# 第 3 章

# 常用句柄操作

Octave 的 GUI 是通过句柄实现的。Octave 使用句柄作为“指针”的用途，根据句柄类型的不同来“指向”不同种类的结构体，而在这些结构体中保存了 GUI 的参数，再根据这些参数即可绘制不同类型的 GUI 控件。

在 Octave 中也设计了基本图形对象，这些对象是专门为初始化结构体而设计的。通过不同的参数可以实例化或者修改不同的基本图形对象，并且对象内部的实例变量会根据参数的不同而产生差异。基本图形对象中的参数和结构体有对应关系，因而在绘制基本类型的图形时，无须处理复杂的结构体，可直接通过基本图形对象进行处理。

在对基本图形对象进行一系列处理后，对象会在低级绘图函数中通过句柄同步到相应的结构体上，实现初始化绘制图形或者刷新图形的功能。

## 3.1 通用句柄操作

### 3.1.1 返回句柄

#### 1. 返回根句柄

调用 groot 函数可以返回根句柄。groot 函数不允许带参数调用，代码如下：

```
>> groot
ans = 0
```

在 Octave 中，根句柄默认被识别为 0。

---

**注意**：在特定情况下，根句柄可能变为一个非 0 的值。为避免程序出错，建议在使用根句柄时使用 groot，而不使用 0 来指代根句柄。

---

#### 2. 返回当前图像的句柄

调用 gcf 函数可以返回当前图像的句柄。gcf 函数不允许带参数调用，代码如下：

```
>> gcf
ans = 1
```

#### 3. 返回当前轴的句柄

调用 gca 函数可以返回当前轴的句柄。gca 函数不允许带参数调用，代码如下：

```
>> gca
ans =  -34.638
```

**4. 返回当前对象的句柄**

调用 gco 函数有两种用法，在 gco 函数不带参数调用时可以返回当前对象的句柄，代码如下：

```
>> gco
ans = [](0x0)
```

若 gco 函数不带参数调用，则函数将返回最后一个用鼠标单击过的图像。此时，如果所有图像均未被鼠标单击过，则函数将返回一个尺寸为 0×0 的矩阵，如上面的代码所示。新绘制一个任意的图像，并且用鼠标单击一下，再不带参数调用 gco 函数，代码如下：

```
>> ezplot(@(x)x+1);
>> #此时单击一下图像
>> gco
ans =  -34.638
```

此外，gco 函数还允许追加一个额外参数，这个参数为一个特定句柄，此时 gco 函数可以返回带有特定句柄的当前对象的句柄，代码如下：

```
>> h = ezplot(@(x)x+1);
>> #此时第 1 幅图像绘制完成
>> gco(h)
ans = [](0x0)
>> #此时单击一下第 1 幅图像
>> gco(h)
ans = 2
>> h2 = figure;
>> #此时第 2 幅图像绘制完成
>> gco(h2)
ans = [](0x0)
>> #此时单击一下第 2 幅图像
>> gco(h2)
ans = 2
```

**5. 返回祖先句柄**

调用 ancestor()函数可以返回当前句柄的类型匹配的第 1 个祖先句柄。ancestor()函数在调用时可以传入两个参数，第 1 个参数是一个特定句柄，第 2 个参数是用于匹配的类型，代码如下：

```
>> ancestor(gcf, 'root')
ans = 0
```

此外，也可以调用 ancestor()函数匹配多种类型，此时第 2 个参数需要传入由多个字符串组成的元胞，代码如下：

```
>> ancestor(gcf, {'root', 'axes'})
ans = 0
```

此外，如果指定的类型包含特定句柄自身的类型，则 ancestor()函数会直接返回自身句柄，代码如下：

```
>> ancestor(gcf, 'figure')
ans = 1
>> ancestor(gcf, {'figure', 'root'})
ans = 1
>> ancestor(gcf, {'root', 'figure'})
ans = 1
```

此外，ancestor()函数还允许追加一个额外参数 toplevel，此时 ancestor()函数将返回最深层的、带有特定句柄的祖先句柄，代码如下：

```
>> ancestor (gcf, {'root', 'figure'}, 'toplevel')
ans = 0
```

**6. 返回子句柄**

调用 allchild()函数可以返回当前句柄的所有子句柄，代码如下：

```
>> allchild(gcf)
ans =

    -45.231
    -54.719
    -67.234
    -73.183
    -78.674
```

