基于控件的操作

基于控件的操作指的是选择屏幕上的控件，获取其信息或对其进行操作。对于一般软件而言，基于控件的操作对不同机型有很好的兼容性；但是对于游戏而言，由于游戏界面并不是由控件构成，无法采用本章节的方法，也无法使用本章节的函数。有关游戏脚本的编写，请参考《基于坐标的操作》。

基于控件的操作依赖于无障碍服务，因此最好在脚本开头使用 `auto()` 函数来确保无障碍服务已经启用。如果运行到某个需要权限的语句无障碍服务并没启动，则会抛出异常并跳转到无障碍服务界面。这样的用户体验并不好，因为需要重新运行脚本，后续会加入等待无障碍服务启动并让脚本继续运行的函数。

您也可以在脚本开头使用 `"auto";` 表示这个脚本需要无障碍服务，但是不推荐这种做法，因为这个标记必须在脚本的最开头(前面不能有注释或其他语句、空格等)，我们推荐使用 `auto()` 函数来确保无障碍服务已启用。

auto([mode])

- `mode` {string} 模式

检查无障碍服务是否已经启用，如果没有启用则抛出异常并跳转到无障碍服务启用界面；同时设置无障碍模式为 mode。mode 的可选值为：

- `fast` 快速模式。该模式下会启用控件缓存，从而选择器获取屏幕控件更快。对于需要快速的控件操作的脚本可以使用该模式，一般脚本则没有必要使用该函数。
- `normal` 正常模式，默认。

如果不加 mode 参数，则为正常模式。

建议使用 `auto.waitFor()` 和 `auto.setMode()` 代替该函数，因为 `auto()` 函数如果无障碍服务未启动会停止脚本；而 `auto.waitFor()` 则会在在无障碍服务启动后继续运行。

示例：

```js
auto("fast");
```

示例 2：

```js
auto();
```

auto.waitFor()

检查无障碍服务是否已经启用，如果没有启用则跳转到无障碍服务启用界面，并等待无障碍服务启动；当无障碍服务启动后脚本会继续运行。

auto.setMode(mode)

- `mode` {string} 模式

设置无障碍模式为 mode。mode 的可选值为：

- `fast` 快速模式。该模式下会启用控件缓存，从而选择器获取屏幕控件更快。对于需要快速的控件查看和操作的脚本可以使用该模式，一般脚本则没有必要使用该函数。
- `normal` 正常模式，默认。

SimpleActionAutomator

> **稳定性: 稳定**

SimpleActionAutomator 提供了一些模拟简单操作的函数，例如点击文字、模拟按键等。这些函数可以直接作为全局函数使用。

click(text[, i])

- `text` {string} 要点击的文本
- `i` {number} 如果相同的文本在屏幕中出现多次，则 i 表示要点击第几个文本, i 从 0 开始计算

返回是否点击成功。当屏幕中并未包含该文本，或者该文本所在区域不能点击时返回 false，否则返回 true。

该函数可以点击大部分包含文字的按钮。例如微信主界面下方的"微信", "联系人", "发现", "我"的按钮。
通常与 while 同时使用以便点击按钮直至成功。例如:

```js
while(!click("扫一扫"));
```

当不指定参数 i 时则会尝试点击屏幕上出现的所有文字 text 并返回是否全部点击成功。

i 是从 0 开始计算的, 也就是，`click("啦啦啦", 0)` 表示点击屏幕上第一个"啦啦啦", `click("啦啦啦", 1)` 表示点击屏幕上第二个"啦啦啦"。

> **文本所在区域指的是，从文本处向其父视图寻找，直至发现一个可点击的部件为止。**

click(left, top, bottom, right)

- `left` {number} 要点击的长方形区域左边与屏幕左边的像素距离
- `top` {number} 要点击的长方形区域上边与屏幕上边的像素距离
- `bottom` {number} 要点击的长方形区域下边与屏幕下边的像素距离
- `right` {number} 要点击的长方形区域右边与屏幕右边的像素距离

注意，该函数一般只用于录制的脚本中使用，在自己写的代码中使用该函数一般不要使用该函数。

点击在指定区域的控件。当屏幕中并未包含与该区域严格匹配的区域，或者该区域不能点击时返回 false，否则返回 true。

有些按钮或者部件是图标而不是文字（例如发送朋友圈的照相机图标以及 QQ 下方的消息、联系人、动态图标），这时不能通过 `click(text, i)` 来点击，可以通过描述图标所在的区域来点击。left, bottom, top, right 描述的就是点击的区域。

至于要定位点击的区域，可以在悬浮窗使用布局分析工具查看控件的 bounds 属性。

通过无障碍服务录制脚本会生成该语句。

longClick(text[, i]))

- `text` {string} 要长按的文本
- `i` {number} 如果相同的文本在屏幕中出现多次，则 i 表示要长按第几个文本, i 从 0 开始计算

返回是否点击成功。当屏幕中并未包含该文本，或者该文本所在区域不能点击时返回 false，否则返回 true。

当不指定参数 i 时则会尝试点击屏幕上出现的所有文字 text 并返回是否全部长按成功。

scrollUp([i])

- `i` {number} 要滑动的控件序号

找到第 i+1 个可滑动控件上滑或**左滑**。返回是否操作成功。屏幕上没有可滑动的控件时返回 false。

另外不加参数时 `scrollUp()` 会寻找面积最大的可滑动的控件上滑或左滑，例如微信消息列表等。

参数为一个整数 i 时会找到第 i + 1 个可滑动控件滑动。例如 `scrollUp(0)` 为滑动第一个可滑动控件。

scrollDown([i])

