面的位置；
2. 双击其中的任何一个流程图或者流程块，可以马上开始编辑。方便您在不同的子流程之间快速跳转。

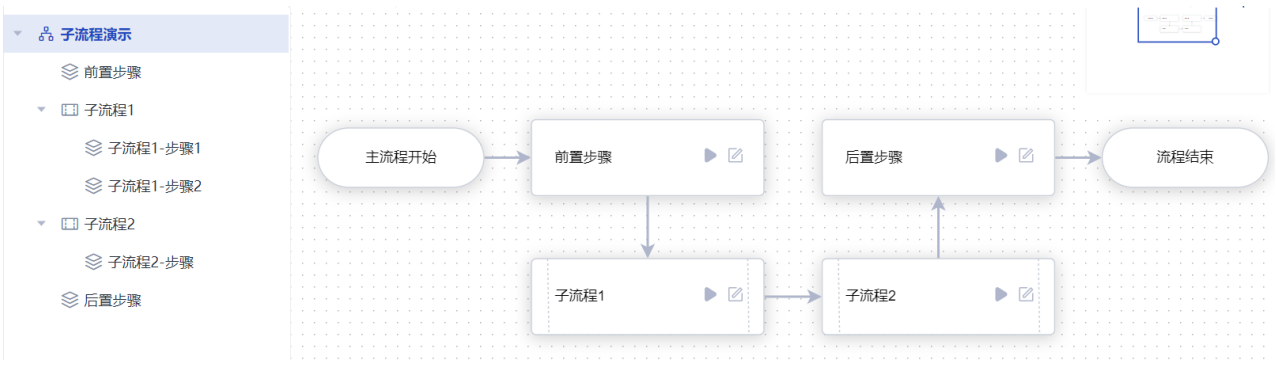

图 87: “流程”选项卡的作用

由于总流程和子流程各自使用了不同的Windows文件夹来保存，我们可以把流程运行过程中需要的文件

（例如“查找图像”命令中用到的图像文件）放在各自的文件夹中，彼此之间不会干扰。按照我们之前学习的内容，在总流程的任意一个流程块中使用`@res""`，得到的是总流程所在文件夹下的`res`文件夹。而在子流程中，则会得到子流程所在文件夹下的`res`文件夹。

当我们在UiBot Creator中编写好了总流程和子流程，并希望在UiBot Worker中使用时，需要使用UiBot Creator的企业版，并将流程打包成一个独立的、扩展名为`.bot`的文件。在打包的时候，虽然总流程和子流程放置在不同的文件夹下，UiBot Creator会自动遍历所需的子流程，以及子流程中的子流程（即“孙流程”）所在的文件夹，把它们的内容全部都打包到同一个`.bot`文件里。也就是说，您在打包的时候并不需要关心总流程和子流程各自放置在哪里，也不用担心他们在运行中用到的文件会有冲突，一切都是自动处理的。

4.2.3 子流程的输入和输出

当我们在总流程中引入了一个子流程的时候，根据前文所述，它们的变量是相互隔离的。即使定义了相同命名的变量，也各有各的取值，不会冲突。这样虽然避免了冲突的问题，但是，如果总流程和子流程需要协作，不可避免的还是要把数据在总流程和子流程之间进行传递。比如，子流程是某个同事负责编写的，其作用是根据我的手机号码，查询手机号码的归属地。那么，在和这位同事协作的过程中，至少有三个需求要满足：

- 负责编写子流程的同事，能够很容易地使用自己的手机号码，对子流程的有效性进行测试；
- 在我们使用子流程的时候，能够很容易地把要查询的手机号码发给子流程；
- 子流程查到手机号码归属地之后，能够很容易地把结果发回给我们。

从UiBot Creator 5.2版本开始，对“流程图变量”的概念进行了扩充，以支持以上的需求。

在UiBot Creator中，打开任何一个流程图，在右侧会有一个选项卡，名为“变量”。点击这个选项卡，可以看到，在弹出的面板里面有三类变量。第一类叫“流程图”，也就是说，这里定义的是“流程图变量”；第二类叫“流程输入”，说明这里定义的是“流程输入变量”，UiBot Creator通常会自动新建一个名为`g_input`的流程输入变量；第三类叫“流程输出”，说明这里定义的是“流程输出变量”，类似地，UiBot Creator会自动新建一个名为`g_output`的流程输出变量。

我们在前文中已经学过，在这里定义的“流程图变量”，在流程图里的各个流程块中都可以共享使用。实际上，“流程输入变量”和“流程输出变量”也都是特殊的流程图变量，也可以在各个流程块中共享使用。当然，他们还有更进一步的用途。

- 对于“流程输入变量”，当这个流程图作为子流程的时候，这个变量可以接收上一级流程（简称“父流程”）传来的值；
- 对于“流程输出变量”，当这个流程图作为子流程的时候，这个变量可以把值传给父流程；
- 对于“流程图变量”，只能在流程图及其流程块中使用，对父流程不可见；
- 对于总流程，其“流程输入变量”和“流程输出变量”还可以在使用UiBot Commander创建任务的时候，作为任务的输入和输出。除此之外，和“流程图变量”并无其他区别。

图 88: 定义流程图变量

除了自动定义好的“流程输入变量”`g_input`和“流程输出变量”`g_output`之外，您可以通过点击下面的“加号”图标去增加，或者点击右边的“垃圾桶”图标去删除这些变量。

细心的读者可能还会发现：对于“流程图变量”和“流程输出变量”来说，在定义的时候都没有“类型”这一栏，而“流程输入变量”则有这一栏。其实，UiBot中的变量是动态类型的，也就是说，一个变量可能先是字符串，后面又变成了数组、数值等类型，所以“流程图变量”和“流程输出变量”中到底存放了什么类型的数据，在流程运行的时候才知道，无法事先预估。而对于“流程输入变量”来说，其实UiBot也并未**强制**限定其数据类型，这一栏中填写的“类型”只是告知其父流程或者UiBot Commander，当需要输入的时候，**建议**输入什么类型的值。

对于上文所述的查询手机号码归属地的例子来说，显然，我们需要在子流程中定义两个变量（如图）：

- 一个是“流程输入变量”，作用是从父流程中获得要查询的手机号码。由于UiBot可以用中文命名变量名，不妨将其命名为 *手机号*。设定其类型为字符串，还可以为其设一个初始值，比如`"13074835678"`；
- 另一个是“流程输出变量”，作用是向父流程传递查到的手机号码归属地。不妨将其命名为 *归属地*。

图 89: 子流程中的变量

这样一来，对于编写这个子流程的同事来说，只需要读取 *手机号* 这个变量的值，即可得到要查询的手机号码。并且，在流程编写和单元测试阶段，还没有作为子流程接入的时候，这个变量的值就是我们设置的初始值，以便调试。当查到归属地信息之后，只需要将其置入 *归属地* 变量，等待被父流程使用即可。

在我们编写总流程（同时也是上文中子流程的父流程）的时候，类似的，只需要定义一个“流程图变量”，用于接收子流程传回的归属地信息，变量名不妨定位为 *归属地结果*。我们还可以为其设置一个初始值，比如"*北京市*"。这样的话，在子流程还没有接入之前，也可以先用这个初始值进行后续步骤的测试和调试。如下图。

图 90: 总流程（父流程）中的变量

最后，假设子流程和总流程都已经编写完毕，开始集成测试了。只需要按前文所述的步骤，把子流程引入到总流程中来，并且，在流程图中选中代表子流程的方块，注意，右侧有一个名为“属性”的标签页，打开这个标签页，在面板里面，除了显示子流程的名称等信息之外，还会显示子流程中定义的所有“流程输入变量”和“流程输出变量”。

图 91: 子流程的“基本信息”面板及其设置方法

如上所示的设置，其含义是：在子流程运行之前，把字符串"13912341234"（注意：上图中是采用普通模式输入，所以不需要输入代表字符串的双引号。如果是专业模式，就需要引号了）作为要查询的手机号，传入到子流程的 手机号 这个“流程输入变量”中；在子流程运行之后，把子流程的 归属地 这个“流程输出变量”的值置入总流程的 归属地结果 变量中。UiBot Creator中采用了标有左右方向的箭头表示数据的传递方向，留意此箭头及其方向，将有助于您理解数据是如何在总流程和子流程之间传递的。

5 人工智能功能

传统RPA实现的是基于固定规则的流程自动化。而在企业实际业务场景中，还有大量不基于固定规则的业务流程，需要人的认知和判断。例如：合同是企业业务场景中常见的文档之一，在业务流程中，经常需要阅读电子合同，识别大量文字并从中抽取出甲乙方名称、合同日期等关键信息。对于这些场景，我们可以使用人工智能（AI，下文将统一使用这一简称）技术，让机器人实现对合同等信息的“认知”，我们称之为“认知自动化”。因此，RPA和AI一起，将流程自动化与认知自动化结合起来，让企业中更多复杂的、高价值的业务场景实现自动化。

对软件机器人来说，如果说AI是它的大脑，认知能力是它的眼睛、嘴、耳朵，RPA是它的双手。结合了AI能力，RPA从只能帮助基于规则的、机械性、重复性的任务实现自动化，拓展到了更丰富的业务场景，将物理世界与数字世界有效连接，满足实际业务中更灵活、多元的自动化需求。而企业采用具备丰富AI能力的RPA平台，也可以快速、经济、灵活地将AI技术应用到业务中。

UiBot中集成了大量的AI能力，并且还会在未来持续增强这部分能力。本章将为大家介绍如何在UiBot中使用AI能力。

5.1 UiBot Mage

UiBot Mage（下文简称为Mage）是专门为UiBot打造的AI能力平台，可提供执行流程自动化所需的各种AI能力。

Mage的中文含义为“具有魔力的人或经过长时间学习具有很多知识的人”，以此命名产品，并取中文名为“魔法师”，可见UiBot希望能用学到的知识赋能RPA机器人。

从UiBot Creator 5.3.0版本开始，在用UiBot编写RPA流程时，对于社区版的UiBot Creator，在联网的前提下，用户登录后，可以使用UiBot Mage提供的各类AI能力，进而将图片、文档中的非结构化信息转变成结构化数据；对于企业客户，使用企业版UiBot Creator时，即使不连接互联网也可以享受UiBot Mage的AI能力，因为企业版可以提供UiBot Mage的私有化部署，并提供标准化的调用接口，灵活适应业务需求。

UiBot Mage的产品特点包括：

- 内置OCR、NLP等多种适合RPA机器人的AI能力。
- 提供预训练的模型，无需AI经验，开箱即用。
- 在预训练之外，也提供定制化的模型，仅需少量配置或训练，即可让AI具有较强的泛化能力。
- 与UiBot Creator无缝衔接，方便在流程中以低代码的形式使用AI模型。
- 能够识别多种类型的文档，适用于财务报销、合同处理、银行开户等不同的业务场景。

目前，UiBot Mage中包含的AI功能如下所示：

定制化程度	数据类型	能力	用途
通用AI能力	图片理解	通用多票据识别	识别普通发票、专用发票、电子发票、销货清单、卷式发票、出租车票、火车票、动车票、飞机行程单、定额发票、购车发票等全票种发票，并返回核心字段值。
		通用卡证识别	识别银行卡、身份证、社保卡、驾驶证、行驶证、户口本、护照、结婚证、房产证、不动产证、营业执照、开户许可证、组织机构代码证、车辆合格证、车辆登记证、基本存款账户信息，并返回核心字段值。
		验证码识别	识别由数字和字母组成的验证码。
		通用表格识别	识别图片中的表外文字和表内文字，并按照单元格的排列顺序，输出表格内容。
		通用文字识别	识别图片中所有文字。
	文本理解	标准地址	提取地址中的省、市、区、街道信息并返回。
	语音能力	语音合成	把一段文本合成为语音，支持7种不同音色。
定制化AI能力	图片理解	自定义模版	上传一组版面样式相对固定的图片文件，通过配置规则的方式，依赖位置关系抽取到业务需要的字段值。
	文本理解	文本分类	创建分类并上传每个分类的相似说法，自动生成AI模型。输入新的文本可以返回匹配到的分类和置信度。
		信息抽取	上传一组文本内容相对固定的文本文件，通过配置规则或训练模型的方式，依赖上下文语义信息抽取到业务需要的字段值。

图 92: UiBot Mage 功能简介

由此可见，UiBot Mage中的AI功能非常丰富，而且还在不断的扩展过程当中。功能虽多，但大致可以分为两类：一类称为“通用AI能力”，是指您基本上不需要在UiBot Mage中进行太多的设置，开箱即用的AI功能，其中比较常用的是对图片的各种识别，如标准化的票据（发票、出租车票等）、标准化的卡证（身份证、营业执照等）。另一类称为“定制化AI能力”，是指您在使用这些AI能力之前，还需要花费一番功夫，在UiBot Mage中先做一定的配置或训练才可以，用起来稍微麻烦一点儿，但能处理更加广泛的数据。

考虑到内容侧重，本章暂时只介绍“通用AI能力”中的图片识别功能。其他功能将在后续篇章中讲述，也可参考UiBot Mage的在线帮助。

5.1.1 UiBot Creator中的Mage AI命令

在UiBot Creator中，已经把UiBot Mage的很多AI功能都包装成了相应的命令，这些命令被放在一个叫做“Mage AI”的分类里面。其中又包含了“信息抽取”、“通用卡证识别”等二级分类，每个二级分类再展开以后，下面还有很多命令。

图 93: Mage AI命令一览

比如，我们把“通用卡证识别”这个二级分类展开，可以看到下面还有“屏幕卡证识别”、“图像卡证识别”、“获取卡证类型”等五条命令，这五条命令有什么区别，各用在什么场合呢？

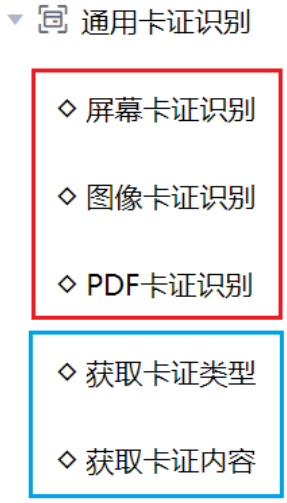

图 94: “通用卡证识别”下面的命令

其实，无论是“通用卡证识别”也好，还是“通用文字识别”或者“通用票据识别”也好，这种“识别”类的Mage AI命令，最主要的使用过程都是分两个步骤进行的：

1. 从指定的图像中，**识别**出所有结果
2. 从识别的结果中，**获取**出所需信息

上图中红框表示的命令，实际上是在做第1步，因为这些命令的最后两个字都是“识别”；上面蓝框中的命令，实际上是在做第2步，因为它们的前两个字都是“获取”。所以，如果我们要用Mage AI来识别一张身份证的话，需要先根据数据源的不同，从红框里面选择一条命令，再根据要获取的具体信息，从蓝框里面选择一条命令。

红框里面的三条命令，主要区别在于图像来源的不同，其中：

- 屏幕识别：图像来源于屏幕上某个窗口，或窗口的某个区域。
- 图像识别：图像来源于本地硬盘上的某个图像文件。
- PDF识别：图像来源于本地硬盘上的某个PDF文件，还可以通过识别全部页或指定页码范围来划分识别范围。

除了数据源不同之外，这三条命令其实没有其他区别，对于同样的图像，识别出的结果也是相同的。但它们的识别结果通常很难直接使用，所以才需要用蓝框里面的两条命令来进一步的获取其中的关键信息。其中：

- 获取卡证类型：由于“通用卡证识别”可以识别很多种卡证，包括身份证、驾驶证、房产证、营业执照等。这条命令可以自动判断被识别的图像属于哪一种卡证。
- 获取卡证内容：对于特定的卡证，这条命令可以提取其中的各个字段。比如身份证，可以提取姓名、性别、民族、身份证号码等字段。

我们来实际测试一下。首先，找一张身份证的图像文件。在本文中，为了不泄漏个人隐私，我们在互联网上找到了下面这张虚构的身份证图像，仅供参考：

图 95: 虚构的身份证图像

这张身份证图像显然是网友们恶搞出来的，形状格式并不规范，而且在照片中还倾斜了一定的角度。但是，UiBot Mage仍然能够准确地获得其中的信息。

假设身份证图像保存在 `C:\temp\id_card.jpg` 文件中。新建一个流程，拖入一条“图像卡证识别”命令，这条命令有两个属性是我们必须填写的，如下图所示：

