Gui界面编程

# 界面简介

在IGG-Hydrate（<https://gitee.com/geomech/hydrate>）安装成功之后，运行zml\_ui.pyw，将启动类似如下的界面：

电脑萤幕的截图

AI 生成的内容可能不正确。

此界面主要包含如下部分：1、菜单区域（用于可以添加自己的菜单）；2、菜单之下的工具栏（可以选择哪些菜单在工具栏显示）；3、工具栏下方为标签栏，可以同时显示多个标签（可以用来显示任意的PyQt控件）；4、标签栏右侧为控制台，用来运行脚本以及显示输出的内容；5、最下方为状态栏，可以显示状态信息。

在基于此界面编程时，用户可以自定义以上几个区域的行为，从而将界面用于不同的场景。

如果要控制图形用户界面的行为，需要使用如下语句来引入gui对象（​**​G​**​raphical ​**​U​**​ser ​**​I​**​nterface, **GUI**），对界面的一切操作，都需要借助gui对象来实现：

from zmlx import gui

这里，gui是我们定义的一个特殊的变量，可以理解为，这个变量就代表了整个图形用户界面，调用这个变量（类的示例）的成员函数，就相当于操作了图形用户界面。

# 程序入口

如果要启动图形用户界面，需要调用如下函数：

gui.execute(func=None, keep\_cwd=True, close\_after\_done=True)

其中：

参数func是在图形用户界面启动之后，会立即在界面的控制台内执行的函数，默认为None，即不执行任何的操作。

参数keep\_cwd是在启动用户界面的时候，是否保留当前工作路径不变。如果为否，则会使用上一次在图形界面上设置的工作路径。

参数close\_after\_done代表在func执行完毕之后，是否立即关闭界面。默认为True，即立即关闭。这是因为，gui.execute函数设计的初衷，是在图形用户界面上执行某一个特定的操作，而在界面执行结束之后，要返回到主线程。如果只是希望启动界面，并在界面上进行额外的操作，请将close\_after\_done设置为False。

示例，执行如下代码（zmlx/demo/gui\_manual/show\_gui.py），将打开主界面：

from zmlx import gui

gui.execute(close\_after\_done=False)

另外，直接执行IGG-Hydrate（<https://gitee.com/geomech/hydrate>）安装包根目录下的zml\_ui.pyw，也可以打开主界面。

# 修改界面

在界面中，菜单栏是操作的入口，而标签页是主要的功能区域。因此，在自定义界面的时候，主要就是去做两个操作：第一，添加自定义的菜单Action；第二，在点击某些菜单Action的时候，在标签页面显示一个自定义的PyQt控件（QWidget）。

## 3.1 向菜单栏/工具栏添加Action

使用如下函数在菜单栏以及工具栏添加Action：

gui.add\_action(menu=None, text=None, name=None, slot=None, icon=None, tooltip=None, on\_toolbar=None, is\_enabled=None, is\_visible=None)

其中：

menu: 一个元素为字符串的list，定义在什么位置添加Action。必须给定。

text: Action上显示的文字. 比如，menu=[‘文件’, ‘打开’], text=‘PDF文件’将会在‘文件’菜单下创建‘打开’子菜单，并在‘打开’子菜单下，添加显示为‘PDF文件’的Action. 必须给定。

slot：在触发此Action的时候执行的函数。slot应该是一个不接受任何参数的Python函数。必须给定。

name: Action的名字。如果此Action只是在菜单中通过点击去触发，则不需要定义名字。如果后续需要在脚本中，使用gui.trigger(name)来触发，则需要去定义一个名字。可选。

icon：此Action显示的图标（默认为Python文件图标），是一个字符串，定义图标的路径。可选。

tooltip：鼠标悬停时的提示（字符串）。可选。

on\_toolbar：是否显示在工具栏。默认为False。可选。如果要将某一个Action添加到工具栏，那么，它首先必须首先在菜单栏内。

is\_enabled：是一个函数，在每次刷新的时候都会被执行，用来判断此Action是否可以启用。默认始终启用。可选。

is\_visible：是一个函数，在每次刷新的时候都会被执行，用来判断此Action是否可见。默认始终可见。可选。

## 3.2 添加标签页

使用如下函数在标签栏显示一个PyQt的控件（并返回控件对象）：

gui.get\_widget(the\_type, caption=None, on\_top=None, init=None, type\_kw=None, oper=None, icon=None, caption\_color=None, set\_parent=False)