- `i` {number} 要滑动的控件序号

找到第 i+1 个可滑动控件下滑或**右滑**。返回是否操作成功。屏幕上没有可滑动的控件时返回 false。

另外不加参数时 `scrollUp()` 会寻找面积最大的可滑动的控件下滑或右滑。

参数为一个整数 i 时会找到第 i + 1 个可滑动控件滑动。例如 `scrollUp(0)` 为滑动第一个可滑动控件。

setText([i,]text)

- i {number} 表示要输入的为第 i + 1 个输入框

- text {string} 要输入的文本

返回是否输入成功。当找不到对应的文本框时返回 false。

不加参数 i 则会把所有输入框的文本都置为 text。例如 `setText("测试")` 。

这里的输入文本的意思是，把输入框的文本置为 text，而不是在原来的文本上追加。

input([i,]text)

- i {number} 表示要输入的为第 i + 1 个输入框
- text {string} 要输入的文本

返回是否输入成功。当找不到对应的文本框时返回 false。

不加参数 i 则会把所有输入框的文本追加内容 text。例如 `input("测试")` 。

UiSelector

UiSelector 即选择器，用于通过各种条件选取屏幕上的控件，再对这些控件进行点击、长按等动作。这里需要先简单介绍一下控件和界面的相关知识。

一般软件的界面是由一个个控件构成的，例如图片部分是一个图片控件(ImageView)，文字部分是一个文字控件(TextView)；同时，通过各种布局来决定各个控件的位置，例如，线性布局(LinearLayout)里面的控件都是按水平或垂直一次叠放的，列表布局(AbsListView)则是以列表的形式显示控件。

控件有各种属性，包括文本(text), 描述(desc), 类名(className), id 等等。我们通常用一个控件的属性来找到这个控件，例如，想要点击 QQ 聊天窗口的"发送"按钮，我们就可以通过他的文本属性为"发送"来找到这个控件并点击他，具体代码为:

```js
var sendButton = text("发送").findOne();
sendButton.click();
```

在这个例子中，`text("发送")` 表示一个条件(文本属性为"发送"), `findOne()` 表示基于这个条件找到一个符合条件的控件，从而我们可以得到发送按钮 sendButton，再执行 `sendButton.click()` 即可点击"发送"按钮。

用文本属性来定位按钮控件、文本控件通常十分有效。但是，如果一个控件是图片控件，比如 Auto.js 主界面右上角的搜索图标，他没有文本属性，这时需要其他属

性来定位他。我们如何查看他有什么属性呢？首先打开悬浮窗和无障碍服务，点击蓝色的图标(布局分析), 可以看到以下界面：

之后我们点击搜索图标，可以看到他有以下属性：

我们注意到这个图标的 desc(描述)属性为"搜索"，那么我们就可以通过 desc 属性来定位这个控件，得到点击搜索图标的代码为：

```js
desc("搜索").findOne().click();
```

可能心细的你可能注意到了，这个控件还有很多其他的属性，例如 checked, className, clickable 等等，为什么不用这些属性来定位搜索图标呢？答案是，其他控件也有这些值相同的属性、尝试一下你就可以发现很多其他控件的 checked 属性和搜索控件一样都是 `false` ，如果我们用 `checked(false)` 作为条件，将会找到很多控件，而无法确定哪一个是搜索图标。因此，要找到我们想要的那个控件，**选择器的条件通常需要是可唯一确定控件的**。我们通常用一个独一无二的属性来定位一个控件，例如这个例子中就没有其他控件的 desc(描述)属性为"搜索"。

另外，对于这个搜索图标而言，id 属性也是唯一的，我们也可以用 `id("action_search").findOne().click()` 来点击这个控件。如果一个控件有 id 属性，那么这个属性很可能是唯一的，除了以下几种情况：

- QQ 的控件的 id 属性很多都是"name"，也就是在 QQ 界面难以通过 id 来定位一个控件
- 列表中的控件，比如 QQ 联系人列表，微信联系人列表等

尽管 id 属性很方便，但也不总是最方便的，例如对于微信和网易云音乐，每次更新他的控件 id 都会变化，导致了相同代码对于不同版本的微信、网易云音乐并不兼容。

除了这些属性外，主要还有以下几种属性：

- `className` 类名。类名表示一个控件的类型，例如文本控件为"android.widget.TextView", 图片控件为"android.widget.ImageView"等。
- `packageName` 包名。包名表示控件所在的应用包名，例如 QQ 界面的控件的包名为"com.tencent.mobileqq"。
- `bounds` 控件在屏幕上的范围。
- `drawingOrder` 控件在父控件的绘制顺序。
- `indexInParent` 控件在父控件的位置。
- `clickable` 控件是否可点击。
- `longClickable` 控件是否可长按。

- `checkable` 控件是否可勾选。
- `checked` 控件是否可已勾选。
- `scrollable` 控件是否可滑动。
- `selected` 控件是否已选择。
- `editable` 控件是否可编辑。
- `visibleToUser` 控件是否可见。
- `enabled` 控件是否已启用。
- `depth` 控件的布局深度。

