Python 环境搭建与项目配置 - 章节介绍

虚拟环境与编辑器配置

- Python 稳定版与发行版的差异，如何选择正确的版本，虚拟环境的优缺点与安装，全局配置国内镜像加速包安装。
- PyCharm 编辑器代码格式化与导入优化，常见快捷键的介绍与使用，插件配置安装，提升开发效率。

项目架构与框架依赖选择

- 介绍 LLMOps 项目的架构与框架依赖选择，配置 pipreqs 确保项目依赖稳定不胡乱升级导致无法运行。

- 整个项目的目录文件约定、命名文件、类文件约定、依赖注入介绍与使用。

数据库与 Postman 接口调试工具

- 介绍项目使用的数据库 Postgres ，数据库选择的差异对比，在不同的系统下如何安装配置。

- 项目 API 工具 Postman 的安装、配置环境变量、基础 URL 、路径参数、授权认证等内容。

Python 环境搭建与配置镜像加速

Python 不同版本之间的差异

Python 版本	维护状态	首次发布	终止支持
3.13	prerelease	2024-10-01	2029-10
3.12	bugfix	2023-10-02	2028-10
3.11	bugfix	2022-10-04	2027-10
3.10	security	2021-10-04	2026-10
3.9	security	2020-10-05	2025-10
3.8	security	2019-10-14	2024-10

Python 虚拟环境的创建与优缺点

创建虚拟环境命令：python –m venv 环境名，该命令在 Python 2.7+ 或 Python 3.3+ 以上可以使用。

Python 虚拟环境与系统环境之间是隔离的，一个项目一个虚拟环境，不同的虚拟环境不会相互影响，最大程度避免包冲突。

占用更大的空间，进入虚拟环境都需要使用命令 activate 激活才能使用，使用完，要使用 deactivate 命令退出。

腾讯云 pip 镜像加速配置

镜像加速原理，国内网站拷贝了整个 pip 的所有包，让 pip 直接连接到国内网站下载，下载会更快。

临时使用：pip install -i 镜像地址 <some-package>
全局使用：pip config set global.index-url 镜像地址

镜像： https://mirrors.cloud.tencent.com/pypi/simple

课程总结

介绍了不同 Python 版本之间的差异，个人与公司该如何选择合适的版本，并演示了 Python 的安装配置。

学习了 Python 创建虚拟环境的命令与演示，探讨了虚拟环境的优缺点，全局配置腾讯云镜像加速源，提升第三方包的安装速度。

PyCharm 编辑器配置与插件安装

项目架构与基础框架选择

LLMOps 项目 7 层架构

语言与项目基础框架选择

使用 Python 作为后端编程语言虽然性能比不上 Java 等编译型等语言，但是 Python 丰富的生态系统，庞大的 AI 开源模型、机器学习基础库等，是其他语言无法媲美的。

Flask 是一个轻量级的框架，核心非常小，功能丰富但不臃肿，性能对比 Django 强了不少，零基础也可以快速入门，大大降低了学习难度。

对于 LLM 应用开发，可以使用 LangChain 极大简化开发流程，提升开发效率，其他语言需要自己封装轮子。

课程总结

分享了 LLMOps 项目的 7 层架构图，分层架构其实就是把系统自上而下分为多个不同的层，每一层只和自己的直接上层和下层“打交道”，不会跨层调用。

了解使用 Python+Flask 技术栈进行开发的优缺点。

项目目录结构约定、规范与依赖注入

依赖注入简介与使用教程

依赖注入是软件工程中的一种设计模式，它允许在创建对象时由外部提供依赖关系，而不是自己创建这些依赖关系。

简单来说，即我需要什么，传递什么，而不是自己内部构建， controller 需要使用 service 的内容，将 service 作为参数传递给 controller 即可。

依赖注入简介与使用教程

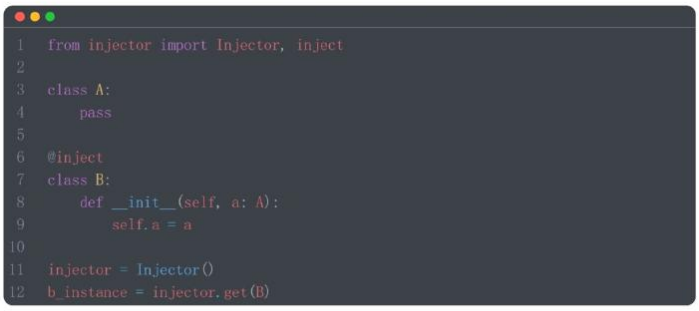
依赖注入是软件工程中的一种设计模式，它允许在创建对象时由外部提供依赖关系，而不是自己创建这些依赖关系。

