

dearpygui 笔记

一个好用的 Python GUI 模块

没有的内容

(原版文档看不懂的)

[Slots](#) · [Basic Example](#) · [Container Stack](#) · [Scene Graph](#) · [3D Operations](#) · [querying](#) · [custom series](#) · [resizing](#) · [stretch](#) · ...

总体

基础

主脚本必须始终

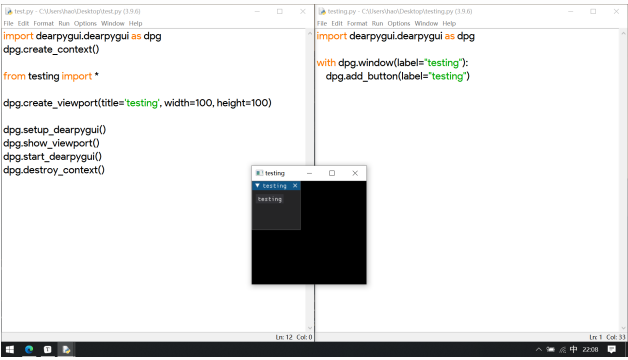
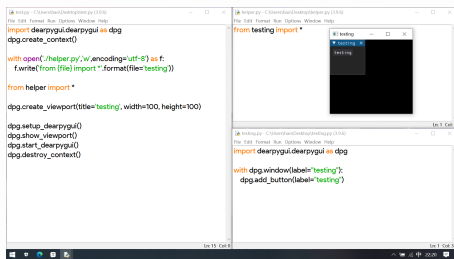
事件	代码	注释
导入	<pre>import dearpygui.dearpygui as dpg</pre>	
创建上下文	<pre>dpg.create_context()</pre>	如果不是第一个 DPG 不会启动/崩溃
创建窗口	<pre>with dpg.window(label="标题"): 组件</pre>	

事件	代码	注释
创建视区	<code>dpg.create_viewport(title='大窗口标题', width=宽, height=高)</code>	需要使用create_viewport 创建该视区 并使用 show_viewport显示
设置 dearpygui	<code>dpg.setup_dearpygui()</code>	
显示该视口中	<code>dpg.show_viewport()</code>	
启动 dearpygui	<code>dpg.start_dearpygui()</code>	
清理环境	<code>dpg.destroy_context()</code>	

创建窗口

`with dpg.window(label="标题", pos=位置(数组)): +组件`

导入窗口

直接	间接
 <pre> test.py - C:\Users\ben\Desktop\test.py (3.9.6) File Edit Format Run Options Window Help import dearpygui.dearpygui as dpg dpg.create_context() from testing import * dpg.create_viewport(title='testing', width=100, height=100) dpg.setup_dearpygui() dpg.show_viewport() dpg.start_dearpygui() dpg.destroy_context() testing.py - C:\Users\ben\Desktop\testing.py (3.9.6) File Edit Format Run Options Window Help import dearpygui.dearpygui as dpg dpg.create_context() with dpg.window(label='testing'): dpg.add_button(label='testing') </pre>	 <pre> test.py - C:\Users\ben\Desktop\test.py (3.9.6) File Edit Format Run Options Window Help import dearpygui.dearpygui as dpg dpg.create_context() dpg.create_viewport(title='testing', width=100, height=100) dpg.setup_dearpygui() dpg.show_viewport() dpg.start_dearpygui() dpg.destroy_context() testing.py - C:\Users\ben\Desktop\testing.py (3.9.6) File Edit Format Run Options Window Help import dearpygui.dearpygui as dpg dpg.create_context() with dpg.window(label='testing'): dpg.add_button(label='testing') </pre>
在 create_context 与 create_viewport 中间用 <code>from xxx import *</code>	使用文件写入自定义

被导入文件内容

(可能) 只需要 `import dearpygui.dearpygui as dpg` 和窗口内容

使用文件写入自定义

```
with open('./helper.py', 'w', encoding='utf-8') as f: # helper.py
    # 写入
    # 是用于帮助导入的文件
    f.write('from {file} import *'.format(file='testing')) # 写入
    # (间接导入)
from helper import *
```

主窗口（不必要）

将填充该视口并始终绘制在其他窗口的后面且铺满视图，自动去除标题栏

```
with dpg.window(tag="Primary Window"):
    ...
    dpg.show_viewport()
    dpg.set_primary_window("Primary Window", True)
    dpg.start_dearpygui()
    ...
```

图标

```
import dearpygui.dearpygui as dpg

dpg.create_context()

with dpg.window(label="Example Window", width=500, height=150):
    dpg.add_text("Hello, world")

dpg.create_viewport(title='Custom Title', width=600, height=200)
# create viewport takes in config options too!

# must be called before showing viewport
dpg.set_viewport_small_icon("path/to/icon.ico")
dpg.set_viewport_large_icon("path/to/icon.ico")

dpg.setup_dearpygui()
dpg.show_viewport()
dpg.start_dearpygui()
dpg.destroy_context()
```

- `dpg.set_viewport_large_icon()` 设置大图标
- `dpg.set_viewport_small_icon()` 设置小图标

组件：辅助

标签系统 (tag)

```
with dpg.window(label="Tutorial"):  
    b0 = dpg.add_button(label="button 0")  
    b1 = dpg.add_button(tag=100, label="Button 1")  
    dpg.add_button(tag="Btn2", label="Button 2")  
  
print(b0)  
print(b1)  
print(dpg.get_item_label("Btn2"))
```

1. `xxx = dpg.add_xxx(...)`
 2. `unique_tag = dpg.generate_uuid()` # 在 `create_context` 前
`dpg.add_xxx(..., tag=xxx)`
- 可以赋值给变量
 - 可以添加tag (`dpg.get_item_label()` 调用)
 - 所有项目都必须具有关联的唯一ID(UUID), 该ID可以是整数或字符串
 - 创建项目时会自动生成标签, 如果以后与组件交互, 则存储此标签
 - `dpg.last_ { item(组件), container(容器), root(根) } ()` 获取最新的对应项

容器

```

with dpg.window(label="Tutorial"):
    dpg.add_button(label="Button 1")
    dpg.add_button(label="Button 2")
    with dpg.group():
        dpg.add_button(label="Button 3")
        dpg.add_button(label="Button 4")
        with dpg.group() as group1:
            pass
dpg.add_button(label="Button 6", parent=group1)
dpg.add_button(label="Button 5", parent=group1)

```

- 在 `with dpg.group():` 内添加组件到组
- 也可以在外部 `dpg.add_... (... parent= 容器名)`

编辑属性

```

btn = dpg.add_button(label="Apply 2") #创建
dpg.set_item_label(btn, "Button 57") #修改label(标签)
dpg.set_item_width(btn, 200) #修改高度

```

代码	功能
<code>get_item_配置</code>	控制其外观和行为的关键字(标签、回调、宽度、高度)
<code>get_item_状态</code>	反映其交互的关键字(可见、悬停、点击等)
<code>get_item_信息</code>	反映其信息的关键字(项目类型、子项、主题等)

回调

```
def button_callback(sender, app_data, user_data):
    ...

with dpg.window(label="Tutorial"):
    # 按钮创建时创建回调
    dpg.add_button(label="Apply", callback=button_callback,
user_data="Some Data")
    # 按钮创建后创建回调
    btn = dpg.add_button(label="Apply 2", )
    dpg.set_item_callback(btn, button_callback)
    dpg.set_item_user_data(btn, "Some Extra User Data")
```

- 用 callback 赋值函数创建

输入函数: 同位置赋值

- 用 dpg.set_item_callback 链接

输入函数: set_item_ 赋值以链接

输入值	功能
sender	发送回调组件的id
app_data	大多数为组件当前值
user_data	(可选) 传输自己的数据 dpg.add_xxx (... callback= ... , user_data= ...) dpg.set_item_user_data()

- 不在主线程上运行

```
import dearpygui.dearpygui as dpg
dpg.create_context()

dpg.configure_app(manual_callback_management=True) #设置

dpg.create_viewport()
dpg.setup_dearpygui()
def callback(sender, app_data, user_data):
    print("Called on the main thread!")
with dpg.window(label="Tutorial"):
```

```
dpg.add_button(label="Press me", callback=callback)
dpg.show_viewport()
```

主循环

```
while dpg.is_dearpygui_running():
    jobs = dpg.get_callback_queue() # 获取
    dpg.run_callbacks(jobs) # 运行
    dpg.render_dearpygui_frame()
```

```
dpg.destroy_context()
```