有时候只靠一个属性并不能唯一确定一个控件，这时需要通过属性的组合来完成定位，例如 `className("ImageView").depth(10).findOne().click()` ，通过链式调用来组合条件。

通常用这些技巧便可以解决大部分问题，即使解决不了问题，也可以通过布局分析的"生成代码"功能来尝试生成一些选择器代码。接下来的问题便是对选取的控件进行操作，包括：

- `click()` 点击。点击一个控件，前提是这个控件的 clickable 属性为 true
- `longClick()` 长按。长按一个控件，前提是这个控件的 longClickable 属性为 true
- `setText()` 设置文本，用于编辑框控件设置文本。
- `scrollForward()` ，`scrollBackward()` 滑动。滑动一个控件(列表等), 前提是这个控件的 scrollable 属性为 true
- `exits()` 判断控件是否存在
- `waitFor()` 等待控件出现

这些操作包含了绝大部分控件操作。根据这些我们可以很容易写出一个"刷屏"脚本(代码仅为示例，请不要在别人的群里测试，否则容易被踢):

```js
while(true){
    className("EditText").findOne().setText("刷屏...");
    text("发送").findOne().clicK();
}
```

上面这段代码也可以写成：

```js
while(true){
    className("EditText").setText("刷屏...");
```

```
    text("发送").clicK();
}
```

如果不加 `findOne()` 而直接进行操作，则选择器会找出**所有**符合条件的控件并操作。

另外一个比较常用的操作的滑动。滑动操作的第一步是找到需要滑动的控件，例如要滑动 QQ 消息列表则在悬浮窗布局层次分析中找到 `AbsListView` ，这个控件就是消息列表控件，如下图：

长按可查看控件信息，注意到其 scrollable 属性为 true，并找出其 id 为"recent_chat_list"，从而下滑 QQ 消息列表的代码为：

```js
id("recent_chat_list").className("AbsListView").findOne().scroll
```

`scrollForward()` 为向前滑，包括下滑和右滑。

选择器的入门教程暂且要这里，更多信息可以查看下面的文档和选择器进阶。

selector()

- 返回 {UiSelector}

创建一个新的选择器。但一般情况不需要使用该函数，因为可以直接用相应条件的语句创建选择器。

由于历史遗留原因，本不应该这样设计(不应该让 `id()` ， `text()` 等作为全局函数，而是应该用 `By.id()` ， `By.text()`)，但为了后向兼容性只能保留这个设计。

这样的 API 设计会污染全局变量，后续可能会支持"去掉这些全局函数而使用 By.***"的选项。

UiSelector.text(str)

- `str` {string} 控件文本
- 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"text 等于字符串 str"的筛选条件。

控件的 text(文本)属性是文本控件上的显示的文字，例如微信左上角的"微信"文本。

UiSelector.textContains(str)

- `str` {string} 要包含的字符串

为当前选择器附加控件"text 需要包含字符串 str"的筛选条件。

这是一个比较有用的条件，例如 QQ 动态页和微博发现页上方的"大家都在搜...."的控件可以用 `textContains("大家都在搜").findOne()` 来获取。

UiSelector.textStartsWith(prefix)

- `prefix` {string} 前缀

为当前选择器附加控件"text 需要以 prefix 开头"的筛选条件。

这也是一个比较有用的条件，例如要找出 Auto.js 脚本列表中名称以"QQ"开头的脚本的代码为 `textStartsWith("QQ").find()` 。

UiSelector.textEndsWith(suffix)

- suffix {string} 后缀

为当前选择器附加控件"text 需要以 suffix 结束"的筛选条件。

UiSelector.textMatches(reg)

- `reg` {string} | {Regex} 要满足的正则表达式。

为当前选择器附加控件"text 需要满足正则表达式 reg"的条件。

有关正则表达式，可以查看**正则表达式 - 菜鸟教程**。

需要注意的是，如果正则表达式是字符串，则需要使用 `\\` 来表达 `\` (也即 Java 正则表达式的形式)，例如 `textMatches("\\d+")` 匹配多位数字；但如果使用 JavaScript 语法的正则表达式则不需要，例如 `textMatches(/\d+/)` 。但如果使用字符串的正则表达式则该字符串不能以"/"同时以"/"结束，也即不能写诸如 `textMatches("/\\d+/")` 的表达式，否则会被开头的"/"和结尾的"/"会被忽略。

UiSelector.desc(str)

- `str` {string} 控件文本
- 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"desc 等于字符串 str"的筛选条件。

控件的 desc(描述，全称为 Content-Description)属性是对一个控件的描述，例如网易云音乐右上角的放大镜图标的描述为搜索。要查看一个控件的描述，同样地可以借助悬浮窗查看。

desc 属性同样是定位控件的利器。

UiSelector.descContains(str)

- `str` {string} 要包含的字符串

为当前选择器附加控件"desc 需要包含字符串 str"的筛选条件。

UiSelector.descStartsWith(prefix)

- `prefix` {string} 前缀

为当前选择器附加控件"desc 需要以 prefix 开头"的筛选条件。

UiSelector.descEndsWith(suffix)

- `suffix` {string} 后缀

为当前选择器附加控件"desc 需要以 suffix 结束"的筛选条件。

UiSelector.descMatches(reg)