事实上，allchild()函数等效于用 get()函数获取一个句柄的 children 键参数，并且还可以额外返回隐藏的句柄。

上面的代码用一个句柄参数返回一组句柄。此外，还可以调用 allchild()函数同时返回多组句柄，代码如下：

```
>> allchild([gcf, gca])
ans =
{
    [1,1] =

        -45.231
        -54.719
        -67.234
        -73.183
        -78.674

    [2,1] =

        -44.229
        -41.701
        -42.317
        -43.329

}
```

### 3.1.2　句柄强制类型转换

**1. 句柄转结构体**

调用 hdl2struct()函数可以将句柄转换为其指向的结构体。调用 hdl2struct()函数可以传入一个参数，这个参数被认为是源句柄，代码如下：

```
>> hdl2struct(groot)
>> #以下输出省略
```

**2. 结构体转句柄**

调用 struct2hdl()函数可以将句柄转换为其指向的结构体。调用 struct2hdl()函数可以传入一个参数，这个参数被认为是源结构体，代码如下：

```
>> struct2hdl(s)
>> #以下输出省略
```

**注意**：如果结构体内的键-值对缺少基本图形对象所描述的字段，则转换将失败。具体地，该结构体必须包含 type、handle、properties、children 和 special 字段。

此外，struct2hdl()函数还允许追加一个额外参数，这个参数被认为是父句柄，此时 struct2hdl()函数将把转换后返回的句柄在父句柄之下创建为子句柄，代码如下：

```
>> struct2hdl(s, groot)
>> #以下输出省略
```

此外，struct2hdl()函数还允许追加第 3 个额外参数。

(1) 如果这个参数为 true，则此时 struct2hdl()函数转换后返回的句柄将保留监听器(或者叫作回调函数)。

(2) 如果这个参数为 false，则此时 struct2hdl()函数转换后返回的句柄将丢弃监听器(或者叫作回调函数)。

(3) 这个参数的默认值为 false。

调用 struct2hdl()函数，将结构体 s 转换为 groot 的子句柄，并且保留监听器(或者叫作回调函数)，代码如下：

```
>> struct2hdl(s, groot, 'true')
>> #以下输出省略
```

### 3.1.3　句柄复制

调用 copyobj()函数可以复制一个句柄。调用 copyobj()函数可以传入一个参数，这个参数被认为是源句柄，代码如下：

```
>> copyobj(gca)
>> #以下输出省略
```

此外，copyobj()函数还允许追加一个额外参数，这个参数被认为是父句柄，此时 copyobj()函

数将把复制后的句柄在父句柄之下作为子句柄，代码如下：

```
>> copyobj(gca, groot)
>> #以下输出省略
```

上面的代码用一个父句柄参数复制一次句柄。此外，还可以调用 allchild()函数复制多次句柄，代码如下：

```
>> copyobj(gca, [groot, gcf])
>> #以下输出省略
```

### 3.1.4 获得句柄

调用 get()函数可以获取句柄对应的结构体。调用 get()函数可以传入一个参数，这个参数被认为是源句柄，代码如下：

```
>> get(groot)
>> #以下输出省略
```

此外，get()函数还允许追加一个额外参数，这个参数被认为是结构体中的键参数，此时 get()函数将返回该键参数的当前值，代码如下：

```
>> get(groot, 'units')
ans = normalized
```

上面的代码用一组键-值对参数获取一次键参数。此外，还可以调用 get()函数同时获取多次键参数，代码如下：

```
>> get(groot, {'units', 'fixedwidthfontname'})
ans =
{
    [1,1] = normalized
    [1,2] = Courier
}
```

### 3.1.5 设置句柄

调用 set()函数可以获取句柄对应的结构体。调用 set()函数可以传入一个参数，这个参数被认为是源句柄，此时，set()函数将返回该句柄所有键参数的可选值和当前值，代码如下：

```
>> set(groot)
>> #以下输出省略
```

set()函数还允许追加一个额外参数，这个参数被认为是结构体中的键参数，此时 set()函数将返回该键参数的可选值和当前值，代码如下：

```
>> set(groot, 'units')
[ centimeters | characters | inches | normalized | {pixels} | points ]
```

set()函数还允许追加第 3 个参数，这个参数被认为是结构体中的值参数，代码如下：

```
>> set(groot, 'units', 'normalized')
```

**注意**：不能调用 set()函数获取或者设置结构体中的只读参数，否则将报错。

```
>> set(groot, 'screendepth')
set: screendepth is read - only
>> set(groot, 'screendepth', 100)
set: screendepth is read - only
```