图 96: 属性示例

对于“识别图片”属性，选择 `C:\temp\id_card.jpg` 文件即可；对于“Mage配置”属性，千万不要手动填写，点击属性右边的按钮（上图红框位置），弹出如下所示的对话框。

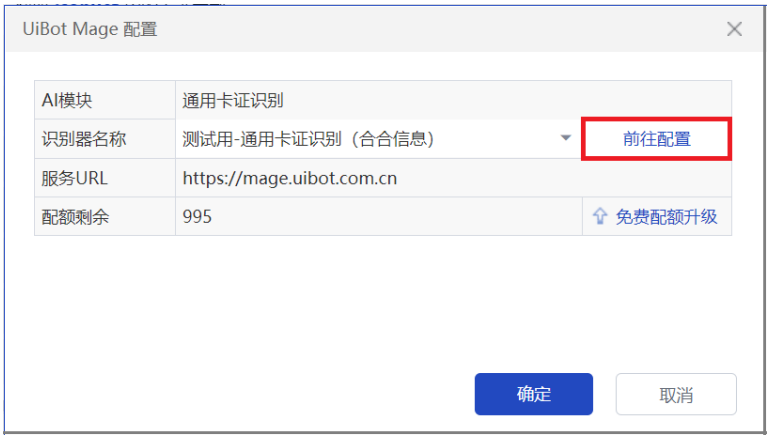

图 97: 身份证识别的识别器配置

对于社区版的UiBot来说，在这个对话框中，唯一需要设置的就是“识别器名称”这个下拉框。如果下拉框中没有内容，点击右边的“前往配置”（上图红框位置），会自动打开浏览器，并跳转到UiBot Mage的主页。可以在UiBot Mage中新建一个通用卡证识别模型，并选择后端的AI引擎，其中标有“原生”的引擎是来也科技自行研发的AI能力，标有“第三方”的引擎是其他AI厂商为来也科技专供的AI能力。不同的AI引擎在不同的场景下各有优势，可以根据实际情况，选择效果较好的AI引擎。

在UiBot Mage上建好了识别模型之后，即可在UiBot Creator中选择到这一模型，并且，之前空缺的“Mage配置”属性也会被自动填写上，我们无需关心它填写的内容。

接下来，我们再依次拖入“获取卡证类型”、“获取卡证内容”命令，获取识别结果中的卡证类型、姓名、

出生日期、地址等信息。注意：这些命令都有一个共同的属性叫“卡证识别结果”，把“图像卡证识别”命令的输出填写在这里即可。每次获取一项信息，可以拖入一条“输出调试信息”的命令将其显示出来。

按照上述步骤，最终形成的流程大致如下所示：

图 98: 通用卡证识别示例

运行后，可以得到如下的结果：

输出

```
[2021-12-21 20:38:58] [INFO] 流程 流程块4.task 开始运行
[2021-12-21 20:39:00] [INFO] 流程块4.task 第4行："id_card"
[2021-12-21 20:39:00] [INFO] 流程块4.task 第6行："奥巴马"
[2021-12-21 20:39:00] [INFO] 流程块4.task 第8行："1961年8月4日"
[2021-12-21 20:39:00] [INFO] 流程块4.task 第10行："华盛顿特区宜宾法尼亚大道1600号白宫"
[2021-12-21 20:39:00] [INFO] 流程块4.task 运行已结束
```

图 99: 通用卡证识别结果示例

值得说明的是，Mage AI类的命令在运行的时候，都需要连接UiBot Mage的服务器。如果您使用的是社区版的UiBot，我们已经在互联网上建好了服务器，直接使用即可。如果是企业版的UiBot，既可以使用互联网上的服务器，也可以自己部署并使用自己的UiBot Mage服务器。如果使用互联网上的服务器，我们限定了每个月的免费使用次数，每运行一次“识别”类的Mage AI命令，就会自动扣除一次。每月初再自动补齐次数的额度。如果免费的次数不够，还可以付费购买更多的使用次数。

当我们想要提取的字段较多，逐一编辑“获取卡证内容”的命令和属性的过程比较繁琐。这个时候，我们可以使用Mage AI识别向导，会更加方便快捷。

5.1.2 Mage AI识别向导

上文描述了如何在UiBot Creator中使用Mage AI功能，来识别一张身份证中的信息。可以看到，当要获取的字段比较多的时候，操作略显繁琐。使用Mage AI还可以识别发票等票据信息，如果按照上述方法去操作，还会更加繁琐。因为UiBot Mage在做卡证识别的时候，每次只识别一张卡证（比如一张身份证）。而在做票据识别的时候，由于很多财务部门都规定要把多张不同的票据贴在同一张纸上去报销，比如把增值税发票、火车票、出租车票都贴在一起，等等。UiBot Mage也会一次性地把这些票据全都识别出来。这样一来，在编写流程的时候，就会有些麻烦了，因为识别完以后，还要判断总共识别出了多少张票据，每张分别是什么类型的票据，不同类型的票据还会有不同的字段，比如火车票上会有出发地、目的地等，而增值税发票没有这些信息。对于UiBot老手来说，编写这样一段流程不算太难，但新手可能就会有些犯难了。

为此，在UiBot Creator中，还提供了Mage AI的向导功能，可以通过图形化界面，快速引导您配置图像识别器，设置提取类型和字段，并自动生成相关的命令框架，过程流畅，简单易用。

打开UiBot Creator，在编写任何一个流程块的时候，工具栏上都可以找到标有“Mage AI”的图标，如下图红框所示。

图 100: Mage AI识别向导在工具栏上的位置

点击这个按钮，即可弹出Mage AI识别向导的窗口。可以看到，这个向导包含了“配置识别器、选择图像来源、提出类型和字段”三个步骤。

使用这个向导，可以自动生成一系列命令，大大简化我们的操作。比如，下面的这张图像中，既有发票，又有出租车票，我们希望一次把这些票据中的关键字段提取出来。发票的方向即使是倒置的也没有关系，UiBot Mage会自动识别。

图 101: 多票据图像示例

只要按照Mage AI识别向导的三个步骤，逐一填写相关信息即可。如下所述：

步骤一：打开Mage AI识别向导，首先进行识别器的配置，即选择AI模块、选择AI能力及其识别器。这和前文中配置Mage识别器的操作基本类似，只需依次选择所需的功能和识别器即可。

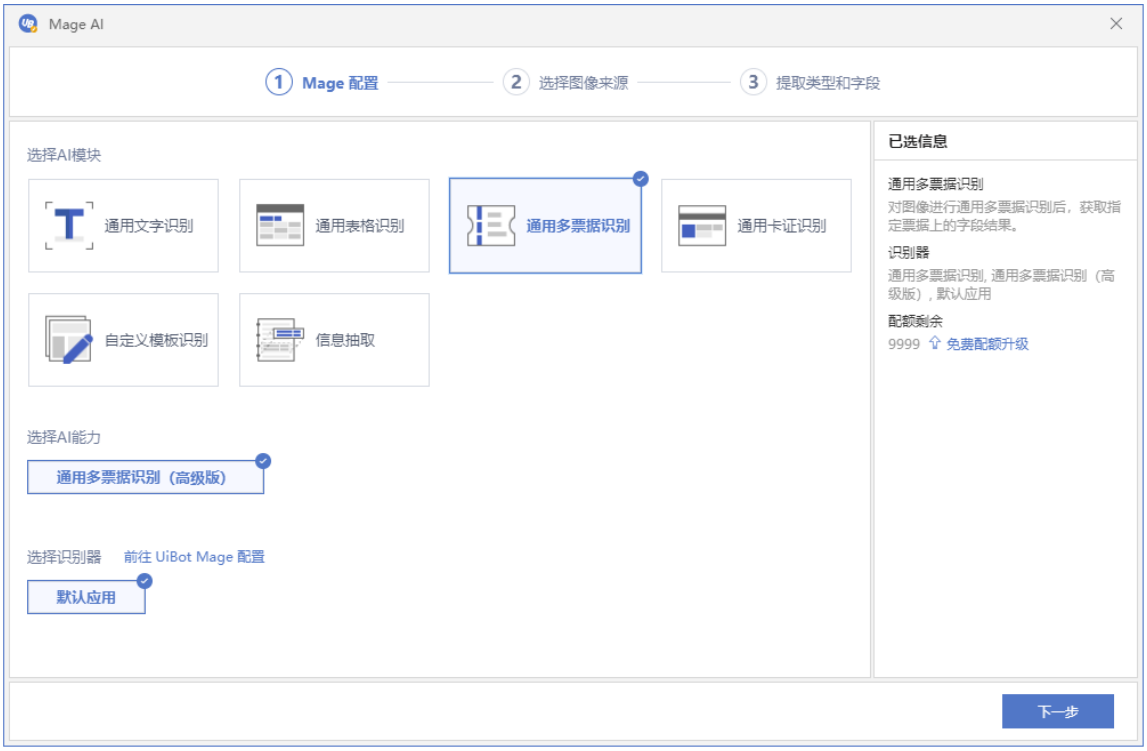

图 102: 配置识别器

步骤二：选择图像来源。可以采取“选择目标”的方式，使用鼠标从电脑屏幕中选择/截取一个识别区域；或者采取“选择图像”，选择或拖拽一个本地的图像文件；或者“选择PDF”，选择一个PDF文件并指定页面范围。

我们以选择本地图像为例，直接在本地硬盘上选取图像文件的路径即可。选择完成后，对话框中会自动把图像显示出来，以方便预览。

图 103: 选择图像来源

步骤三：选择提取类型和提取字段。支持同时配置多种票据及其提取字段，配置之后，可以在右侧“已选信息”中确认自己的选择。

图 104: 提取类型和字段示例

上述三个步骤都完成后，点击“完成”，UiBot Creator会自动生成一系列的命令，如下图所示。

图 105: 通用多票据识别示例

可以看到，对于上述场景，UiBot Creator自动生成了17行命令，节省了我们的工作量。但这些命令仍然只是一个框架，还需要您再继续往里面填充其他命令，才能满足业务要求。比如，您可能需要把识别出的每一张增值税发票的信息填写到一个Excel文件里，把识别出的每一张出租车票填写到另外一个Excel文件里。识别后的效果如下图所示：

图 106: 运行结果示例

那么，您可能需要在流程中恰当的位置，插入打开和关闭Excel文件的命令，并在恰当的位置，插入写入Excel文件的命令。基于自动生成的框架，插入这些命令应该对您来说已经没有难度了，本章不再列出示例流程，请自行练习。通过练习，不难发现，上述识别发票并把不同类型的发票自动录入Excel文件的流程，只需要不到10分钟就可以开发完成，在开发过程中，大多数时候也只需要用鼠标点选，连敲键盘的场景都很少。可见UiBot Creator中“Mage AI识别向导”的便利性。

5.2 本地OCR

OCR的全称是“光学字符识别”，这是一项历史悠久的技术，早在上个世纪，OCR就可以从纸质的书本中扫描并获得其中的文字内容。如今，OCR的技术也在不断演进，已经融入了流行的深度学习等技术，识别率不断提高。我们现在用OCR去识别屏幕上的文字，由于这些文字不像纸质书本一样存在印刷模糊、光线不好等问题，所以识别率是非常高的。

其实，前文提到的UiBot Mage中就包含了OCR的功能。但有的场合并不适合使用UiBot Mage，例如下面的场景：

我们在前面的内容中提到，有些情况是无法获取界面元素的。此时，使用“图像”类命令，可以找到准确的操作位置。但还不能像有目标的命令那样，把界面元素中的内容读出来。

比如著名的游戏平台Steam，其界面使用了DirectUI技术绘制，我们无法获得其中的任何文字（虽然这些内容用肉眼很容易看到），如图所示。

图 107: 很难直接获取Steam界面中的文字

使用UiBot Mage，固然可以得到其中的文字，但未免“高射炮打蚊子”。而且UiBot Mage的AI能力必须连接互联网才能使用，免费版也有配额限制。此时，就需要祭出UiBot的“本地OCR”命令了。

“本地OCR”具体包含了以下的OCR命令：

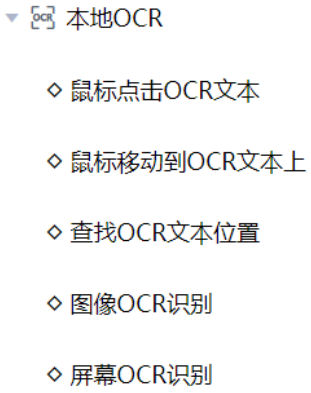

图 108: UiBot的本地OCR命令

顾名思义，这些命令都是不需要连接互联网的，直接在您运行UiBot的计算机上即可执行。

我们先试一下“屏幕OCR识别”命令。双击或拖动插入一条“屏幕OCR”命令，点击命令上的“查找目标”按钮（此时UiBot Creator的窗口会暂时隐藏）；把鼠标移动到Steam的登录窗口上，此窗口会被红框蓝底的遮罩遮住；此时拖动鼠标，划出一个要进行文字识别的区域，这个区域会用紫色框表示。如图所示。

图 109: 选择OCR目标

当然，您也可以不划出要识别的区域，直接在窗口上点击鼠标左键，代表识别整个窗口。这样的一条命令，会在运行的时候，自动找到Steam的登录窗口，并在指定的区域（相对于窗口的位置）截图，然后识别截图里面的文字，最后把识别到的文字输出到指定的变量中。

根据前文学到的经验，可以直接点击命令右侧的三角形，运行单条命令，会在运行完成后，自动输出结果。可以看到，只要Steam的登录窗口存在，且窗口大小没有发生变化，就能识别出我们所划的区域中的文字“账户名称”。

上文演示了“屏幕OCR识别”。此外还有“图像OCR识别”命令，和“屏幕OCR识别”类似，只不过前者需要提供一个图像文件，UiBot会在流程运行到这一条命令的时候，不考虑屏幕图像内容，直接采用指定的图像文件去进行识别。

另外，还有“鼠标点击OCR文本”、“鼠标移动到OCR文本上”、“查找OCR文本位置”等命令，他们类似于“图像”类命令中的“点击图像”、“鼠标移动到图像上”，“查找图像”命令，只不过不需要传入图像了，只需要在属性中标明要找的文字即可。UiBot会在运行的时候，自动在屏幕上找到指定的文本，并根据文本的位置，进行鼠标点击或移动等操作。

5.3 百度OCR

市面上很多云厂商都提供了在线的OCR服务，其中，百度OCR的口碑相对较好，很多用户都自行购买了百度OCR的服务。在这种情况下，UiBot也提供了百度OCR的相关命令，可以方便快捷地使用百度OCR的功能。而且，百度OCR也对发票、身份证、火车票等票据、卡证的图像进行了特别优化，能较为准确的识别其中的关键内容，其识别效果和UiBot Mage相比各有千秋。用户可以根据实测效果和实际情况选用。

为了能够正常接入百度云的OCR，首先需要满足以下三个要求：

- 要能够接入互联网。百度云是基于互联网的云服务，而不是本地运行的软件，个人使用的话，必须接入互联网。如果是企业用途，不能接入互联网，可能需要和百度云进行商务谈判，购买其离线服务。
- 可能需要向百度付费。百度OCR服务是收费的，但提供了每天若干次（通用文字识别每天5000次，证照等识别每天500次）的免费额度。个人使用的话，免费额度也基本够用了。当然，百度可能会随时修改免费额度和收费价格等政策，我们无法预估您需要向百度付多少费用。
- 由于百度云是收费的，不可能UiBot的用户都共用一个账号。所以每个用户要申请自己的百度云账号，以及百度OCR服务的账号（一般称为Access Key和Secret Key），申请方法很简单，请点击这里查看我们的在线教程。

UiBot中包含了以下的百度OCR命令：

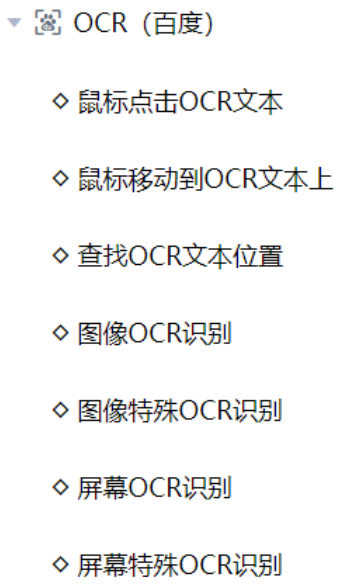

图 110: UiBot的百度OCR命令

可以看到，与上文所述的“本地OCR”命令相比，百度OCR命令中也有“鼠标点击OCR文本”、“鼠标移动到OCR文本上”、“查找OCR文本位置”、“图像OCR识别”和“屏幕OCR”这五条命令，这五条命令的使用方法与UiBot的“本地OCR”命令大体类似，唯一的区别是，需要在“属性”中填写我们在百度云上申请的Access Key和Secret Key。