- `reg` {string} | {Regex} 要满足的正则表达式。

为当前选择器附加控件"desc 需要满足正则表达式 reg"的条件。

有关正则表达式，可以查看**正则表达式 - 菜鸟教程**。

需要注意的是，如果正则表达式是字符串，则需要使用 `\\` 来表达 `\` (也即 Java 正则表达式的形式)，例如 `textMatches("\\d+")` 匹配多位数字；但如果使用 JavaScript 语法的正则表达式则不需要，例如 `textMatches(/\d+/)` 。但如果使用字符串的正则表达式则该字符串不能以"/"同时以"/"结束，也即不能写诸如 `textMatches("/\\d+/")` 的表达式，否则会被开头的"/"和结尾的"/"会被忽略。

UiSelector.id(resId)

- `resId` {string} 控件的 id，以"包名:id/"开头，例如"com.tencent.mm:id/send_btn"。**也可以不指定包名**，这时会以当前正在运行的应用的包名来补全 id。例如 id("send_btn"),在 QQ 界面想当于 id("com.tencent.mobileqq:id/send_btn")。

为当前选择器附加"id 等于 resId"的筛选条件。

控件的 id 属性通常是可以用来确定控件的唯一标识，如果一个控件有 id，那么使用 id 来找到他是最好的方法。要查看屏幕上的控件的 id，可以开启悬浮窗并使用界面工具，点击相应控件即可查看。若查看到的控件 id 为 null, 表示该控件没有 id。另外，在列表中会出现多个控件的 id 相同的情况。例如微信的联系人列表，每个头像的 id 都是一样的。此时不能用 id 来唯一确定控件。

在 QQ 界面经常会出现多个 id 为"name"的控件，在微信上则每个版本的 id 都会变化。对于这些软件而言比较难用 id 定位控件。

UiSelector.idContains(str)

- `str` {string} id 要包含的字符串

为当前选择器附加控件"id 包含字符串 str"的筛选条件。比较少用。

UiSelector.idStartsWith(prefix)

- `prefix` {string} id 前缀

为当前选择器附加"id 需要以 prefix 开头"的筛选条件。比较少用。

UiSelector.idEndsWith(suffix)

- `suffix` {string} id 后缀

为当前选择器附加"id 需要以 suffix 结束"的筛选条件。比较少用。

UiSelector.idMatches(reg)

- reg {Regex} | {string} id 要满足的正则表达式

附加 id 需要满足正则表达式。

需要注意的是，如果正则表达式是字符串，则需要使用 `\\` 来表达 `\` (也即 Java 正则表达式的形式)，例如 `textMatches("\\d+")` 匹配多位数字；但如果使用 JavaScript 语法的正则表达式则不需要，例如 `textMatches(/\d+/)` 。但如果使用字符串的正则表达式则该字符串不能以"/"同时以"/"结束，也即不能写诸如 `textMatches("/\\d+/")` 的表达式，否则会被开头的"/"和结尾的"/"会被忽略。

```js
idMatches("[a-zA-Z]+")
```

UiSelector.className(str)

- `str` {string} 控件文本
- 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"className 等于字符串 str"的筛选条件。

控件的 className(类名)表示一个控件的类别，例如文本控件的类名为 android.widget.TextView。

如果一个控件的类名以"android.widget."开头，则可以省略这部分，例如文本控件可以直接用 `className("TextView")` 的选择器。

常见控件的类名如下：

- `android.widget.TextView` 文本控件
- `android.widget.ImageView` 图片控件
- `android.widget.Button` 按钮控件
- `android.widget.EditText` 输入框控件
- `android.widget.AbsListView` 列表控件
- `android.widget.LinearLayout` 线性布局
- `android.widget.FrameLayout` 帧布局
- `android.widget.RelativeLayout` 相对布局

- `android.widget.RelativeLayout` 相对布局
- `android.support.v7.widget.RecyclerView` 通常也是列表控件

UiSelector.classNameContains(str)

- `str` {string} 要包含的字符串

为当前选择器附加控件"className 需要包含字符串 str"的筛选条件。

UiSelector.classNameStartsWith(prefix)

- `prefix` {string} 前缀

为当前选择器附加控件"className 需要以 prefix 开头"的筛选条件。

UiSelector.classNameEndsWith(suffix)

- `suffix` {string} 后缀

为当前选择器附加控件"className 需要以 suffix 结束"的筛选条件。

UiSelector.classNameMatches(reg)

- `reg` {string} | {Regex} 要满足的正则表达式。

为当前选择器附加控件"className 需要满足正则表达式 reg"的条件。

有关正则表达式，可以查看**正则表达式 - 菜鸟教程**。

需要注意的是，如果正则表达式是字符串，则需要使用 `\\` 来表达 `\` (也即 Java 正则表达式的形式)，例如 `textMatches("\\d+")` 匹配多位数字；但如果使用 JavaScript 语法的正则表达式则不需要，例如 `textMatches(/\d+/)` 。但如果使用字符串的正则表达式则该字符串不能以"/"同时以"/"结束，也即不能写诸如 `textMatches("/\\d+/")` 的表达式，否则会被开头的"/"和结尾的"/"会被忽略。

UiSelector.packageName(str)

- `str` {string} 控件文本

- 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"packageName 等于字符串 str"的筛选条件。

控件的 packageName 表示控件所属界面的应用包名。例如微信的包名为"com.tencent.mm", 那么微信界面的控件的 packageName 为"com.tencent.mm"。

要查看一个应用的包名，可以用函数 `app.getPackageName()` 获取，例如 `toast(app.getPackageName("微信"))` 。

UiSelector.packageNameContains(str)

