Dash 是一个通过 python 语言来开发 web 界面，并在运行时将前端编译成 html/js/css 并运行的框架。其主要特点是：

1. 完全使用 Python 来开发，无论是界面元素还是服务端逻辑。Dash 不仅负责将 python 编写的界面元素转化为 html/js/css，而且负责前端与后端的交互。因此，程序员可以完全在 python 语言框架内完成全部工作。
2. Dash 内置 flask 服务器，因此 flask 的诸多运行机制也一样适用，比如可以如下获取 cookie:

```
import flask

flask.request.cookies.get("your cookie name")
```

上述代码可以在全局使用，也可以 callback 方法中使用（这里的 callback 特指 Dash 的 app.callback）。

3. 与 plotly 无缝集成，具有较强的可视化能力。尤为可贵的是，可视化是 web 交互式的，但又完全是通过 Python 来编程的（注意 javascript 提供了交互能力，而 python 是不可能的）。

1. 核心概念

在一般的 web 编程中，我们通常要为视图准备两个 handler，一个响应 GET 请求，提供初始化的界面，供用户操作；另一个响应 POST 请求，接收用户提交的数据，进行处理后，再重定向到新的视图。

在 Dash 中，上述交互对用户是不可见的。我们准备一个视图，通过 callback 来处理用户的输入。当用户在前端进行输入时，dash 自动将这些消息 (click 事件或者值变更）及时传递给后端，应用在则 callback 中得到这些输入值，完成校验，将结果更新到同一个视图中的某些控件上，或者输出一个新的视图（如果指定的 Output 中绑定了某个 html 控件的 children 属性的话）。

```
    layout = dbc.Container(
        [
            html.Br(),
            dbc.Container(
                [
                    dcc.Location(id="urlLogin", pathname="/login", refresh=True),
                    html.Div(
                        [
                            dbc.Container(
                                html.Img(
                                    src="/assets/dash-logo-stripe.svg",
    className="center"
                                ),
                            ),
                            dbc.Container(
                                id="loginType",
                                children=[
                                    dcc.Input(
                                        placeholder="Enter your username",
                                        type="text",
                                        id="usernameBox",
                                        className="form-control",
                                        n_submit=0,
                                    ),
                                    html.Br(),
                                    dcc.Input(
                                        placeholder="Enter your password",
                                        type="password",
                                        id="passwordBox",
                                        className="form-control",
                                        n_submit=0,
                                    ),
                                    html.Br(),
                                    html.Button(
                                        children="Login",
                                        n_clicks=0,
                                        type="submit",
                                        id="loginButton",
                                        className="btn btn-primary btn-lg",
                                    ),
                                    html.Br(),
                                ],
                                className="form-group",
                            ),
                        ]
                    ),
                ],
                className="jumbotron",
            ),
        ]
```

```
)

################################################################################
# LOGIN BUTTON CLICKED / ENTER PRESSED - REDIRECT TO PAGE1 IF LOGIN DETAILS ARE
CORRECT
################################################################################
@callback(
    Output("urlLogin", "pathname"),
    Input("loginButton", "n_clicks"),
    [State("usernameBox", "value"), State("passwordBox", "value")],
    suppress_callback_exceptions=True,
)
def on_login(n_clicks, username, password):
    if n_clicks == 0:
        print("first loaded")
    else:
        print("login button clicked with:", username, password)
        response = dash.callback_context.response
        if login_user(response, username, password):
            # JUMP TO INDEX PAGE
            return "/"

@callback(
    Output("usernameBox", "className"),
    [
        Input("loginButton", "n_clicks"),
        Input("usernameBox", "n_submit"),
        Input("passwordBox", "n_submit"),
    ],
    [State("usernameBox", "value"), State("passwordBox", "value")],
)
def update_output(n_clicks, usernameSubmit, passwordSubmit, username, password):
    print("update_output by usernameBox")
    if (n_clicks > 0) or (usernameSubmit > 0) or (passwordSubmit > 0):
        if get_current_user() is None:
            response = dash.callback_context.response
            if login_user(response, username, password):
                return "form-control"
            else:
                return "form-control is-invalid"
        else:
            return "form-control is-invalid"
    else:
        return "form-control"

################################################################################
# LOGIN BUTTON CLICKED / ENTER PRESSED - RETURN RED BOXES IF LOGIN DETAILS
INCORRECT
################################################################################
@callback(
    Output("passwordBox", "className"),
```

```
    [
        Input("loginButton", "n_clicks"),
        Input("usernameBox", "n_submit"),
        Input("passwordBox", "n_submit"),
    ],
    [State("usernameBox", "value"), State("passwordBox", "value")],
)
def update_output(n_clicks, usernameSubmit, passwordSubmit, username, password):
    print("in update_output: passwordBox")
    if (n_clicks > 0) or (usernameSubmit > 0) or (passwordSubmit) > 0:
        if get_current_user() is None:
            response = dash.callback_context.response

            if login_user(response, username, password):
                return "form-control"
            else:
                return "form-control is-invalid"
        else:
            return "form-control is-invalid"
    else:
        return "form-control"
```