1. 使用 `dpg.configure_app(manual_callback_management=True)` 设置
2. 在主循环内用 `xxx = dpg.get_callback_queue()` 获取
3. 使用 `dpg.run_callbacks(xxx)` 运行

组件值

```
with dpg.window(width=300):
    input_txt = dpg.add_input_text(
        label="InputTxt",
        default_value="This is a default value!", # 默认值
    )
    slider_float = dpg.add_slider_float(
        label="SliderFloat",
        default_value=50.0, # 默认值
    )
    dpg.add_slider_int(default_value=15, tag="slider_int")
    # 获取值
    print(dpg.get_value(input_txt))
    print(dpg.get_value(slider_float))

dpg.set_value("slider_int", 40) # 设置值
```

- 添加默认值: `default_value`
- 获取值: `get_value(赋值为组件的变量)`
- 设置值: `set_value(tag , 值)`

侦听与事件

```
def change_text(sender, app_data):
    if dpg.is_item_hovered("text item"): #组件侦听
        dpg.set_value("text item", f"爬开!!")
    else:
        dpg.set_value("text item", f"把鼠标放我上面!")

with dpg.handler_registry():
    dpg.add_mouse_move_handler(callback=change_text) #鼠标移动侦听

with dpg.window(width=500, height=300):
    dpg.add_text("把鼠标放我上面!", tag="text item") # 添加文本
```

```
def change_text(sender, app_data):
    dpg.set_value("text item", f"Mouse Button ID: {app_data}")

with dpg.window(width=500, height=300):
    dpg.add_text("Click me with any mouse button", tag="text item")
    with dpg.item_handler_registry(tag="widget handler") as handler: # 监听器设置
        dpg.add_item_clicked_handler(callback=change_text) # 鼠标点击监听器并设置回调
    dpg.bind_item_handler_registry("text item", "widget handler") # 监听器绑定
```

- 设置监听器 `with dpg.item_handler_registry(tag="监听器名") as handler:`
- 添加事件 `dpg.add_item_..._handler(callback=回调)`
- 绑定监听器 `dpg.bind_item_handler_registry("组件tag", "监听器tag")`

组件事件:

名称	译意
Activated	激活
Active	活跃

名称	译意
Clicked	点击
Deactivated	停用
Deactivated After Edited	编辑后停用
Focus	聚焦
Hover	悬停
Resize	调整大小
Toggled	切换
Visible	可见

全局事件：

名称	译意
——Keys——	——以下为键盘的操作——
Down	按下
Press	按 (Down+Release)
Release	释放
——Mouse——	——以下为鼠标的操作——
Click	单击
Double Click	双击
Down	按下
Drag	拖动
Move	移动
Release	抬起
Wheel	滚轮

添加和删除组件

添加:回调 `dpg.add_ xxx`

删除: `dpg.delete_item(" tag ")`

注: 删除容器时, 默认情况下会删除该容器及其子项, 除非 `delete_item` 中 `children_only` 设置为`True`时不会删除该容器

属性编辑

```
with dpg.window(width=500, height=300):  
    dpg.add_button(enabled=True, label="Press me", tag="item")  
    dpg.configure_item("item", enabled=False, label="New Label")
```

- `dpg.configure_item` 编辑属性
- `dpg.get_item_configuration` 获取属性 (字典类型)

值注册器

```
with dpg.value_registry():  
    dpg.add_bool_value(default_value=True, tag="bool_value")  
    dpg.add_string_value(default_value="Default string",  
tag="string_value")  
  
with dpg.window(label="Tutorial"):  
    dpg.add_checkbox(label="Radio Button1", source="bool_value")  
    dpg.add_checkbox(label="Radio Button2", source="bool_value")  
  
    dpg.add_input_text(label="Text Input 1",  
source="string_value")  
    dpg.add_input_text(label="Text Input 2",  
source="string_value", password=True)
```

- 在 `with dpg.value_register():` 中定义
- 定义: `dpg.add_类型{ bool(布尔), string(字符串) }_value(default_value=默认值, tag=" tag ")`
- 使用: `dpg.add_ xxx (... , source=" 对应的tag ")`

其他

开发工具

```
import dearpygui.dearpygui as dpg

dpg.create_context()
# start
dpg.show_documentation()
dpg.show_style_editor()
dpg.show_debug()
dpg.show_about()
dpg.show_metrics()
dpg.show_font_manager()
dpg.show_item_registry()
# end
dpg.create_viewport(title='Custom Title', width=800, height=600)
dpg.setup_dearpygui()
dpg.show_viewport()
dpg.start_dearpygui()
dpg.destroy_context()
```

其中几个：

代码	功能
<code>dpg.show_style_editor()</code>	样式编辑器
<code>dpg.show_item_registry()</code>	组件信息查看器
<code>dpg.show_font_manager()</code>	字体管理器
<code>dpg.show_metrics()</code>	性能监视器

渲染循环

```
import dearpygui.dearpygui as dpg
dpg.create_context()
dpg.create_viewport(title='Custom Title', width=600, height=200)
dpg.setup_dearpygui()
with dpg.window(label="Example Window"):
    dpg.add_text("Hello, world")
dpg.show_viewport()
# here ↓
while dpg.is_dearpygui_running():
    print("每一帧都会运行这里")
    dpg.render_dearpygui_frame()
# here ↑
dpg.destroy_context()
```

`dpg.is_dearpygui_running()` 判断 dpg 是否运行
`dpg.render_dearpygui_frame()` 让 dpg 渲染一帧

组件

简单组件

代码	功能
<code>dpg.add_button(label="名称")</code>	按钮
<code>dpg.add_slider_int/float (label="名称", width=限制)</code>	滑块
<code>dpg.add_text("xxx ")</code>	文本
<code>dpg.add_input_text(label="xxx ")</code>	文本输入
<code>dpg.add_checkbox(label="xx ", callback=xxx)</code>	复选框
<code>dpg.add_color_picker(label="xxx ", callback=xxx)</code>	颜色选择

容器和上下文管理器

核心代码	上下文管理器	翻译
------	--------	----

核心代码	上下文管理器	翻译
<code>add_table</code>	<code>with table(...):</code>	表格
<code>add_table_row</code>	<code>with table_row(...):</code>	表格_行
<code>add_window</code>	<code>with window(...):</code>	窗口
<code>add_menu_bar</code>	<code>with menu_bar(...):</code>	菜单栏
<code>add_child</code>	<code>with child(...):</code>	子项
<code>add_clipper</code>	<code>with clipper(...):</code>	?
<code>add_collapsing_header</code>	<code>with collapsing_header(...):</code>	?
<code>add_colormap_registry</code>	<code>with colormap_registry(...):</code>	color map 注册表 ?
<code>add_group</code>	<code>with group(...):</code>	组
<code>add_node</code>	<code>with node(...):</code>	节点 ?
<code>add_node_attribute</code>	<code>with node_attribute(...):</code>	节点属性 ?
<code>add_node_editor</code>	<code>with node_editor(...):</code>	节点编辑器 ?
<code>add_staging_container</code>	<code>with staging_container(...):</code>	staging 容器 ?
<code>add_tab_bar</code>	<code>with tab_bar(...):</code>	标签栏/选项卡栏
<code>add_tab</code>	<code>with tab(...):</code>	标签/选项卡
<code>add_tree_node</code>	<code>with tree_node(...):</code>	树状节点 ?
<code>add_tooltip</code>	<code>with tooltip(...):</code>	工具提示
<code>add_popup</code>	<code>with popup(...):</code>	弹出
<code>add_drag_payload</code>	<code>with payload(...):</code>	?
<code>add_drawlist</code>	<code>with drawlist(...):</code>	绘图列表
<code>add_draw_layer</code>	<code>with draw_layer(...):</code>	绘图层

