

RAG 技术详解与实战应用

第4讲： RAG项目工程化入门：
从脚本走向模块化与可维护性



目录

-  1. 上节回顾
-  2. Why
-  3. Git管理
-  4. 代码质量保证
-  5. 工程发布



LazyLLM介绍

- LazyLLM开发初衷
- 大模型应用开发思路
- 亮点：
 - 以数据流为核心的应用开发范式
 - 复杂应用一键部署
 - 跨平台
 - 不同技术选型提供统一体验
 - 代码直观简洁

如何使用大模型

- 本地模型
- 在线模型
- 流式输出
- Prompt格式
- 在模型中使用Prompt

数据流

- Pipeline: 顺序执行
- Parallel: 并发执行
- Diverter: 分流并发执行
- Warp: 数据并行/批处理
- IFS: 条件分支
- Switch: 多路选择
- Loop: 循环

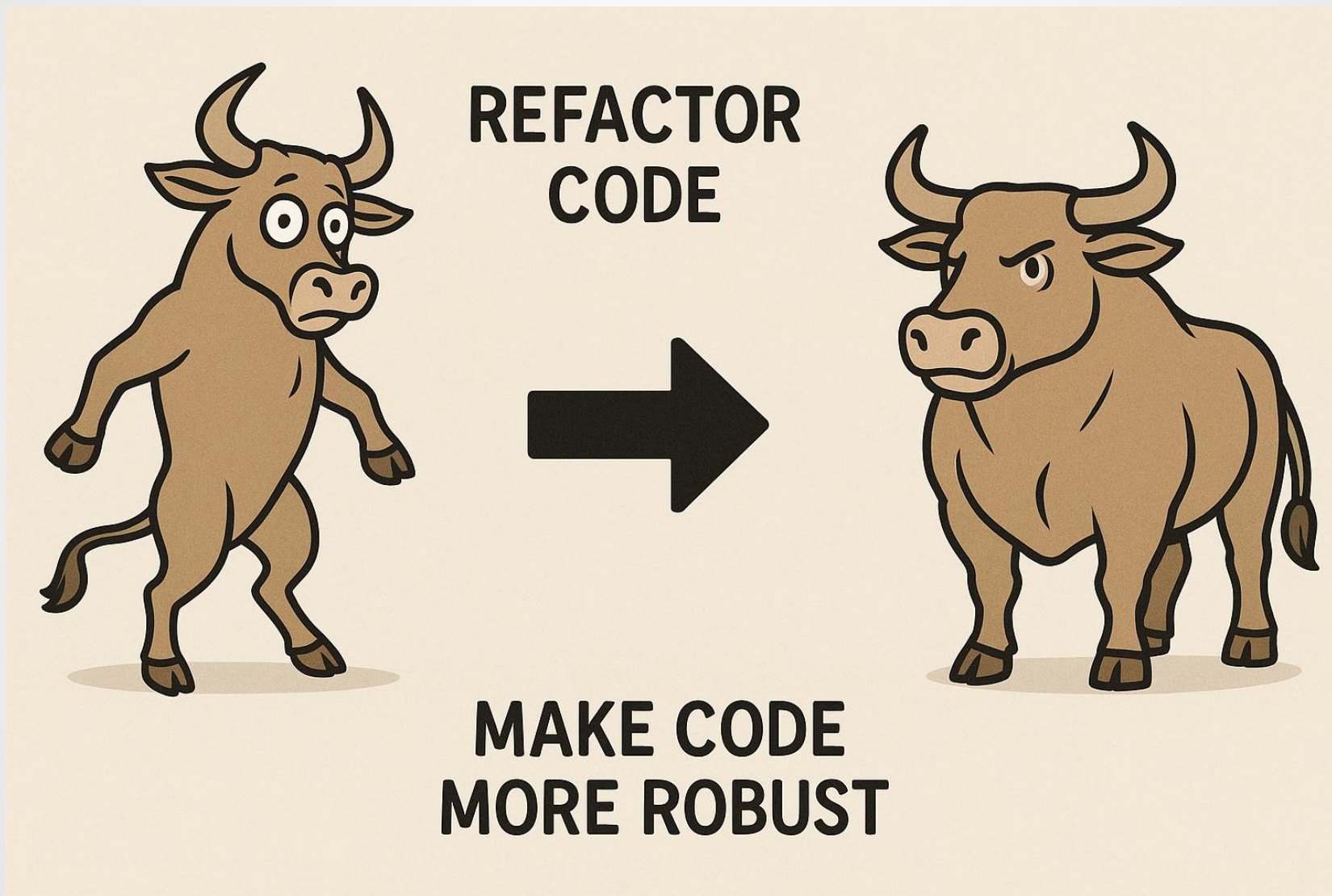


目录

-  1. 上节回顾
-  2. Why
-  3. Git管理
-  4. 代码质量保证
-  5. 工程发布



为什么要工程化



目录

-  1. 上节回顾
-  2. Why
-  3. Git管理
-  4. 代码质量保证
-  5. 工程发布



Git, What and Why?