再来看看“图像特殊OCR识别”命令。所谓“特殊”，是指我们要测试的是某种特定的图像，如身份证、火车票等。假设在 `D:\1.png` 文件中保存了如下的图像：

图 111: 要进行特殊OCR的图像

插入一条“图像特殊OCR识别”命令，按图示修改其属性。除了前文提到的Access Key和Secret Key之外，还需要指定要识别的图片的文件名，以及选择OCR引擎为“火车票识别”。其他属性均保持默认值，运行后，可以在输出栏看到识别的结果。这个结果实际上是一个JSON文档，如果需要进一步处理，需要采用UiBot提供的JSON类命令，但与本章关系不大，略过不表。

图 112: 特殊OCR的属性设置

6 UB语言参考

除了可视化视图之外，还有很多用户喜欢使用UiBot的源代码视图来编写一个流程块。源代码视图使用一种UiBot自创的编程语言BotScript（以下简称UB语言）来描述流程块。在这一章，我们先学习UB语言的基本规则，为后面学习源代码视图打下基础。

本章需要读者有一点点编程基础，任何编程语言都可以，只要了解变量、函数等基本概念即可。如果完全没有基础，请先阅读初级开发者指南的附录部分，以便快速入门。

6.1 概述

前文提到，UiBot的设计理念是“强大”、“简单”、“快捷”。简言之，UiBot既要让没有计算机基础的初学者，通过简单的学习，即可快速掌握流程的编写方法；又要让有一定编程基础的专业人员，能够以最快的速度实现自己的流程。

为了实现这些指标，UiBot提供了可视化的流程编写界面，便于初学者快速掌握；同时提供了一种简单、易学、接近自然语言的UB编程语言，便于专业人员的快速实现。当然，同一个流程块，可以用两种界面来显示，并可以在开发过程中随时切换。

这一章主要介绍UB语言的基本语法规则。具有基本编程基础的读者，大约在两小时内即可掌握此规则，再经过数个小时的熟悉，即可灵活运用。对于有按键精灵基础的读者，还能进一步缩短学习时间。

对于UiBot来说，编程语言只是表达逻辑的工具，关键的功能还是由命令库或插件来实现。所以，语言设计只包括基本的逻辑，所有具体的功能，哪怕是最基本的“延时”功能，都不列入语言设计中，而在命令库中单独设计。本章内容亦不包命令库的介绍。

UB语言是专门设计的，而不是市面上流行的编程语言如Python、JavaScript等，是因为UiBot的主要受众是那些非计算机专业科班出身，但足够熟悉业务流程的非技术人员。UB语言的设计尽可能的接近自然语言，对于理解基本英文单词的人来说，即使没有学习过，也能大致读懂。

相比之下，以JavaScript为例，虽然JavaScript是一种很优秀的语言，在专业的程序员手里能发挥出很高的效率，甚至UiBot本身都有一部分代码是使用JavaScript编写的。但这种语言里面大量使用的括号，容易给非专业人员的学习带来障碍。如下图：

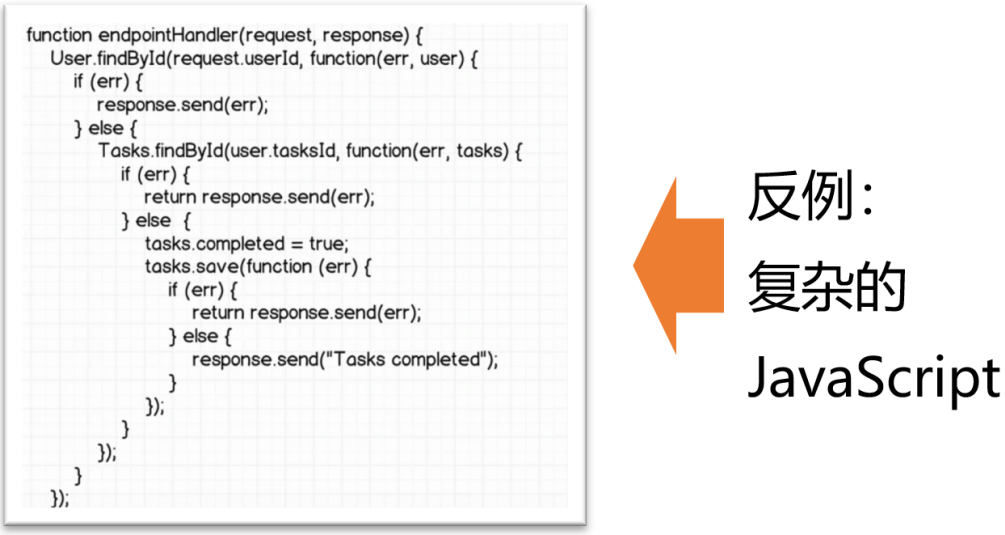

图 113: 复杂的JavaScript

因此，我们设计了专门的UB语言，并使这门语言尽可能简化，甚至尽可能少用除了字母和数字之外的元素。实际上，我们也考虑过使用市面上流行的编程语言的可能性，因为如果这样做，我们的开发UiBot的工作量会大大降低，但与此同时，您的学习难度则会大大提高。所以，我们否定了这种思路，决定不采用流行的编程语言如Python等，非不能也，是不为也。

但是，在UB语言中，吸取了很多其他编程语言的优点。您会在UB语言的设计中看到Basic语言、Python语言、JavaScript语言的一些特点。因为我们在充分理解的基础上，博取众家之长，吸取最容易理解且常用的部分，删去复杂、不常用的部分，使UB语言精简、简单、易学、易用。

我们认为UB语言是目前最适合RPA领域的编程语言。

6.2 基本结构

UB语言的源代码文件是纯文本格式，扩展名不限，一律采用UTF-8编码。

UB语言的源代码由多条语句组成，和一般的脚本型语言，如Python、JavaScript等一样，UB语言并没有严格的结构和显式指定的入口。执行一个流程块的时候，从第一行开始执行，遇到函数定义暂时跳过，然后继续从函数结束后的一行开始执行（函数的概念请参考这里）。

一般来说，我们推荐一行只写一个语句。如果一定要写多个语句，则用冒号分隔符（:）进行分隔。

如果一行内容不够，需要折行，可以在任意语句中出现的逗号（,）或二元运算符（“二元运算符”的概念请参考这里）之后直接折行，不需要增加其他额外的符号，也不推荐在其他地方折行。但如果一定要在其他地方折行，则用反斜杠（\）作为折行符号。例如：

```
Dim a= \
1
```

当一行中存在 `//` 时，表示从这以后的内容都是注释。包含在 `/*  */` 中的内容，无论多少行，都视作注释。例如：

```
// 这里是注释
/*
这里
也是
注释
*/
```

注释在流程运行过程中没有任何作用，仅供阅读方便。

UB语言中所有关键字、变量名、函数名均**不区分大小写**。例如：变量名abc、ABC或者Abc都被认为是同一个变量。

6.3 变量、常量和数据类型

6.3.1 数据类型

变量是编程语言中最基础的功能，变量中可以存放数字、字符串等值，并且可以在运行的过程中，随时改变变量中的值。UB语言中的变量是动态类型的，即变量的值和类型都可以在运行过程中动态改变。这符合一般脚本型语言如Python、JavaScript的习惯。变量的类型分为以下几种：整数型、浮点数型、布尔型、字符串型、函数型、复合型和空值型。

整数型的值可以以十进制或者十六进制的方式表示，其中十六进制需加前缀 `&H` 或 `&h`。

浮点数的值可以用常规方式或者科学计数法方式表示。如 `0.01` 或者 `1E-2` 或者 `1e-2` 均代表同一个浮点数。

布尔型的值仅有 `True` 或者 `False`，两者皆不区分大小写。

字符串型的值用一对单引号（'）或一对双引号（"）所包围，字符串中可以用 `\t` 代表制表符，用 `\n` 代表换行，用 `\'` 代表单引号，用 `\"` 代表双引号，用 `\\` 代表反斜杠本身（这种表示方式称为“转义”）。字符串中间可以直接换行，无需增加任何其他符号，换行符也会作为字符串的一部分。

也可以用前后各三个单引号（'''）来表示一个字符串，这种字符串被称为长字符串。在长字符串中，可以直接写回车符、单引号和双引号，无需用 `\n`，`\'` 或者 `\"` 进行转义。

函数型的值只能是已经定义好的函数，在后文详述。

复合型的值包括数组、字典等，在下一节详细阐述。

空值型的值总是 `Null`，不区分大小写。

例如：

```
a = 1            // a是整数型变量
a = &HFF         // a还是整数型变量
a = True         // a是布尔型变量。作为动态类型语言，a的类型可以随时变化
a = FALSE        // a是布尔型变量，注意True和False都不区分大小写
a = 'UiBot'      // a是字符串型变量
a = "UiBot
RPA"             // a是字符串型变量，字符串中可以换行
a = null         // a是空值型变量，可以写为Null、NULL或null（不区分大小写）
```

6.3.2 变量和常量

变量的定义方式是：

```
Dim 变量名
```

定义变量名的同时，可以给变量赋值一个初始值：

```
Dim 变量名=值
```

想要定义多个变量的话，可以这样定义：

```
Dim 变量名1 = 值1, 变量名2
Dim 变量名1 = 值1, 变量名2 = 值2
```

常量的定义方式和变量类似，只是把Dim改为Const，并且必须在定义时就指定值：

```
Const 常量名=值, 常量名=值
```

常量和变量的唯一区别是，常量只能在定义时指定一次值，后面不允许再修改。

例如：

```
Dim a               // 定义名为a的变量，暂不赋值
Dim b = 1           // 定义名为b的变量，并赋值为1
Dim c, d = True     // 定义名为c和d的两个变量，为d赋值True
Const e = 'UiBot'   // 定义名为e的常量，为其赋值为字符串'UiBot'
Const f             // 错误：常量必须有初始赋值
```

对于有命名的变量、常量、函数等，其名字统称为标识符，标识符需要遵循一定规则定义。

标识符可以用英文字母、下划线（_），任意UTF-8编码中包含的除英语以外其他语言的字符（当然，也包括汉字）来表示，除了第一个字符外，后面还可以使用0-9的数字。当使用英文字母时，变量名不区分大小写。

UB语言推荐变量经过定义再使用（除了For语句中的循环变量、Try语句中的异常变量、函数参数等）。变量在函数范围内定义时，属于局部变量，在函数退出时即清空。在函数范围之外任何位置定义时，属

于流程块级变量，在一个流程块的运行过程中都不会清空。流程块级变量可以定义在函数范围外任何位置，不影响其使用，甚至可以在使用变量之后定义。

如果变量未经定义而直接使用，UiBot也不会报错，但会有一个警告提示。这个警告能避免您在输入变量名时产生手误，比如把变量名 cat 误输入成为了 cart，通过警告信息，您就会发现 cart 是未经定义的，并进一步查出是因为手误所致。

6.3.3 复合类型

除了常用的整数型、字符串型等简单数据类型之外，UiBot还支持两种复合类型：数组、字典。两者在定义时和简单数据类型变量的定义并无区别。

数组类型变量的表示方法为：使用小写方括号包围起来，使用逗号来分隔每个元素，和 VBScript 中的数组定义类似。范例：

```
Dim 数组变量 = [值1, 值2, 值3, 值4]
```

同一个数组中的多个元素的值可以是任意类型，例如：元素的值是整数，就构成一个整数数组；同一个数组中的多个元素也可以是不同类型，例如：第一个元素是整数，第二个元素是字符串等；甚至，一个数组中的元素也可以是另外一个数组，这样就构成了一般意义上的多维数组。范例：

Dim 数组变量 = [值 1, 值 2, [值 11, 值 22], 值 4]

字典类型变量的表示方法为：使用大括号来包围起来，名字和其对应的值为一对，用逗号分隔。范例：

```
{ 名字1:值1, 名字2:值2, 名字3:值3 }
```

其中 **名字** 只能是字符串，**值** 可以是任意类型的表达式。如果您熟悉 JavaScript 或者 JSON，会发现这种初始化方法和 JSON 的表示形式高度相似。

数组和字典类型变量的使用方法为：无论是数组还是字典，要引用其中的元素，均采用方括号作为索引。范例：

```
变量名[索引1]
```

使用这种方法引用数组或者字典中的元素，既可以作为左值也可以作为右值，也就是说，既可以读取该变量的值，也可以为该变量的内容赋值，甚至可以在其中增加新的值。范例：

```
Dim 变量 = [486, 557, 256]                          // 变量可以用中文命名，初值是一个数组
a = 变量[1]                                         // 此时a被赋值为557
变量 = {"key1":486, "key2":557, "key3":256}         // 变量的类型改为一个字典
a = 变量["key1"]                                    // 此时a被赋值为486
```

注意：在引用数组或字典中的元素时，数组的索引只能是整数类型，用0作为起始索引；字典的索引只能是字符串类型。如果未能正确的使用，会在运行时报错。

数组或者字典的引用是可以嵌套的，如果要引用数组中的数组（即多维数组），或者字典中的数组，可以继续在后面写新的方括号。范例：

```
变量 = {"key1":486, "key2":557, "key3":256}      // 变量的类型为一个字典
变量["key4"] = [235, 668]                        // 往字典中增加一个新值，该值是一个数组
//此时，字典中的内容为 {"key1":486, "key2":557, "key3":256, "key4":[235, 668]}
a = 变量["key4"][0]                              // 此时a被赋值为235
```

6.4 运算符和表达式

UB语言中的运算符及其含义如下表：

+	-	*	/	&	^	<	<=
加法	减法/求负	乘法	除法	连接字符串	求幂	小于	小于等于

>	>=	<>	=	And	Or	Not	Mod
大于	大于等于	不相等	相等/赋值	逻辑与	逻辑或	逻辑非	取余数

把变量、常量和值用运算符和圆括号 **( )** 连接到一起，称为表达式。在上述运算符中，**Not** 是一元运算符、**-** 既可以用作一元运算符，也可以用作二元运算符，其他都是二元运算符。一元运算符只允许在右边出现一个元素（变量、常量、表达式或值），二元运算符只允许在左右两边同时出现两个元素。

注意：当 **=** 出现在表达式内部时，其含义是判断是否相等。当 **=** 构成一个独立的语句时，其含义是赋值。这里 **=** 的设计虽然具有二义性，但能更好的被初学者所接受。

UB语言中删掉一些其他语言中具备、但不常用的运算符，如整数除运算符、位操作运算符等等。因为这些运算符的使用场景较少，即使需要，也可以采用其他方式实现。

表达式常用于赋值语句，可以给某个变量赋值，其形式为：

```
变量名 = 表达式
```

注意，当表达式为一个独立的（没有使用任何运算符计算）数组、字典类型的变量时，赋值操作只赋值其引用，也就是说，只是为这个变量增加一个“别名”。当一个数组、字典中的元素发生改变时，另一个也会改变。

例如：

```
a = [486, 557, 256]      // a是一个数组
b = a                    // b是a的“别名”
b[1] = 558               // 改变b里面的值，a里面的值也会跟着改变
c = a[1]                 // 此时c的值是558，而不是原来的557
a = 557                  // 此时a被赋值为557（变为整数型）
```

```
b = a                    // 此时b里面的值也是557，但和a分别保存
b = 558                  // b里面的值发生改变，a的值不改变
c = a                    // 此时c的值仍然是原来的557，因为a不是字典、数组
```

6.5 逻辑语句

6.5.1 条件分支语句

即一般编程语言中最常用的If...Else语句，主要用于对某一个或者多个条件进行判断，从而执行不同流程。在UB语言中，有以下几种形式：

```
If 条件
    语句块1
End If
```

```
If 条件
    语句块1
Else
    语句块2
End If
```

```
If 条件1
    语句块1
ElseIf 条件2
    语句块2
Else
    语句块3
End If
```

