Reflex & FastAPI 快速上手指南

本指南帮助您快速掌握使用 **Reflex** 进行全栈开发以及 **FastAPI** 进行后端模型服务开发的核心概念与代码片段。

1. Reflex 全栈开发 (前端 + 逻辑)

Reflex 是一个纯 Python 的全栈框架，它将前端 UI 编译为 React，后端逻辑运行在 FastAPI 上。

核心概念

- **Compoents (组件)**: 定义 UI (如 `rx.text`, `rx.button`, `rx.input`).
- **State (状态)**: 定义应用变量和后端逻辑函数 (Event Handlers).
- **Pages (页面)**: 将 UI 绑定到路由.

快速入门代码

```
import reflex as rx

# 1. State: 定义状态和逻辑
class State(rx.State):
    prompt: str = ""
    result: str = ""

    # Event Handler: 处理事件的后端函数
    def handle_submit(self):
        # 这里可以调用模型
        self.result = f"收到请求: {self.prompt}"

# 2. UI: 定义前端界面
def index():
    return rx.container(
        rx.vstack(
            rx.heading("SoulPet AI Chat"),
            rx.input(
                placeholder="输入你的问题...",
                on_blur=State.set_prompt,  # 绑定输入到 State
            ),
            rx.button("发送", on_click=State.handle_submit),
            rx.text(State.result), # 响应式显示结果
            spacing="4",
        )
    )

# 3. App: 定义应用
app = rx.App()
app.add_page(index)
```

常用命令

- `reflex init`: 初始化项目结构.
- `reflex run`: 运行开发服务器 (默认端口 3000).

---

2. FastAPI 后端开发 (模型 API 服务)

虽然 Reflex 内部集成了 FastAPI，但在需要独立的微服务或非前端 API 时，直接使用 FastAPI 非常高效。

核心代码结构

```
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

# 1. 定义数据模型 (用于请求体校验)
class QueryRequest(BaseModel):
    text: str
    max_length: int = 50

class QueryResponse(BaseModel):
    generated_text: str

# 2. 定义 API 路由
@app.post("/api/predict", response_model=QueryResponse)
async def predict(request: QueryRequest):
    # 这里加载和调用深度学习模型
    # model_output = model.generate(request.text)
    fake_output = f"模型处理了: {request.text}"

    return {"generated_text": fake_output}

# 3. 启动 (使用 uvicorn)
# uvicorn main:app --reload
```

---

3. 进阶：Reflex 与 FastAPI 混合使用

Reflex 应用本身就是一个 FastAPI 应用，你可以挂载自定义的 API 路由，实现"前端界面 + 纯 API 接口"共存。

```
import reflex as rx
from fastapi import FastAPI

# 获取 Reflex 的底层 FastAPI 实例
# 注意：这通常在 reflex init 生成的 rxconfig.py 或主入口中配置
# 或者在 Reflex 页面代码中添加 API 路由

async def custom_api_handler():
    return {"message": "Hello from custom API"}
```

```
app = rx.App()
app.api.add_api_route("/custom-api", custom_api_handler)
```

4. 核心组件速查表

功能	**Reflex 组件 (rx)**	常用属性/事件
布局	rx.box, rx.vstack, rx.hstack, rx.container	spacing, align_items, padding
文本	rx.text, rx.heading, rx.markdown	font_size, color
输入	rx.input, rx.text_area	on_change, on_blur, value
交互	rx.button	on_click, loading (加载状态)
反馈	rx.spinner, rx.progress	-

5. UV 环境操作提醒

您修改了依赖配置，请记得同步环境：

```
uv sync
```