上面的代码用一组键-值对参数设置一次键参数。此外，还可以调用 set()函数同时设置多次键参数，代码如下：

```
>> set(groot, 'units', 'normalized', 'hittest', 'on')
>> set(groot, {'units', 'hittest'}, {'normalized', 'on'})
>> set(groot, struct('units', 'normalized', 'hittest', 'on'))
```

### 3.1.6　查找非隐藏的句柄

调用 findobj()函数可以获取句柄。如果这个句柄含有子句柄，findobj()函数则有可能也返回子句柄。

findobj()函数可以不传入参数而直接调用，此时 findobj()函数将从根句柄开始依次查找子句柄，再将这些句柄全部返回，代码如下：

```
>> findobj
ans =

     0
     2
     1
```

findobj()函数还允许传入一个参数，这个参数被认为是源句柄，此时，findobj()函数将从该句柄开始依次查找子句柄，再将这些句柄全部返回，代码如下：

```
>> findobj(0)
ans =

     0
     2
     1
```

上面的代码用一个句柄参数获取一次句柄。此外，还可以调用 findobj()函数同时获取多次句柄，代码如下：

```
>> findobj([0 1])
ans =

     0
     1
     2
     1
```

findobj()函数还允许追加传入一个参数 flat，此时，findobj()函数将只返回这些句柄，而不返回子句柄，代码如下：

```
>> findobj([0 1], "flat")
ans =

    0
    1
```

findobj()函数还允许追加传入参数-depth和查找深度，此时，findobj()函数将从该句柄开始依次查找子句柄，直到达到查找深度，或者不存在更多的子句柄为止。再将这些句柄全部返回，代码如下：

```
>> findobj([0 1], "-depth", 0)
ans =

    0
    1
```

findobj()函数还允许以键-值对的形式传入查找的属性，此时，findobj()函数将按照条件先查找句柄，再对初步查找到的句柄进行筛选。筛选方法如下：

(1) 句柄对应的结构体必须含有指定的键参数。

(2) 如果句柄对应的结构体含有指定的键参数，则值参数必须和指定的值参数相等。

最后将满足条件的句柄全部返回，代码如下：

```
>> findobj('units', 'pixels')
ans =

    2
    1
```

此外，findobj()函数还允许在键-值对之间配置逻辑参数“-and”“-or”“-xor”或“-not”，两种逻辑参数的区别如下：

(1) 如果在键-值对之间配置逻辑参数“-and”，则代表在对相邻键-值对进行筛选时，相邻的这两对键-值对必须都判定为满足条件，对应的句柄才可能满足条件。

(2) 如果在键-值对之间配置逻辑参数“-or”，则代表在对相邻键-值对进行筛选时，至少一对键-值对判定为满足条件，对应的句柄才可能满足条件。

(3) 如果在键-值对之间配置逻辑参数“-xor”，则代表在对相邻键-值对进行筛选时，如果一对键-值对判定为满足条件，并且另一对键-值对判定为不满足条件，则对应的句柄才可能满足条件。

(4) 如果在键-值对之间配置逻辑参数“-not”，则代表在对相邻键-值对进行筛选时，在参数“-not”之后的那一对键-值对必须判定为不满足条件，对应的句柄才可能满足条件。

最后将满足条件的句柄全部返回，代码如下：

```
>> findobj('units', 'pixels', '-and', 'pointer', 'arrow', '-or', 'resize', 'on')
ans = 1
```

findobj()函数还允许在键-值对之前配置参数“-regexp”，此时的值参数被认为是一个正则表达式，通过这个正则表达式进行筛选，代码如下：

```
>> findobj('-regexp', 'units', '[pn]')
ans =

             0
        1.0000
    -190.0581
```

findobj()函数还允许只传入查找的属性名，此时必须将第1个参数配置为“-property”，然后findobj()函数将按照条件先查找句柄，再筛选出含有指定的键参数的句柄，最后返回这些句柄，代码如下：

```
>> findobj('-property', 'units')
ans =

             0
        1.0000
    -190.0581
```

### 3.1.7　查找全部句柄

findall()函数的用法和findobj()的用法相同，区别在于findall()函数会额外返回隐藏的句柄，可以查找全部句柄。findall()函数相比于findobj()函数返回的句柄更多，代码如下：

```
>> findall('-property', 'units')
ans =

             0
        1.0000
    -190.0581
    -189.8896
    -186.9252
    -187.0560
    -188.9455

>> findobj('-property', 'units')
ans =

             0
        1.0000
    -190.0581
```