- `str` {string} 要包含的字符串

为当前选择器附加控件"packageName 需要包含字符串 str"的筛选条件。

UiSelector.packageNameStartsWith(prefix)

- `prefix` {string} 前缀

为当前选择器附加控件"packageName 需要以 prefix 开头"的筛选条件。

UiSelector.packageNameEndsWith(suffix)

- `suffix` {string} 后缀

为当前选择器附加控件"packageName 需要以 suffix 结束"的筛选条件。

UiSelector.packageNameMatches(reg)

- `reg` {string} | {Regex} 要满足的正则表达式。

为当前选择器附加控件"packageName 需要满足正则表达式 reg"的条件。

有关正则表达式，可以查看**正则表达式 - 菜鸟教程**。

UiSelector.bounds(left, top, right, buttom)

- `left` {number} 控件左边缘与屏幕左边的距离

- `top` {number} 控件上边缘与屏幕上边的距离
- `right` {number} 控件右边缘与屏幕左边的距离
- `bottom` {number} 控件下边缘与屏幕上边的距离

一个控件的 bounds 属性为这个控件在屏幕上显示的范围。我们可以用这个范围来定位这个控件。尽管用这个方法定位控件对于静态页面十分准确，却无法兼容不同分辨率的设备；同时对于列表页面等动态页面无法达到效果，因此使用不推荐该选择器。

注意参数的这四个数字不能随意填写，必须精确的填写控件的四个边界才能找到该控件。例如，要点击 QQ 主界面的右上角加号，我们用布局分析查看该控件的属性，如下图：

可以看到 bounds 属性为(951, 67, 1080, 196)，此时使用代码 `bounds(951, 67, 1080, 196).clickable().click()` 即可点击该控件。

UiSelector.boundsInside(left, top, right, buttom)

- `left` {number} 范围左边缘与屏幕左边的距离
- `top` {number} 范围上边缘与屏幕上边的距离
- `right` {number} 范围右边缘与屏幕左边的距离
- `bottom` {number} 范围下边缘与屏幕上边的距离

为当前选择器附加控件"bounds 需要在 left, top, right, buttom 构成的范围里面"的条件。

这个条件用于限制选择器在某一个区域选择控件。例如要在屏幕上半部分寻找文本控件 TextView，代码为：

```
js
var w = className("TextView").boundsInside(0, 0, device.width, d
log(w.text());
```

其中我们使用了 `device.width` 来获取屏幕宽度， `device.height` 来获取屏幕高度。

UiSelector.boundsContains(left, top, right, buttom)

- `left` {number} 范围左边缘与屏幕左边的距离
- `top` {number} 范围上边缘与屏幕上边的距离
- `right` {number} 范围右边缘与屏幕左边的距离
- `bottom` {number} 范围下边缘与屏幕上边的距离

为当前选择器附加控件"bounds 需要包含 left, top, right, buttom 构成的范围"的条件。

这个条件用于限制控件的范围必须包含所给定的范围。例如给定一个点(500, 300), 寻找在这个点上的可点击控件的代码为:

```js
var w = boundsContains(500, 300, device.width - 500, device.heig
w.click();
```

UiSelector.drawingOrder(order)

- order {number} 控件在父视图中的绘制顺序

为当前选择器附加控件"drawingOrder 等于 order"的条件。

drawingOrder 为一个控件在父控件中的绘制顺序，通常可以用于区分同一层次的控件。

但该属性在 Android 7.0 以上才能使用。

UiSelector.clickable([b = true])

- `b` {Boolean} 表示控件是否可点击

为当前选择器附加控件是否可点击的条件。但并非所有 clickable 为 false 的控件都真的不能点击，这取决于控件的实现。对于自定义控件(例如显示类名为 android.view.View 的控件)很多的 clickable 属性都为 false 都却能点击。

需要注意的是，可以省略参数 `b` 而表示选择那些可以点击的控件，例如 `className("ImageView").clickable()` 表示可以点击的图片控件的条件，

`className("ImageView").clickable(false)` 表示不可点击的图片控件的条件。

UiSelector.longClickable([b = true])

- `b` {Boolean} 表示控件是否可长按

为当前选择器附加控件是否可长按的条件。

UiSelector.checkable([b = true])

- `b` {Boolean} 表示控件是否可勾选

为当前选择器附加控件是否可勾选的条件。勾选通常是对于勾选框而言的，例如图片多选时左上角通常有一个勾选框。

UiSelector.selected([b = true])

- `b` {Boolean} 表示控件是否被选

为当前选择器附加控件是否已选中的条件。被选中指的是，例如 QQ 聊天界面点击下方的"表情按钮"时，会出现自己收藏的表情，这时"表情按钮"便处于选中状态，其 selected 属性为 true。

UiSelector.enabled([b = true])

- `b` {Boolean} 表示控件是否已启用

为当前选择器附加控件是否已启用的条件。大多数控件都是启用的状态(enabled 为 true)，处于“禁用”状态通常是灰色并且不可点击。

UiSelector.scrollable([b = true])

- `b` {Boolean} 表示控件是否可滑动

为当前选择器附加控件是否可滑动的条件。滑动包括上下滑动和左右滑动。

可以用这个条件来寻找可滑动控件来滑动界面。例如滑动 Auto.js 的脚本列表的代码为:

```js
className("android.support.v7.widget.RecyclerView").scrollable()
//或者classNameEndsWith("RecyclerView").scrollable().findOne().sc
```