什么是Git?

Git 是一个**分布式版本控制系统**，用于跟踪文件的更改，特别适用于软件开发。
它允许多个开发者协作开发代码，同时保留所有更改的历史记录。

为什么要使用 Git?

- 代码历史可**追溯**，支持版本回滚
- 支持多人**协作**，提高团队效率
- 便于分支管理，降低开发冲突
- 可**托管**到 github gitlab等，方便备份与共享



创建git仓库 – 以github为例



- 首先你要有一个 GitHub 账号，如果没有的话请先 [注册](#)
- 登陆账号，[创建一个项目](#) (New repository) 例如: **my-project**
- 只需要输入项目名称 (Repository name) 即可，项目描述 (Description) 选填。Public 是公开，可以在 GitHub 搜到，Private 是私密项目，只有自己和项目成员能看到。点击 Create repository

创建新仓库

仓库包含项目中的所有文件，包括修订历史记录。在其他地方已有仓库？[导入仓库](#)

带星号(*) 的为必填项。

仓库模板

不设置模板

使用模板仓库的内容新建仓库。

所有者 *

hwj-st

仓库名称 *

my-project

my-project 名称可用。

好的仓库名称应该简单且容易记忆。需要灵感吗？这个怎么样：miniature-tribble？

描述 (可选)

☒ 公共

任何人都可以看到这个仓库。您可以选择谁可以提交。

☐ 私有

您可以选择谁可以看到和提交到这个仓库。

使用以下方式初始化此仓库：

☐ 添加 README 文件

您可以在此处为您的项目撰写详细描述。[了解更多关于 README 的信息。](#)

添加 .gitignore 文件

.gitignore 模板：无

从模板列表中選擇哪些文件不需要跟踪。[了解更多关于忽略文件的信息。](#)

选择许可证

许可证：无

许可证告诉其他人，他们可以使用您的代码做什么和不能做什么。[了解更多关于许可证的信息。](#)

① 您正在个人账户中创建公共仓库



初始化 Git 仓库并推送到 GitHub



在创建好项目之后，可以在本地初始化 Git 仓库并推送代码；如果是通过http，则按需输入账号密码（或者token，见[创建token教程](#)）

```
>>> mkdir my-project && cd my-project
>>> echo "# test" >> README.md
>>> git init
```

首次使用需要设置用户名和邮箱

```
>>> git config --global user.name "你的用户名"
>>> git config --global user.email "你的邮箱"
```