简单来说， controller 需要使用 service 的内容，将 service 作为参数传递给 controller 即可，我需要什么，传递什么。

每页最多三行正文，正文多于三行要分页

Python 中的依赖注入框架 injector

Injector 是一个 Python 的轻量级依赖注入框架，通过装饰器 @inject 和 Injector 实例轻松实现依赖注入功能，不需要显式创建依赖对象。



```python
from injector import Injector, inject

class A:
    pass

@inject
class B:
    def __init__(self, a: A):
        self.a = a

injector = Injector()
b_instance = injector.get(B)
```

课程总结

约定了 LLMOps 项目的目录 / 文件规范，了解了各个目录的作用，对于很多人来说，适应规范会觉得很不舒服，但在企业团队开发中，这是一件必须的事，越规矩，越自由。

分享了依赖注入的介绍，在 Python 中如何 injector 框架快速实现依赖注入的技巧。

项目目录结构约定、规范与依赖注入

Flask 是一个优美的微框架，这即是一件好事——你可以按照自己的习惯和想法来组织你的项目，不过对于团队来说这可能是一件坏事——团队每个人都有自己的喜好，这会使整体项目的结构很混乱。因此我们为 LLMOps 项目设计提炼了一种规范，它不仅仅是结构，风格还有诸多细节，你会在后续逐渐了解到。

01. 项目结构

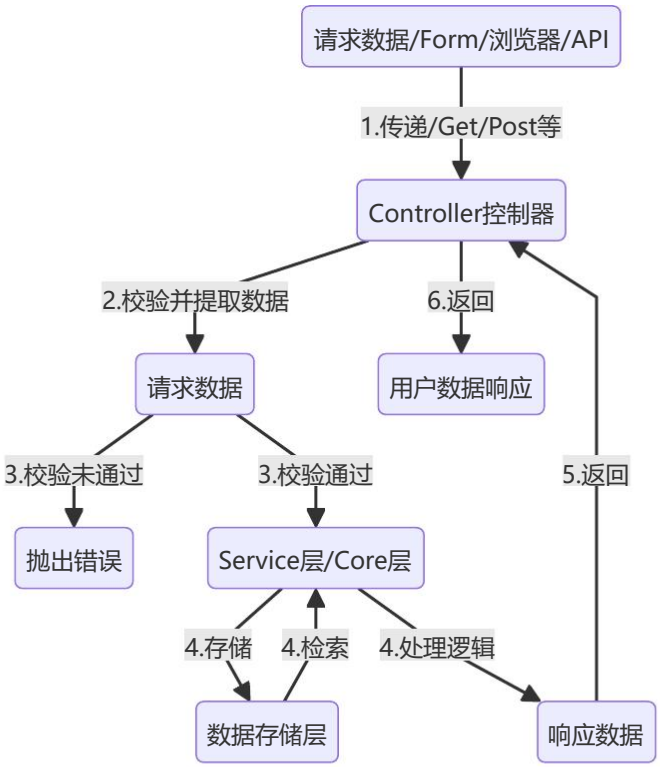
对于很多人来说，适应规范会觉得很不舒服，但在开发中，这是一件必须的事，越规矩，越自由。

```
|---app   // 应用入口集合
|   ├---__init__.py
|   └---http
|---config   // 应用配置文件
|   ├---__init__.py
|   ├---config.py
|   └---default_config.py
|---internal   // 应用所有内部文件夹
|   ├---core   // LLM核心文件，集成LangChain、LLM、Embedding等非逻辑的代码
|   |   |---agent
|   |   |---chain
|   |   |---prompt
|   |   |---model_runtime
|   |   |---moderation
|   |   |---tool
|   |   |---vector_store
|   |   └---...
|   ├---exception   // 通用公共异常目录
|   |   ├---__init__.py
|   |   ├---exception.py
|   |   └---...
|   ├---extension   // Flask扩展文件目录
|   |   ├---__init__.py
|   |   ├---database_extension.py
|   |   └---...
|   ├---handler   // 路由处理器、控制器目录
|   |   ├---__init__.py
|   |   ├---account_handler.py
|   |   └---...
|   ├---middleware   // 应用中间件目录，包含校验是否登录
|   |   ├---__init__.py
|   |   └---middleware.py
|   |   └---...
|   ├---migration   // 数据库迁移文件目录，自动生成
|   |   ├---versions
|   |   └---...
|   ├---model   // 数据库模型文件目录
|   |   ├---__init__.py
|   |   ├---account.py
|   |   └---...
|   ├---router   // 应用路由文件夹
|   |   ├---__init__.py
```

```
|  |  ├---router.py
|  |  └---...
|  ├---schedule  // 调度任务、定时任务文件夹
|  |  ├---__init__.py
|  |  └---...
|  ├---schema  // 请求和响应的结构体
|  |  ├---__init__.py
|  |  └---...
|  ├---server  // 构建的应用，与app文件夹对应
|  |  ├---__init__.py
|  |  └---...
|  ├---service // 服务层文件夹
|  |  ├---__init__.py
|  |  ├---oauth_service.py
|  |  └---...
|  ├---task  // 任务文件夹，支持即时任务+延迟任务
|  |  ├---__init__.py
|  |  └---...
|---pkg  // 扩展包文件夹
|  ├---__init__.py
|  |---oauth
|  |  ├---__init__.py
|  |  ├---github_oauth.py
|  |  └---...
|  └---...
├---storage  // 本地存储文件夹
├---test  // 测试目录
├---venv  // 虚拟环境
├---.env  // 应用配置文件
├---.gitignore  // 配置git忽略文件
├---requirements.txt  // 第三方包依赖管理
└---README.md  // 项目说明文件
```