核心代码	上下文管理器	翻译
<code>add_viewport_drawlist</code>	<code>with viewport_drawlist(...):</code>	视口绘图列表? `
<code>add_file_dialog</code>	<code>with file_dialog(...):</code>	文件对话框
<code>add_filter_set</code>	<code>with filter_set(...):</code>	筛选器
<code>add_font</code>	<code>with font(...):</code>	字体
<code>add_font_registry</code>	<code>with font_registry(...):</code>	字体注册
<code>add_handler_registry</code>	<code>with handler_registry(...):</code>	handler 注册器 ?
<code>add_plot</code>	<code>with plot(...):</code>	?
<code>add_subplots</code>	<code>with subplots(...):</code>	子plot ?
<code>add_texture_registry</code>	<code>with texture_registry(...):</code>	纹理注册器
<code>add_value_registry</code>	<code>with value_registry(...):</code>	值注册器
<code>add_theme</code>	<code>with theme(...):</code>	主题
<code>add_item_pool</code>	<code>with item_pool(...):</code>	组件池 ?
<code>add_template_registry</code>	<code>with template_registry(...):</code>	模板注册器

demo:

```

with dpg.window(label="Main"):
    with dpg.menu_bar():
        with dpg.menu(label="Themes"):
            dpg.add_menu_item(label="Dark")
            dpg.add_menu_item(label="Light")
            dpg.add_menu_item(label="Classic")
        with dpg.menu(label="Other Themes"):
            dpg.add_menu_item(label="Purple")
            dpg.add_menu_item(label="Gold")
            dpg.add_menu_item(label="Red")

```

```

with dpg.menu(label="Tools"):
    dpg.add_menu_item(label="Show Logger")
    dpg.add_menu_item(label="Show About")
with dpg.menu(label="Oddities"):
    dpg.add_button(label="A Button")
    dpg.add_simple_plot(label="Menu plot", default_value=
(0.3, 0.9, 2.5, 8.9), height=80)

```

```

w = dpg.add_window(label="Main")
mb = dpg.add_menu_bar(parent=w)
themes = dpg.add_menu(label="Themes", parent=mb)
dpg.add_menu_item(label="Dark", parent=themes)
dpg.add_menu_item(label="Light", parent=themes)
other_themes = dpg.add_menu(label="Other Themes", parent=themes)
dpg.add_menu_item(label="Purple", parent=other_themes)
dpg.add_menu_item(label="Gold", parent=other_themes)
dpg.add_menu_item(label="Red", parent=other_themes)
tools = dpg.add_menu(label="Tools", parent=mb)
dpg.add_menu_item(label="Show Logger", parent=tools)
dpg.add_menu_item(label="Show About", parent=tools)
oddities = dpg.add_menu(label="Oddities", parent=mb)
dpg.add_button(label="A Button", parent=oddities)
dpg.add_simple_plot(label="A menu plot", default_value=(0.3, 0.9,
2.5, 8.9), height=80, parent=oddities)

```

```

dpg.add_window(label="Main", tag="w")
dpg.add_menu_bar(parent="w", tag="mb")
dpg.add_menu(label="Themes", parent="mb", tag="themes")
dpg.add_menu_item(label="Dark", parent="themes")
dpg.add_menu_item(label="Light", parent="themes")
dpg.add_menu(label="Other Themes", parent="themes",
tag="other_themes")
dpg.add_menu_item(label="Purple", parent="other_themes")
dpg.add_menu_item(label="Gold", parent="other_themes")
dpg.add_menu_item(label="Red", parent="other_themes")
dpg.add_menu(label="Tools", parent="mb", tag="tools")
dpg.add_menu_item(label="Show Logger", parent="tools")
dpg.add_menu_item(label="Show About", parent="tools")
dpg.add_menu(label="Oddities", parent="mb", tag="Oddities")
dpg.add_button(label="A Button", parent="Oddities")

```

```
dpg.add_simple_plot(label="A menu plot", default_value=(0.3, 0.9, 2.5, 8.9), height=80, parent="Oddities")
```

```
dpg.push_container_stack(dpg.add_window(label="Main"))
dpg.push_container_stack(dpg.add_menu_bar())
dpg.push_container_stack(dpg.add_menu(label="Themes"))
dpg.add_menu_item(label="Dark")
dpg.add_menu_item(label="Light")
dpg.pop_container_stack()
dpg.push_container_stack(dpg.add_menu(label="Tools"))
dpg.add_menu_item(label="Show Logger")
dpg.add_menu_item(label="Show About")
dpg.pop_container_stack()
# remove menu_bar from container stack
dpg.pop_container_stack()
# remove window from container stack
dpg.pop_container_stack()
</details>
```

容器槽

代码	功能
<code>is_item_container</code>	检查是否为容器类型
<code>get_item_slot</code>	返回项目的槽
<code>get_item_parent</code>	返回项目的父UUID
<code>get_item_children</code>	返回项的子项
<code>reorder_items</code>	在一次调用中重新排序子项
<code>move_item_up</code>	将组件在其槽内上移
<code>move_item_down</code>	将组件在其槽内下移
<code>move_item</code>	在容器之间移动组件
<code>set_item_children</code>	将一个舞台移到一个组件的子槽中

绘图API

- 绘图命令可以添加到 `DrawList`、`VIEPORT_DrawList` 或窗口等容器中
- 创建一个绘图列表项 `add_drawlist`，可以调用绘制命令来添加项
- 坐标系是右手坐标系，原点位于左上角，x轴指向左侧，y轴指向下方，z轴指向屏幕

```
with dpg.window(label="xxx"):  
    with dpg.drawlist(width=xxx, height=xxx):  
        ...
```

图层

- 在 `with dpg.draw_layer():` 中添加内容
- `dpg.draw_layer():` 内可添加tag

图像

- 可以显示PNG、JPEG或BMP类型的图像
- 使用`DRAW_IMAGE`添加图像。

使用 `pmin` 和 `pmax` 我们可以定义矩形的左上角和右下角区域

图像将缩放以适合指定区域。

使用 `uv_min` 和 `uv_max` 我们可以指定规格化的纹理坐标，以仅使用图像上区域的一部分。默认为 `uv_min= [0,0]` 和 `UV_max= [1,1]` 将在 `UV_MIN=[0,0]` 时显示整个图像, `UV_max= [0.5,0.5]` 将仅显示绘图的第一个四分之一。

```

# 加载图片
width, height, channels, data = dpg.load_image('xxx.png') # 0:
width, 1: height, 2: channels, 3: data
#注册图片
with dpg.texture_registry():
    dpg.add_static_texture(width, height, data, tag="image_id")

with dpg.window(label="Tutorial"):
    with dpg.drawlist(width=700, height=700):
        # 绘制*3
        dpg.draw_image("image_id", (0, 400), (200, 600), uv_min=
(0, 0), uv_max=(1, 1))
        dpg.draw_image("image_id", (400, 300), (600, 500), uv_min=
(0, 0), uv_max=(0.5, 0.5))
        dpg.draw_image("image_id", (0, 0), (300, 300), uv_min=(0,
0), uv_max=(2.5, 2.5))

```

1. 加载图片：四个变量 = `dpg.load_image(' 图片路径 ')`
2. 注册图片： `with dpg.texture_registry():\n`
`dpg.add_static_texture(前面四个变量 , tag=" 图片id ")`
3. 使用图片： `dpg.draw_image(" 图片id ", 左上、右下位置, uv规格化)`

窗口与视口

```

with dpg.viewport_drawlist():
    dpg.draw_circle((100, 100), 25, color=(255, 255, 255, 255))
# 置顶

dpg.add_viewport_drawlist(front=False, tag="viewport_back")
dpg.draw_circle((200, 200), 25, color=(255, 255, 255, 255),
parent="viewport_back") # 置底

with dpg.window(label="Tutorial", width=300, height=300):
    dpg.add_text("Move the window over the drawings to see the
effects.", wrap=300)
    dpg.draw_circle((100, 100), 25, color=(255, 255, 255, 255))
# 窗口内

```

- 窗口内：跟随窗口
- 一般 `dpg.viewport_drawlist` 内：置顶

- 带有 `viewport_back` tag的 `dpg.viewport_drawlist` 内: 置底

文件和目录选择器

(dpg内部)

可用于选择单个文件、多个文件或目录, 当用户单击 **Ok** 按钮, 则运行对话框的回调。

信息通过APP_DATA参数传递, 例如: 文件路径 文件名 当前路径 当前过滤器(文件类型过滤器)

```
def callback(sender, app_data):
    print("Sender: ", sender)
    print("App Data: ", app_data)

dpg.add_file_dialog(directory_selector=True, show=False,
callback=callback, tag="file_dialog_id")

with dpg.window(label="Tutorial", width=800, height=300):
    dpg.add_button(label="Directory Selector", callback=lambda:
dpg.show_item("file_dialog_id"))
```

Sender: xxx

App Data:

{'file_path_name': 'xxx', ---文件路径

'file_name': 'xxx', ---文件名

'current_path': 'xxx', ---当前路径

'current_filter': 'xxx', ---当前过滤器

'min_size': [xxx, xxx], ---最大大小

'max_size': [xxx, xxx], ---最小大小

'selections': {'xxx': 'xxx'}} ---选择该文件/目录的相对路径和绝对路径

- `callback=lambda: dpg.show_item("file_dialog_id")`

扩展名

```
with dpg.file_dialog(directory_selector=False, show=False,
callback=callback, id="file_dialog_id"):
    dpg.add_file_extension(".*")
    dpg.add_file_extension("", color=(150, 255, 150, 255))
    dpg.add_file_extension("Source files (*.cpp *.h *.hpp)
{.cpp,.h,.hpp}", color=(0, 255, 255, 255))
    dpg.add_file_extension(".h", color=(255, 0, 255, 255),
custom_text="[header]")
    dpg.add_file_extension(".py", color=(0, 255, 0, 255),
custom_text="[Python]")
```

- 懒得写了，自己看:D

加点别的

```
import dearpygui.dearpygui as dpg
dpg.create_context()
def callback(sender, app_data):
    print("Sender: ", sender)
    print("App Data: ", app_data)
with dpg.file_dialog(directory_selector=False, show=False,
callback=callback, tag="file_dialog_tag"):
    dpg.add_file_extension(".*")
    dpg.add_file_extension("", color=(150, 255, 150, 255))
    dpg.add_file_extension(".cpp", color=(255, 255, 0, 255))
    dpg.add_file_extension(".h", color=(255, 0, 255, 255))
    dpg.add_file_extension(".py", color=(0, 255, 0, 255))

with dpg.group(horizontal=True):
    dpg.add_button(label="fancy file dialog")
    dpg.add_button(label="file")
    dpg.add_button(label="dialog")
dpg.add_date_picker()
with dpg.child_window(height=100):
    dpg.add_selectable(label="bookmark 1")
    dpg.add_selectable(label="bookmark 2")
    dpg.add_selectable(label="bookmark 3")

with dpg.window(label="Tutorial", width=800, height=300):
```

```
dpg.add_button(label="File Selector", callback=lambda:
dpg.show_item("file_dialog_tag"))
dpg.create_viewport(title='Custom Title', width=800, height=600)
dpg.setup_dearpygui()
dpg.show_viewport()
dpg.start_dearpygui()
dpg.destroy_context()
```

- 在里面加组或子窗口

选择多个文件

通过在 `dpg.file_dialog` 设置 `file_count` (数量限制) 关键字来选择多个文件

Ctrl/Shift 选择

过滤器/搜寻

```
def callback(sender, filter_string):
    dpg.set_value("filter_id", filter_string)

with dpg.window(label="about", width =500, height=300):
    dpg.add_input_text(label="Filter (inc, -exc)",
callback=callback)
    with dpg.filter_set(id="filter_id"):
        dpg.add_text("aaa1.c", filter_key="aaa1.c", bullet=True)
        dpg.add_text("bbb1.c", filter_key="bbb1.c", bullet=True)
        dpg.add_text("ccc1.c", filter_key="ccc1.c", bullet=True)
        dpg.add_text("aaa2.cpp", filter_key="aaa2.cpp",
bullet=True)
        dpg.add_text("bbb2.cpp", filter_key="bbb2.cpp",
bullet=True)
        dpg.add_text("ccc2.cpp", filter_key="ccc2.cpp",
bullet=True)
        dpg.add_text("abc.h", filter_key="abc.h", bullet=True)
        dpg.add_text("hello, world", filter_key="hello, world",
bullet=True)
```

设置:

```
with dpg.filter_set(id=" id "):
```

```
dpg.add_XXX ( ... , filter_key=" xxx ", bullet=True)
```

使用:

```
dpg.add_input_text( ... , callback=xxx)
```

callback:

```
def xxx ( ... ,filter_string):
```

```
dpg.set_value(" id ", filter_string)
```

字体

默认字体: ProggyClean.ttf

```
# 注册字体
with dpg.font_registry():
    # .ttf or .otf file
    default_font = dpg.add_font("xxx.otf", 20)
    second_font = dpg.add_font("xxx.otf", 10)

with dpg.window(label="Font Example", height=200, width=200):
    dpg.add_button(label="Default font")
    b2 = dpg.add_button(label="Secondary font")
    dpg.add_button(label="default")

    dpg.bind_font(default_font) # 设为默认
    dpg.bind_item_font(b2, second_font) # 专门指定
```

注册:

默认情况下, 仅添加基本拉丁语和拉丁语补充字形(0x0020-0x00FF)

```
with dpg.font_registry():
```

```
xxx = dpg.add_font(" path ", pixel[px] )
```

添加其他字形:

```
with dpg.font("xxx.otf", 20) as font:
    # 添加默认字形
    dpg.add_font_range_hint(dpg.mvFontRangeHint_Default)
```

```
# 其他字形
#         mvFontRangeHint_Japanese
#         mvFontRangeHint_Korean
#         mvFontRangeHint_Chinese_Full
#         mvFontRangeHint_Chinese_Simplified_Common
#         mvFontRangeHint_Cyrillic
#         mvFontRangeHint_Thai
#         mvFontRangeHint_Vietnamese
dpg.add_font_range_hint(dpg.mvFontRangeHint_Japanese)

# 添加区间字形
dpg.add_font_range(0x3100, 0x3ff0)

# 添加指定字形
dpg.add_font_chars([0x3105, 0x3107, 0x3108])

# 刷新区间字形
dpg.add_char_remap(0x3084, 0x0025)
```

设置:

设为默认: `dpg.bind_font(xxx)`

专门指定: `dpg.bind_item_font(item , xxx)`

附: [Unicode Characters](#)

菜单栏

视口菜单栏: `with dpg.viewport_menu_bar():`

窗口菜单栏: `with dpg.menu_bar():`

里面的内容:

```
with dpg.menu(label=" xxx "):
```

```
    dpg.add_menu_item
```

其他组件也可以

```
dpg.add_menu_item(label=" xxx ", callback= xxx )
```

node 编辑器

它有4个主要组成部分

node编辑器:node所在的区域

node:包含属性的自由浮动“窗口”

属性:带有Pins的小部件集合, 用于创建指向/来自的连接, 可以是输入、输出或静态

链接:属性之间的连接

属性可以包含任何UI项。当用户单击并拖动node的属性时, 将运行编辑器的回调。DPG将属性的标记通过 *app_data* 回调参数

```
# 连接属性时运行回调
def link_callback(sender, app_data):
    # app_data -> (link_id1, link_id2)
    dpg.add_node_link(app_data[0], app_data[1], parent=sender)

# 断开属性时运行回调
def delink_callback(sender, app_data):
    # app_data -> link_id
    dpg.delete_item(app_data)

with dpg.window(label="Tutorial", width=400, height=400):
    with dpg.node_editor(callback=link_callback,
delink_callback=delink_callback):
        with dpg.node(label="Node 1"):
            with dpg.node_attribute(label="Node A1"):
                dpg.add_input_float(label="F1", width=150)

            with dpg.node_attribute(label="Node A2",
attribute_type=dpg.mvNode_Attr_Output):
                dpg.add_input_float(label="F2", width=150)

        with dpg.node(label="Node 2"):
            with dpg.node_attribute(label="Node A3"):
                dpg.add_input_float(label="F3", width=200)
```



```
with dpg.node_attribute(label="Node A4",
attribute_type=dpg.mvNode_Attr_Output):
    dpg.add_input_float(label="F4", width=200)
```

在 with dpg.node_editor(callback=link_callback,
delink_callback=delink_callback): 内添加内容

在 with dpg.node(label=" xxx "): 内添加node

在 with dpg.node_attribute(label=" xxx "): 内添加属性

选择查询

检索选定的node和链接(并使用以下命令清除此选择)

```
dpg.get_selected_nodes(editor_id)
dpg.get_selected_links(editor_id)
dpg.clear_selected_nodes(editor_id)
dpg.clear_selected_links(editor_id)
```

属性类型

以下常量可用于 attribute_type 属性的参数

```
mvNode_Attr_Input # (默认)
mvNode_Attr_Output
mvNode_Attr_Static
```

属性形状

以下常量可用于shape节点属性的参数

```
mvNode_PinShape_Circle
mvNode_PinShape_CircleFilled #(默认)
mvNode_PinShape_Triangle
mvNode_PinShape_TriangleFilled
mvNode_PinShape_Quad
mvNode_PinShape_QuadFilled
```