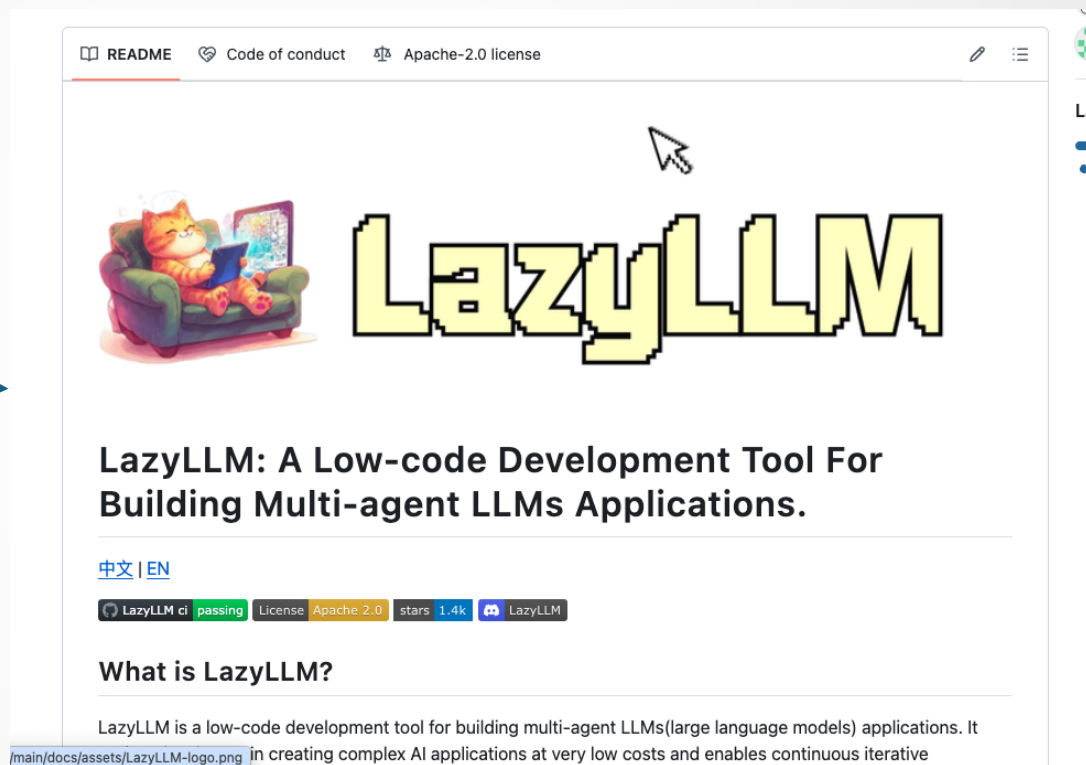
```
>>> git add README.md
>>> git commit -m "test"
>>> git branch -M main
>>> git remote add origin <your-repo-http-url>
>>> git push -u origin main
```

```
>>> git push -u origin main
Username for 'https://github.com': lwj-st
Password for 'https://lwj-st@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 223 bytes | 223.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/lwj-st/my-project.git
* [new branch] main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```



Git工程中重要的几个文件


- README.md
- .gitignore
- requirements.txt
- Dockerfile
- setup.py
- ...



Git工程中重要的几个文件

- README.md
- .gitignore
- requirements.txt
- Dockerfile
- setup.py
- ...

Gitignore记录了**不需要提交**的文件，如编译生成的二进制文件、日志文件等



__pycache__
*.pyc
test/
dist/
tmp/
.vscode
build
*.lock
*.db



Git工程中重要的几个文件

- README.md
- .gitignore
- requirements.txt
- Dockerfile
- setup.py
- ...

requirements.txt记录了**项目依赖**

lazyllm
fastapi==0.111.0
loguru>=0.7.2
pydantic>=2.5.0
requests>=2.32.2
pymilvus>=2.4.11, <2.5.0

Dockerfile文件用于**镜像构建**

setup.py文件则用于**打包与发布**



Git能力展示



Your Repositories

github.com/lwj-st?tab=repositories

lwj-st

Overview Repositories 14 Projects Packages Stars 14

lwj-st

Edit profile

3 followers • 1 following

Achievements

Organizations

Find a repository...

Type

Language

Sort

New

lazyplatform-docker-compose

Private

Updated 3 days ago

Star

test

Public

Updated 4 days ago

Star

scripts

Public

Shell Updated last week

Star

Authorized

Private

Shell Updated last week

Star

kube-prometheus

Public

Forked from prometheus-operator/kube-prometheus

Use Prometheus to monitor Kubernetes and applications running on Kubernetes

Jsonnet Apache License 2.0 Updated 2 weeks ago

Star

LazyLLM

Public

Forked from lazyagi/LazyLLM

Easiest and laziest way for building multi-agent LLMs applications.

Python Apache License 2.0 Updated 2 weeks ago

Star

LazyLLM_image

Public

Dockerfile 1 Updated 2 weeks ago

Star

liwenjian.test

Private

C++ Updated 3 weeks ago

Star



我们以前面讲过的召回器为例，添加一段代码作为项目的主文件，命名为my_project/retriever.py。

```
1.  from lazyllm import Retriever, Document
2.  def create_retriever(path: str, query: str):
3.      """
4.      创建并执行检索
5.      Args:
6.          path (str): 文档的绝对路径
7.          query (str): 查询语句
8.      Returns:
9.          list: 检索结果
10.     """
11.     doc = Document(path)
12.     retriever = Retriever(doc, group_name="CoarseChunk",
13.                           similarity="bm25_chinese", topk=3)
14.     return retriever(query)
```

Google风格的文档

- 简短描述
- Args（参数列表）
- Returns（返回值）
- Raises（异常）
- Examples（示例代码）



规范的python代码都会在包文件夹添加__init__.py，我们需要添加名为my_project/__init__.py的文件

1. `from .retriever import create_retriever`
2. `__version__ = '0.1.0'`
3. `__all__ = ['create_retriever']`

__init__.py是 Python 包管理的核心，它的作用主要是 **标识包并控制导入的行为。**

1. `# example_pkg/__init__.py`
2. `from .module_a import func_a`
3. `from .module_b import func_b`
4. `__all__ = ["func_a", "func_b"]`



分支管理和标签管理



Git 分支是代码版本控制中的**指针**，指向某一次提交（Commit），可以让你并行地开发、测试和修复 Bug。

Git 标签是某个特定提交的**快照**，是一种用于版本发布的标记，不会随开发进度变化

常见分支策略

- 主分支（main）：始终保持稳定可发布状态
- 开发分支（dev）：用于日常开发
- 版本开发分支（dev/*）：用于维护已发布版本的开发工作
- 个人分支（name/*）：用于开发新功能，完成后合并回 dev
- 功能分支（feature/*）：用于开发新功能，完成后合并回 dev
- 修复分支（hotfix/*）：用于紧急修复生产环境 bug

常见标签策略

- 语义化版本(v0.0.0)： <主版本>.<次版本>.<修订版本>
- 里程碑标签(v0.0.0-alpha)：适合项目阶段性发布
- 时间戳标签(2025-04-03-hotfix)：追踪时间版本
- 环境标签(v1.0.0-prod)：指明版本的发布环境



Git 分支管理操作示例

创建并切换到开发分支

```
>>> git checkout -b dev
```

在dev分支上新创建功能分支

```
>>> git checkout -b feature/new-feature
```

...修改代码...

