后端 & 写一个后端

包含两篇文章；这篇文章主要是为了帮助大家理解所谓后端的概念，backend这个概念在matplotlib中还是经常会见到的；

如果你并不计划自己写一个后端，其实这篇文章不必看完...你只需要直到后端是一个负责将你用代码描述的图形实实在在的在屏幕上画出的东西就可以了；

Backends

什么是backends?

网络上的许多文章和资料中都会提到“backends”，很多初学用户很难都会对这个概念感到困扰；Matplotlib致力于开发更多不同的应用场景和不同的输出格式；一些用户使用python shell与matplotlib交互，他们键入代码就可以把绘图窗口弹出来；另一些用户使用Jupyter记事本，并且希望在文字中插入图像以进行快速的数据分析；还有一用户将Matplotlib嵌入到用户图形界面，例如PyQt或者PyGObject，以构建丰富的应用程序；有一些人使用matplotlib批处理脚本，由数字的模拟结果生成图片；还有一些人运行web应用程序服务器来动态的提供图片；

为了支持这些不同的使用场景，matplotlib可以针对不同的输出，这些输出功能中的每一个都被称为后端；对应的前端是用户所面对的代码，例如绘图代码就是前端；而后端则承担了绘制一幅图片的幕后的一切复杂工作；实际上matplotlib中有两种后端：用户接口后端（用在PyQt/PySide,PyGObject,Tkinter,wxPython,或者macOS/Cocoa中），还有一种是硬拷贝，用于生成图片文件（PNG,SVG,PDF...也被称为非交互式后端）

选择一个后端

有善终方法配置后端：

- 在matplotlibrc文件中修改`rcParams["backend"]`参数
- 修改`MPLBACKEND`环境变量（至于什么是环境变量...不懂的话请百度一下，这是一个系统级别的概念）
- 使用函数`matplotlib.use()`

接下来是更详细的描述：

如果有多于一个的配置存在，那么列表中的最后一个生效；例如调用了`matplotlib.use()`函数将会覆盖你在`matplotlibrc`中的配置；

若没有一个明确的配置，那么matplotlib会根据你的系统中可用的后端以及是 否有GUI事件轮询已经准备好运行 来自动的选择一个后端；自动选用的优先顺序：MacOSX,QtAgg,GTK4Agg,Gtk3Agg, TkAgg, WxAgg, Agg.最后一个，Agg是一个非交互性的后端，这个后端只可以输出文件；在linux中，如果matplotlib不能链接上X display或者Wayland display，那么Agg会被使用；

以下配置的细节：

1. 在`matplotlibrc`中设置`rcParams["backends"]`

```
backend : qtagg # 使用antigrain（agg）渲染的pyqt
```

更多详情https://matplotlib.org/stable/tutorials/introductory/customizing.html

2. 设置`MPLBACKEND`环境变量

   你可以对当前的shell或者对单个脚本设置环境变量；

   在unix中：

   ```
   > export MPLBACKEND=qtagg
   > python simple_plot.py

   > MPLBACKEND=qtagg python simple_plot.py
   ```

   在windows中：

   ```
   > set MPLBACKEND=qtagg
   > python simple_plot.py
   ```

   环境变量一旦被设置就会覆盖掉所有`matplotlibrc`中的设置；因为`MPLBACKEND`设置是全局的，就像我们在`.bashrc`或者`.profiles`中的的设置一样（注：这两个文件是Unix-linux系统中常见的配置文件，系统中的很多基础配置都能在这两个文件中被配置）；但是并不推荐您去配置环境变量，因为可能会导致matplotlib出现一些反直觉的行为；