当条件满足时，会执行条件之后的语句块，否则，语句块不会执行。Else后面的语句块则会在前面所有条件都不满足的时候，才会执行。

例如：

```
// Time.Hour() 可以取得当前时间中的小时数
// TracePrint() 可以把指定的内容输出到UiBot的输出栏中

If Time.Hour() > 18              // 取得当前时间中的小时数
    TracePrint("下班时间")       // 如果大于18，则执行这里的语句
Else
    TracePrint("上班时间")       // 如果不满足前面的条件，则执行这里的语句
End If
```

6.5.2 选择分支语句

根据一定的条件，选择多个分支中的一个。先计算Select Case后面的表达式，然后判断是否有某个Case分支和这个表达式的值是一致的。如果没有一致的Case分支，则执行Case Else（如果有）后面的语句块。

```
Select Case 表达式
    Case 表达式1, 表达式2
        语句块1
    Case 表达式3, 表达式4
        语句块2
    Case Else
        语句块3
End Select
```

例如：

```
Select Case Time.Month()            // 取得当前时间中的月份
    Case 1,3,5,7,8,10,12            // 如果是1、3、5、7、8、10、12月
        DayOfMonth = 31             // 当月有31天
    Case 4,6,9,11                   // 如果是4、6、9、11月
        DayOfMonth = 30             // 当月有30天
    Case Else                       // 如果是其他（也就是2月）
        DayOfMonth = 28             // 当月有28天（不考虑闰年的情况）
End Select
TracePrint(DayOfMonth)
```

6.5.3 条件循环语句

在UB语言中，使用Do...Loop语句来实现条件循环，即满足一定条件时，循环执行某一语句块。Do...Loop语句有以下五种不同的形式，用法较为灵活：

1. **前置条件成立**则循环：先判断条件，条件成立则循环执行语句块，否则自动退出循环。

```
Do While 条件
    语句块
Loop
```

2. **前置条件不成立**则循环：和前一条相反，条件成立则退出循环，否则循环执行语句块。

```
Do Until 条件
    语句块
Loop
```

3. **后置条件成立**则循环：先执行语句块，再判断条件，条件成立则继续循环执行语句块，否则自动退出循环。

```
Do
    语句块
Loop While 条件
```

4. **后置条件不成立**则循环：先执行语句块，再判断条件，条件成立则自动退出循环，否则继续循环执行语句块。

```
Do
    语句块
Loop Until 条件
```

5. **无限**循环：该循环语句本身不进行任何条件的判断，需要在语句块中自行做判断，如果语句块中没有跳出循环的语句，则会无限的执行该循环

```
Do
    语句块
Loop
```

例如：

```
Do Until Time.Hour() > 18                // 判断当前时间中的小时数，只要不大于18就循环
    TracePrint("还没有到下班时间")       // 每次循环，都会执行这里的语句
    Delay(1000)                          // 每判断一次，休息一秒钟
Loop
TracePrint("下班时间到啦")               // 如果大于18，则跳出循环，执行这里的语句
```

6.5.4 计次循环语句

计次循环语句主要用于执行一定次数的循环，其基本形式为：

```
For 循环变量 = 起始值 To 结束值 Step 步长
    语句块
Next
```

在计次循环语句中，起始值、结束值、步长都只允许是整数型或者浮点数型；步长可以省略，默认值为1。变量从起始值开始，每循环一次自动增加步长，直到大于结束值，循环才会结束。

在计次循环语句中，循环变量可以不用Dim语句定义，直接使用，但在循环结束后就不能再使用了。

例如：

```
Dim count = 0                // 定义变量count
For i=1 To 100               // 每次循环，变量i都会加1。这里变量i不需要定义
    count = count + i
```

```
Next
TracePrint(count)            // 这里会显示1+2+3+…+100的结果，即5050
```

6.5.5 遍历循环语句

遍历循环语句可以用于处理数组、字典中的每一个元素。遍历循环语句有以下两种形式：

```
For Each 循环变量 In 数组或字典
    语句块
Next
```

在这种形式的循环语句中，会自动遍历数组、字典中的每一个值，并将其置入循环变量中，直到遍历完成为止。

或者：

```
For Each 循环变量1, 循环变量2 In 数组或字典
    语句块
Next
```

在这种形式的循环语句中，会自动遍历数组、字典中的每一个索引和值，并将其分别置入循环变量1、循环变量2中，直到遍历完成为止。

和计次循环语句类似，在遍历循环语句中，循环变量可以不用Dim语句定义，直接使用，但在循环结束后就不能再使用了。

例如：

```
Dim days = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31] // 定义数组型变量days
Dim count = 0
For Each i In days              // 每次循环，变量i的值分别为days中的每个值
    count = count + i           // 把数组中的每个值依次加起来
Next
TracePrint(count)               // 这里会显示一年中每个月的天数的累加和，即365
```

6.5.6 跳出语句

在UB语言中，支持以下形式的循环跳出语句：

```
Break
```

只能出现在条件循环、计次循环或遍历循环等循环语句的内部语句块中，其含义是立即跳出当前循环。

```
Continue
```

只能出现在条件循环、计次循环或遍历循环等循环语句的内部语句块中，其含义是立即结束当前循环，并开始下一次循环。

例如：

```
Dim days = { '一月':31, '二月':28, '三月':31,
             '四月':30, '五月':31, '六月':30,
             '七月':31, '八月':31, '九月':30,
             '十月':31, '十一月':30, '十二月':31 }  // 定义字典型变量days

For Each i,j In days              // 每次循环，变量i，j分别为days中每个名字和值
    If j Mod 2 = 0                // 如果j是偶数
           Continue               // 结束本次循环，开始下一次循环
    End If
    TracePrint(i)                 // 把days中的名字（其值不是偶数）显示出来
Next
```

另外，在流程块中的任何地方，只需要书写

```
Exit
```

不需要任何参数，即可在执行到此行的时候，自动结束整个流程（不是当前流程块）的执行。

6.6 函数

所谓函数，是指把一组常用的功能包装成一个语句块（称之为“定义”），并且可以在其他语句中运行这个语句块（称之为“调用”）。使用函数可以有效的梳理逻辑，以及避免重复代码的编写。

函数的定义和调用没有先后关系，可以先出现调用，再出现定义。但函数必须定义在全局空间中，也就是说，函数定义不能出现在其他函数定义、分支语句、循环语句下面的语句块中。

函数定义中可以包含参数，参数相当于是一个变量，但在调用时，可以由调用者指定这些变量的值。

定义函数的格式如下：

- 无参数的函数

```
Function 函数名( )
    语句块
End Function
```

- 有参数的函数

```
Function 函数名(参数定义1, 参数定义2)
    语句块
End Function
```

其中，参数定义的格式可以只是一个变量名，也可以是变量名 = 表达式的形式。对于后者来说，表示这个参数带有一个“默认值”，其默认值由“表达式”来确定。

如果函数有参数，则参数中的每个变量名都被认为是此函数内已经定义好的局部变量，无需使用Dim语句定义。

在函数定义中，要退出函数并返回，采用以下写法：

```
Return 返回值
```

当执行到这一语句时，将跳出函数并返回到调用语句的下一行。返回的时候可以带一个返回值（具体作用下文叙述）。返回值可以忽略，默认为Null。当执行到函数末尾的时候，无论有没有写Return语句，都会返回。

例如：

```
Function Add(x, y=1)          // 定义了两个参数的函数，第二个参数有默认值
    Return x + y              // 返回值为x+y的值
End Function
```

调用函数的格式如下：

```
返回 = 函数名(表达式1, 表达式2)
```

或者

```
函数名(表达式1, 表达式2)
```

按照第一种格式调用，可以指定一个变量作为返回，当函数调用完成后，函数的返回值会自动赋值给这里的返回变量，调用者可以通过返回值，了解到函数调用的情况。此时，必须在被调用的函数名后面加圆括号。而当按照第二种格式调用时，调用者不需要返回值，则可以省略圆括号，使语句更符合自然语言习惯。

当调用时，相当于对函数中的参数进行了一次赋值运算，用表达式的值对其赋值。与赋值运算的规则相同，当表达式为一个独立的（没有使用任何运算符计算）数组、字典时，赋值操作只赋值其引用，也就是说，只是为变量增加一个“别名”。

调用函数时，传入的表达式的数量可以少于参数的数量。如果某个参数没有传入值，或者传入值为Null，则采用其默认值。没有默认值的参数，调用函数时必须传入值或者表达式。

例如，对于上面定义的函数，可以按照如下的方式调用：

```
a = Add(100)              // 调用Add函数，第二个参数取默认值1，所以a的值是101
b = Add(100, 200)         // 调用Add函数，指定了两个参数，所以b的值是300
Add 100, 200              // 调用Add函数，不关心返回值，所以可以不写括号
```

当函数定义完成后，其名称可以作为一个函数类型的常量使用，也可以把函数名称赋值给某个变量，用这个变量也可以调用这个函数。

例如，对于上面定义的函数Add，可以按照如下的方式使用：

```
Dim Plus = Add
TracePrint Plus(100, 200)
// 相当于先调用了Add函数，再用其返回值调用了TracePrint函数，结果是300
```

除了在流程块中定义的函数之外，UB语言中的各种命令实际上就是内置的函数。比如上面例子中的TracePrint就是一个内置函数，同时也是一条命令。所以，在使用TracePrint命令的时候，以下两种方式都是可以的：

- TracePrint("Hello")
- TracePrint "Hello"

请注意："函数"和"参数"的称呼符合一般编程语言的习惯。但为了更好地让非IT人员理解，我们在UiBot软件本身中多采用"子程序"和"属性"的称呼，来指代"函数"和"参数"。但在本章中，由于介绍的是UB语言，所以仍然维持与其他编程语言一致的习惯，称为"函数"和"参数"。

6.7 其他

6.7.1 多模块

UB语言支持多模块，可以在UB语言中，调用另一个用UB语言编写的流程块。虽然UB语言并未规定扩展名，但如果要调用另一个UB语言的流程块，则两个流程块需要有相同的扩展名。

我们把被调用的流程块称为"模块"，则去掉扩展名以后，剩下的文件名就是模块的名字。比如某个流程块的文件名为UBTest.task，则其模块名为UBTest。

在UB语言中，采用以下方式导入一个模块：

```
Import 模块名
```

注意这里的模块名的书写规则和变量名一致，不需要采用双引号，也不需要加扩展名。如Import UBTest。UiBot在编译和运行时会自动按照当前流程块文件的扩展名，为其补充扩展名，并在当前目录下查找。在Windows中，由于文件名不区分大小写，所以Import语句后面的模块名也可以不区分大小写。在其他操作系统中，需要注意模块名的大小写要和文件一致。

每个导入的模块，都会被放置在一个与模块名同名的"命名空间"中，可以通过下面这种方式来调用导入模块中的函数：

```
命名空间.函数名
```

即在命名空间和函数名之间加一个点号（.）进行分隔。

导入一个模块之后，既可以调用模块中定义的函数，又可以直接以模块名作为函数名，直接运行这个流程块中的所有命令。例如，有一个流程块 ABC.task。在其他流程块中Import之后，直接采用下面的格式即可直接调用ABC.task（相当于运行了ABC.task这个流程块）：

```
ABC()
```

假设流程块 ABC.task中定义了一个函数，名为test，则可以采用下面的格式调用这个函数

```
ABC.test()
```

6.7.2 异常

作为动态类型语言，有很多错误在编译时难以检查，只能在运行时报错。而且，由于UiBot不强调运行速度，而更强调运行的稳定性，也会在运行时加入比较多的检查。当出错的时候，比较合适的报错手段是抛出异常。比如，对于界面元素自动化中的“有目标命令”，在运行的时候，如果到了超时时间都不能找到目标，就会自动抛出一个异常。

除了自动抛出的异常之外，在流程块中，还可以采用Throw语句抛出一个异常：

```
Throw 字符串
```

在抛出异常时，可以把异常相关信息以字符串的形式一起抛出，也可以省略这个字符串。

如果在流程块中没有对异常进行处理，当出现异常时，整个流程都会终止执行，并且把异常相关信息显示出来。如下图所示：

图 114: 流程运行的时候出现异常

如果不希望流程在发生异常的时候终止，可以采用以下语句对异常进行处理：

```
Try
    语句块
Catch 变量名
    语句块
Else
    语句块
End Try
```

如果在Try后面的语句块中发生了异常，会跳到Catch后面的语句块中执行。如果在Try语句块中没有发生异常，且定义了Else语句块（当然，也可以省略Else语句块），则会跳到Else语句块中执行。

Catch语句后面的变量名可以省略。如果不省略，可以不用Dim语句提前定义，当发生异常时，这个变量的值是一个字典，其中包含“File”、“Line”和“Message”三个字段，分别代表发生异常的文件名、发生异

常的行号、异常包含的信息。

对于界面元素自动化来说，有的时候可能会因为界面卡顿等引起失败，实际上再试一次可能又正常了。因此，在Try语句后面，还可以加一个“重试次数”。如下所示：

```
Try N
    语句块1
Catch 变量名
    语句块2
Else
    语句块3
End Try
```

这里的N可以是变量、表达式或常量，但通常应为整数类型。其含义是：

在Try和Catch之间的语句（如上例中的“语句块1”），如果发生了异常，会自动回到Try的地方去重试，当重试N次之后，如果仍然有异常，才会跳到Catch后面的语句（如上例中的“语句块2”）去执行。当在N次尝试中，只要有1次成功了，就不会再继续后面的尝试（比如第1次异常，第2次没有异常，就不会再试第3次了），而是跳到Else后面的语句（如上例中的“语句块3”）去执行。

7 高级开发功能

7.1 流程调试

当我们兴致勃勃地用UiBot写完一个流程并运行后，总是期待得到成功的结果。但是有时候往往达不到预期的效果，尤其是对于新手而言，要么运行的时候UiBot报错，要么UiBot不报错，但是流程运行没有得到预想的结果。这个时候就需要对流程进行调试了。

所谓调试，是将编制的程序投入实际运行前，用手工或自动等方式进行测试，修正语法错误和逻辑错误的过程，是保证计算机软件程序正确性的必不可少的步骤。

其实，我们在前面已经大量使用了一种最原始、最朴素、但也是最常用、最实用的一种程序调试方法：“输出调试信息”命令。在关键代码的上一行或下一行添加输出调试信息，查看参数、返回值是否正确。严格意义上来说，这并不能算是一种程序调试的方法，但是确实可以用以测试和排除程序错误，同时也是某些不支持调试的情况下一个重要的补充方法。

本节将会介绍**“真正意义上”**的程序调试方法，可以根据提示的错误信息、监测的运行时变量，准确定位错误原因及错误位置。

7.1.1 调试的原则

首先，我们要清晰地认识到：程序，是人脑中流程落实到编程工具的一种手段；程序调试，本质上是帮助厘清人脑思路的一种方式。因此，在调试的过程中，人脑一定要清晰，这样才能迅速和准确地定位和解决问题。

1. 冷静分析和思考与错误相关的提示信息。
2. 思路要开阔，避开钻死胡同。一个问题，如果一种方法已验证行不通，就需要换种尝试思路。
3. 避免漫无目的试探，试探也是要有目的性地缩减排查的范围，最终定位出错的地方。
4. 调试工具只是定位错误位置、查找错误原因的辅助方法和手段。利用调试工具，可以帮你理清楚程序中数据流转逻辑，可以辅助思考，但不能代替思考，解决实际问题时仍需要根据调试的提示信息，自己思考后做出正确的判断。
5. 不要只停留于修正了一个错误，而要思考引起这个错误的本质原因，是粗心写错了名称？还是用错了命令？还是流程设计上就有问题？只有找到了引起错误的本质原因，才能从根本上规避错误，以后不再犯类似错误。

7.1.2 调试的方法

首先，要对系统的业务流程非常清楚。业务产生数据，数据体现业务，流程的运行逻辑也代表着业务和数据的运转过程。当错误发生时，首先应该想到并且知道这个问题的产生所依赖的业务流程和数据。

比如：当点击“提交”按钮时，表单提交出现错误。这时应该思考：点击“提交”按钮后，发生了哪些数据流转？再根据错误现象及报错提示信息，推测该错误可能会发生在这个业务数据流转过程中的哪个位置，从而确定我们调试的断点位置。