```
>>> git add .
```

```
>>> git commit -m "Add new feature"
```

```
>>> git push origin feature/new-feature
```

开发完成后合并回 dev

```
>>> git checkout dev
```

```
>>> git merge feature/new-feature
```

```
>>> git push origin dev
```

通常情况下，此步骤会在网页上通过pull request完成



在github上提交pull request



- 当改好的代码提到远程仓库后就可以提交pull request了
- 进入 GitHub 仓库主页，点击 "Pull Requests" -> "New Pull Request"
- 如果是同一仓库的Pull Requests 选择对应的 **base (目标分支)** 和 **compare (比对分支)** 即可
- 如果是fork仓库的Pull Requests 则有 **base repository (目标仓库)** 和 **base (目标分支)** 默认是fork的源仓库信息，**head repository (源仓库)** 和 **compare (比对分支)** 默认是自己仓库的信息
- 添加 PR 说明，点击 "Create Pull Request"



冲突及其解决方法

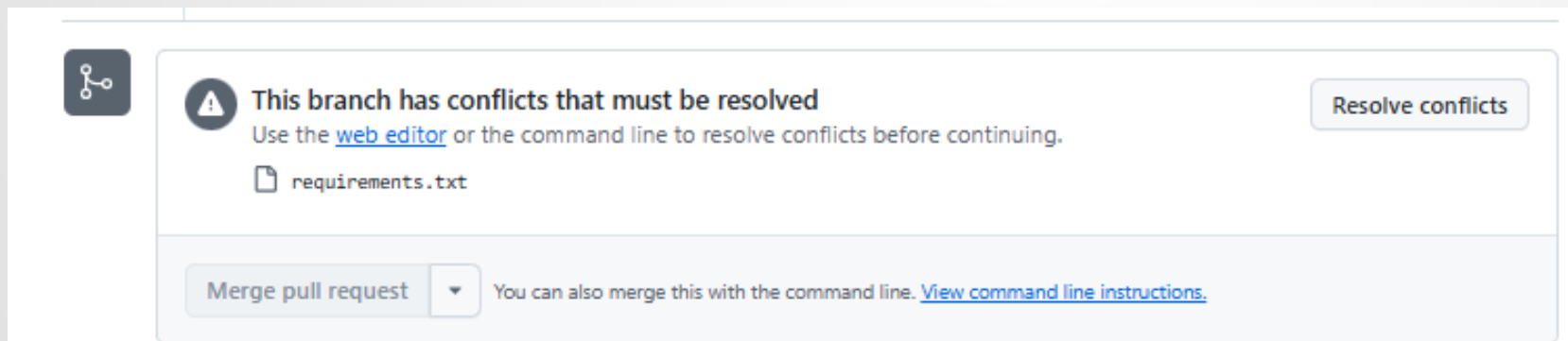
当多个开发者同时修改同一部分代码并尝试合并时，Git 可能会提示 **conflict**。目前主要有**两种可能导致冲突的情况**：

1. 多名开发者**协作开发同一个分支**，会出现同分支冲突

```
To https://github.com/lwj-st/my-project.git
! [rejected] dev -> dev (fetch first)
error: failed to push some refs to 'https://github.com/lwj-st/my-project.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

尽可能避免多人
修改同一分支

2. 在网页上**提交Pull Request**时，和目标分支产生了冲突。



同分支冲突解决策略



- 执行git pull merge远程分支

```
$ git pull
```

CONFLICT (content): Merge conflict in requirements.txt

Automatic merge failed; fix conflicts and then commit the result.

- 此时，你可以运行git status查看哪些文件存在冲突

```
$ git status
```

On branch dev

Your branch and 'origin/dev' have diverged, and have 1 and 1 different commits each, respectively.