UiSelector.editable([b = true])

- b {Boolean} 表示控件是否可编辑

为当前选择器附加控件是否可编辑的条件。一般来说可编辑的控件为输入框(EditText)，但不是所有的输入框(EditText)都可编辑。

UiSelector.multiLine([b = true])

- b {Boolean} 表示文本或输入框控件是否是多行显示的

为当前选择器附加控件是否文本或输入框控件是否是多行显示的条件。

UiSelector.findOne()

- 返回 UiObject

根据当前的选择器所确定的筛选条件，对屏幕上的控件进行搜索，直到屏幕上出现满足条件的一个控件为止，并返回该控件。如果找不到控件，当屏幕内容发生变化时会重新寻找，直至找到。

需要注意的是，如果屏幕上一直没有出现所描述的控件，则该函数会阻塞，直至所描述的控件出现为止。因此此函数不会返回 `null` 。

该函数本来应该命名为 `untilFindOne()` ，但由于历史遗留原因已经无法修改。如果想要只在屏幕上搜索一次而不是一直搜索，请使用 `findOnce()` 。

另外，如果屏幕上有多个满足条件的控件， `findOne()` 采用深度优先搜索(DFS)，会返回该搜索算法找到的第一个控件。注意控件找到的顺序有时会起到作用。

UiSelector.findOne(timeout)

- `timeout` {number} 搜索的超时时间，单位毫秒

- 返回 UiObject

根据当前的选择器所确定的筛选条件，对屏幕上的控件进行搜索，直到屏幕上出现满足条件的一个控件为止，并返回该控件；如果在 timeout 毫秒的时间内没有找到符合条件的控件，则终止搜索并返回 `null` 。

该函数类似于不加参数的 `findOne()` ，只不过加上了时间限制。

示例：

```js
//启动Auto.js
launchApp("Auto.js");
//在6秒内找出日志图标的控件
var w = id("action_log").findOne(6000);
//如果找到控件则点击
if(w != null){
    w.click();
}else{
    //否则提示没有找到
    toast("没有找到日志图标");
}
```

UiSelector.findOnce()

- 返回 UiObject

根据当前的选择器所确定的筛选条件，对屏幕上的控件进行搜索，如果找到符合条件的控件则返回该控件；否则返回 `null` 。

UiSelector.findOnce(i)

- `i` {number} 索引

根据当前的选择器所确定的筛选条件，对屏幕上的控件进行搜索，并返回第 i + 1 个符合条件的控件；如果没有找到符合条件的控件，或者符合条件的控件个数 < i, 则返回 `null` 。

注意这里的控件次序，是搜索算法深度优先搜索(DSF)决定的。

UiSelector.find()

- 返回 UiCollection

根据当前的选择器所确定的筛选条件，对屏幕上的控件进行搜索，找到所有满足条件的控件集合并返回。这个搜索只进行一次，并不保证一定会找到，因而会出现返回的控件集合为空的情况。

不同于 `findOne()` 或者 `findOnce()` 只找到一个控件并返回一个控件， `find()` 函数会找出所有满足条件的控件并返回一个控件集合。之后可以对控件集合进行操作。

可以通过 empty()函数判断找到的是否为空。例如：

```js
var c = className("AbsListView").find();
if(c.empty()){
    toast("没找到┌(ㅑ^ㅗ)┐");
}else{
    toast("找到啦");
}
```

UiSelector.untilFind()

- 返回 UiCollection

根据当前的选择器所确定的筛选条件，对屏幕上的控件进行搜索，直到找到至少一个满足条件的控件为止，并返回所有满足条件的控件集合。

该函数与 `find()` 函数的区别在于，该函数永远不会返回空集合；但是，如果屏幕上一直没有出现满足条件的控件，则该函数会保持阻塞。

UiSelector.exists()

- 返回 {Boolean}

判断屏幕上是否存在控件符合选择器所确定的条件。例如要判断某个文本出现就执行某个动作，可以用：

```js
if(text("某个文本").exists()){
    //要支持的动作
}
```

UiSelector.waitFor()

等待屏幕上出现符合条件的控件；在满足该条件的控件出现之前，该函数会一直保持阻塞。

例如要等待包含"哈哈哈"的文本控件出现的代码为：

```js
textContains("哈哈哈").waitFor();
```

UiSelector.filter(f)

- `f` {Function} 过滤函数，参数为 UiObject，返回值为 boolean

为当前选择器附加自定义的过滤条件。

例如，要找出屏幕上所有文本长度为 10 的文本控件的代码为：

```js
var uc = className("TextView").filter(function(w){
    return w.text().length == 10;
});
```

UiObject

UiObject 表示一个控件，可以通过这个对象获取到控件的属性，也可以对控件进行点击、长按等操作。

获取一个 UiObject 通常通过选择器的 `findOne()` ， `findOnce()` 等函数，也可以通过 UiCollection 来获取，或者通过 `UiObject.child()` ， `UiObject.parent()` 等函数来获取一个控件的子控件或父控件。

UiObject.click()

- 返回 {Boolean}

点击该控件，并返回是否点击成功。

如果该函数返回 false，可能是该控件不可点击(clickable 为 false)，当前界面无法响应该点击等。

UiObject.longClick()

- 返回 {Boolean}

长按该控件，并返回是否点击成功。

如果该函数返回 false，可能是该控件不可点击(longClickable 为 false)，当前界面无法响应该点击等。

UiObject.setText(text)