7.1.3 UiBot的调试方法

添加和删除断点

在UiBot中，可以设置断点，在调试的过程中，遇到断点会自动停下来。考虑到UiBot的主要业务逻辑在流程块中，所以只需要在流程块中设置断点，即可满足调试要求。

我们知道，流程块包含了“可视化”和“源代码”两种视图，无论哪一种，都可以用以下的方式来添加和删除断点：

1. 点击任意一行命令左边的空白位置，都可以添加断点。再次点击这个位置，可以删除这个断点。
2. 选中一行命令，在菜单中选择“运行”->“设置/取消断点”，原先没有断点的，会加上断点；原先有断点的，会删掉这个断点。
3. 选中一行命令，直接按热键F4，效果同上。

设置断点后，这一行命令的左边空白处会出现一个红色的圆形，同时这一行命令本身的背景也会变红。如下图：

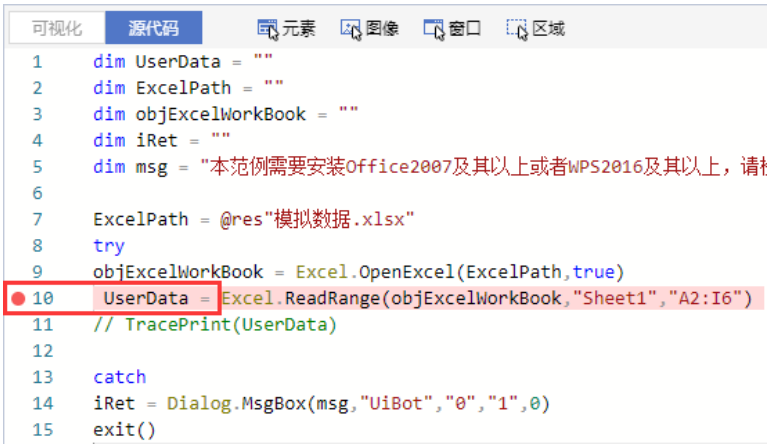

图 115: 添加和删除断点

调试运行

在编写流程块的过程中，我们可以发现：在菜单栏的“运行”一栏下面，分别有四个菜单项：运行、运行全流程、调试运行、调试运行全流程。点击工具栏的“运行”图标右边的下拉按钮，也有类似的四个选项。它们的含义分别是：

1. 运行：只运行当前流程块，并且忽略其中所有的断点。
2. 运行全流程：运行整个流程图，并且忽略其中所有的断点。
3. 调试运行：只运行当前流程块，遇到断点会停下来。

4. 调试运行全流程：运行整个流程图，遇到断点会停下来。

图 116: 调试运行菜单项

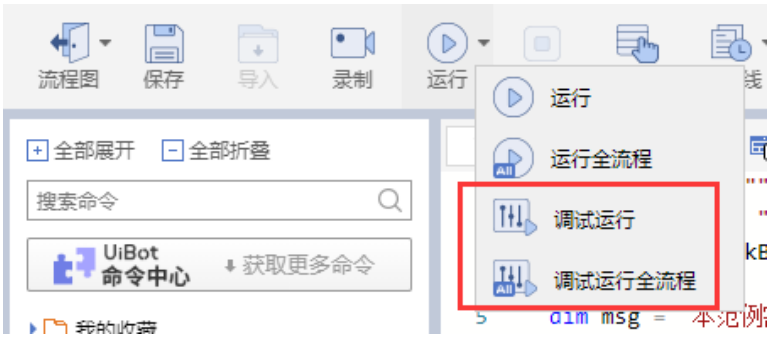

图 117: 调试运行工具栏

单步调试

当调试运行时，程序运行到断点处，会自动停下来。此时，在调试状态栏列出了常见的四个调试运行动作：继续运行(F6)、步过(F7)、步入(F8)、步出(F9)。“继续运行”指的是继续运行到下一个断点；“步过”指的是继续运行下一条命令；“步入”指的是继续运行下一条命令，如果下一条命令是函数，那么进入函数，在函数内的第一条命令处停下来；“步出”指的是跳出本层函数，并返回到上一层。

调试状态栏的左下方列出了本流程块变量的值，在程序运行到断点位置暂停时，进行下一步调试，这时需要特别**注意观察**程序运行的每一步的数据是否为业务流程处理的正确数据，来判断程序是否正确执行。这些数据包括输入数据、返回数据等，如果程序运行起来后，并没有进入我们预先设定的断点处，此时需要根据错误信息和业务处理流程逻辑重新推测错误发生位置，重新设置断点。最终不断将一个大的问题细化拆解，最终精确定位错误点。

图 118: 单步调试

调试状态栏的右下方列出了本流程块的断点列表，大家可以根据需要启用、禁用和删除断点。

图 119: 断点列表

打断点的技巧

一般打断点的方式及位置是：

- 在有可能发生错误的方法的第一行逻辑程序打断点。
- 方法中最有可能发生错误的那一行打程序断点。

7.2 单元测试块

一般来说，一个流程图由一个或多个流程块组成，如果该流程比较复杂，那么流程中包含的流程块数量一般比较多，或者单个流程块的命令条数比较多。我们对流程图中靠后执行的流程块进行调试时，如果靠后流程块依赖靠前流程块的数据输入，那么靠后流程块的调试将会非常费时费力。我们举一个具体实例：假设某个流程由两个流程块组成，分别叫做“靠前流程块”和“靠后流程块”，流程中定义了两个全局变量x和y，“靠前流程块”中分别为x和y赋值为4和5，“靠后流程块”分别打印出x、y和x+y的值。

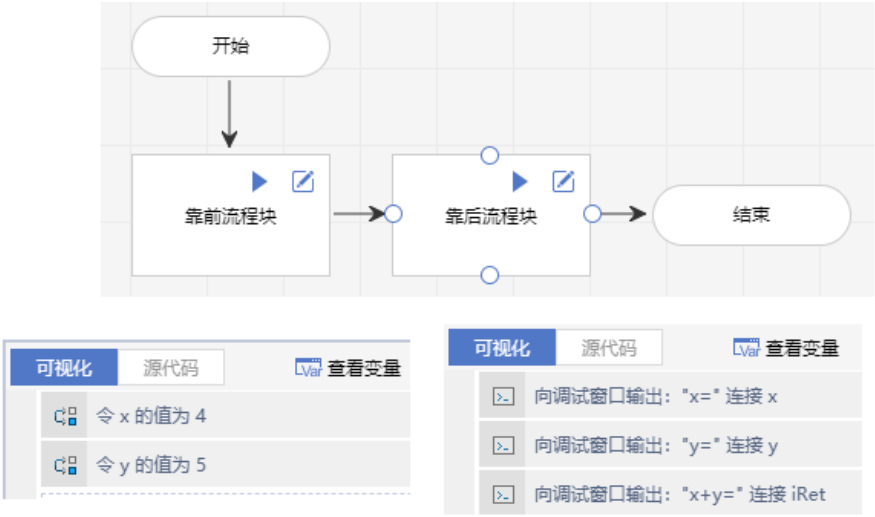

图 120: 单元测试块的实例

我们在流程图视图下点击运行，可以看到，UiBot可以输出正确的结果：

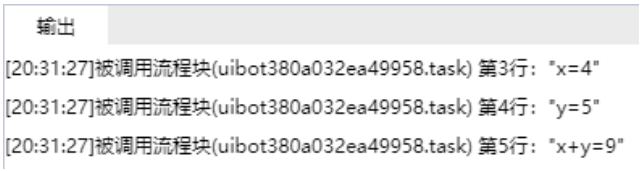

图 121: 运行全流程得到正确结果

假如我们要单独测试一下“靠后流程块”的功能（其实就是一个加法模块）是否正确，此时“靠后流程块”是无法单独执行的，我们在“靠后流程块”的可视化视图或源代码视图下点击运行，会报错：

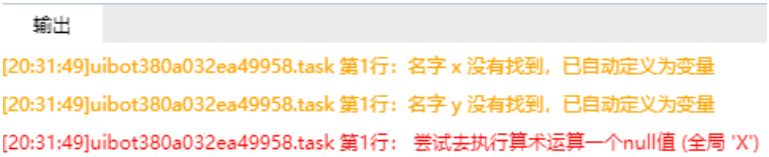

图 122: 运行单个流程块报错

为了测试这个“靠后流程块”，必须要执行“靠前流程块”，因为x和y赋值操作来源于“靠前流程块”。这里“靠前流程块”比较简单，只做了几个赋值操作，如果“靠前流程块”比较复杂，例如x和y的值分别来源于抓取天猫和京东两个网站某类商品后的统计数量。那么测试这个“靠后流程块”的代价将非常大。

那怎么办呢？强大的UiBot从5.0版本开始，提供了一种单元测试块，可对单个流程块进行测试。回到刚才的例子，我们来看看单元测试块的具体用法。

打开“靠后流程块”的源代码视图，在命令中心“基本命令”的“基本命令”目录下，插入一条“单元测试块”命令。我们可以看到，在源代码视图下，`UnitTest`和`End UnitTest`中间就是单元测试块，我们在中间填入测试命令分别为x和y赋值3和2。

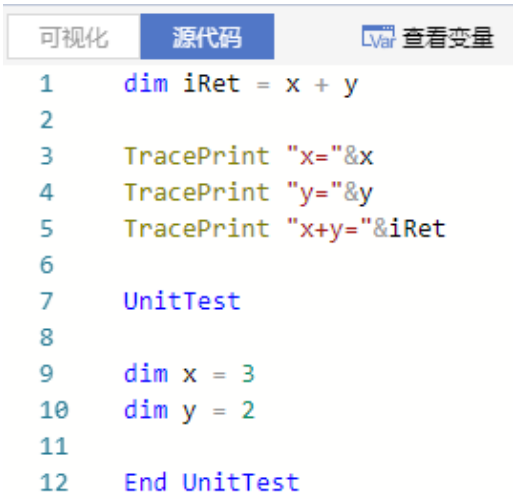

图 123: 撰写单元测试块

我们在“靠后流程块”的可视化视图或源代码视图下再次点击运行，此时执行正确：

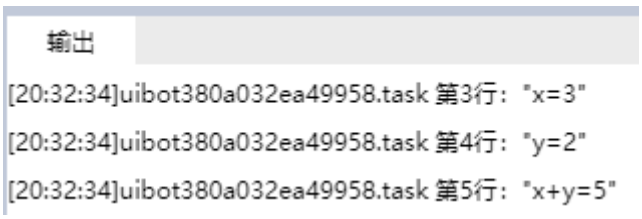

图 124: 运行单元测试块

单元测试块具有如下特性：

第一、单元测试块不管放置在流程块中的什么位置，都会被优先执行。

第二、只有在运行单个流程块时，这个流程块中的单元测试块才会被执行；如果运行的是整个流程，流程块中的单元测试块将不会被执行。

第一条特性保证了调试单个流程块时，单元测试块肯定会被执行到；第二条特性保证了单元测试块的代码不会影响整个流程的运行，不管是运行单个流程块，还是运行整个流程，都可以得到正确的结果。

7.3 时间线

源代码的版本控制是软件开发中一个十分重要的工程手段，它可以保存代码的历史版本，可以回溯到任意时间节点的代码进度。版本控制是保证项目正常进展的必要手段。对初学者学习而言，建议在开始进行实践小项目的阶段即进行源代码版本控制，这在以后的工作中会大有裨益。

UiBot通过集成著名的代码版本控制软件Git，提供了强大的版本控制手段：**“时间线”**。所谓时间线，指的是不同时间点的代码版本。

1. 手动保存时间线

用户鼠标移到工具栏“时间线”按钮上，“时间线”按钮上出现“保存时间线”按钮，点击“保存时间线”，即可保存将该时间点的流程。保存时间线时，需要填写备注信息，用以描述该时间点修改了代码的哪些内容。

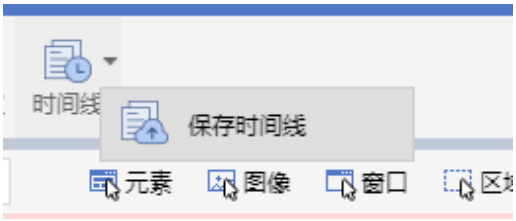

图 125: 保存时间线

2. 自动保存时间线

如果用户不记得保存时间线，没关系，UiBot每隔五分钟，会自动保存时间线；如果这段时间内用户未修改流程内容，则不保存时间线。

3. 查看时间线

点击工具栏“时间线”按钮，“时间线”页面按照“今天”、“七天之内”、“更早之前”列出已保存的时间点，单击任意一个时间点，可查看当前文件和选中时间点文件的内容差异，内容差异会用红色背景标识出。

如果要恢复该时间线的部分代码，可以直接点击代码对比框的蓝色箭头，直接将该段代码恢复到现有代码中。

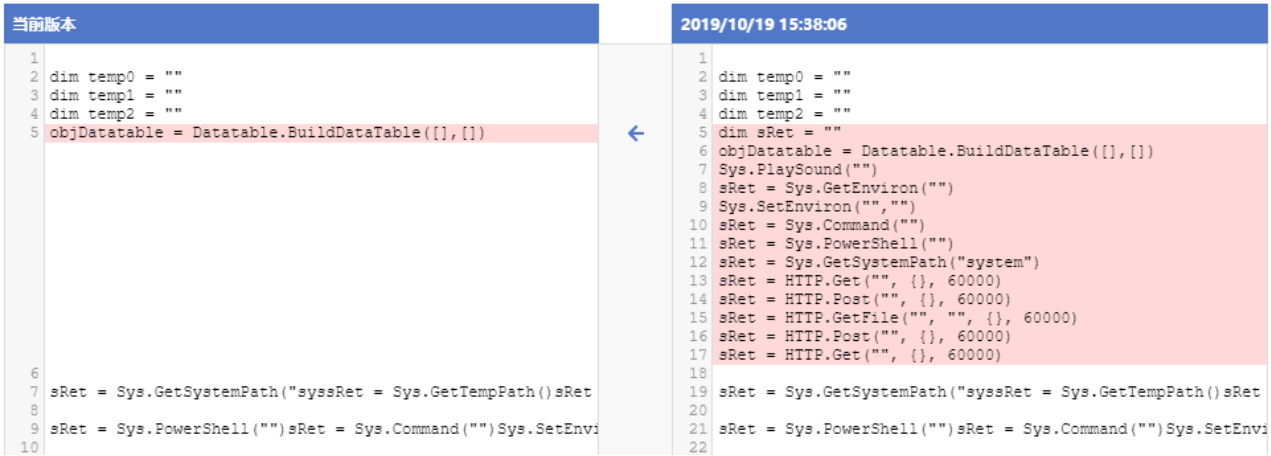

图 126: 时间线对比代码

4. 在“时间线”页面，点击任意一个时间点的备注详情，可查看该时间点备注的详细信息，如图：

图 127: 查看时间线备注

5. 在“时间线”页面，点击任意一个时间点的恢复按钮，可将该时间线的代码内容恢复至现有代码，恢复后的时间点，会在左上角有个绿色的R标记，表示Revert（恢复），鼠标移动到R标记上，会显示从哪个时间点恢复的具体时间点。

图 128: 恢复时间线

7.4 命令库

7.4.1 模块化思想

模块化的思想在许多行业中早已有之，并非计算机科学所独创。

例如，建筑行业很早就提出了模块化建筑概念，即在工厂里预制各种房屋模块构件，然后运到项目现场组装成各种房屋。模块构件在工厂中预制，便于组织生产、提高效率、节省材料、受环境影响小。模块组装时施工简便快速、灵活多样、清洁环保，盖房子就像儿童搭建积木玩具一样。

又如，现代电子产品功能越来越复杂、规模越来越大，利用模块化设计的功能分解和组合思想，可以选用模块化元件（如集成电路模块），利用其标准化的接口，搭建具有复杂功能的电子系统。模块化设计不但能加快开发周期，而且经过测试的模块化元件也使得电子系统的可靠性大大提高，标准化、通用化的元件使得系统易构建、易维护。

总之，模块化的思想就是在对产品进行功能分析的基础上，将产品分解成若干个功能模块，预制好的模块再进行组装，形成最终产品。

UiBot中的预制件是模块化的一个典型示例，现在UiBot已经提供了四百多个预制件，涵盖了鼠标键盘、各种界面元素的操作、常见软件的自动化操作、数据处理、文件处理、网络和系统操作等方方面面。这些预制件采用模块化，各自相对独立，而又能组合起来完成复杂的功能。