其中：

the\_type：待显示控件的class；比如，要显示一个日历，可以令the\_type=QtWidgets. QCalendarWidget；

caption：控件显示在标签栏的时候，标签上显示的文字。注意，gui.get\_widget函数，将会遍历标签栏的所有的控件，找到类型和the\_type匹配，且标签文字和caption匹配的标签。如果找到，则返回此标签对象。否则，会创建标签并返回；

on\_top：是否将找到或者创建的标签放在最前面。当此函数找到了现有的标签，则默认只是返回标签对象，而不是将标签在最前端显示。

init：在创建标签之后，对标签对象进行的初始化的工作。init是一个函数，只接收一个参数，即标签对象。init函数只有在创建新的标签的时候会被执行。如果是找到了已有的标签，则不会执行init函数。

type\_kw：在构造标签对象的时候，传递的关键词参数（一个字典）。默认不传递任何参数。

oper：在标签显示之后执行的操作。oper是一个函数，只接收一个参数，即标签对象。oper在每次调用gui.get\_widget的时候都会被执行（无论是创建新的标签对象，还是找到已有的标签，都会执行oper操作）；

icon：标签栏显示的图标（图标文件的路径）。

set\_parent：在构造标签对象的时候，是否在type\_kw中附加parent=self这个参数。

## 3.3 示例

执行如下代码（zmlx/demo/gui\_manual/show\_calendar.py），将在启动界面之后，在菜单上添加一个Action，并在点击这个Action之后，在主界面显示一个日历：

代码如下：

from zmlx import gui  
from zmlx.ui.pyqt import QtWidgets  
  
def show\_calendar():  
 gui.get\_widget(  
 the\_type=QtWidgets.QCalendarWidget, caption='日历')  
  
def setup\_ui():  
 gui.add\_action(  
 menu='Gui Manual', text='显示日历', slot=show\_calendar)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 gui.execute(func=setup\_ui, close\_after\_done=False)

启动界面之后，点击“显示日历”菜单，效果如下：

电脑屏幕和键盘

AI 生成的内容可能不正确。

在这个示例中，定义了setup\_ui函数，并使用gui.execute调用这个函数来进行图形界面的配置。

**一个更加建议的方案，是使用zml\_gui\_setup.py文件**，即在自定义的Python包内，添加名为zml\_gui\_setup.py的文件。在图形界面启动的时候，会自动搜索所有可用的包内，是否存在zml\_gui\_setup.py；如果找到，则会直接执行。

## 3.4 标签的标准函数

如果一个显示在当前标签页中的标签包含特定的函数，则可以使用预定义好的菜单来调用。主要支持的命令如下：

|  |  |
| --- | --- |
| **标签控件的成员函数** | **界面菜单及说明** |
| import\_data(self) | 当标签页定义了此函数时，则当标签在前端显示时，”文件”-“导入”菜单将自动可用，用于从文件导入数据 |
| save\_file(self) | 对应“文件”-“保存”，用于保存数据 |
| need\_save**(self)** | 用于判断当前是否需要去保存数据。如果此函数没有定义，则“文件”-“保存”按钮将一直可用。否则，保存菜单将根据need\_save的结果来确定是否显示 |
| export\_data**(self)** | 导出数据，对应“文件”-“导出” |
| refresh(self) | 对应“显示”-“刷新”，并且注意，刷新操作可能会在更多条件下触发，比如一个标签从后台到前台的时候；程序内部也可能会根据情况进行主动的刷新 |
| console\_exec(self) | 对应“操作”-“运行”，用于在控制台运行脚本，或者是特定的命令 |

# 界面交互

## 4.1 基本操作

在任何时候，如果需要在代码中操作图形用户界面，必须借助gui对象。尽管使用gui.get\_widget可以返回一个控件对象，但是，除非确信位于主线程中，否则，不要直接调用界面对象的成员函数（如果在不同的线程中直接调用界面对象的成员函数，将造成界面的崩溃）。更安全地操作界面对象，需要借助gui.call函数：

gui.call(func, \*args, \*\*kwargs)