- `text` {string} 文本
- 返回 {Boolean}

设置输入框控件的文本内容，并返回是否设置成功。

该函数只对可编辑的输入框(editable 为 true)有效。

UiObject.copy()

- 返回 {Boolean}

对输入框文本的选中内容进行复制，并返回是否操作成功。

该函数只能用于输入框控件，并且当前输入框控件有选中的文本。可以通过 `setSelection()` 函数来设置输入框选中的内容。

```js
var et = className("EditText").findOne();
//选中前两个字
et.setSelection(0, 2);
//对选中内容进行复制
```

```
if(et.copy()){
    toast("复制成功");
}else{
    toast("复制失败");
}
```

UiObject.cut()

对输入框文本的选中内容进行剪切，并返回是否操作成功。

该函数只能用于输入框控件，并且当前输入框控件有选中的文本。可以通过 `setSelection()` 函数来设置输入框选中的内容。

UiObject.paste()

- 返回 {Boolean}

对输入框控件进行粘贴操作，把剪贴板内容粘贴到输入框中，并返回是否操作成功。

js

```
//设置剪贴板内容为“你好”
setClip("你好");
var et = className("EditText").findOne();
et.paste();
```

UiObject.setSelection(start, end)

- `start` {number} 选中内容起始位置
- `end` {number} 选中内容结束位置(不包括)
- 返回 {Boolean}

对输入框控件设置选中的文字内容，并返回是否操作成功。

索引是从 0 开始计算的；并且，选中内容不包含 end 位置的字符。例如，如果一个输入框内容为"123456789"，要选中"4567"的文字的代码为 `et.setSelection(3, 7)` 。

该函数也可以用来设置光标位置，只要参数的 end 等于 start，即可把输入框光标设置在 start 的位置。例如 `et.setSelection(1, 1)` 会把光标设置在第一个字符的后面。

UiObject.scrollForward()

- 返回 {Boolean}

对控件执行向前滑动的操作，并返回是否操作成功。

向前滑动包括了向右和向下滑动。如果一个控件既可以向右滑动和向下滑动，那么执行 `scrollForward()` 的行为是未知的(这是因为 Android 文档没有指出这一点，同时也没有充分的测试可供参考)。

UiObject.scrollBackward()

- 返回 {Boolean}

对控件执行向后滑动的操作，并返回是否操作成功。

向后滑动包括了向右和向下滑动。如果一个控件既可以向右滑动和向下滑动，那么执行 `scrollForward()` 的行为是未知的(这是因为 Android 文档没有指出这一点，同时也没有充分的测试可供参考)。

UiObject.select()

- 返回 {Boolean}

对控件执行"选中"操作，并返回是否操作成功。"选中"和 `isSelected()` 的属性相关，但该操作十分少用。

UiObject.collapse()

- 返回 {Boolean}

对控件执行折叠操作，并返回是否操作成功。

UiObject.expand()

- 返回 {Boolean}

对控件执行操作，并返回是否操作成功。

UiObject.show()

对集合中所有控件执行显示操作，并返回是否全部操作成功。

UiObject.scrollUp()

对集合中所有控件执行向上滑的操作，并返回是否全部操作成功。

UiObject.scrollDown()

对集合中所有控件执行向下滑的操作，并返回是否全部操作成功。

UiObject.scrollLeft()

对集合中所有控件执行向左滑的操作，并返回是否全部操作成功。

UiObject.scrollRight()

children()

- 返回 UiCollection

返回该控件的所有子控件组成的控件集合。可以用于遍历一个控件的子控件，例如：

```js
className("AbsListView").findOne().children()
    .forEach(function(child){
        log(child.className());
    });
```

childCount()

- 返回 {number}

返回子控件数目。

child(i)

- i {number} 子控件索引
- 返回 {UiObject}

返回第 i+1 个子控件。如果 i>=控件数目或者小于 0，则抛出异常。

需要注意的是，由于布局捕捉的问题，该函数可能返回 `null` ，也就是可能获取不到某个子控件。

遍历子控件的示例：

```js
var list = className("AbsListView").findOne();
for(var i = 0; i < list.childCount(); i++){
    var child = list.child(i);
    log(child.className());
}
```

parent()

- 返回 {UiObject}

返回该控件的父控件。如果该控件没有父控件，返回 `null` 。

bounds()

- 返回 **Rect**

返回控件在屏幕上的范围，其值是一个**Rect**对象。

示例：

```js
var b = text("Auto.js").findOne().bounds();
toast("控件在屏幕上的范围为" + b);
```

如果一个控件本身无法通过 `click()` 点击，那么我们可以利用 `bounds()` 函数获取其坐标，再利用坐标点击。例如：

```js
var b = desc("打开侧拉菜单").findOne().bounds();
click(b.centerX(), b.centerY());
//如果使用root权限，则用 Tap(b.centerX(), b.centerY());
```

boundsInParent()

- 返回 **Rect**

返回控件在父控件中的范围，其值是一个**Rect**对象。

drawingOrder()

- 返回 {number}

返回控件在父控件中的绘制次序。该函数在安卓 7.0 及以上才有效，7.0 以下版本调用会返回 0。

id()

- 返回 {string}

获取控件的 id，如果一个控件没有 id，则返回 `null` 。

text()