除了UiBot中的预制件之外，您也可以把用UiBot实现的一部分功能组装成模块，将来如果要再用到类似的功能，就不需要重写了，直接拿这个模块来用即可。比如，在某个项目中，我们使用UiBot做了“银行账户流水下载”的功能，即可将其组装成模块。在今后的项目中，只要导入模块，即可直接使用“银行账户流水下载”的功能，省时省力。

在UiBot中，这样的模块称之为命令库。一个命令库里面包含了若干条命令，使用起来就像UiBot中的预制件一样，可以在可视化视图中拖拽，也可以用接近自然语言的形式来展示，便于理解。

7.4.2 建立命令库

这里用一个实际的例子来说明如何建立命令库：假设我们设计了一个模块，其中包含四个功能：加法、减法、乘法、除法。我们希望把这四个功能作为四条命令包装在一个UiBot的命令库里面，以便今后使用。当然，在实际使用UiBot的过程中，四则运算这样的命令过于简单了，意义不大。但读者通过这个例子掌握了命令库的用法，自然就会实现更实用、更复杂的功能。

从UiBot Creator 5.1版本开始，当我们点击首页上的“新建”按钮时，会弹出一个对话框，可以选择要新建的是一个流程，还是一个命令库（如下图）。

图 129: 新建命令库

选择命令库之后，可以看到命令库的编写界面和编写流程块类似。实际上，命令库确实可以视为一个特殊的流程块，但它不会像普通的流程块那样，从第一行开始执行，而是需要设置若干个“**子程序**”。如果您熟悉其他编程语言，“子程序”的称呼实际上就相当于其他语言中的“函数”（function）或者“过程”(procedure)。在UiBot中，之所以称为“子程序”，是为了让IT基础较少，不了解其他编程语言的开发者不至于感到困惑（比如和数学中的“函数”概念产生混淆）。

命令库中的每个子程序，对于命令库的使用者看来，就是一条“命令”。所以，就像UiBot预制的命令一样，我们可以为其设置一个名称，和一组属性，这些名称和属性也会被使用者看到。

新建一个命令库之后，作为例子，UiBot Creator已经帮我们生成了一个子程序的框架，在可视化视图和源代码视图下，其内容如下图所示。在源代码视图下，还会生成一段注释，以助理解。

图 130: 命令库的可视化视图

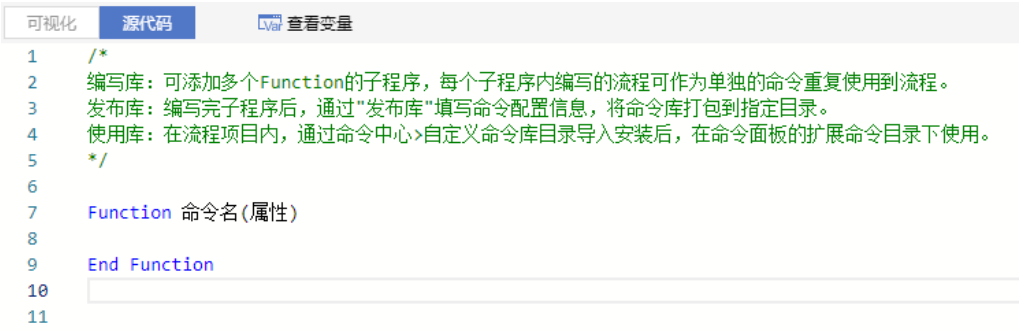

图 131: 命令库的源代码视图

我们已经学习了使用源代码来编写流程内容，直接切换到源代码视图，把下面的源代码粘贴进去。

```
Function 加法(被加数, 加数)
    Return 被加数 + 加数
End Function

Function 减法(被减数, 减数)
    Return 被减数 - 减数
End Function

Function 乘法(被乘数, 乘数)
    Return 被乘数 * 乘数
End Function

Function 除法(被除数, 除数)
    Return 被除数 / 除数
End Function
```

切换到可视化视图，即可看到我们已经完成了一个加减乘除的四则运算命令库，其中包含四条命令。如下图。当然，您也可以直接在可视化视图编写命令库及命令，具体过程比较简单，不再赘述。

图 132: 四则运算命令库

命令库至此已建立完毕，但为了方便他人使用，推荐使用“发布”功能，把这个命令库发布成一个独立的文件，以便发给他人。

在编写命令库的时候，我们可以看到，工具栏上有一个“发布库”的按钮，如下图所示。

图 133: “发布库”的按钮

点击这个按钮，UiBot Creator会校验命令库中是否存在错误，如果没有错误，则会弹出如图所示的对话框。这个对话框中的默认值已经填写好了，可以不填。而且即使不填，也不会对使用命令库有任何影响。但在这个例子中，我们仍然对红框所在的内容进行了修改，这样修改是为了让用户使用起命令库来更加容易。

图 134: 发布命令库界面

这些修改的意义在于：

- 填写“使用说明”一栏，使得其他人在用命令库的时候，鼠标移动到这条命令上面，会有一个浮窗说明命令的具体说法；
- 填写“可视化翻译”一栏，使得其他人在用命令库的时候，这条命令在可视化视图中能以更容易理解的形式出现。其中的`%1%`和`%2%`，会在可视化视图中，分别用命令的第一个属性和第二个属性替代。例如，第一个属性为`1`，第二个属性为`2`，则可视化视图中显示的内容是“将1和2相加”，而不是默认的“`四则运算.加法(1,2)`”。显然，前者的可读性要好得多；
- 在“输出到”一栏打勾并填写一个变量名，如“相加结果”。使得其他人在用命令库的时候，每次插入这条命令，还会把命令的执行结果放置到这个变量里面。

填写完成后，只要点击“发布”按钮，即可把命令库发布为一个独立的、以`.zip`为扩展名的文件。把这个文件用各种方式（如邮件、U盘等）发给其他同事，他们只要导入命令库，就可以像使用UiBot Creator中的其他预制件一样，使用其中的命令。

下面我们来看看如何导入命令库。

7.4.3 导入和使用命令库

假如我们的某个同事拿到了我们发布的命令库，具体的使用方法是：

1. 用UiBot Creator打开任意一个流程，然后再打开任意一个流程块；
2. 在左侧的面板中找到“UiBot命令中心”的按钮，点击此按钮，选择“自定义命令”下面的“自定义库命令”，如下图中红框所示。
3. 找到“导入命令库”按钮，点击后，选择已发布的命令库文件（扩展名为`.zip`）。导入完成后，在界面上会出现已导入的命令库，如下图中绿框所示。

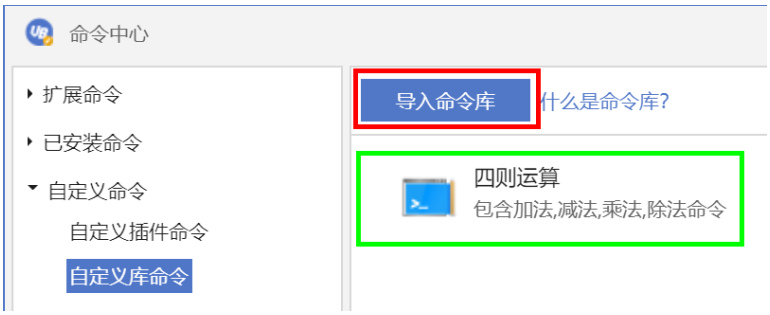

图 135: 导入命令库

4. 回到编写流程块的界面中，可以看到左侧的命令列表中，增加了一项“扩展命令”，其中包含了我们导入的“四则运算”，里面又有四条命令，对应着编写命令库时定义的四个“子程序”。

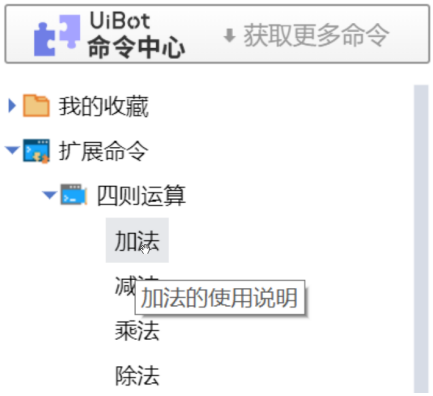

图 136: 导入命令库之后

这些命令的用法和UiBot Creator中的其他预制件一样，具体不再赘述。

值得注意的是：

- 如果我们在编写流程块的时候导入一个命令库，这个命令库在当前流程里面的所有流程块中都是可用的。但如果换了另外一个流程，就需要重新导入了；

- 使用了命令库的流程，在打包给UiBot Worker或者UiBot Store使用的时候，命令库会被自动打包，而不需要我们再做额外的处理。

8 扩展UiBot命令

说起**“插件”**，很多人脑海中都会浮现出IE/Chrome/Firefox浏览器插件、Eclipse、Visual Studio、Sublime Text等各种编程工具的插件，这些应用工具层面上的插件，依托于原平台运行，但又扩展了原平台的功能，极大丰富了原有工具和平台。基于插件，用户甚至可以定制化地打造个性化的浏览器和编程工具。

其实，绝大部分编程语言也提供这样一种插件机制，我们一般称之为**“类库”**。比如Java语言，除了提供最基本的语言特性之外，还额外提供了内容极为丰富的核心库，涵盖了网络通信、文件处理、加解密、序列化、日志等方方面面，几乎无所不包。但是，即便是如此完善、如此强大的Java官方核心库，仍有很多用户还是觉得不够用，或者说在自己特定的应用场景中不好用。因此，一部分具备编程能力的用户，根据自己的应用需求和场景特点，将某一部分的功能打造得非常强大，弥补或者超越了官方核心库。这些用户将这部分功能抽取和贡献出来，这就形成了公认的第三方类库，这些第三方类库和官方核心库一起，共同构成了繁荣的Java生态圈。JavaScript和Python语言同样也是如此。

我们再回到UiBot上来。前文提到过，UiBot本质上是一个平台工具，这个平台有几个特点，第一个特点就是**“强大”**，UiBot提供用以搭建RPA流程的零部件数量非常丰富，大大小小一整套的功能模块，从基础的键盘鼠标操作、各种界面元素操作，到常见办公软件、浏览器的自动化操作，从各种各样的数据处理，到文件处理、网络和系统操作等，一应俱全。但是这个平台还有第二个特点，那就是**“简单”**，UiBot将最通用、最常用、最基本、最核心的功能抽取出来，集成到平台中，形成一套简明、精干的核心库。如果一味地堆砌功能，将大大小小的所有功能一股脑地集成到UiBot的框架中，那UiBot的框架就会变得非常的臃肿，学习难度也会大幅上升。

那么，如果用户遇到了一个UiBot框架不能直接解决的问题，那么应该怎么办呢？类似于Java或JavaScript，UiBot也提供了**插件**机制，如果您擅长市面上的通用编程语言，那么您可以利用这些编程语言实现特定的功能，然后在UiBot中使用这个功能。

更有意思的是，UiBot还支持用多种不同的编程语言来编写插件。包括Python语言、Java语言、C#语言和C/C++语言。您可以在此范围内任意选择喜欢的语言，无论哪种语言编写的插件，在UiBot里面使用起来几乎没有差异。

当然，由于不同的编程语言之间有比较大的差异，使用不同的编程语言为UiBot编写插件的方法也是不一样的。本文分别介绍使用Python语言、Java语言、C#语言为UiBot编写插件的方法。考虑到C/C++语言比较难学，鉴于篇幅，本文就不对这两种语言的插件机制进行介绍了。

在以下描述中，经常会涉及到文件目录，如无特殊说明，均指相对于UiBot Creator/Worker安装目录的相对路径。为了便于书写，我们采用/符号来作为路径的分隔符，而不是Windows习惯的\符号。

8.1 用Python编写插件

8.1.1 编写方式

用Python编写UiBot插件是最简单的，只需要用任意文本编辑器书写扩展名为.py的文件（下文简称py文

件），并且保存为UTF-8格式，放置在extend/python目录下，即可直接以插件名.函数名的形式，调用这个py文件里面定义的函数。

注意，这里的**插件名**是指文件名去掉扩展名.py以后的部分，例如文件名为test.py，则插件名为test。

我们来看一个完整的例子：

1. **编写插件源代码**。打开extend/python目录，在这个目录下创建test.py文件，使用记事本打开test.py文件，写入如下内容：

```
def Add(n1, n2):
    return n1 + n2
```

2. 将test.py文件另存为UTF-8编码格式，如下图：

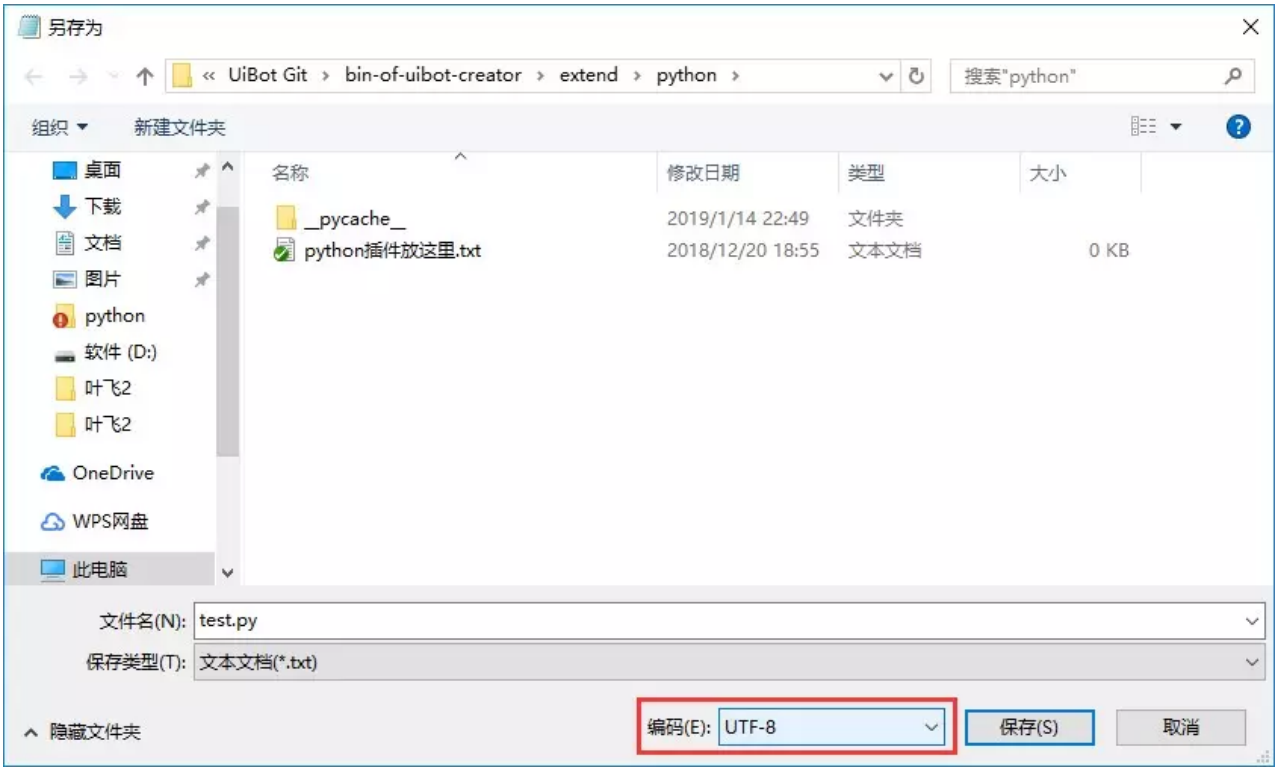

图 137: Python插件编写

3. **调用插件功能**。打开UiBot，新建一个流程，在源代码视图写入代码：

```
Traceprint test.add(1, 1)
```

4. **验证插件功能是否正确**。运行此流程，结果如下所示，代表插件调用正常。

图 138: Python插件运行结果

8.1.2 插件API

在用Python编写插件的时候，除了可以调用Python本身的功能之外，插件还可以反过来调用UiBot的一部分功能。我们称之这些被调用的功能为插件API。

插件API的调用方法如下：

1. 在Python插件中写入如下代码：

```
import UiBot
```

2. 直接调用插件API，例如：

```
def CommanderInfo():
    return UiBot.GetCommanderInfo()
```