其中：func为需要调用的函数，而\*args, \*\*kwargs是要传递给func的参数。比如，通过如下代码，获得了一个控件：

obj = gui.get\_widget(the\_type=QtWidgets.QCalendarWidget, caption='日历')

如果希望调用obj的成员函数，如obj.setEnabled(False)，更加安全的方法，是：

gui.call(obj.setEnabled, False)

函数gui.call会通过PyQt的信号-槽函数的形式，将任意线程的调用，都转移到图形用户界面的主线程中去执行，并在执行完毕之后，将结果返回到调用的线程中。

与使用gui.call等效，我们也可以将需要对界面进行的操作注册给gui对象：

gui.add\_func(key, func)

其中key是函数名，func是需要添加的函数（当func=None的时候，则相当于移除了函数）。

例如：

gui.add\_func(‘test’, obj.setEnabled)

会将obj的setEnabled注册给gui对象，后续，调用gui.test(False)则等效于gui.call(obj.setEnabled, False)。除非后续覆盖，否则注册给gui对象的函数，将会一直存在。

将某个界面操作通过gui.add\_func添加到gui对象的另一个好处，是可以方便其它模块去调用。

## 4.2 其它接口函数

为了方便与界面的交互，在图形界面启动的瞬间，会默认使用gui.add\_func函数将一些界面相关的操作注册给了gui对象，下面，是一些常用的函数（更多的接口需要去查看源代码自行获取，或者，如有特殊的调用需求，请联系作者）。