3. 如果你的脚本依赖于一个特殊的backend，那么你可以使用matplotlib.use()函数；

   ```
   import matplotlib
   matplotlib.use('qtagg')
   ```

   这个函数要在figure被绘制之前使用，否则matplotlib或许会选择backend失败并且会抛出一个错误；

   如果用户希望使用不同的后端，再用了`use`函数情况下就需要修改代码；因此如非必要不要显示的调用`use`函数；

内置后端（the builtin backends）

默认情况下，matplotlib应当自动的选择一个后端，这个被选中的后端应该同时支持交互式的工作，由脚本打印图像，以及将结果输出到屏幕或者文件，所以至少在最开始，你是不用担心后端问题的；常见的意外是你的Python环境中没有安装`tkinter`同时没有安装其他的GUI工具包；这种情况在linux环境下比较常见，在linux下你需要安装一个叫做`python-tk`的包；

如果你想写一个用户图形界面，或者构建一个web应用服务器，或者需要更好的理解代码执行过程中发生的事情，请继续阅读下去；

为了使用户图形接口更容易被定制，matplotlib从画布（canvas：用于绘制的地方）中分离了渲染器（render：实际执行绘制的东西）的概念；对用户接口而言最典型的渲染器是`Agg`，这是一个使用叫做`Anti-Grain Geometry`的c++库去渲染figure的像素的渲染器；`QtAgg`，`GTK4Agg`，`GTK3Agg`，`wxAgg`，`TkAgg`，和 `macosx` 后端用的都是`Agg`；还有一个替代性的渲染器是基于Cairo库的，`QtCairo`等用的是这个；

对于渲染引擎这个概念概念而言，用户需要区分矢量渲染器和点阵渲染器；矢量图像语言以“从这个点到那个点画一条线”这样的方式描述绘图命令，因此尺寸是随意的（所以矢量图的特征是无论怎么放大都不会变糊）；点阵后端生成一个像素绘制结果，这个结果是依赖DPI设置的；

静态后端

这是一些matplotlib渲染器的总结（是一些非交互式的后端）

Renderer	Filetypes	Description
AGG	png	raster graphics -- high quality images using the Anti-Grain Geometry engine.
PDF	pdf	vector graphics -- Portable Document Format output.
PS	ps, eps	vector graphics -- PostScript output.
SVG	svg	vector graphics -- Scalable Vector Graphics output.
PGF	pgf, pdf	vector graphics -- using the pgf package.
Cairo	png, ps, pdf, svg	raster or vector graphics -- using the Cairo library (requires pycairo or cairocffi).

为了保存非交互式后端生成的结果，我们可以使用`matplotlib.pyplot.savefig('filename')`方法；

交互式后端

这些是支持用户界面和渲染器的后端；也是交互式后台，能够将结果显示到屏幕上，搭配非交互式渲染器也能将结果输出到文件中;

Backend	Description
QtAgg	Agg rendering in a Qt canvas (requires PyQt or Qt for Python, a.k.a. PySide). This backend can be activated in IPython with `%matplotlib qt`. The Qt binding can be selected via the `QT_API` environment variable; see Qt Bindings for more details.
ipympl	Agg rendering embedded in a Jupyter widget (requires ipympl). This backend can be enabled in a Jupyter notebook with `%matplotlib ipympl`.
GTK3Agg	Agg rendering to a GTK 3.x canvas (requires PyGObject and pycairo). This backend can be activated in IPython with `%matplotlib gtk3`.
GTK4Agg	Agg rendering to a GTK 4.x canvas (requires PyGObject and pycairo). This backend can be activated in IPython with `%matplotlib gtk4`.
macosx	Agg rendering into a Cocoa canvas in OSX. This backend can be activated in IPython with `%matplotlib osx`.
TkAgg	Agg rendering to a Tk canvas (requires TkInter). This backend can be activated in IPython with `%matplotlib tk`.
nbAgg	Embed an interactive figure in a Jupyter classic notebook. This backend can be enabled in Jupyter notebooks via `%matplotlib notebook`.