(use "git pull" to merge the remote branch into yours)

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: requirements.txt

- 直接修改文件，保留正确的代码然后添加并提交：

```
$ git add requirements.txt
```

```
$ git commit -m "解决冲突"
```

```
$ git push
```



- 页面修改，直接点击 **Resolve conflicts** (解决冲突)
- 编辑冲突文件保留需要的内容后点击 标记为 **Mark as resolved** (已解决) 即可
- Git 会在有冲突的文件中标记冲突部分，格式如下：

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< dev (Current Change)
XXXX
=====
XXXX
>>>>>>> main (Incoming Change)
```

- <<<<<<< dev 表示 **提pr的分支dev** 的代码
- ===== 是分隔符
- >>>>>>> main 表示 **主分支 (main)** 的代码



lwj-st/my-project at dev

GitHub

github.com/lwj-st/my-project/tree/dev

lwj-st / my-project

Type to search

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

my-project

Public

Pin

Unwatch 1

Fork 0

Star 0

feature/new-feature had recent pushes 33 minutes ago

Compare & pull request

dev had recent pushes 7 minutes ago

Compare & pull request

dev

3 Branches

0 Tags

Go to file

Add file

<> Code

This branch is 12 commits ahead of main.

Contribute

lwj-st 解决冲突

a0f3439 · 8 minutes ago

14 Commits

my_project

test

36 minutes ago

.gitignore

Add new feature

34 minutes ago

README.md

test

36 minutes ago

requirements.txt

Update requirements.txt

9 minutes ago

README

my-project

介绍

这是一个示例 Python 项目，支持自动化测试、Docker 部署，并符合 PEP 8 代码规范。

安装

pip install my-project

About

No description, website, or topics provided.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

Python 100.0%

Suggested workflows

Based on your tech stack:

Django

Configure

Build and Test a Django Project

Python application

Configure

Create and test a Python application.

SLSA Generic generator

Configure

Generate SLSA3 provenance for your existing release workflows.

More workflows

Dismiss suggestions

目录

-  1. 上节回顾
-  2. Why
-  3. Git管理
-  4. 代码质量保证
-  5. 工程发布



- 安装 flake8
\$ pip install flake8
- 检查整个项目，有问题的话会有日志提示，没日志信息就是最好的信息
\$ flake8 .
- 特点
 - **代码风格检查**：基于 PEP 8 规范，检测缩进、命名、行长度等问题。
 - **语法错误检测**：发现未定义变量、语法错误等潜在 Bug。
 - **复杂性分析**：通过 McCabe 复杂度检查，提示代码是否过于复杂。
 - **插件扩展性**：支持第三方插件，可扩展检查规则（如类型检查）。



代码格式化工具

- 安装 black

```
$ pip install black
```

- 格式化整个项目

```
$ black .
```

```
reformatted /root/my_project/__init__.py
```

```
reformatted /root/my_project/retriever.py
```

All done! 🌟 🍰 🌟

2 files reformatted.



- 特点

- **强制代码格式化** (不能自定义样式, 减少团队争议)
- **默认 88 字符换行** (可调整)
- **自动调整引号** (优先使用双引号)
- **对 if-else、列表等结构进行优化**



在代码提交前进行风格检查

- 安装pre-commit
\$ pip install pre-commit
- 初始化pre-commit
\$ pre-commit install
- 在 .pre-commit-config.yaml 中添加规则:
repos:
 - repo: <https://github.com/psf/black>
rev: 23.1.0
hooks:
 - id: black
 - repo: <https://github.com/pycqa/flake8>
rev: 6.0.0
hooks:
 - id: flake8



Why?

- 可以确保改动后仍然保持正确性，提高代码质量
- 提高代码的可维护性，降低维护成本
- 单元测试可以发现 Bug，避免意外改动影响已有功能，提高开发效率
- 单元测试可以作为文档，帮助团队理解代码

How?

Pytest 是 Python 语言中最流行的测试框架之一，它可以帮助我们自动化测试代码，确保代码按照预期运行，减少人工测试的工作量。

- 文件名 必须以 test_ 开头或 _test.py 结尾。
- 类名 必须以 Test 开头，并且不能有 init 方法。
- 函数/方法 必须以 test_ 开头。
- 可通过 pytest.ini 自定义规则。



使用 pytest 编写和运行测试



Step1. 安装 pytest:

```
$ pip install pytest
```

Step2. 创建 tests/test_retriever.py:

1. `import` pytest
2. `from` my_project `import` create_retriever
3. `TEST_PATH = "./data_kb"`
`# 测试文档路径`
4. `def` test_retriever_contains_keyword():
5. test_query = "为我介绍一下2008年北京奥运会"
6. expected_keyword = "奥运比赛"
7. results = create_retriever(TEST_PATH, test_query)
8. top_content = results[0].get_content() `if` results `else` ""
9. `assert` expected_keyword `in` top_content, (
 f"检索结果中未找到关键词 '{expected_keyword}' ")
10. `def` test_retriever_empty_query():
11. results = create_retriever(TEST_PATH, "")
12. `assert` isinstance(results, list), "结果应该是列表类型"

Step3. 运行 pytest:

```
$ PYTHONPATH=${PWD}:${PYTHONPATH}
$ pytest --disable-warnings tests/test_retriever.py
```

```
===== test session starts =====
platform linux -- Python 3.10.9, pytest-8.3.3, pluggy-1.5.0
rootdir: /root
plugins: anyio-4.4.0, hydra-core-1.3.2
collected 2 items

tests/test_retriever.py .. [100%]

