一、快速了解LangGraph

LangGraph是一个功能非常强大的大语言模型本地应用构建框架。不只是包含了各种基于大语言模型构建本地应用的工具，更重要的是，他积累了非常多使用大语言模型构建本地应用的经验，并且将这些经验总结成了非常多的案例，让大家可以直接使用。

但是，LangGraph并不是一个独立的框架，他是LangChain框架的一个生态组件。所以，如果脱离LangChain来介绍LangGraph，那纯粹是耍流氓。因此后面的内容，是预设大家有LangChain的基础的，并且最好是看过我之前分享的LangChain视频。

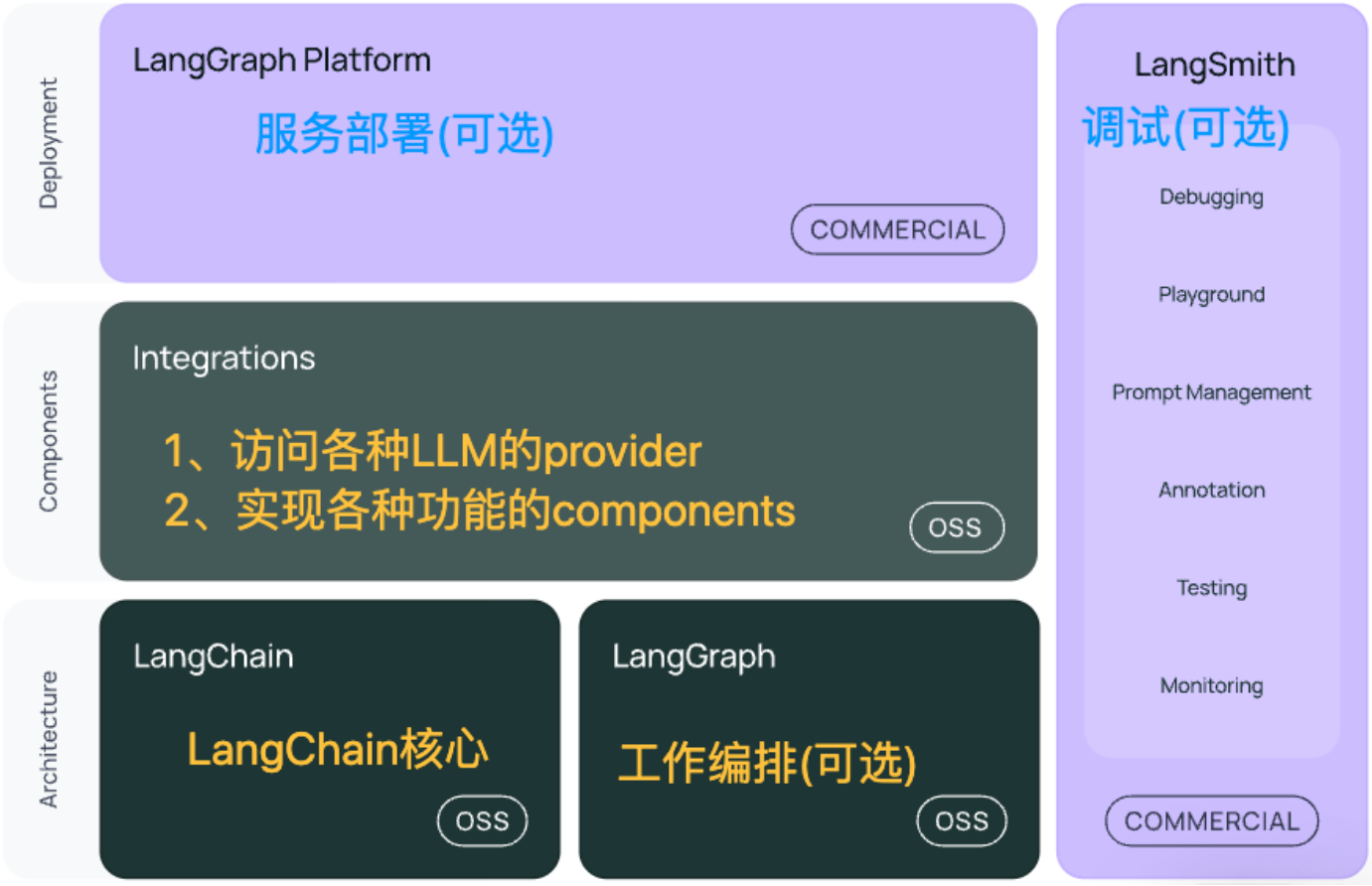
课程目标：

- 理解LangGraph与LangChain
- 理解Agent智能体
- 理解智能体编排
- 如何部署LangChain应用

一、LangGraph到底是干什么的?

LangChain ，官网地址：https://python.langchain.com/docs/introduction/ 是一个用于开发由大语言模型LLM提供支持的应用程序的框架。简单来说，就是一个用LLM快速构建本地应用的框架。当前最新的版本是V0.3。后续内容也以这个版本为准。

在LangChain中，除了LangChain外，还有LangGraph、LangSmith等一系列的生态组件。



其中，LangChain是整个生态的基础。LangSmith主要是针对LangChain的应用进行测试、监控和分析的平台。LangGraph则是基于LangChain的应用程序开发框架，它可以帮助开发者更方便地构建和管理复杂的应用程序。

LangGraph的官网地址：https://langchain-ai.github.io/langgraph/ 。 从页面左侧的菜单可以看到，使用LangGraph构建应用的标准流程是这样：