绘制图表

地块由多个组件组成。

Y轴：这是一个容器，是添加到绘图中的所有数据系列的父级。打印一次可以有多个Y轴(最多3个)。

X轴：这是x数据刻度(只允许1个x轴)。

系列：要显示的数据的容器。需要将数据系列添加为要在绘图上显示的Y轴的子项。有许多不同类型的数据系列可用。系列还可以包含当右键单击图例中的系列标签作为上下文菜单时将显示的UI项

图例(可选)：

这是一个普通的图例，ALOS允许用户切换哪些数据序列是可见的。

绘图具有一些内置功能：

切换数据系列：单击所需数据系列的图例名称进行切换

设置：双击鼠标右键

平移绘图：在绘图上单击并拖动

平移轴：在轴上单击并拖动

缩放：滚动鼠标滚轮

缩放轴：悬停轴和滚动鼠标滚轮

缩放区域：单击鼠标右键并拖动

最大化显示：双击

缩放轴区域：按住Shift键并单击鼠标右键并拖动

添加数据

```
sindatax = []
sindatay = []
for i in range(0, 500):
    sindatax.append(i / 1000)
    sindatay.append(0.5 + 0.5 * sin(50 * i / 1000))
```

使用 `.append()`

创建视图：`with dpg.plot(label=" xxx ", height= xxx , width= xxx):`

可选择创建图例：`dpg.add_plot_legend()`

创建x、y轴: `dpg.add_plot_axis(dpg.mvXAxis, label="x")` 和

`dpg.add_plot_axis(dpg.mvYAxis, label="y", tag="y_axis")`

添加系列 (内容, 可添加多个): `dpg.add_line_series(数据 (前面的sindatax和sindatay) , label=" x ", parent=" y ")`

更新数据

- `dpg.add_line_series` 中添加tag
- `dpg.set_value(' tag ', [x和y])`
- `dpg.set_item_label(' tag ', " 值 ")`

轴限制

以下命令可用于控制地块轴限制

```
set_axis_limits(...)  
get_axis_limits(...)  
set_axis_limits_auto(...)  
fit_axis_data(...)
```

例子:

```
with dpg.window(label="Tutorial", width=400, height=400):  
    with dpg.group(horizontal=True):  
        dpg.add_button(label="fit y", callback=lambda:  
dpg.fit_axis_data("y_axis"))  
        dpg.add_button(label="unlock x limits", callback=lambda:  
dpg.set_axis_limits_auto("x_axis"))  
        dpg.add_button(label="unlock y limits", callback=lambda:  
dpg.set_axis_limits_auto("y_axis"))  
        dpg.add_button(label="print limits x", callback=lambda:  
print(dpg.get_axis_limits("x_axis")))  
        dpg.add_button(label="print limits y", callback=lambda:  
print(dpg.get_axis_limits("y_axis")))  
    with dpg.plot(label="Bar Series", height=-1, width=-1):  
        dpg.add_plot_legend()  
        # create x axis
```

```

    dpg.add_plot_axis(dpg.mvXAxis, label="Student",
no_gridlines=True, tag="x_axis")
    dpg.set_axis_limits(dpg.last_item(), 9, 33)
    dpg.set_axis_ticks(dpg.last_item(), (("S1", 11), ("S2",
21), ("S3", 31)))
    # create y axis
    dpg.add_plot_axis(dpg.mvYAxis, label="Score",
tag="y_axis")
    dpg.set_axis_limits("y_axis", 0, 110)
    # add series to y axis
    dpg.add_bar_series([10, 20, 30], [100, 75, 90],
label="Final Exam", weight=1, parent="y_axis")
    dpg.add_bar_series([11, 21, 31], [83, 75, 72],
label="Midterm Exam", weight=1, parent="y_axis")
    dpg.add_bar_series([12, 22, 32], [42, 68, 23],
label="Course Grade", weight=1, parent="y_axis")
dpg.create_viewport(title='Custom Title', width=800, height=600)
dpg.setup_dearpygui()
dpg.show_viewport()
dpg.start_dearpygui()
dpg.destroy_context()

```

自定义轴标签

可以使用设置轴标签 `set_axis_ticks` , 可以使用 `reset_axis_ticks` 重置它们

```

# create x axis
dpg.add_plot_axis(dpg.mvXAxis, label="Student", no_gridlines=True)
dpg.set_axis_ticks(dpg.last_item(), (("S1", 11), ("S2", 21),
("S3", 31)))

# create y axis
dpg.add_plot_axis(dpg.mvYAxis, label="Score", tag="yaxis_tag")

# add series to y axis
dpg.add_bar_series([10, 20, 30], [100, 75, 90], label="Final
Exam", weight=1, parent="yaxis_tag")
dpg.add_bar_series([11, 21, 31], [83, 75, 72], label="Midterm
Exam", weight=1, parent="yaxis_tag")
dpg.add_bar_series([12, 22, 32], [42, 68, 23], label="Course
Grade", weight=1, parent="yaxis_tag")

```

多个y轴

(最多3个y轴)

```
# create y axis 1
dpg.add_plot_axis(dpg.mvYAxis, label="y1")
dpg.add_line_series(sindatax, sindatay, label="y1 lines",
parent=dpg.last_item())
# create y axis 2
dpg.add_plot_axis(dpg.mvYAxis, label="y2")
dpg.add_stem_series(sindatax, sindatay, label="y2 stem",
parent=dpg.last_item())
# create y axis 3
dpg.add_plot_axis(dpg.mvYAxis, label="y3 scatter")
dpg.add_scatter_series(sindatax, sindatay, label="y3",
parent=dpg.last_item())
```

注释

使用 `dpg.add_plot_annotation()` 标记

内容:

- `label`: 里面的文字
- `default_value=(x, y)`: 默认值
- `offset=(x, y)`: 偏移, 离指向的位置, 用直线指出, 可省略
- `color=[rgba]`: 标记的颜色
- `clamped`: 固定 (当视角不包含该点时标签在边缘显示), 可省略 (默认 `True`)

可拖动的点和线

- 点: `dpg.add_drag_point()` (用元组表示坐标)
- 线: `dpg.add_drag_line()` (用浮点数表示坐标)

内容:

- `label`: 里面的文字
- `color=[rgba]`: 标记的颜色
- `default_value`: 默认值
- `callback`: 回调 (获取: `dpg.get_value(sender)`)
- `vertical`: (线) 纵向, 默认 `True`

自定义右键菜单

- 右击图例中对应项时弹出的菜单
- 在 `dpg.add_line_series` 后添加组件，并 `parent=` 指向其

例子：

```
dpg.add_line_series(sindatax, sindatay, label="series 1",
parent="yaxis", tag="series_1")

dpg.add_button(label="Delete Series 1", parent=dpg.last_item(),
callback=lambda: dpg.delete_item("series_1"))
```

主题

详见下面的 `## 主题`

1. 在前面加 `with dpg.theme(tag=" xxx ")`：
2. 添加主题的对象 `with`
`dpg.theme_component(dpg.mvStemSeries/dpg.mvScatterSeries/...)`：
3. 主题的内容
4. 在图标里用 `dpg.bind_item_theme(" 图表系列tag ", " 主题tag ")` 应用

主题的内容：

- 颜色：`dpg.add_theme_color(dpg.mvPlotCol_Line, (rgb), category=dpg.mvThemeCat_Plots)`
- 点的形状：`dpg.add_theme_style(dpg.mvPlotStyleVar_Marker, dpg.mvPlotMarker_Diamond/Square/... , category=dpg.mvThemeCat_Plots)`
- 点的大小：`dpg.add_theme_style(dpg.mvPlotStyleVar_MarkerSize, int , category=dpg.mvThemeCat_Plots)`

弹出窗口

- 右键菜单：在 `with dpg.popup(链接到组件 (例如 dpg.last_item()))`
- 全屏警示框：设置 `modal=True` (默认 `False`)

按键：`mousebutton=` 添加

1. `dpg.mvMouseButton_Left` 左键
2. `dpg.mvMouseButton_right` 右键 (默认)

3. `_mvMouseButton_Middle_ 中键`
4. `_mvMouseButton_X1_ X1`
5. `_mvMouseButton_X2_ X2`

将窗口作为弹出窗口:

- 创建: `with dpg.window(label=" xxx ", modal=True, show=False, tag=" tag ", no_title_bar=True):`
- 调用: `callback=lambda: dpg.configure_item(" tag ", show=True)`

简易图表

添加: `dpg.add_simple_plot(label=" xxx ", default_value=(int/float), height= xxx)`

参数:

- 柱状 (默认线状) : `histogram=True`
- 覆盖在图表顶部的文字: `overlay=" xxx "`

更新数值: `dpg.set_value(" tag ", 更新后列表)`

暂存

```
def stage_items(): # 存入
    with dpg.stage(tag="stage1"):
        dpg.add_text("hello, i was added from a stage",
tag="text_tag")

def present_stage_items(): # 导出
    dpg.move_item("text_tag", parent="main_win")

with dpg.window(label="Tutorial", tag="main_win"):
    dpg.add_button(label="stage items", callback=stage_items)
    dpg.add_button(label="present stages items",
callback=present_stage_items)
```

- 存储: 在 `dpg.stage()`: 内添加有tag的组件
- 清理: 使用 `dpg.unstage(" tag ")` 和 `dpg.pop_container_stack()`

使用类(class)包装项

示例:

```
class Window:
    def __init__(self, label):
        self._children = []
        with dpg.stage() as stage:
            self.id = dpg.add_window(label=label)
        self.stage = stage
    def add_child(self, child):
        dpg.move_item(child.id, parent=self.id)
    def submit(self):
        dpg.unstage(self.stage)

class Button:
    def __init__(self, label):
        with dpg.stage():
            self.id = dpg.add_button(label=label)
    def set_callback(self, callback):
        dpg.set_item_callback(self.id, callback)

my_button = Button("Press me")
my_button.set_callback(lambda: print("I've been pressed!"))

my_window = Window("Tutorial")
my_window.add_child(my_button)
my_window.submit()
```

```
class Button:
    def __init__(self, label):
        with dpg.stage() as self._staging_container_id:
            self._id = dpg.add_button(label=label)
    def set_callback(self, callback):
        dpg.set_item_callback(self._id, callback)
    def get_label(self):
        return dpg.get_item_label(self._id)
    def submit(self, parent):
        dpg.push_container_stack(parent)
        dpg.unstage(self._staging_container_id)
        dpg.pop_container_stack()
```



```
my_button = Button("Press me")
my_button.set_callback(lambda: print("I've been pressed!"))

print(my_button.get_label())

with dpg.window(label="Tutorial", tag="main_win"):
    dpg.add_text("hello world")
my_button.submit("main_win")
```

表格

基础工作

- 创建: `with dpg.table():`
- 添加列: `dpg.add_table_column(label)` (最多64)
- 添加行内容: `with dpg.table_row():` (添加组件)
- 添加单元格: `with dpg.table_cell():` (行和列内) (相当于在这一行这一列里 `
`)

边框/背景

在 `dpg.table(加上以下内容)`

关键字	解释
<code>borders_innerH</code>	内边框H
<code>borders_innerV</code>	内边框V
<code>borders_outerH</code>	外边框H
<code>borders_outerV</code>	外边框V
<code>row_background</code>	每一行颜色交错

(H: 横向, V: 竖向)

列标题

在 `dpg.table()` 中把 `header_row` 设置为 `True`)

将显示 `dpg.add_table_column()` 的 `label`

列选项

关键字	默认值	描述
<code>init_width_or_weight</code>	0.0	设置列的起始宽度(固定策略)或比例(弹性策略)
<code>default_hide</code>	False	默认为隐藏/禁用列
<code>default_sort</code>	False	默认为排序列
<code>width_stretch</code>	False	列将被拉伸。最好是禁用水平滚动 (如果是 <code>_SizingStretchSame</code> 或 <code>_SizingStretchEqual</code> 默认)。
<code>width_fixed</code>	False	列将不会被拉伸。最好是启用水平滚动 (如果是 <code>_SizingFixedFit</code> 且表可调整大小, 否则是 <code>_SizingFixedFit</code>)
<code>no_resize</code>	False	禁用手动调整大小
<code>no_reorder</code>	False	禁用此列的手动重新排序, 这也将防止其他列交叉
<code>no_hide</code>	False	禁用隐藏/禁用此列的功能
<code>no_clip</code>	False	禁用此列的剪裁
<code>no_sort</code>	False	禁用此列的排序
<code>no_sort_ascending</code>	False	禁用按升序排序的功能
<code>no_sort_descending</code>	False	禁用按降序排序的功能

<code>no_header_width</code>	False	禁用页眉文本宽度对自动列宽的影响
<code>prefer_sort_ascending</code>	True	在此列上首次排序时将初始排序方向设置为升序(默认)
<code>prefer_sort_descending</code>	False	在此列上首次排序时, 将初始排序方向设置为降序

indent_enabled	默认 False 值	输入单元格时使用描述缩进值(第0列)
indent_disable	False	输入单元格时忽略当前的缩进值 (默认为0) 单元格内的缩进变化仍将被尊重(

###列排序

1. 在 `dpg.table` 添加属性 `sortable=True` 和 `callback`
2. 不要排序的列添加属性 `no_sort=True` (默认 `False`)
3. 自定义排序

排序回调示例:

```
def sort_callback(sender, sort_specs):

    # sort_specs scenarios:
    # 1. no sorting -> sort_specs == None
    # 2. single sorting -> sort_specs == [[column_id,
direction]]
    # 3. multi sorting -> sort_specs == [[column_id, direction],
[column_id, direction], ...]
    #
    # notes:
    # 1. direction is ascending if == 1
    # 2. direction is ascending if == -1

    # no sorting case
    if sort_specs is None: return

    rows = dpg.get_item_children(sender, 1)

    # create a list that can be sorted based on first cell
    # value, keeping track of row and value used to sort
    sortable_list = []
    for row in rows:
        first_cell = dpg.get_item_children(row, 1)[0]
        sortable_list.append([row, dpg.get_value(first_cell)])

    def _sorter(e):
        return e[1]
```

```

sortable_list.sort(key=_sorter, reverse=sort_specs[0][1] < 0)

# create list of just sorted row ids
new_order = []
for pair in sortable_list:
    new_order.append(pair[0])

dpg.reorder_items(sender, 1, new_order)

```

裁剪

优化性能

```

def clipper_toggle(sender):
    dpg.configure_item("table_clip",
clipper=dpg.get_value(sender))

with dpg.window(label="Tutorial"):
    dpg.add_checkbox(label="clipper", callback=clipper_toggle,
default_value=True)

    with dpg.table(header_row=False, tag="table_clip",
clipper=True):

        for i in range(5):
            dpg.add_table_column()
        for i in range(30000):
            with dpg.table_row():
                for j in range(5):
                    dpg.add_text(f"Row{i} Column{j}")

```

1. `dpg.table` 添加属性 `clipper=True`
2. 添加 `dpg.configure_item(" tag ", clipper=True)`

纹理

提供3种类型的纹理：静态的、原始纹理、动态的

可以在以下组件中使用纹理：`mvDrawImage`、`mvImage`、`mvImageButton`、`mvImageSeries`

它们总是一维列表或数组

使用关键字Show将打开纹理注册表

静态纹理

用于不经常更改的图像，通常在启动时加载

如果需要更新它们，您可以删除并重新创建它们。它们可以是列表、元组、NumPy数组，以及支持具有连续数据的Python缓冲区协议的任何类型

例子：

```
texture_data = []
for i in range(0, 100 * 100):
    texture_data.append(255 / 255)
    texture_data.append(0)
    texture_data.append(255 / 255)
    texture_data.append(255 / 255)

with dpq.texture_registry(show=True):
    dpq.add_static_texture(width=100, height=100,
        default_value=texture_data, tag="texture_tag")

with dpq.window(label="Tutorial"):
    dpq.add_image("texture_tag")
```

动态纹理

用于**每帧可以更改**的中小型纹理

可以使用 set_value 更新这些选项，但宽度和高度必须与第一次创建纹理时相同

除了执行安全检查和转化外，这些纹理与原始纹理相似

例子：

```
texture_data = []
for i in range(0, 100 * 100):
    texture_data.append(255 / 255)
    texture_data.append(0)
    texture_data.append(255 / 255)
    texture_data.append(255 / 255)
```

```

with dpg.texture_registry(show=True):
    dpg.add_dynamic_texture(width=100, height=100,
default_value=texture_data, tag="texture_tag")

def _update_dynamic_textures(sender, app_data, user_data):
    new_color = dpg.get_value(sender)
    new_color[0] = new_color[0] / 255
    new_color[1] = new_color[1] / 255
    new_color[2] = new_color[2] / 255
    new_color[3] = new_color[3] / 255

    new_texture_data = []
    for i in range(0, 100 * 100):
        new_texture_data.append(new_color[0])
        new_texture_data.append(new_color[1])
        new_texture_data.append(new_color[2])
        new_texture_data.append(new_color[3])

    dpg.set_value("texture_tag", new_texture_data)

with dpg.window(label="Tutorial"):
    dpg.add_image("texture_tag")
    dpg.add_color_picker((255, 0, 255, 255), label="Texture",
no_side_preview=True, alpha_bar=True, width=200,
callback=_update_dynamic_textures)