===== 2 passed in 0.01s =====
```



pytest中常用的标记 (扩展)

pytest 提供了 markers (标记) 机制, 可以用来分类测试、控制测试执行、参数化测试等。以下是一些常见的 pytest 标记及其用途。

```
@pytest.mark.device(serial="abc")
def test_another():
    pass
```

```
class TestClass:
    def test_method(self):
        pass
```

```
@pytest.mark.webtest
def test_send_http():
    # perform some webtest test for your app
    pass
```

```
@pytest.mark.device(serial="123")
def test_something_quick():
    pass
```

您可以仅运行有特定标记的测例, 例如

```
$ pytest -v -m webtest
```

更进一步, 您可以仅运行与一个或多个标记关键字参数匹配的测试, 例如

```
$ pytest -v -m "device(serial='123' )"
```

更多信息参考: <https://docs.pytest.org/en/stable/example/markers.html>



- 在持续集成（CI）中，测试覆盖率是衡量代码质量的重要指标。使用 pytest 结合 pytest-cov 插件，可以生成详细的测试覆盖率报告，帮助开发者分析哪些代码未被测试。

- 执行测试，添加覆盖率统计：

```
$ export PYTHONPATH=${PWD}:$PYTHONPATH
```

```
$ pytest --cov=my_project --cov-append --cov-report=html
```

- 测试完成后，可以在 [htmlcov](#) 目录 查看HTML 报告。
- `pytest --cov=my-project` 统计 my-project 目录下的代码覆盖率。
- `--cov-append` 确保多次运行时，覆盖率不会被重置。
- `--cov-report=html` 生成可视化 HTML 报告，帮助分析测试覆盖率。
- 详细可参考 [pytest-cov](#)



测试覆盖率



Coverage report: 67%

Files Functions Classes

coverage.py v7.6.12, created at 2025-04-11 10:41 +0800

File ▲

- lazyllm/__init__.py
- lazyllm/cli/__init__.py
- lazyllm/cli/deploy.py
- lazyllm/cli/install.py
- lazyllm/cli/main.py
- lazyllm/cli/run.py
- lazyllm/client.py
- lazyllm/common/__init__.py
- lazyllm/common/bind.py
- lazyllm/common/common.py
- lazyllm/common/deprecated.py
- lazyllm/common/globals.py
- lazyllm/common/logger.py
- lazyllm/common/multiprocessing.py
- lazyllm/common/option.py
- lazyllm/common/queue.py
- lazyllm/common/registry.py
- lazyllm/common/text.py
- lazyllm/common/threading.py
- lazyllm/common/utils.py
- lazyllm/components/__init__.py
- lazyllm/components/auto/__init__.py