02. 代码运行流程图

项目通过路由接收用户的发起的请求，并调用控制器特定的方法来处理，控制器接收到数据后，对数据进行校验，校验未通过则抛出错误；校验通过后将数据传递给Service/Core层进行相应的逻辑处理、数据存储和检索等操作，完成逻辑计算后得到响应数据，返回给控制器，控制器在将数据响应给用户，至此，一个最简单的流程结束。



03. 文件与Python类函数命名规范

3.1 文件名

- 使用全部小写字母。
- 使用下划线分隔单词，例如：`app_service.py`。
- 尽可能保证文件作用的单一，不要把所有代码一次性写在同一个文件中。

3.2 类名

- 使用驼峰命名法，例如：class AppService。
- 类名应该以大写字母开头，每个单词的首字母都大写。
- 如果类名由多个单词组成，单词之间不使用下划线分割。

3.3 函数名和方法

- 使用小写字母。
- 使用下划线分隔单词。
- 例如：`get_account`，`generate_token`等。

3.4 变量名

- 使用小写字母。
- 使用下划线分隔单词。
- 例如：`my_variable`，`token_count`。

3.5 常量

- 使用全大写字母。
- 使用下划线分隔单词。
- 例如：`MAX_SIZE`，`PI`等。

3.6 私有变量与方法

- 以一个下划线开头表示私有，例如：`_my_private_variable`，`_my_private_method()`等。
- 在 Python 中并没有严格的私有变量/方法，这种命名约定只是一种约定，而不是强制规则，实际上这些变量/方法仍然可以被使用，但是作为一种约定，在外部调用时，不应该调用私有的变量与方法。

3.7 模块名

- 与文件名类似，使用全部小写字母，使用下划线分隔单词。
- 例如：`my_module.py`对应的模块名应该是`my_module`。
- 模块下创建`__init__.py`文件代表当前目录为一个模块，并尽可能在`__init__.py`中使用`__all__`简化导出。

04. Python中的依赖注入

依赖注入是软件工程中的一种设计模式，它允许在创建对象时由外部提供依赖关系，而不是自己创建这些依赖关系。

简单来说，即我需要什么，传递什么，而不是自己内部构建，`Controller`需要使用`Service`的内容，将`Service`作为参数传递给`Controller`即可，在 Python 中可以使用`injector`包来快速实现依赖注入。

安装：

```
pip install injector
```

使用示例：

```python
from injector import Injector, inject

class A:
    pass

@inject
class B:
    def __init__(self, a: A):
        self.a = a

injector = Injector()
b_instance = injector.get(B)
```

其中`@inject`装饰器用于注入，使用@inject装饰的函数/类，不需要显式创建依赖对象就可以直接使用。要获取被注入的对象，可以使用`Injector`实例的`get()`方法传入特定的类型，即可获取。

依赖库介绍、安装与测试

Postman 基础配置与使用介绍

几种环境下的 Postgres 数据库安装

几种环境下的Postgres数据库安装

01. Postgres 数据库介绍

PostgreSQL（又称 Postgres）是一种强大、开源的关系型数据库管理系统（RDBMS），它具备高度的可靠性、稳定性和可扩展性，主要特点如下：

1. 开源：PostgreSQL 是基于开源许可证发布的，任何人都可以免费使用、修改和分发它。
2. 关系型数据库：PostgreSQL 是一种关系型数据库，支持 SQL 查询语句，具有强大的数据处理能力。
3. 可扩展性：PostgreSQL 支持多种插件扩展，可以满足各种不同规模和需求的应用场景。
4. 支持复杂数据类型：除了传统的数据类型外，PostgreSQL 还支持数组、JSON、XML 等复杂数据类型。
5. 事务支持：PostgreSQL 提供完整的 ACID 事务支持，确保数据的一致性和可靠性。
6. 触发器和存储过程：支持触发器和存储过程，可以在数据库层面实现业务逻辑。
7. 并发控制：具备强大的并发控制能力，能够处理大量并发访问请求。
8. 安全性：提供强大的用户认证和权限管理能力，确保数据安全。
9. 扩展性：可以轻松地通过扩展模块来增加额外的功能，如全文搜索、地理信息系统等。