- 返回 {string}

获取控件的文本，如果控件没有文本，返回 `""` 。

findByText(str)

- `str` {string} 文本
- 返回 UiCollection

根据文本 text 在子控件中递归地寻找并返回文本或描述(desc)**包含**这段文本 str 的控件，返回它们组成的集合。

该函数会在当前控件的子控件，孙控件，曾孙控件...中搜索 text 或 desc 包含 str 的控件，并返回它们组合的集合。

findOne(selector)

- `selector` UiSelector
- 返回 UiOobject

根据选择器 selector 在该控件的子控件、孙控件...中搜索符合该选择器条件的控件，并返回找到的第一个控件；如果没有找到符合条件的控件则返回 `null` 。

例如，对于酷安动态列表，我们可以遍历他的子控件(每个动态列表项)，并在每个子控件中依次寻找点赞数量和图标，对于点赞数量小于 10 的点赞：

```js
//找出动态列表
var list = id("recycler_view").findOne();
//遍历动态
list.children().forEach(function(child){
    //找出点赞图标
    var like = child.findOne(id("feed_action_view_like"));
    //找出点赞数量
    var likeCount = child.findOne(id("text_view"));
    //如果这两个控件没有找到就不继续了
    if(like == null || likeCount == null){
        return;
    }
    //判断点赞数量是否小于10
    if(parseInt(likeCount.text()) < 10){
        //点赞
        like.click();
    }
});
```

find(selector)

- `selector` UiSelector
- 返回 UiCollection

根据选择器 selector 在该控件的子控件、孙控件...中搜索符合该选择器条件的控件，并返回它们组合的集合。

UiCollection

UiCollection, 控件集合, 通过选择器的 `find()` ， `untilFind()` 方法返回的对象。

UiCollection"继承"于数组，实际上是一个 UiObject 的数组，因此可以使用数组的函数和属性，例如使用 length 属性获取 UiCollection 的大小，使用 forEach 函数来遍历 UiCollection。

例如，采用 forEach 遍历屏幕上所有的文本控件并打印出文本内容的代码为：

```js
console.show();
className("TextView").find().forEach(function(tv){
    if(tv.text() != ""){
        log(tv.text());
    }
});
```

也可以使用传统的数组遍历方式：

```js
console.show();
var uc = className("TextView").find();
for(var i = 0; i < uc.length; i++){
    var tv = uc[i];
    if(tv.text() != ""){
        log(tv.text());
    }
}
```

UiCollection 的每一个元素都是 UiObject，我们可以取出他的元素进行操作，例如取出第一个 UiObject 并点击的代码为 `ui[0].click()` 。如果想要对该集合的所有元

素进行操作，可以直接在集合上调用相应的函数，例如 `uc.click()` ，该代码会对集合上所有 UiObject 执行点击操作并返回是否全部点击成功。

因此，UiCollection 具有所有 UiObject 对控件操作的函数，包括 `click()` ，`longClick()` ，`scrollForward()` 等等，不再赘述。

UiCollection.size()

- 返回 {number}

返回集合中的控件数。

历史遗留函数，相当于属性 length。

UiCollection.get(i)

- `i` {number} 索引
- 返回 **UiObject**

返回集合中第 i+1 个控件(UiObject)。

历史遗留函数，建议直接使用数组下标的方式访问元素。

UiCollection.each(func)

- `func` {Function} 遍历函数，参数为 UiObject。

遍历集合。

历史遗留函数，相当于 `forEach` 。参考**forEach**。

empty()

- 返回 {Boolean}

返回控件集合是否为空。

nonEmpty()

- 返回 {Boolean}

返回控件集合是否非空。

UiCollection.find(selector)

- `selector` UiSelector
- 返回 UiCollection

根据 selector 所确定的条件在该控件集合的控件、子控件、孙控件...中找到所有符合条件的控件并返回找到的控件集合。

注意这会递归地遍历控件集合里所有的控件以及他们的子控件。和数组的 `filter` 函数不同。

例如：

```js
var names = id("name").find();
//在集合
var clickableNames = names.find(clickable());
```

UiCollection.findOne(selector)

- `selector` UiSelector
- 返回 UiOobject

根据选择器 selector 在该控件集合的控件的子控件、孙控件...中搜索符合该选择器条件的控件，并返回找到的第一个控件；如果没有找到符合条件的控件则返回 `null` 。

Rect

`UiObject.bounds()` , `UiObject.boundsInParent()` 返回的对象。表示一个长方形(范围)。

Rect.left

- {number}

长方形左边界的 x 坐标、

Rect.right

- {number}

长方形右边界的 x 坐标、

Rect.top

- {number}

长方形上边界的 y 坐标、

Rect.bottom

- {number}

长方形下边界的 y 坐标、

Rect.centerX()

- 返回 {number}

长方形中点 x 坐标。

Rect.centerY()

- 返回 {number}

长方形中点 y 坐标。

Rect.width()

- 返回 {number}

长方形宽度。通常可以作为控件宽度。

Rect.height()

- 返回 {number}

长方形高度。通常可以作为控件高度。

Rect.contains(r)

- r **Rect**

返回是否包含另一个长方形 r。包含指的是，长方形 r 在该长方形的里面(包含边界重叠的情况)。

Rect.intersect(r)

- r **Rect**

返回是否和另一个长方形相交。

UiSelector 进阶

未完待续。

上一页
全局变量与函数
基础

下一页
基于坐标的操作
基础