上述代码定义了一个视图。有几点需要注意：

1. 这个视图中，存在一个 dcc.Location 对象。在视图中只要存在这个对象，我们就可以通过 callback 来改变其 url 属性，从而引起重新加载（重新加载既可能是原视图，也可能是新的页面）
2. 第一个 callback 方法中，我们接收来自 usernameBox 和 passwordBox 中的值，判断用户能否登录。如果允许登录，我们就修改上述 dcc.Location 对象的路径为"/"，在某个地方，我们将这个路径指向应用程序的缺省显示页（即 homepage）。这主要是通过将上述两个控件（以及 loginButton）作为输入，将 urlLogin 控件（即 dcc.Location）作为输出绑定在一起，并且在成功的情况下，返回路径"/"来实现的。
3. 其它两个方法分别处理上述输入控件激发数据变化，但验证不能通过的情况下，向用户提示哪个控件的输入有错误。这时我们不修改 urlLogin 控件的 pathname 值，所以我们将仍然停留在本页面。

2. 路由

一般而言，Dash 应用程序没有路由。但是，只要是构建大型应用程序，都不可避免地涉及路由问题：在传统 c/s 程序中，功能都被组织成一个个 page（每个 page 对应一个 url），服务端通过响应浏览器提交的路径，根据路由配置来生成响应页面。

在大型 SPA 应用程序中，即使 server 端只处理少数几个路由，我们也往往在前端生成路由表来切换视图，以保持代码的简洁可读（另一个原因是可以实现分步加载，提高初次响应速度）。

很显然，要构建复杂的 Dash 应用，我们也必须使用路由。Dash 目前没有提供前端路由机制，相反，它提供了一种多页应用的方式。其核心是，如果当前视图处理完成了，那么它可以改变页面中的 Location 控件的 pathname 属性（当然是通过 callback 机制，见上一节），从而引起重定向。

下面是一个简单的路由实现。

首先，我们按照 Dash 的惯例，定义一个 layout:

```
# ROUTING.PY

layout = html.Div(
    [
        dcc.Location(id="router", refresh=False),
        html.Div(id="page-content")
    ],
    id="rootElement",
)
```

这个 layout 非常简单，因为我们不打算用它来展示任何东西，只是简单地用来做路由转发。为了做到这一点， 我们需要提供修改 ur 这个 Location 控件的 pathname 属性（假设所有的跳转都在本服务以内）的机制，这是通过将这个控件作为 Output，Location 控件作为输入进行绑定来实现的：

```
# ROUTING.PY

@callback(Output("page-content", "children"),
          [Input("router", "pathname")])
def _routing(pathname):
    # ENSURE AUTH
    if not auth.get_current_user():
        return auth.layout

    handler = routes.get(pathname, None)
    if handler is None:
        return homepage.layout

    return handler()
```

这里的 Location 控件比较特殊，它并不会出现在页面元素中。似乎一个页面也允许多个 Location 控件，但都将更新当前窗口的地址。

上述 callback 的机制是，当检测到当前窗口的 pathname（dash 用语，指 http://host:port/server_path ? query_string 中的 server_path）发生改变后，将会触发这个 callback 运行，并且函数得到新的 pathname。在这里，我们检测用户是否已登录，如果没有，则返回登录页面（auth.layout），否则，调用路径对应的事件处理函数，一般地，我们在事件处理函数中，返回一个新