- Prebuilt agents：第一步，构建Agent智能体。这是LangGraph应用的基础
- LangGraph framework：第二步，构建LangGraph应用。主要是以Graph图的方式将多个Agent智能体整合成一个整体。这也是LangGraph最核心的部分
- LangGraph Platform：第三步，通过LangGraph Platform平台部署应用。这是一个商业化的平台，可以以标准化的形式部署LangGraph应用，并提供测试、监控、分析等功能。

其实，对于LangGraph框架，如果你把这几个部分都搞清楚了，那么，整个LangGraph框架，你也就通了一大半了。

对于构建应用来说，前两步必不可少。而第三步则通常不是一个必须选项。所以接下来我们重点介绍前两个部分。

这一章先来介绍第一步，构建智能体。这对于LangGraph来说，是一个非常重要的部分。因为LangGraph是基于Agent的，所以构建Agent是LangGraph的基础。

这里的Agent智能体，其实本质上就是将大语言模型的各种功能，封装成独立的整体。Agent构建完成后，未来我们有什么问题，直接交给Agent处理就行了。不用过多关注Agent的细节。而与大模型交互这个事情，LangChain框架已经实现了非常多的核心功能，所以，这一部分也是和langChain联系非常紧密的一个部分。

二、快速体验LangChain与LangGraph

LangGraph和LangChain都是用于构建和管理大型语言模型应用的工具，它们都提供了一种简单易用的方式来构建和管理复杂的应用程序。只不过，LangChain更关注于应用程序的整体流程，而LangGraph更关注于如何处理特定的任务。

关于LangGraph的细节，在后溪章节中会详细介绍。 这里，我们先用最简单直白的方式来对比一下LangChain和LangGraph在和大语言模型做交互时的基础思想有什么区别。

要使用LangGraph，首先需要安装LangGraph的依赖库

```
! pip install langgraph
## 后面的案例中会用到langchain,所以同时也安装下langchain的依赖
! pip install langchain
! pip install langchain_community
```

实际上，从依赖库的安装过程就能看到，LangGraph是依赖于LangChain库的。

然后，先从基础的访问大模型的API开始。比较一下LangChain和LangGraph的访问大模型的API的区别。

使用LangChain访问大模型最基础的方式是使用init_chat_model创建一个ChatModel，大模型对象。通过这个大模型对象完成与大模型的交互

```
import os
from config.load_key import load_key
# 制定OpenAI的API_KEY。
if not os.environ.get("OPENAI_API_KEY"):
    os.environ["OPENAI_API_KEY"] = load_key("OPENAI_API_KEY")
from langchain.chat_models import init_chat_model
# 创建访问OpenAI的Model。
# model = init_chat_model("gpt-4o-mini",model_provider="openai")
# openai在国内是无法直接访问的，需要科学上网。这里指定base_url是因为使用的是openai的国内代理，
2233.ai。
model = init_chat_model("gpt-4o-
mini",model_provider="openai",base_url="https://api.gptsapi.net/v1")

model.invoke("你是谁？能帮我解决什么问题？")
```

然后，使用LangGraph访问基础大模型的方式是这样的：

```
from langgraph.prebuilt import create_react_agent

agent = create_react_agent(
    model=model,
    tools=[],
    prompt="You are a helpful assistant",
)

agent.invoke({"messages":[{"role":"user","content":"你是谁？能帮我解决什么问题？"}]})
```

而如果要个大模型提供一些自定义的辅助工具，使用LangChain方式是这样的：

```
import datetime
from langchain.tools import tool

from config.load_key import load_key
from langchain_community.chat_models import ChatTongyi
# 构建阿里云百炼大模型客户端
llm = ChatTongyi(
    model="qwen-plus",
    api_key=load_key("BAILIAN_API_KEY"),
)

# 定义工具 注意要添加注释
@tool
def get_current_date():
    """获取今天日期"""
    return datetime.datetime.today().strftime("%Y-%m-%d")
# 大模型绑定工具
llm_with_tools = llm.bind_tools([get_current_date])
# 工具容器
all_tools = {"get_current_date": get_current_date}
# 把所有消息存到一起
```

```
query = "今天是几月几号"
messages = [query]
# 询问大模型。大模型会判断需要调用工具，并返回一个工具调用请求
ai_msg = llm_with_tools.invoke(messages)
print(ai_msg)
messages.append(ai_msg)
# 打印需要调用的工具
print(ai_msg.tool_calls)
if ai_msg.tool_calls:
    for tool_call in ai_msg.tool_calls:
        selected_tool = all_tools[tool_call["name"].lower()]
        tool_msg = selected_tool.invoke(tool_call)
        messages.append(tool_msg)
llm_with_tools.invoke(messages).content
```

使用LangGraph后，使用的方式是这样的：

```
from langgraph.prebuilt import create_react_agent

agent = create_react_agent(
    model=llm,
    tools=[get_current_date],
    prompt="You are a helpful assistant",
)

agent.invoke({"messages":[{"role":"user","content":"今天是几月几号"}]})
```

三、章节小结

从这个案例大概可以简单的感受到，LangGraph的一部分核心功能就是要在LangChain的基础上，以Agent智能体的方式，提供更简单实用的功能封装，从而让我们可以更方便的使用LangChain的功能。
当然，Agent的功能封装远不止这个案例中这么简单。通过LangGraph的Agent功能，可以将与大大模型交互的各种基础功能统一封装成独立的Agent，而不用过多关注Agent内部的实现细节。
接下来，有了Agent之后，LangGraph还通过Graph图的方式，可以将多个Agent智能体串联起来，实现更加复杂的应用。
例如，我们之前实现了一个查询今天日期的Agent，接下来可以再实现航班信息的Agent，将这两个Agent串联起来，就可以完成一个出行规划的复合任务。
这些在后面章节会逐步介绍。