目前Python插件中能使用的插件API包括：

- UiBot.IsStop()

这个函数用于检测当前流程是否需要马上停下来（比如用户按下了“停止”按钮）。当需要停下来时，返回`True`，否则返回`False`。

当某个插件函数需要执行比较长时间的时候，在执行过程中，如果用户决定停止流程，但插件函数还没有执行完成，流程将无法立即停下来。因此，建议在编写插件时，考虑到插件函数执行时间比较长的情况，并且在函数执行过程中定期的调用这个插件API，来确定流程是否要停下来。如果要停，应该立即退出插件函数。

- UiBot.GetString(string_path)

这个函数用于获得当前语言版本的某个字符串，参数是一个字符串路径（下面解释），返回值是获得的字符串。

我们在插件中可能会用到字符串，有的字符串内容是需要区分语言版本的。比如我们在插件中提示一个报错信息，这个报错信息应该包含中文版、英文版或其他语言版本。如果用户使用的是中文版的UiBot，那么就报中文的错误；如果用户使用的是英文版的UiBot，就报英文的错误。

如何做到这一点呢？我们可以在UiBot的安装目录下看到`lang/en-us/extend.json`和`lang/zh-cn/extend.json`这两个文件（其他语言版本也有类似的路径，不再赘述），分别表示插件中要用到的英文版和中文版的字符串。可以把我们要用的字符串的不同语言版本分别写到这些文件中去，然后在插件中用`UiBot.GetString()`来获得需要的字符串即可。

当然，UiBot的插件有很多，每个插件中也有很多字符串，这么多字符串都放在一个文件中，如何保证不冲突呢？很容易看到，这个文件是JSON格式的，其中用多个嵌套的JSON Object来区分不同的字符串。当我们需要使用一个字符串的时候，只需要在`UiBot.GetString()`的参数中填入**字符串路径**即可。所谓**字符串路径**，是指这个字符串所在的Object及其往上各级Object的Key的组合，其中用/分隔。比如`UiBot.GetString('App/Prefix')`获得的就是这个文件中，Key为'App'的JSON Objet中的Key为'Prefix'的字符串。

- UiBot.GetCommanderInfo()

当UiBot Worker在运行流程时，和UiBot Commander建立了连接，则可以通过这个API获得Commander的一些信息，如URL等。除了UiBot官方之外，一般用户的插件不会用到UiBot Commander，所以并不需要使用这个API。

8.1.3 插件的导入模块

单纯的一个py文件，功能往往比较有限。只有在py文件中通过import语句，导入其他的一些Python模块，其功能才更加丰富。

实际上，在UiBot安装目录的`lib/site-packages`路径下，已经预置了很多的Python模块（或者Python包。Python包和模块的定义和差异请查阅相关说明，本文不作解释）。这些模块都是在Python插件中可以直接使用的。如果我们在插件中还需要导入其他的模块，一种方式是将其放置在`lib/site-packages`路径下，还有一种方式是将其放置在`extend/python/<插件名>.lib`路径下。注意这里的`<插件名>.lib`也是一个目录，如果我们有个Python插件，文件名是test.py，则这个目录就是`test.lib`。

在编写插件时，我们更推荐把插件中导入的模块（假设这些模块是UiBot本身没有预置的）放在`extend/python/<插件名>.lib`路径下，而不是`lib/site-packages`路径下。因为`lib/site-packages`是一个公用目录，当我们删除掉一个插件的时候，很难从中分辨出到底哪个模块是被这个插件所使用的，而现在已经不再需要了。但如果把这些模块放在`extend/python/<插件名>.lib`路径下，就很清晰了，因为在删除插件时，只需要把和插件同名，且扩展名为`.lib`的目录一并删掉，就可保证不错不漏。

另外，值得注意的是：有的py文件会导入一些扩展名为pyd的模块，这些模块实际上是二进制格式的动态链接库。请注意动态链接库区分32位版本和64位版本，如果您使用的UiBot是32位版本，那么这些pyd模块也应该是32位版本的；否则，pyd模块就应该是64位版本的。

8.1.4 隐藏源代码

对于py文件来说，其源代码是完全公开的。如果我们既要让其他人使用我们编写的Python插件，又不希望被其他人看到插件的源代码，该怎么办呢？

我们只需要在UiBot Creator中至少调用一次这个插件，就会看到有一个`extend/python/__pycache__`目录被创建出来了。到这个目录里面去看一看，里面有一些以插件名开头，中间是诸如`.cpython-37`这样

的内容，以扩展名.pyc结束的文件。例如，我们的py文件为test.py，那么会自动创建这样的一个文件：extend/python/__pycache/test.cpython-37.pyc。

把这个文件改名为test.pyc，并且放在extend/python目录下，同时删除掉原来的test.py（删除前请自行备份），我们仍然可以在UiBot中使用test这个插件，且用法不变。因为它的代码已经以二进制的格式保存在test.pyc中了。我们只需要把这个文件发送给其他人去使用，就可以避免被人直接读到源代码。

当然，test.pyc实际上并不是加密的，仍然有可能被人反编译，得到一部分源代码。如果要做比较彻底的加密，还需要配合其他手段，本文不再赘述。

8.1.5 其他注意事项

1. 如果Python插件的函数中定义了N个参数，那么在UiBot中调用的时候，可以传入少于N个参数，多余的参数会自动补为None。但不可以传入多于N个参数。
2. 可以把UiBot中的数组或者字典类型作为参数，传入Python插件中，对应为Python中的list或dict类型。也可以把Python中的list, tuple或dict类型作为返回值，传回到UiBot，前两者都被转换为数组类型，后者被转换为字典类型。无论传入参数，还是返回值，这些复合类型在Python插件和UiBot之间都采用值传递的方式，而不是引用传递的方式。
3. 可以在Python插件的函数中抛出异常，异常可以由Python插件自行捕获，也可以不捕获。如果Python插件不捕获，那么异常会自动被传到UiBot中，UiBot可以捕获。如果UiBot也不捕获，那么流程的运行会出错退出，并且会在出错信息中说明是由于Python插件中的异常导致的，以便排查问题。
4. UiBot中已经内置了Python的运行环境，无需额外安装Python。即使安装了，UiBot也不会使用您安装的Python。目前UiBot内置的Python是3.7.1版本。
5. Python中的变量、函数都是区分大小写的，但在UiBot中使用Python插件时，仍然可以不区分大小写的调用其中的函数。比如，在前面的例子中，可以在UiBot中写test.add(1,1)，也可以写Test.ADD(1,1)，其效果完全一样。
6. 可以在Python中使用全局变量，比如可以把变量写到函数之外。全局变量的值被Python插件中的所有函数所共享，但不同的插件不共享全局变量。
7. 使用Python编写UiBot插件很容易，但Python本身是一门独立的编程语言，使用文本编辑器开发和调试都很不方便，因此建议使用集成开发环境，例如Visual Studio Code进行Python插件开发。

8.2 用Java编写插件

8.2.1 编写方式

从UiBot Creator 5.0版开始，支持用Java语言写UiBot的插件。用过Java的读者都知道，Java的源代码文

件一般以.java扩展名结尾，需要先用**JDK**（Java Development Kit）编译成扩展名为.class的字节码（Byte Code）文件，然后才能运行。运行的时候不一定要安装JDK，也可以只安装**JRE**（Java Runtime Environment）。

由于版权的限制，UiBot中没有内置JDK，但内置了由Oracle公司发布的JRE 1.7版本。所以，为了用Java编写插件，需要您自行下载和安装Oracle JDK 1.7版本。下载和安装的方法在互联网上有大量资料讲述，本文不再重复。

为了方便您用Java语言写UiBot的插件，我们设计了一个插件的例子并将其源码放在GitHub上，点击这里即可获取。如果您习惯使用git，也可以从这个URL拉取：https://github.com/Laiye-UiBot/extend-example。后续内容将围绕这个例子展开。

按照Java语言的规范，首先我们需要设计一个插件名，然后将源代码文件命名为<插件名>.java，并在文件中写一个Java类，这个类的名字也必须是插件名。在例子中，我们可以看到插件名为JavaPlugin，所以源代码文件名必须是JavaPlugin.java，而在这个文件中会定义一个名为JavaPlugin的类：

```
public class JavaPlugin
{
}
```

为了让UiBot能够正常使用这个类，这个类必须是public的，也不能包含在任何包里。类里面可以定义public、private或protected的函数，但只有public函数是UiBot可以直接调用的。比如，我们在例子中定义了一个叫Add的函数，这个函数是public的，所以，可以在UiBot中调用它。

怎么调用呢？需要先用JDK中的javac程序，编译这个源码文件，在命令行输入：

```
javac -encoding utf8 JavaPlugin.java
```

当然，这里需要javac程序在当前的搜索路径下。另外，例子中的JavaPlugin.java是UTF-8编码的，且里面有中文字符，所以需要加上-encoding utf8的选项。如果没有中文字符，则此选项可以省略。

如果编译成功，会自动生成名为JavaPlugin.class的文件，把这个文件放到extend/java目录下，然后就可以像使用Python插件一样的使用它。例如，我们可以打开UiBot，新建一个流程，在源代码视图写入代码：

```
Traceprint javaPlugin.add(1, 1)
```

运行此流程，结果如下所示，代表插件调用正常。

图 139: Java插件运行结果

8.2.2 插件API

和Python插件类似，在Java插件中，也可以使用插件API，反过来调用UiBot的一部分功能。如果要调用插件API，无需import任何包，只需要在编译Java插件的时候，把插件例子中的UiBot目录复制到Java插件源代码所在目录下即可。

目前Java插件中能使用的插件API包括：

- UiBot.API.IsStop()

用于检测当前流程是否需要马上停下来（比如用户按下了“停止”按钮）。当需要停下来时，返回`True`，否则返回`False`。

其具体作用请参考Python插件中使用的UiBot.IsStop()函数。

- UiBot.API.GetString(string_path)

用于获得当前语言版本的某个字符串，参数是一个字符串路径（下面解释），返回值是获得的字符串。

其具体作用请参考Python插件中使用的UiBot.GetString()函数。另外，在插件例子中，我们也使用到了这个API，来获得字符串路径为’Excel/SaveBook’的字符串，即extend.json文件中，名为’Excel’的JSON Object中的名为’SaveBook’的字符串。

下面这段UiBot源代码会先调用Java插件中的GetString()函数，再反过来调用UiBot中的UiBot.API.GetString()。您可以在UiBot中输入这段代码，运行试试，看能得到什么结果。

```
Traceprint JavaPlugin.getString()
```

- UiBot.API.GetCommanderInfo()

当UiBot Worker在运行流程时，和UiBot Commander建立了连接，则可以通过这个API获得Commander的一些信息。除UiBot官方之外，一般用户的插件不会用到UiBot Commander，所以并不需要使用这个API。

8.2.3 变量的传递

Java是静态类型的编程语言，也就是说，变量在使用之前需要先定义，且定义时必须指定变量的类型（整数、浮点数、字符串等），在运行的时候，变量也只能是指定的这种类型。而且，数组中通常只能包含同一种类型的数据。

但这与UiBot有很大的不同，UiBot的变量是动态类型的，可以不指定类型，运行的时候还可以随意更换类型，数组中也可以包含各种不同类型的数据。

所以，为了在UiBot中顺利使用Java插件，需要符合以下规定：

- 如果Java插件的参数是整数、浮点数、字符串、布尔类型，UiBot传入的参数也必须是同样的类型（除了下面几条所述的例外情况），否则会出错

- 如果Java插件的参数是浮点数，可以传入整数，不会出错。但反之不成立，也就是说，如果Java插件的参数是整数，不能传入浮点数
- 如果Java插件的参数是长整数型（long），可以传入小于 2^31 的整数，不会出错。但反之不成立，也就是说，如果Java插件的参数是整数型（int），不能传入大于等于 2^31 的整数
- 如果需要把字典或数组类型从UiBot中传到Java插件中，Java插件中的参数类型只能使用org.json.JSONArray（对应数组）或者org.json.JSONObject（对应字典）
- 如果需要把字典或数组类型从Java插件中传到UiBot中，Java插件中的返回值类型只能使用org.json.JSONArray或者org.json.JSONObject。UiBot会自动把org.json.JSONArray类型的返回值转换成UiBot中的数组，而把org.json.JSONObject类型的返回值转换成UiBot中的字典
- 无论传入参数，还是返回值，这些复合类型在Java插件和UiBot之间都采用值传递的方式，而不是引用传递的方式
- 可以在Java源代码中写`import org.json.*;`，这样就可以直接使用JSONArray或者JSONObject类型，避免`org.json`的前缀。在插件例子中就是这样写的。另外，org.json这个包已经被UiBot包含在运行环境中了，无需额外下载和安装。

在插件例子中，有一个Concat函数，用于演示如何把两个数组从UiBot传到Java插件中，又如何把两个数组连接后的结果返回到UiBot中。建议读者仔细阅读。

8.2.4 插件的引用模块

和Python类似，单纯的一个Java文件，功能往往比较有限。只有在源代码中通过import语句，导入其他的一些Java包，其功能才更加丰富。

我们在UiBot中已经内置了Oracle JRE 1.7，当然也包含了JRE中自带的包，比如`com.sun.javafx`等。此外，我们还在UiBot中内置了要用到的org.json包。除了上述内容以外，其他第三方的Java包可以以`.class`格式的文件存在，也可以以`.jar`格式的文件存在。熟悉Java语言的读者，对以上两种文件应该有足够的了解，前者是Java代码的Byte Code，后者是多个Byte Code打包而成的压缩文件。

在启动UiBot的时候，会自动把`extend/java`目录加入到Java的classpath中。此外，当加载一个Java插件的时候，还会把`extend/java/<`插件名`>.lib`这个目录，以及这个目录下**所有**扩展名为`.jar`的文件，都自动加入到Java的classpath中。比如，我们有个Java插件，名为`A.class`，且放置在`extend/java`目录下。那么，`extend/java`目录、`extend/java/A.lib`目录、以及`extend/java/A.lib/*.jar`，都会加入classpath中。我们的插件中如果需要引用任何第三方的Java包，只要把包放置在这些路径下，并且符合Java的classpath规范，即可使用。

8.2.5 其他注意事项

1. Java插件中的函数不支持可变参数或默认参数，在调用时必须传入指定数量、指定类型的参数。
2. 可以在Java插件的函数中抛出异常，异常可以由Java插件自行捕获，也可以不捕获。如果Java插件不捕获，那么异常会自动被传到UiBot中，UiBot可以捕获。如果UiBot也不捕获，那么流程的运行会出错退出，并且会在出错信息中说明是由于Java插件中的异常导致的，以便排查问题。

3. Java中的变量、函数都是区分大小写的，但在UiBot中使用Java插件时，仍然可以不区分大小写的调用其中的函数。比如，在前面的例子中，可以在UiBot中写`javaPlugin.add(1,1)`，也可以写`JavaPlugin.ADD(1,1)`，其效果完全一样。

4. 在写Java插件的时候，实际上是定义了一个Java类，并且把类里面的public函数给UiBot去调用。这个类可以有构造函数，也可以有成员变量，它们的初始化都会在流程刚刚开始运行的时候自动完成。

5. UiBot中内置了Oracle JRE 1.7版本，您需要自行下载Oracle JDK 1.7版本去编译Java插件。虽然有时JDK和JRE的版本不一致也可以工作，但为了减少麻烦，还是推荐用同一版本。另外，也推荐使用集成开发环境，来进行Java插件的开发，例如Eclipse或IntelliJ IDEA。

8.3 用C#.Net编写插件

8.3.1 编写方式

UiBot本身的部分代码就是基于微软的.Net框架，用C#语言编写的。所以，也可以用C#语言编写UiBot的插件（以下简称为.Net插件）。实际上，微软的.Net框架支持多种编程语言，包括VB.Net、C++/CLI等等，这些编程语言都遵循.Net框架的规范，它们都可以用来编写.Net插件，但因为C#是微软主推的编程语言，所以本文用C#举例，有经验的读者亦可将其移植到.Net框架上的其他语言。另外，UiBot对.Net插件的支持也是在不断升级的，本文以UiBot Creator 5.1版为例，如果在老版本的UiBot上，一些例子可能无法正常运行，请及时升级。