### 3.1.8　重置句柄

调用reset()函数可以将当前句柄重置为默认状态，即将句柄对应的结构体键-值对复原为默认状态。reset()函数允许传入一个参数，这个参数被认为是源句柄，代码如下：

```
>> reset(groot)
```

### 3.1.9　查找可见的图形

调用findfigs函数可以将所有可见的图形，但不在当前屏幕上显示的窗口移动到当前窗

口上,代码如下:

```
>> a = figure;
>> #假如 Octave 的主软件当前显示的屏幕为屏幕 1
>> #那么用 a 句柄首次绘制图形时,对应的窗口也在屏幕 1 上
>> #然后,手动把该图形窗口移动到屏幕 2 上,并保持 Octave 的主软件当前显示的窗口在屏幕 1 上
>> findfigs
>> #此时,该图形窗口会被自动移动到屏幕 1 上
```

## 3.2 句柄组

### 3.2.1 创建句柄组

调用 hggroup()函数可以创建句柄组。句柄组用于存放图线对象等的句柄,方便对多个句柄做出批量的属性上的变换,也方便轴对象做出关于子对象的操作。hggroup()函数可以不传入参数而直接调用,此时将在当前的轴对象的基础上创建句柄组,并返回组对象的句柄,代码如下:

```
>> h = hggroup
h =  -40.136
```

hggroup()函数在调用时可以传入一个参数,这个参数表示组对象的父轴对象。在 axes 的基础上创建句柄组的代码如下:

```
>> h = hggroup(axes)
h =  -48.731
```

hggroup()函数在调用时还可以以键-值对的形式追加传入一对或多对参数,用于初始化句柄组对象的一个或多个属性。

### 3.2.2 增加句柄键参数

调用 addproperty()函数可以增加句柄键参数。addproperty()函数至少需要传入 3 个参数进行调用,第 1 个参数表示键参数,第 2 个参数表示要增加键参数的句柄,第 3 个参数表示值参数类型。向 gcf 中增加键参数为 aaa、值参数类型为 string 的代码如下:

```
>> addproperty('aaa', gcf, 'string')
```

addproperty()函数支持的值参数类型如表 3-1 所示。

**表 3-1 addproperty()函数支持的值参数类型**

| 值参数类型 | 含 义 | 值参数类型 | 含 义 |
|---|---|---|---|
| string | 字符串类型 | double | double 类型 |
| any | 任意类型 | handle | 句柄类型 |
| radio | 单选项类型 | data | 数据类型 |
| boolean | 布尔类型 | color | 颜色类型 |

此外,addproperty()函数允许追加传入第 4 个参数进行调用,这个参数表示值参数。向

gcf 中增加键参数为 aaa、值参数类型为 string、值参数为 bbb 的代码如下：

```
>> addproperty('aaa', gcf, 'string', 'bbb')
```

### 3.2.3　绑定监听器

调用 addlistener()函数可以绑定监听器，用于在句柄的某个值参数发生变动时执行监听器函数。addlistener()函数需要传入 3 个参数进行调用，第 1 个参数表示句柄，第 2 个参数表示键参数，第 3 个参数表示监听器函数。向 gcf 中绑定键参数为 aaa、监听器函数为 add([2,3])的代码如下：

```
>> addproperty('aaa', gcf, 'string')
>> addlistener(gcf, 'aaa', {@add, [2, 3]})
```

### 3.2.4　解绑监听器

调用 dellistener()函数可以解绑监听器，用于解绑监听器函数。dellistener()函数需要传入 3 个参数进行调用，第 1 个参数表示句柄，第 2 个参数表示键参数，第 3 个参数表示监听器函数。向 gcf 中解绑键参数为 aaa、监听器函数为 add([2,3])的代码如下：

```
>> dellistener(gcf, 'aaa', {@add, [2, 3]})
```

### 3.2.5　连接句柄键参数

调用 linkprop()函数可以连接多个句柄的一个或多个键参数，键参数在被连接后，对应的值参数会发生变化。linkprop()函数需要传入两个参数进行调用，第 1 个参数表示句柄，第 2 个参数表示一个键参数字符串，或多个键参数构成的字符串元胞。连接两个轴对象 a1 和 a2 的字号的代码如下：

```
>> a1 = axes
a1 =  - 34.057
>> a2 = axes
a2 =  - 39.331
>> linkprop([a1, a2], 'fontsize')
ans =

onCleanup (@() unlink_linkprop (h, prop))
```