的页面视图，而 dash 则将这个新的视图装载到上述"page-content"中。page-content 中原来的元素则被清除（这里可能引起内存问题）。

这里并没有看到任何实际的路由和路由处理函数。所有的路由，都收集在 routing.py 中的 routes 集合中，并且我们提供一个装饰器来让组件自己注册路由：

```
# ROUTING.PY

routes = {}

def on(pathname: str):
    """
    dispatch function.
    """
    def decorator(func):
        global routes

        routes[pathname] = func
        return func
    return decorator
```

auth 视图稍微复杂一些，与路由相关的部分是：

```
layout = dbc.Container(
    [
        html.Br(),
        dbc.Container(
            [
                dcc.Location(id="urlLogin", pathname="/login", refresh=True),
              ...

# CALLBACK
@callback(
    Output("urlLogin", "pathname"),
    Input("loginButton", "n_clicks"),
    [State("usernameBox", "value"), State("passwordBox", "value")],
    suppress_callback_exceptions=True,
)
def on_login(n_clicks, username, password):
    if n_clicks == 0:
       ...
```

在上面的代码中，loginButton 的点击、或者 usernameBox、passwordBox 的值变化都会引起这个 callback 被调用，并且当它被调用时，我们已经拿到了用户输入的 username 和 password（按声明顺序绑定到 Input, State 对象上）。我们现在要做的就是，检查 username 和 password 是否有效，如果有效，则让用户登录，生成 session，最终返回一个路径（str），dash 会将这个路径更新到 Location

控件，最终导致 dash 向后台请求新的页面。
注册路由

在 auth 模块的 controller.py 中，我们注册两个路由：

```
from .models import get_current_user, remove_current_user
from .view import layout
from alpha.web import routing

@routing.on('/logout')
def logout():
    if  get_current_user():
        remove_current_user()

    return layout

@routing.on('/login')
def login():
    return layout
```

当上述 controller.py 被导入时，注册将自动完成。但我们需要有方法来保证这个注册一定会在程序开始前就完成。因此 routing 中提供了一个 build_blueprints() 的方法：

```
def build_blueprints():
    """
    collect all routes by import controller from web/*/controller.py
    """
    _dir = os.path.dirname(os.path.abspath(__file__))
    package_prefix = "alpha.web."
    for pyfile in glob.glob(f"{_dir}/**/controller.py"):
        sub = pyfile.replace(f"{_dir}/", "").replace(".py", "").replace("/", ".")
        module_name = package_prefix + sub
        importlib.import_module(module_name)
```

这个方法要求，所有的页面模块都放在 alpha.web 目录下，并且事件响应都在 controller.py 中。如果你的代码有其它的组织方式，需要对此进行修改。

我们再来看另一个视图，即根视图。当用户登录后，就会进入到这个视图。这个视图位置在 alpha.web.homepage 下，它的 view 特别简单：

```
from dash import html, dcc, callback, Input, Output
from alpha.web import auth

layout = html.Div(
    [
        dcc.Location(id="homepage", refresh=False)
        html.H1("logout"),
        dcc.Link("Logout", href="/logout"),
    ], id="rootElement"
)

```

它提供了一个链接，当用户点击后，就会跳转到/logout 这个路径上（这是 Dash 中改变视图的另一个方法）。然后 routing 模块检测到新的 pathname，于是触发 callback，找到/logout 的处理函数，退出当前用户，并跳转到登录页面。因此，这里的 Location 组件并没有起作用，我们声明这样一个组件，只是为了将来之用。

3. Gotcha

routing.py 能够提供路由的关键原因是，rootElement 是后面所有视图的父结点，只要这个节点存在，路由机制就有效。

从上图可以看出，即使切换到了 logout 视图，这个 rootElement 仍然存在。因此，在 routing 中，我们必须将新的视图更新到“page-content"节点下，而不是 rootElement 中。否则，我们将摧毁这个总路由。