```
420     return store.get_nodes(group_name)
421
422     def retrieve(self, query: str, group_name: str, similarity: str, similarity_cut_off: Union[float, Dict[str, float]],
423                 index: str, topk: int, similarity_kws: dict, embed_keys: Optional[List[str]] = None,
424                 filters: Optional[Dict[str, Union[str, int, List, Set]]] = None) -> List[DocNode]:
425         self._lazy_init()
426         self._dynamic_create_nodes(group_name, self.store)
427
428         if index is None or index == 'default':
429             return self.store.query(query=query, group_name=group_name, similarity_name=similarity,
430                                     similarity_cut_off=similarity_cut_off, topk=topk,
431                                     embed_keys=embed_keys, filters=filters, **similarity_kws)
432
433         index_instance = self.store.get_index(type=index)
434         if not index_instance:
435             raise NotImplementedError(f"index type '{index}' is not supported currently.")
436
437         try:
438             return index_instance.query(query=query, group_name=group_name, similarity_name=similarity,
439                                         similarity_cut_off=similarity_cut_off, topk=topk,
440                                         embed_keys=embed_keys, filters=filters, **similarity_kws)
441         except Exception as e:
442             raise RuntimeError(f"index type `{index}` of store `{type(self.store)}` query failed: {e}")
443
444     def find(self, nodes: List[DocNode], group: str) -> List[DocNode]:
445         if len(nodes) == 0: return nodes
446         self._lazy_init()
447         self._dynamic_create_nodes(group, self.store)
448
449     def get_depth(name):
450         cnt = 0
451         while name != LAZY_ROOT_NAME:
452             cnt += 1
453             name = self.node_groups[name]['parent']
```



持续集成 (CI) : 让测试自动化



1.传统项目开发测试会遇到下面等问题

1. 传统手动运行测试，可能会忘记或执行不完整。
2. 多人协作时，每个人的测试环境可能不同，导致**“在我电脑上能跑”**的问题。
3. 如果代码合并后才手动测试，可能要到后期才发现 Bug，修复成本更高。
4. 代码变更较多时，手动测试可能遗漏边界情况。
5. 不同开发者的代码风格不同，可能导致代码风格混乱。

2.使用GitHub Actions CI有下面等优点

1. 每次**提交 (push)** 或**拉取请求 (pull request)** 时，自动运行测试，避免问题进入主分支。
2. 开发者能在第一时间知道代码是否通过所有测试。
3. 代码测试在标准化的GitHub Runner环境运行，不受开发者本地环境影响。
4. 可以并行运行多个测试，加快反馈速度。
5. 使用 **prettier**、**black**、**flake8**、**eslint** 等工具，自动检查代码格式，保持代码风格一致。
6. Github 提供免费的runner使用，可以白嫖



添加 `.github/workflows/test.yml`，配置测试脚本，即可实现每次提交pr或合入分支时进行测试

name: Run Tests

on: [push, pull_request]

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v3

with:

python-version: "3.10"

- name: Install dependencies

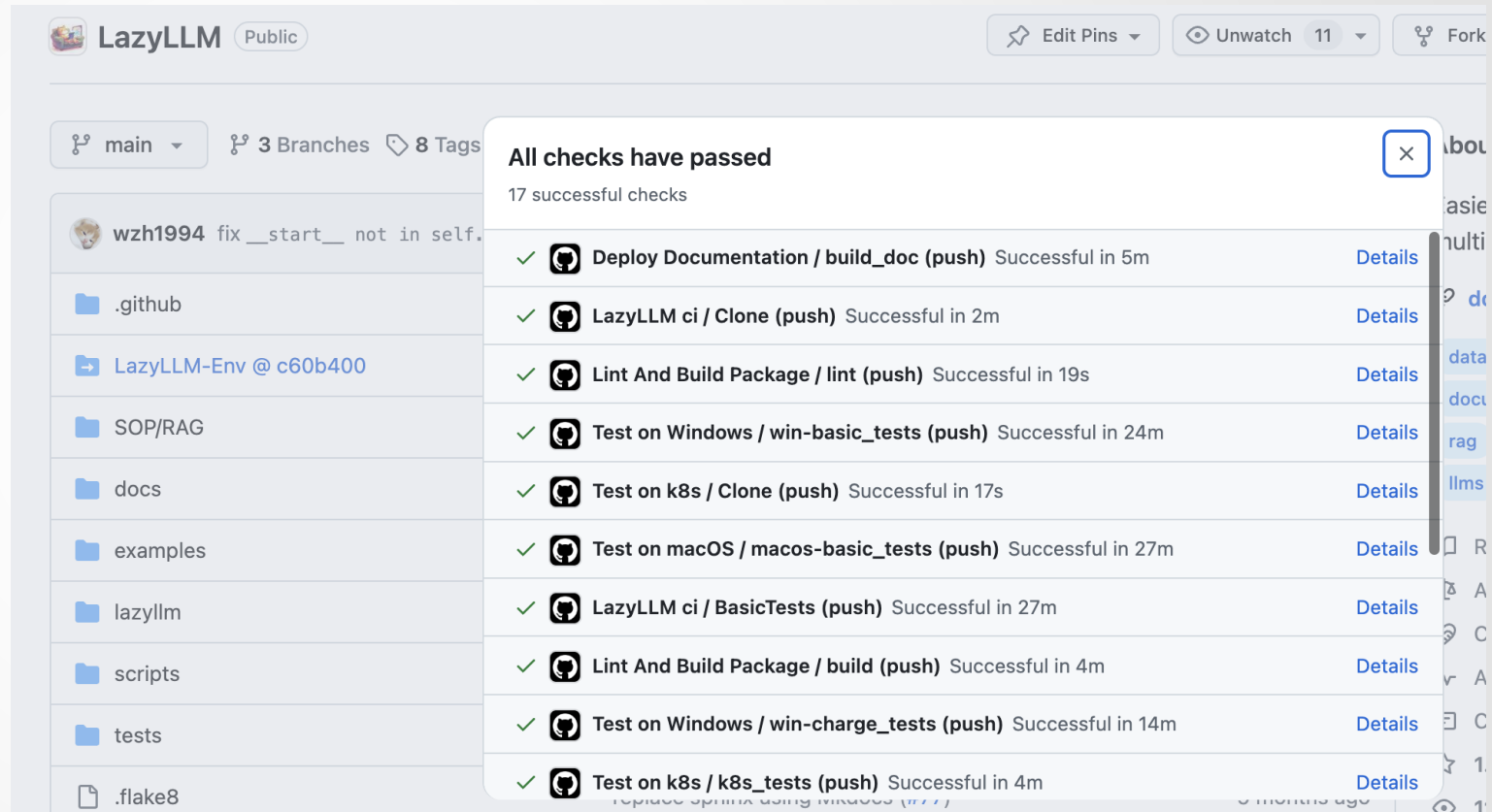
run: pip install -r requirements.txt

- name: Run tests

run: |

export PYTHONPATH=\${PWD}:\$PYTHONPATH

pytest



目录

-  1. 上节回顾
-  2. Why
-  3. Git管理
-  4. 代码质量保证
-  5. 工程发布



用 MkDocs 搭建文档系统

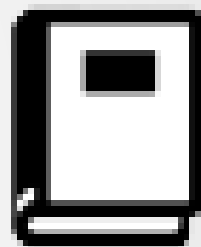
为什么要用MkDocs

1.在项目开发过程中，文档管理常常遇到以下问题：

1. 不同开发者可能使用 **Word**、**Markdown**、**PDF**、**Wiki** 等不同格式，导致难以维护。
2. 许多文档存储在本地，缺少版本控制，难以追踪更改历史。
3. 需要手动更新编译，过程繁琐。

2.使用MkDocs 可以很好的这些问题

1. 使用 Markdown 编写，简单易读，统一格式，降低学习成本。
2. Git 版本控制，文档与代码一起管理，随代码更新而更新。
3. 结合 GitHub Actions 或 Read the Docs，提交代码后自动生成最新文档，无需手动更新。



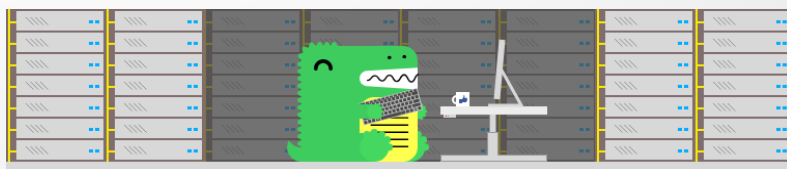
除了MkDocs还可以用什么



Sphinx



pdoc



Docusaurus



使用 MkDocs 生成文档



1.安装依赖包，有关详细信息，请参阅[安装指南](#)。