同时连接两个轴对象 a1 和 a2 的字号和字体名的代码如下：

```
>> linkprop([a1, a2], {'fontsize', 'fontname'})
ans =

onCleanup (@() unlink_linkprop (h, prop))
```

在连接句柄键参数成功后，linkprop()函数会创建一个特殊的对象，专门用于处理建好的链接。

### 3.2.6　连接轴对象范围

调用 linkaxes()函数可以连接多个轴对象的范围，轴对象在被连接后，对应的范围会发生

变化。linkaxes()函数至少需要传入一个参数进行调用,这个参数表示不同的轴对象。连接两个轴对象 a1 和 a2 的范围的代码如下:

```
>> linkaxes([a1, a2])
```

linkaxes()函数还允许追加传入第 2 个参数,这个参数被认为是链接类型。linkaxes()函数支持的链接类型如表 3-2 所示。

**表 3-2 linkaxes()函数支持的链接类型**

| 链接类型 | 含义 | 链接类型 | 含义 |
|---|---|---|---|
| x | 链接 $x$ 轴范围 | xy | 链接 $x$ 轴范围和 $y$ 轴范围 |
| y | 链接 $y$ 轴范围 | off | 关闭链接 |

只连接两个轴对象 a1 和 a2 的 $x$ 轴范围的代码如下:

```
>> linkaxes([a1, a2], 'x')
```

## 3.3 判断绘图句柄

### 3.3.1 判断图形句柄

调用 ishghandle()函数可以判断一个变量是不是图形句柄。ishghandle()函数可以传入一个参数直接调用,代码如下:

```
>> ishghandle(groot)
ans = 1
```

此外,还可以调用 ishghandle()函数同时判断多个变量是不是图形句柄,代码如下:

```
>> ishghandle([groot gca])
ans =

   1 1
```

### 3.3.2 通过类型判断图形句柄

调用 isgraphics()函数也可以判断一个变量是不是图形句柄。

isgraphics()函数可以传入一个参数直接调用,此时其返回值和 ishghandle()函数相同,等效于传入一个参数并调用 ishghandle()函数,代码如下:

```
>> isgraphics(groot)
ans = 1
>> isgraphics([groot gca])
ans =

   1 1
```

isgraphics()函数还允许追加传入一个参数,这个参数被认为是句柄类型,此时,如果句柄类型和图形句柄中的 type 值参数相同,则那些句柄才会被 isgraphics()函数返回,代码如下:

```
>> isgraphics([groot gca], 'root')
ans =

    1 0
```

### 3.3.3　判断图形句柄或 Java 对象

调用 ishandle()函数可以判断一个变量是不是图形句柄或者 Java 对象。ishandle()函数可以传入一个参数直接调用，此时若变量是一个图形句柄或者 Java 对象，则返回 1，否则返回 0，代码如下：

```
>> j = javaObject('java.lang.Object')
j =

<Java object: java.lang.Object>

>> ishandle(j)
ans = 1

>> ishandle(groot)
ans = 1
```

此外，还可以调用 ishandle()函数同时判断多个变量是不是图形句柄，代码如下：

```
>> ishandle([groot groot])
ans =

    1 1
```

**注意**：在默认情况下，ishandle()函数不能同时判断多个变量是不是 Java 对象，否则代码将报错，所报的错误提示如下：

```
>> ishandle([j j])
error: octave_base_value::resize (): wrong type argument 'octave_java'
```

### 3.3.4　判断坐标轴句柄

调用 isaxes()函数可以判断一个变量是不是坐标轴句柄。isaxes()函数可以传入一个参数而直接调用，此时若变量是一个坐标轴句柄，则返回 1，否则返回 0，代码如下：

```
>> isaxes(gca)
ans = 1
```

此外，还可以调用 isaxes()函数，以便同时判断多个变量是不是图形句柄，代码如下：

```
>> isaxes([gca gca])
ans =

    1 1
```

### 3.3.5 判断图像句柄

调用 isfigure()函数可以判断一个变量是不是图像句柄。isfigure()函数可以传入一个参数而直接调用,此时若变量是一个图像句柄,则返回 1,否则返回 0,代码如下:

```
>> isfigure(gcf)
ans = 1
```

此外,还可以调用 isfigure()函数同时判断多个变量是不是图像句柄,代码如下:

```
>> isfigure([gcf gcf])
ans =

    1 1
```