为了方便您用C#语言写.Net插件，我们设计了一个插件的模板，并将其源码放在GitHub上，点击这里即可获取。如果您习惯使用git，也可以从这个URL拉取：https://github.com/Laiye-UiBot/extend-example。建议您在写.Net插件的时候，直接在这个模板的基础上写，而无需从头开始。后续讲述的内容，也将围绕这个模板中的例子展开。

和Java插件类似，.Net插件也需要编译成扩展名为`.dll`的文件，才能被UiBot使用。微软的集成开发环境Visual Studio兼具编写和编译的功能，并且也提供了免费的社区版，推荐下载使用。我们提供的模板是基于Visual Studio 2015版本的，您可以选择这个版本，也可以选更高版本的Visual Studio，但不建议使用低于2015版本的Visual Studio。

安装了Visual Studio，并下载了我们的.Net插件模板后，可以双击`UiBotPlugin.sln`文件，这是一个“解决方案”，名字起得很唬人，实际上就是多个相关联的文件的集合。用Visual Studio打开这个解决方案后。可以看到，里面包含了很多内容，其中唯一需要我们动手修改的是UiBotPlugin.cs文件，其他的文件、引用、Properties等都可以不去动。如下图：

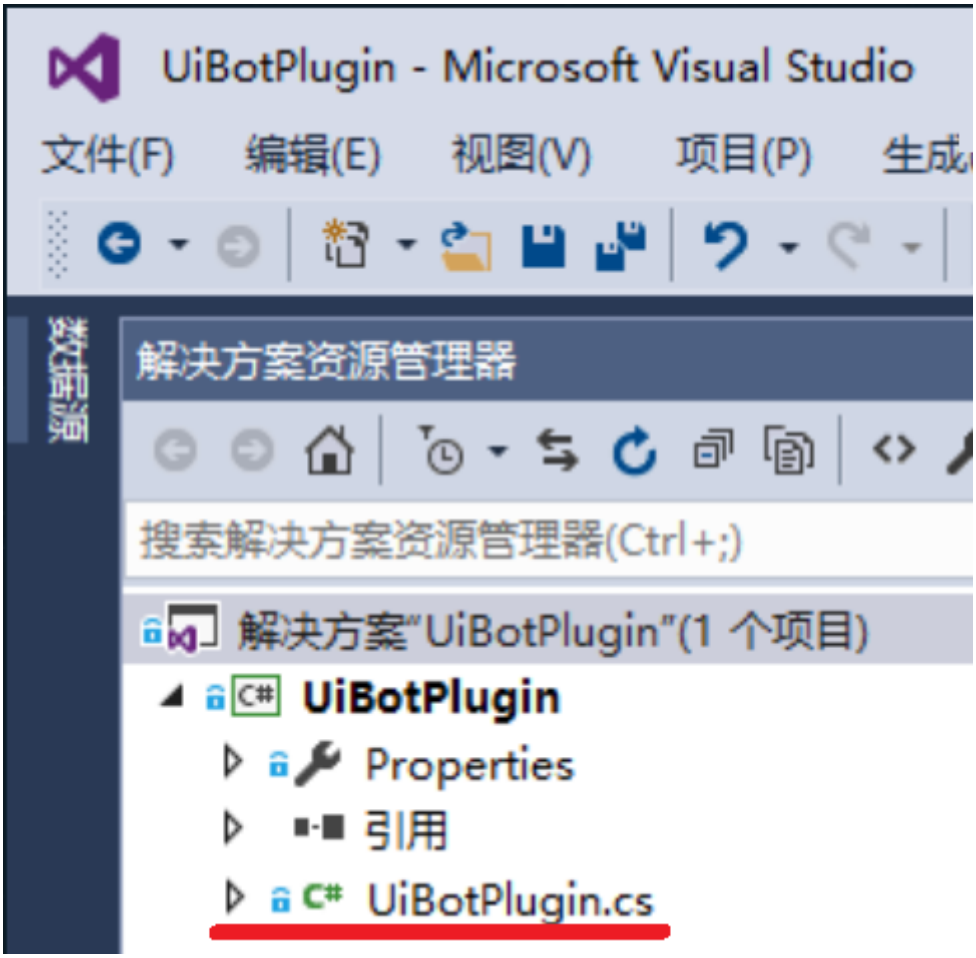

图 140: .Net插件的模板

在`UiBotPlugin.cs`文件里，有一个叫`UiBotPlugin`的命名空间，其中包含了一个接口（interface）和一个类（class）。为了避免混淆，我们推荐把这个命名空间的名字改为您的插件名。比如最终的插件文件是`DotNetPlugin.dll`，那么插件名就是`DotNetPlugin`，这个命名空间的名字也改为`DotNetPlugin`为宜。

从模板中可以看出：在接口里面声明了三个函数，在类里面写了这三个函数的实现。这三个函数都是例子，您随时可以把它们的声明和实现都删掉，加入您自己的插件函数。但请特别注意：在加入函数的时候，也要保持类似的写法，需要在接口中声明，在类中实现，否则，UiBot不能正常识别这个插件函数。

我们用例子中的Add函数为例，尝试编译插件，并在UiBot中调用这个函数：

1. 选择Visual Studio的“生成”（Build）菜单项，编译这个解决方案之后，会看到在插件的目录下有个叫Release的目录，里面产生了一个叫UiBotPlugin.dll的文件。
2. 把这个文件手动改名为您的插件名，并保留`.dll`的扩展名。如改名为`DotNetPlugin.dll`。
3. 把这个文件放到UiBot的`extend/DotNet`目录下。
4. 打开UiBot，新建一个流程，在源代码视图写入代码：

```
Traceprint DotNetPlugin.add(1, 1)
```

运行此流程，结果如下所示，代表插件调用正常。

图 141: .Net插件运行结果

您可能注意到了，在前面的Python插件、Java插件的例子中，都有Add这个例子函数，而除了插件名之外，UiBot调用它们的方式和运行结果都没有区别。实际上，不同的插件内部实现是有很大差异的，比如在Python语言里，默认用UTF-8编码来保存字符串，而在.Net里默认用UTF-16保存。但UiBot已经帮您抹平了这些差异，让您在使用的过程中不必关心这些细节。

8.3.2 插件API

和Python、Java插件类似，在.Net插件中，也可以使用插件API，反过来调用UiBot的一部分功能。如果要调用插件API，只需要基于UiBot提供的模板编写插件即可，无需做其他任何设置。

.Net插件中能使用的插件API的名字、参数和含义都和Java插件完全一致，例如，可以用`UiBot.API.IsStop()`来检测当前流程是否需要马上停下来，等等。请参考Java插件的中关于插件API的讲解，不再赘述。

在模板中，您可能会看到一个名叫`DotNetAdapter.dll`的文件。实际上，这个文件是UiBot每个版本都包含的。从UiBot 5.1版开始，您调用的.Net版的插件API，实际上都在这个文件里面实现。因此，当您的插件发布的时候，并不需要包含这个文件，因为UiBot已经自带了。

同时，如果您的UiBot更新到了更高的版本，`DotNetAdapter.dll`中也可能会包含了更多的插件API。您可以自行从UiBot中拿到新版本的`DotNetAdapter.dll`文件，并放在您编写的插件的源代码所在的目录下，即可使用到新版的插件API。

8.3.3 变量的传递

和Java类似，C#.Net也是静态类型的编程语言，变量在使用之前需要先定义，且定义时必须指定变量的类型。而且，数组中通常只能包含同一种类型的数据。这与UiBot的动态类型有很大的不同。

因此，在编写和使用.Net插件的时候，需要符合以下规定：

- 对于整数、浮点数、字符串、布尔类型等基本类型的参数，UiBot对.Net插件的类型检查不是很严格，它会尽量进行转换，即使转换不成功，也不会报错。所以，请在使用时特别留意每个参数的类型，避免传入了不正确的值，而没有及时发现。
- 如果需要把字典或数组类型从UiBot中传到.Net插件中，.Net插件中的参数类型只能使用Newtonsoft.Json.Linq.JA（对应数组）或者Newtonsoft.Json.Linq.JObject（对应字典）。在模板中，由于我们已经写了`using`

`Newtonsoft.Json.Linq;`，所以可以省略前缀，简写为JArray（对应数组）或JObject（对应字典），下文亦使用此简化写法。

- 如果需要把字典或数组类型从.Net插件中传到UiBot中，.Net插件中的返回值类型只能使用JArray（对应数组）或JObject（对应数组）。UiBot会自动把JArray类型的返回值转换成UiBot中的数组，而把 JObject类型的返回值转换成UiBot中的字典。
- 无论传入参数，还是返回值，这些复合类型在.Net插件和UiBot之间都采用值传递的方式，而不是引用传递的方式。

在插件模板中，有一个作为例子的Concat函数，用于演示如何把两个数组从UiBot传到.Net插件中，又如何把两个数组连接后的结果返回到UiBot中。建议读者仔细阅读。

8.3.4 插件的引用模块

UiBot本身是依赖于.Net Framework的，并且假设用户已经安装了.Net Framework 4.5.2（含）以上的版本。如果没有安装.Net Framework，或者版本不对，UiBot本身都不能运行，当然就更不能使用您编写的插件了。所以，在编写插件的时候，只要您的插件依赖的也是.Net Framework 4.5.2版本，就不必担心环境不匹配的问题。

微软已经在.Net Framework里面内置了非常丰富的功能，但难免有的功能仍然没有包含，需要引用第三方的.Net dll文件。

和Java插件类似，UiBot在加载一个.Net插件的时候，如果这个.Net插件引用了其他第三方的.Net dll文件，UiBot首先会试图到.Net插件所在的目录下去搜索被引用的dll文件。如果没有找到，还会再到`<插件名>.lib`这个目录下去找一次。比如，我们有个.Net插件，名为`A.dll`，放置在`extend/DotNet`目录中，且引用了`B.dll`。那么UiBot会先尝试找`extend/DotNet/B.dll`，再尝试找`extend/DotNet/A.lib/B.dll`。如果这两个目录下都没有找到，会抛出异常。

8.3.5 其他注意事项

1. JArray和JObject并不是.Net Framework里面自带的，而是使用了开源的Json.Net。在编译和运行的时候，都需要依赖一个名为`Newtonsoft.Json.dll`的文件。在UiBot提供的模板中，已经包含了这个文件。同时，在每个版本的UiBot中，也会自带这个文件。因此，您可以直接使用JArray和JObject，而并不需要把这个文件包含在插件当中。

2. 在编译插件的时候，编译器可能会警告“`DotNetAdapter`的处理器架构不匹配”之类的信息。实际上没有影响，无需理睬这个警告。

3. .Net插件中的函数支持默认参数。在调用时，如果某些参数有默认值，则可以不传值，此参数会自动取默认值。

4. 可以在.Net插件的函数中抛出异常，异常可以由.Net插件自行捕获，也可以不捕获。如果.Net插件不捕获，那么异常会自动被传到UiBot中，UiBot可以捕获。如果UiBot也不捕获，那么流程的运行会出错退出，并且会在出错信息中说明是由于.Net插件中的异常导致的，以便排查问题。

5. .Net中的变量、函数都是区分大小写的，但在UiBot中使用.Net插件时，仍然可以不区分大小写的调用其中的函数。比如，在前面的例子中，可以在UiBot中写`DotNet.add(1,1)`，也可以写`dotnet.ADD(1,1)`，其效果完全一样。

8.4 插件的分享

无论是用哪种语言编写插件，都可以放在`extend`目录的相应路径中，例如Python插件放在`extend/python`目录中，即可直接使用。

但这种情况只适合您自己使用，不能分享给其他人。另外，使用了这个插件的流程，如果打包在UiBot Worker上运行的时候，并不会把插件打包进去，而需要单独把插件复制到UiBot Worker的`extend`目录的相应路径中。

如果您的插件确实具有分享的意义，我们推荐您把插件分享到UiBot命令中心，供互联网上的其他人学习使用。您也可以在UiBot命令中心找到其他人分享的一些插件。在UiBot Worker上运行的时候，如果用到了这些插件，也不需要把它们复制到UiBot Worker中。

假设我们用Python语言写了一个“四则运算”的插件，文件名为`Arith.py`，其内容如下：

```
def Add(a, b):
    return a+b

def Sub(a, b):
    return a-b

def Mul(a, b):
    return a*b

def Div(a, b):
    return a/b
```

按照如下的步骤，可以把这个插件分享给其他用户：

1. 用UiBot Creator打开任意一个流程，然后再打开任意一个流程块；
2. 在左侧的面板中找到“UiBot命令中心”的按钮，点击此按钮，选择“自定义”下面的“自定义插件”，如下图中红框所示；
3. 选择“新增自定义插件”，并在对话框中，参考图示内容进行填写：

图 142: 新增自定义插件

由于我们的插件文件名为`Arith.py`，所以，主文件名就是`Arith`。可以把`Arith.py`和它依赖的文件都压缩成一个zip文件（可采用任意第三方压缩工具，或者Windows自带的压缩功能来完成），这个zip文件的命名随意。然后选中这个zip文件，再填好主文件名，最关键的信息就填完了。其他的栏目如“插件名称”、“插件主页”、“插件描述”都是供使用者阅读的，按照您的习惯详略得当的填写即可。

4. 保存之后，一个插件就添加完成了，可以在界面上看到这个新增插件的信息。为了让使用者能够更清晰的了解这个插件的具体用法，我们还有必要对插件中的每个函数（在使用者看来，是每个命令）进行相关配置。配置方法如下：

- 在界面上可以看到，添加完成插件之后，会多出一个“新增命令”的按钮。这个按钮允许我们为插件中的函数（在使用者看来是“命令”）增加一些配置信息，以便使用。
- 点击“新增命令”按钮，进入新增命令页面，左侧是关于命令的说明信息，请根据页面提示和要求填写。其中，可视化翻译中填写的“把 %1% 和 %2% 两个数字相加”，“ %1% ”和“ %2%”分别表示读取第一个和第二个属性。

图 143: 新增命令

- 还可以进一步填写命令的属性信息。找到右侧的“属性编辑”窗口。点击“必选属性”右侧的“添加”按钮，通过“属性编辑”的对话框添加第一个属性“a”及其属性说明，参数类型选“数字”；组件类型选择“输入框”；默认值填写0。同理，添加第二个属性“b”。这两个值分别代表被加数和加数。

图 144: 新增属性

- 陆续加入其它命令的信息，让使用者在用到您的插件时，能够在可视化视图中清晰的显示出命令的含义，并清晰的说明每项属性的含义，使用起来才足够容易。

到此为止，我们已经做好了插件发布的一切准备工作，点击对话框右上方的“安装调试”按钮，即可把这

个插件安装到当前流程中。回到编写流程的界面中，即可在左侧的命令列表中找到我们添加的“四则运算”，以及里面的四条命令：我的加法、我的减法、我的乘法和我的除法。对于使用者来说，就像使用一般的UiBot预制命令一样，拖动到UiBot的可视化视图中，并配置其属性即可。

我们来尝试一下，拖入“我的加法”命令，同时在属性栏里面，设置第一个加数为“2”，第二个加数为“3”。如图所示：

图 145: “我的加法”命令属性设置

还可以再拖入一条“向调试窗口输出”命令，把结果输出出来。这样一来，可视化视图的显示效果类似于下图所示。可见我们设置的信息生效了，在使用插件的时候，插件的翻译是易于理解的形式。

图 146: 在可视化视图中的显示

运行这个流程，可以看到输出栏的结果符合预期，说明这个自定义命令达到了要求，可以发布。

接下来，再次打开“UiBot命令中心”，找出我们刚才添加的插件，并点击“发布”，在“发布命令”页面中填写说明和版本号之后，点击“提交”按钮。之后请等待UiBot官方的审核，如审核通过，就可以在命令中心的“可安装”一栏中找到您发布的插件及其命令了。

除了您发布的插件之外，还能看到其他用户发布的插件，包含了各种特定的功能。您可以选择喜欢的插件进行安装，并在您的流程中使用。

值得说明的是，和前文中提到的“命令库”类似:如果我们在编写流程块的时候，使用了一个插件或命令库，这个插件或命令库在当前流程里面的所有流程块中都是可用的。但如果换了另外一个流程，就需要重新安装或者导入了；另外，使用了插件或命令库的流程，在打包给UiBot Worker或者UiBot Store使用的时候，命令库会被自动打包，而不需要我们再做额外的处理。