Backend	Description
WebAgg	On `show()` will start a tornado server with an interactive figure.
GTK3Cairo	Cairo rendering to a GTK 3.x canvas (requires PyGObject and pycairo).
GTK4Cairo	Cairo rendering to a GTK 4.x canvas (requires PyGObject and pycairo).
wxAgg	Agg rendering to a wxWidgets canvas (requires wxPython 4). This backend can be activated in IPython with `%matplotlib wx`.

ipympl

Juypter小插件的生态系统迭代的太快了...以至于matplotlib难以直接进行支持；情啊安装ipympl

```
pip install ipympl
```

或

```
conda install ipympl -c conda-forge
```

使用非内置的后端（non-builtin backends）

更一般的说，任何能够导入的后端都可以通过上述的方法来选择使用；如果`name.of.the.bakcend`是包含后端的模块，请使用`module://name.of.the.backend`作为后端的名称，例如`matplotlib.use('module://name.of.the.backend')`

关于后端实现的信息可以在这里获得：Writing a backend -- the pyplot interface.

写一个后端--pyplot接口

接下来的内容建立在你已经理解了上面那篇文章的内容的基础上，这篇文章是对第三方后端开发者的参考；也仅仅介绍backend和pyplot之间的互动，不涉及渲染部分，渲染在`backend_template`中有描述；

有两个API用于定义后端：一个是新的 canvas-based API（于matplotlib3.6引入），另一个是老的function-based API；新的API更容易实现，因为其中的许多方法可以从“parent backends”中继承；如果matplotlib<3.6的向后兼容性不是问题，建我们还是建议使用新的API；不过旧有的API也仍然受支持；

总的来说，一个backend模组需要为`pyplot`提供信息，因此：

1. `pyplot.figure()`可以创建一个新的`Figure`实例，并且将这个Figure实例与后端提供的画布类的实例相关联，它本身托管在后端提供的管理器类的实例中；
2. `pyplot.show()`可以展示所有的figures，并且可以启动GUI事件轮询（注：启动GUI事件轮询就意味着show所展示的窗口是可是实现一部分GUI窗口功能的，例如放大，关闭，平移等）

为了做到这些，backend module必须定义一个`FigureCanvasBase`的子类`backend_module.FigureCanvas`；在canvas-based API中，这是对后端模块唯一的一个严格要求；function-based API需要定义许多额外的包级别的函数；

Canvas-based API (matplotlib >= 3.6)

1. 创建一个figure：`pyplot.figure()`调用`figure = Figure()`; `FigureCanvas.new_manager(figure, num)`（`new_manager`是一个类方法）以实例化一个canvas，一个manager，并且设置`figure.canvas`和`figure.canvas.manager`的属性；Figure的反序列化使用同样的方式，但是要用反序列化figure的函数替代实例化figure的`Figure()`

   交互式的后端应当通过设置`FigureCanvas.manager_class`属性为所需要管理的类，来自定义`new_manager`的效果，此外（如果不能在管理器之前创建canvas，例如wx后端这中情况）还应重写（override）`FigureManager.create_with_canvas`类方法；（非交互式后端通常可以使用琐碎的FigureManagerBase，因此可以跳过这个步骤）

   在一个新的figure通过`pyplot`被注册后，如果是在交互模式下，`pyplot`将会调用他的canvas`draw_idle()`方法，这个方法可以按照我们希望的方式重写；

2. 展示figures：`pytplot.show()`调用`FigureCanvas.manager_class.pyplot_show()`（一个类方法），以开启主事件轮询；

   默认情况下，`pyplot_show()`会检查是否存在任何使用pyplot所注册的管理器(manager)，对所有的manager调用`manager.show()`，然后如果使用`block=True`调用`FigureCanvas.manager_class.start_main_loop()`，则启动主事件轮询；因此交互式的后端应当相应的重写`FigureCanvas.manager_class.start_main_loop`类方法

Function-based API

1. 创建一个figure：`pyplot.figure()`调用`new_figure_manager(num,*args,**kwargs)`，反序列化调用`new_figure_manager_given_figure(num,figure)`

   此外，在交互式模式中，可以通过提供模块级别的`draw_if_interactive()`函数来订制新注册图形的第一个绘图

2. 展示figures：`pyplot.show()`调用一个模块级别的`show()`函数，这个函数通常是由`ShowBase`类和他的`mainloop`方法生成的