PostgreSQL 官网：https://www.postgresql.org/

02. 不同环境下的 Postgres 数据库安装

2.1 Windows 下 Postgres 安装

Windows/Mac 下支持使用可执行文件快速安装，安装后像普通软件一样启动 Postgres 即可。

下载地址：https://www.enterprisedb.com/downloads/postgres-postgresql-downloads。

安装后可通过 pgAdmin 工具来连接 PostgreSQL 数据库。

默认安装的 PostgreSQL 会开机自启，可以通过以下步骤关闭开机自启：

1. 按下 `win+r` 打开运行对话框，输入 `services.msc` 并回车。
2. 找到 `postgres-x64-16`，右击选择 `属性`，将启动方式修改为 `手动`。
3. 可以右击选择 `停止`，关闭 `postgres` 服务。

Windows 下的启动与停止命令：

```
pg_ctl start -D "D:\Software\PostgreSQL\16\data"
pg_ctl stop -D "D:\Software\PostgreSQL\16\data"
```

`-D` 参数告诉 `pg_ctl` 命令应该使用哪个目录中的数据文件和配置文件。

也可以写成 `bat` 批处理，快速实现启动+关闭。

启动 bat：

```
@echo off
set PG_PATH="D:\Software\PostgreSQL\16\bin"
set PG_DATA="D:\Software\PostgreSQL\16\data"

echo Starting PostgreSQL...
%PG_PATH%\pg_ctl.exe start -D %PG_DATA%
echo PostgreSQL started.
```

停止 bat：

```
@echo off
set PG_PATH="D:\Software\PostgreSQL\16\bin"
set PG_DATA="D:\Software\PostgreSQL\16\data"

echo Stoping PostgreSQL...
%PG_PATH%\pg_ctl.exe stop -D %PG_DATA%
echo PostgreSQL stoped.
```

2.2 Ubuntu 下 Postgres 安装

对于 Debian 的系统（如 Ubuntu），可以使用如下命令：

```
sudo apt update
sudo apt install postgresql postgresql-contrib
```

检测 PostgreSQL 是否启动：

```
sudo systemctl status postgresql
```

通过以下两个命令启动或者停止 PostgreSQL 服务：

```
sudo systemctl start postgresql
sudo systemctl stop postgresql
```

安装完成后，可以通过 postgresql 提供命令工具 `psql` 连接到 PostgreSQL 数据库，亦或者使用 pgAdmin 可视化界面进行连接：

```
psql -U postgres -h localhost -W
```

也可以通过切换到 postgres 用户直接运行 `psql` 命令：

```
sudo -i -u postgres
psql
```

在 `psql` 中修改 `postgres` 密码：

```
\password postgres
```

如果无法通过 `psql -U postgres` 进行登录，则大概率是 `postgresql` 仅开启了本地登录，可以通过编辑 `pg_hba.conf` 修改配置：

```
sudo vim /etc/postgresql/<版本号>/main/pg_hba.conf
```

```
# "local" is for Unix domain socket connections only
local   all             all                                     md5
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

PostgreSQL 中常见的身份验证方法如下：

1. peer：仅适用于本地连接。客户端必须作为相同的操作系统用户连接。例如，如果你以 `postgres` 用户登录操作系统，那么连接到数据库时也必须以 `postgres` 用户身份。
2. md5：使用MD5哈希进行密码验证。客户端必须提供正确的密码，密码在传输过程中会被加密。
3. password：以明文方式传输密码进行验证。不推荐使用，因为密码在网络上以明文形式传输，安全性较低。
4. trust：不需要密码，直接允许连接。不推荐在生产环境中使用，因为安全性较低。

2.3 Docker 快捷安装 PostgreSQL

Docker 中安装 PostgreSQL 非常简单，官方配置了镜像支持一件安装。

首先，从 Docker Hub 上拉去 PostgreSQL 的官方镜像：

```
docker pull postgres
```

然后运行 PostgreSQL 容器：

```
docker run --name postgres-dev -e POSTGRES_USER=postgres -e
POSTGRES_PASSWORD=postgres -p 5432:5432 -d postgres
```

停止与开启 PostgreSQL 容器：

```
docker start postgres-dev
docker stop postgres-dev
```

Docker 删除镜像与删除容器命令：

```
docker rmi <镜像id或名称>
docker rm <容器id或名称>
```