主要接口函数如下：

|  |  |
| --- | --- |
| 函数名称 | 使用说明 |
| gui.list\_actions() | 返回所有Action的名字，这些名字可以使用来gui.trigger(name)触发 |
| gui.trigger(name) | 触发给定名字的Action（相当于点击了相应的菜单） |
| gui.tab\_count() | 返回标签栏中所有页面的数量 |
| gui.close\_all\_tabs() | 关闭所有的标签页面 |
| gui.cls() | 清除控制台的输出内容 |
| gui.kill\_thread() | 杀死当前正在执行的控制台线程（不安全，特别是控制台线程正在操作界面的时候） |
| gui.toolbar\_warning(  text) | 显示一个警告提示（用红色字体） |
| gui.get\_figure\_widget(  clear=True, \*\*kwargs) | 返回一个Matplotlib绘图控件，其中clear为是否清除已有的绘图，而\*\*kwargs是传递给gui.get\_widget的参数 |
| gui.status(message) | 显示状态栏消息（临时显示，默认显示5秒钟） |
| gui.title(title) | 设置主窗口的标题 |
| gui.view\_cwd() | 打开一个新的标签页，显示当前工作路径下的内容 |
| gui.progress(  label=None, val\_range=None, value=None,  visible=None) | 显示一个进度条。其中label为进度条左侧的提示，val\_range为值的范围，value为当前的值，visible为是否可见 |
| gui.new\_code() | 打开对话框，新建Python脚本文件 |
| gui.exec\_current() | 执行当前标签页面中的脚本（如果当前页面不是Python脚本，则自动忽略） |
| gui.add\_file\_handler(  desc, exts, func) | 添加文件处理函数。其中desc为文件类型的描述（字符串），且desc对于不同的文件类型应该是唯一的（如果两种文件类型的desc相同，则会被视为一种文件），exts为这种类型支持的扩展名（exts是一个list，且每一个元素都是字符串，且扩展名以点号开始，例如 exts=[‘.txt’, ‘.xml’]），func可以是两种类型：1、func可以是一个函数，且此函数进接收一个参数，即文件路径；2、func可以是一个dict，并定义open\_file和new\_file；例如，func={‘open\_file’: func1, ‘new\_file’: func2}，这样，当在界面点击新建的时候，会调用func2 (filepath)在给定的路径下创建一个文件，而在界面上打开文件的时候，会调用func1(filepath)来打开给定的文件。 |
| gui.reg\_file\_type(  desc, exts, name, save, load, init, widget\_type) | 注册一种文件类型。注意，此函数会自动调用gui.add\_file\_handler。其中desc为文件类型的描述（字符串），且desc对于不同的文件类型应该是唯一的（如果两种文件类型的desc相同，则会被视为一种文件）；exts为这种类型支持的扩展名（exts是一个list，且每一个元素都是字符串，且扩展名以点号开始，例如 exts=[‘.txt’, ‘.xml’]）；name为文件类型的名字，必须是一个标准的Python变量，将以此添加一个gui.open\_name函数，用来打开这种文件类型；save: save(data, filename)，用于将数据存储到文件；load: load(filename) 读取文件的函数；init: init() 返回初始化之后的数据的函数；widget\_type: 编辑器控件类型。此类需要有set\_data和get\_data函数。 |
| gui.edit\_in\_tab(  widget\_type,  set\_data, get\_data=None, caption=None, support\_refresh  =False) | 在另外一个标签里面，使用给定的编辑器控件去编辑数据. 这个给定的控件，需要有get\_data和set\_data函数。其中 get\_data函数用于从编辑器获取数据，set\_data函数用于设置数据（将数据推送到编辑器上显示）。  widget\_type: 编辑器控件类型（需要定义set\_data和get\_data成员函数）;  set\_data: 一个函数，调用此函数用于返回编辑器编辑之后的结果;  get\_data: 一个函数，调用此函数用于获得初始化/刷新编辑器的数据;  caption: 标签的标题.  support\_refresh: 是否支持刷新操作 (默认为否，即控件在创建之后，不再接收数据) |
| gui.open\_file(  filepath) | 打开文件。会调用gui.add\_file\_handler注册的函数 |
| gui.open\_file\_by\_dlg() | 通过对话框打开文件；会调用gui.add\_file\_handler注册的函数 |
| gui.set\_cwd(folder) | 设置当前工作路径 |
| gui.get\_current\_widget() | 返回当前的控件对象；如果标签栏为空，则返回None |
| gui.exec\_file(  filename) | 在控制台线程中执行给定的文件 |
| gui.start(code) | 在控制台线程中执行给定的函数或者代码，其中code是一个函数（不接受任何参数），或者是一个字符串（待执行的Python脚本）。**注意，此函数非常重要**，对于比较繁重的任务，比如一个大的循环，如果在界面的主线程里面执行，则会使得界面卡死。此时，可以使用此函数，将需要执行的任务打包成一个函数，并将此函数发送给控制台执行。使用程序主界面的控制台来执行一个线程，其主要的优势，是可以监控线程的执行，特别是，可以在繁重的任务中插入gui.break\_point()，从而可以在界面上暂停或者终止线程。 特别需要注意，传递给start函数的code中，如果需要操作界面，务必使用gui.call函数来实现安全的调用（通过PyQt的信号-槽机制） |
| gui.is\_running() | 控制台线程是否处于繁忙的状态 |
| gui.play\_sound(  filename) | 播放一个声音 |
| gui.show\_tab\_details() | 打开一个标签列表详情页面 |
| gui.open\_url(url, caption=None) | 打开一个网址 |
| gui.close() | 关闭界面，退出程序 |
| gui.show\_next() | 切换到下一个标签 |
| gui.show\_prev() | 切换到上一个标签 |
| gui.window() | 返回主窗口对象 |

# 控件

基于PyQt进行界面编程的最核心的工作是编写各种功能的控件。除了PyQt自带的控件之外，还包括MatplotWidget, ImageView, CodeEdit等，这些控件可以使用如下语句import到需要的位置：

from zmlx.ui.widget import \*

**例如**：

当需要进行Matplotlib绘图的时候，可以嵌入MatplotWidget控件，MatplotWidget最重要的接口函数是：

MatplotWidget.plot\_on\_figure(on\_figure)

其中on\_figure是一个仅接收参数figure的回调函数，其中figure是matplotlib.figure.Figure对象。给定此回调函数，即可以在控件上进行任何Matplotlib支持的绘图。例如，为了在控件上绘制曲线，可以定义回调函数如下：

def on\_figure(fig):  
 ax = fig.subplots()  
 ax.plot([1, 2, 3], [4, 5, 6])

然后调用：

widget.plot\_on\_figure(on\_figure)

即可在控件widget上绘制出需要的曲线。