```

原始纹理

使用方式与动态纹理相同，但只接受数组(Numpy、PYTHON等)且不执行任何安全检查。

这些纹理用于需要每帧更新大型纹理的高性能应用程序

```

import array

texture_data = []
for i in range(0, 100 * 100):
    texture_data.append(255 / 255)
    texture_data.append(0)
    texture_data.append(255 / 255)

```

```

texture_data.append(255 / 255)

raw_data = array.array('f', texture_data)

with dpd.texture_registry(show=True):
    dpd.add_raw_texture(width=100, height=100,
        default_value=raw_data, format=dpd.mvFormat_Float_rgba,
        tag="texture_tag")

def update_dynamic_texture(sender, app_data, user_data):
    new_color = dpd.get_value(sender)
    new_color[0] = new_color[0] / 255
    new_color[1] = new_color[1] / 255
    new_color[2] = new_color[2] / 255
    new_color[3] = new_color[3] / 255

    for i in range(0, 100 * 100 * 4):
        raw_data[i] = new_color[i % 4]

with dpd.window(label="Tutorial"):
    dpd.add_image("texture_tag")
    dpd.add_color_picker((255, 0, 255, 255), label="Texture",
        no_side_preview=True, alpha_bar=True, width=200,
        callback=update_dynamic_texture)

```

格式

支持:

格式化	静态纹理	动态纹理	原始纹理
mvFormat_Float_rgba	✓	✓	✓
mvFormat_Float_rgb			✓
mvFormat_Int_rgba			
mvFormat_Int_rgb			

加载图像

使用 `load_image` 文件加载图像数据

此函数返回一个元组，其中

```
0->宽度
1->高度
2->途径
3->DATA(一维数组, mvBuffer)

如果失败，则返回 None。
```

接受的文件类型包括：

- JPEG(无12位/通道JPEG或采用算术编码的JPEG)
- PNG
- BMP
- PSD
- GIF
- HDR
- PIC
- PPM
- PGM

例子：

```
width, height, channels, data = dpj.load_image("xxx.png")

with dpj.texture_registry(show=True):
    dpj.add_static_texture(width=width, height=height,
        default_value=data, tag="texture_tag")

with dpj.window(label="Tutorial"):
    dpj.add_image("texture_tag")
```

保存图像

使用 `save_image` 将图像数据保存到文件。

该图像是从左到右、从上到下存储的像素的矩形。每个像素最多包含4个数据分量，每个通道8位交织，顺序如下：1=Y、2=YA、3=RGB、4=RGBA (y表示单色)

PNG创建的输出文件具有与输入相同数量的组件，BMP格式将文件格式中的Y扩展为RGB且不输出Alpha。

接受的文件类型包括：

- PNG
- JPG
- BMP
- TGA
- HDR

文件类型由扩展名决定。必须为小写(PNG、JPG、BMP、TGA、HDR)。

例子：

```
width, height = 255, 255

data = []
for i in range(width*height):
    data.append(255)
    data.append(255)
    data.append(0)

with dpg.window(label="Tutorial"):
    dpg.add_button(label="Save Image",
callback=lambda:dpg.save_image(file="newImage.png", width=width,
height=height, data=data, components=3))

dpg.show_viewport()
while dpg.is_dearpygui_running():
    dpg.render_dearpygui_frame()
```

主题

主题是由以下内容组成的容器：

主题组件：是主题中的容器，可以指定主题颜色/样式的目标项目类型

主题颜色：添加到主题组件并设置颜色的项

主题样式：添加到主题组件并设置样式的项目

主题可以：

绑定为默认主题。这将在所有窗口中产生全局效果

绑定到一个容器。如果适用的主题组件在该主题中，将应用到它的子项

如果主题中有适用的主题组件，则绑定到项类型、项容器或特定项

主题组件必须具有指定的项目类型。这可以是 `MvA11` 适用于所有组件或特定组件类型。

样式和颜色项目有一个命名的常量，并将该常量应用于其组件的命名项目类型。风格和颜色项目必须有一个命名的类别，常量在名称中包含其类别

主题颜色和样式分为以下类别：

`mvThemeCat_Plots` : 与绘图关联的项目。标识的样式/颜色常量：

`mvPlotCol_*` 或 `mvPlotStyle_*`

`mvThemeCat_Nodes` : 与bode关联的项。标识的样式/颜色常量：

`mvNodeCol_*` 或 `mvNodeStyle_*`

`mvThemeCat_Core` : 所有其他在dearpygui内组件。标识的样式/颜色常量：

`mvThemeCol_*` 或 `mvThemeStyle_*`

可参考 `dpg.show_style_editor()`

默认主题(全局)

将在所有窗口中全局应用该主题，并适用于子窗口

```

with dpg.theme() as global_theme:

    with dpg.theme_component(dpg.mvAll):
        dpg.add_theme_color(dpg.mvThemeCol_FrameBg, (255, 140,
23), category=dpg.mvThemeCat_Core)
        dpg.add_theme_style(dpg.mvStyleVar_FrameRounding, 5,
category=dpg.mvThemeCat_Core)

    with dpg.theme_component(dpg.mvInputInt):
        dpg.add_theme_color(dpg.mvThemeCol_FrameBg, (140, 255,
23), category=dpg.mvThemeCat_Core)
        dpg.add_theme_style(dpg.mvStyleVar_FrameRounding, 5,
category=dpg.mvThemeCat_Core)

dpg.bind_theme(global_theme)

```

只特定于组件

将主题应用于特定组件，将覆盖该指定项之前的任何主题

```

with dpg.window(label="Tutorial", pos=(20, 50), width=275,
height=225) as win1:
    t = dpg.add_input_text(default_value="some text")

with dpg.theme() as item_theme:
    with dpg.theme_component(dpg.mvAll):
        dpg.add_theme_color(dpg.mvThemeCol_FrameBg, (200, 200,
100), category=dpg.mvThemeCat_Core)
        dpg.add_theme_style(dpg.mvStyleVar_FrameRounding, 0,
category=dpg.mvThemeCat_Core)

dpg.bind_item_theme(t, item_theme)

```

主题的优先顺序

应用的主题的优先顺序按以下顺序排列:

1. 具体组件
2. 继承的容器
3. 全局

禁用组件的主题

每个组件都有单独的禁用主题，在禁用时会使用该主题

如果未设置禁用主题，则将使用默认禁用主题

1. 组件加 `enabled=False`
2. `with dpg.theme_xxx():` 添加参数 `enabled_state=False`

标记

图表图形

参数	意
<code>mvPlotMarker_None</code>	无
<code>mvPlotMarker_Circle</code>	圆圈
<code>mvPlotMarker_Square</code>	方形
<code>mvPlotMarker_Diamond</code>	钻石形
<code>mvPlotMarker_Up</code>	上
<code>mvPlotMarker_Down</code>	下
<code>mvPlotMarker_Left</code>	左
<code>mvPlotMarker_Right</code>	右
<code>mvPlotMarker_Cross</code>	叉号
<code>mvPlotMarker_Plus</code>	加号
<code>mvPlotMarker_Asterisk</code>	星号

支持颜色的

参数	意
<code>mvThemeCol_Text</code>	文本
<code>mvThemeCol_TabActive</code>	激活Tab
<code>mvThemeCol_SliderGrabActive</code>	滑块抓取活动

参数	意
mvThemeCol_TextDisabled	已禁用的文本
mvThemeCol_TabUnfocused	未聚焦Tab
mvThemeCol_Button	按钮
mvThemeCol_WindowBg	窗口背景
mvThemeCol_TabUnfocusedActive	激活未聚焦Tab
mvThemeCol_ButtonHovered	被鼠标悬停的按钮
mvThemeCol_ChildBg	子背景
mvThemeCol_DockingPreview	?
mvThemeCol_ButtonActiv	被点击的按钮
mvThemeCol_Border	边缘
mvThemeCol_DockingEmptyBg	?
mvThemeCol_Header	标题
mvThemeCol_PopupBg	?
mvThemeCol_PlotLines	图表的线
mvThemeCol_HeaderHovered	被悬停的标题
mvThemeCol_BorderShadow	边缘阴影
mvThemeCol_PlotLinesHovered	被悬停的图表的线
mvThemeCol_HeaderActive	激活的标题 ?
mvThemeCol_FrameBg	?
mvThemeCol_PlotHistogram	图表直方图
mvThemeCol_Separator	分隔符
mvThemeCol_FrameBgHovered	?
mvThemeCol_PlotHistogramHovered	被悬停的图表直方图
mvThemeCol_SeparatorHovered	被悬停的分隔符
mvThemeCol_FrameBgActive	?

参数	意
mvThemeCol_TableHeaderBg	表头背景
mvThemeCol_SeparatorActive	激活的分隔符 ?
mvThemeCol_TitleBg	标题背景
mvThemeCol_SeparatorTableBorderStrong	?
mvThemeCol_ResizeGrip	?
mvThemeCol_TitleBgActive	激活的标题背景 ?
mvThemeCol_TableBorderLight	?
mvThemeCol_ResizeGripHovered	?
mvThemeCol_TitleBgCollapsed	崩溃的标题背景 ?
mvThemeCol_TableRowBg	表格行背景
mvThemeCol_ResizeGripActive	?
mvThemeCol_MenuBarBg	菜单栏背景
mvThemeCol_TableRowBgAlt	表格行背景符号 ?
mvThemeCol_Tab	Tab
mvThemeCol_ScrollbarBg	滚动栏背景
mvThemeCol_TextSelectedBg	文本选择背景
mvThemeCol_TabHovered	被悬停的Tab
mvThemeCol_ScrollbarGrab	?
mvThemeCol_DragDropTarget	拖动目标
mvThemeCol_ScrollbarGrabHovered	?
mvThemeCol_NavHighlight	导航栏高亮
mvThemeCol_ScrollbarGrabActive	?
mvThemeCol_NavWindowingHighlight	导航栏窗口高亮 ?
mvThemeCol_CheckMark	打勾 ?
mvThemeCol_NavWindowingDimBg	?

参数	意
mvThemeCol_SliderGrab	?
mvThemeCol_ModalWindowDimBg	?

图表颜色

参数	意
mvPlotCol_Line	线
mvPlotCol_LegendBg	图例背景
mvPlotCol_YAxisGrid	Y轴网格
mvPlotCol_Fill	填充
mvPlotCol_LegendBorder	图例边框
mvPlotCol_YAxis2	Y轴2
mvPlotCol_MarkerOutline	?
mvPlotCol_LegendText	?
mvPlotCol_YAxisGrid2	Y轴网格2
mvPlotCol_MarkerFill	标记填充
mvPlotCol_TitleText	标题文本
mvPlotCol_YAxis3	Y轴3
mvPlotCol_ErrorBar	错误栏
mvPlotCol_InlayText	?
mvPlotCol_YAxisGrid3	Y轴网格3
mvPlotCol_FrameBg	?
mvPlotCol_XAxis	X轴
mvPlotCol_Selection	选择器
mvPlotCol_PlotBg	图表背景
mvPlotCol_XAxisGrid	X轴网格

参数	意
mvPlotCol_Query	查询
mvPlotCol_PlotBorder	图表边框
mvPlotCol_YAxis	Y轴
mvPlotCol_Crosshairs	十字线 (坐标)

node颜色

参数	意
mvNodeCol_NodeBackground	Node背景
mvNodeCol_TitleBarSelected	已选择标题栏
mvNodeCol_BoxSelector	盒子选择器
mvNodeCol_NodeBackgroundHovered	被悬停的Node背景
mvNodeCol_Link	链接
mvNodeCol_BoxSelectorOutline	?
mvNodeCol_NodeBackgroundSelected	已选择Node背景
mvNodeCol_LinkHovered	被悬停的链接
mvNodeCol_GridBackground	网格背景
mvNodeCol_NodeOutline	?
mvNodeCol_LinkSelected	已选择背景
mvNodeCol_GridLine	网格线
mvNodeCol_TitleBar	标题栏
mvNodeCol_Pin	钉
mvNodeCol_PinHovered	已悬停的钉
mvNodeCol_TitleBarHovered	已悬停的标题栏

核心样式

参数	组件	意
mvStyleVar_Alpha	1	透明度
mvStyleVar_WindowPadding	2	窗口填充
mvStyleVar_WindowRounding	1	窗口圆角
mvStyleVar_WindowBorderSize	1	窗口边缘大小
mvStyleVar_WindowMinSize	2	窗口最小大小
mvStyleVar_WindowTitleAlign	2	窗口标题对齐
mvStyleVar_ChildRounding	1	子边距
mvStyleVar_ChildBorderSize	1	子边缘大小
mvStyleVar_PopupRounding	1	?
mvStyleVar_PopupBorderSize	1	?
mvStyleVar_FramePadding	2	?
mvStyleVar_FrameRounding	1	?
mvStyleVar_FrameBorderSize	1	?
mvStyleVar_ItemSpacing	2	组件间隔
mvStyleVar_ItemInnerSpacing	2	组件内间隔
mvStyleVar_IndentSpacing	1	缩进间隔
mvStyleVar_CellPadding	2	?
mvStyleVar_ScrollbarSize	1	滚动条大小
mvStyleVar_ScrollbarRounding	1	滚动条圆角
mvStyleVar_GrabMinSize	1	?
mvStyleVar_GrabRounding	1	?
mvStyleVar_TabRounding	1	Tab圆角
mvStyleVar_ButtonTextAlign	2	按钮文本对齐
mvStyleVar_SelectableTextAlign	2	可选文本对齐

打印样式

参数	组件	意
mvPlotStyleVar_LineWeight	1	线宽
mvPlotStyleVar_Marker	1	标记
mvPlotStyleVar_MarkerSize	1	标记大小
mvPlotStyleVar_MarkerWeight	1	标记宽
mvPlotStyleVar_FillAlpha	1	填充透明度
mvPlotStyleVar_ErrorBarSize	1	错误栏大小
mvPlotStyleVar_ErrorBarWeight	1	错误栏宽度
mvPlotStyleVar_DigitalBitHeight	1	?
mvPlotStyleVar_DigitalBitGap	1	?
mvPlotStyleVar_PlotBorderSize	1	图表边框大小
mvPlotStyleVar_MinorAlpha	1	?
mvPlotStyleVar_MajorTickLen	2	?
mvPlotStyleVar_MinorTickLen	2	?
mvPlotStyleVar_MajorTickSize	2	?
mvPlotStyleVar_MinorTickSize	2	?
mvPlotStyleVar_MajorGridSize	2	?
mvPlotStyleVar_MinorGridSize	2	?
mvPlotStyleVar_PlotPadding	2	图表填充
mvPlotStyleVar_LabelPadding	2	标签填充
mvPlotStyleVar_LegendPadding	2	图例填充
mvPlotStyleVar_LegendInnerPadding	2	内图例填充
mvPlotStyleVar_LegendSpacing	2	图例边距
mvPlotStyleVar_MousePosPadding	2	鼠标位置填充

参数	组件	意
mvPlotStyleVar_AnnotationPadding	2	注释填充
mvPlotStyleVar_FitPadding	2	?
mvPlotStyleVar_PlotDefaultSize	2	图表默认大小
mvPlotStyleVar_PlotMinSize	2	图表最小大小

Node样式

参数	组件	意
mvNodeStyleVar_GridSpacing	1	网格间距
mvNodeStyleVar_NodeCornerRounding	1	Node边角圆角
mvNodeStyleVar_NodePaddingHorizontal	1	Node横向填充
mvNodeStyleVar_NodePaddingVertical	1	Node竖向填充
mvNodeStyleVar_NodeBorderThickness	1	边缘大小
mvNodeStyleVar_LinkThickness	1	链接厚度
mvNodeStyleVar_LinkLineSegmentsPerLength	1	链接线长
mvNodeStyleVar_LinkHoverDistance	1	链接悬停距离
mvNodeStyleVar_PinCircleRadius	1	钉 圆半径
mvNodeStyleVar_PinQuadSideLength	1	钉 四边形长
mvNodeStyleVar_PinTriangleSideLength	1	钉 三角形边长
mvNodeStyleVar_PinLineThickness	1	钉 线条厚度
mvNodeStyleVar_PinHoverRadius	1	钉 悬停半径
mvNodeStyleVar_PinOffset	1	钉 偏移

工具提示

是当项目悬停时显示的窗口，可以保存任何其他组件的容器，必须具有父参数

```
dpg.add_text("Hover me", tag="tooltip_parent")
with dpg.tooltip("tooltip_parent"):
    dpg.add_text("A tooltip")
```