```
$ pip install mkdocs
```

2.初始化 MkDocs:

```
$ mkdocs new my-project >>> cd my-project
```

```
$ cd my-project
```

3.创建文件目录如下：有一个配置文件`mkdocs.yml`，以及一个名为 `docs` 的文件夹，其中包含您的文档源文件（是`docs_dir` 配置设置的默认值）。目前该 `docs` 文件夹仅包含一个名为`index.md` 的文档页面。

```
my-project/  
├── docs  
│   └── index.md  
└── mkdocs.yml
```

4.查看项目

然后通过运行以下命令启动服务器`mkdocs serve`

端口被占用时可以用 `-a` 指定端口 `mkdocs serve -a 0.0.0.0:8008`

```
$ mkdocs serve
```

```
INFO - Building documentation...
```

```
INFO - Cleaning site directory
```

```
INFO - Documentation built in 0.22 seconds
```

```
INFO - [15:50:43] Watching paths for changes: 'docs', 'mkdocs.yml'
```

```
INFO - [15:50:43] Serving on http://127.0.0.1:8000/
```

在浏览器中打开<http://127.0.0.1:8000/>，您将看到显示的默认主页：

详细配置参考：<https://www.mkdocs.org/>



托管文档到 Read the Docs



1. 为什么选择 Read the Docs ?

在开发项目时，良好的 **文档** 是不可或缺的。相比把文档散落在本地文件、Markdown 文件或 Wiki 页面上，**Read the Docs** 提供了一个高效的在线文档托管和自动构建平台，特别适合 **开源项目** 和 **持续更新的技术文档**。

以下是托管到 Read the Docs 的几个核心优势：

- **自动构建**：推送代码到 GitHub，Read the Docs 就会自动构建并更新文档。
- **版本管理**：支持多个文档版本，可以让用户查看不同版本的文档（比如 latest、stable、v1.0）
- **在线搜索**：提供在线搜索功能，可以快速查找内容。
- **免费托管**：完全免费，不需要额外购买服务器或域名。且自带 **https 安全访问**，无需配置 SSL 证书。

2. 添加.readthedocs.yaml 文件 [配置说明](#)

```
# .readthedocs.yaml
# Read the Docs 配置文件

# 必需的版本字段
version: 2

# 设置构建环境
build:
  os: ubuntu-24.04
  tools:
    python: "3.10" # MkDocs 需要 Python

# 配置 MkDocs
mkdocs:
  configuration: mkdocs.yml # 默认的 MkDocs 配置文件

# 可选：指定 Python 依赖（如果有）
#python:
# install:
#   - requirements: requirements.txt # 如果你有依赖文件
```



托管文档到 Read the Docs



3.注册 [Read the Docs](#) 账号 （选择github自动登录）

4.在 Read the Docs 主页，点击 导入一个项目 **(Import a Project)** 。

5.选择需要托管的 GitHub 仓库，点击 **+** 继续。

6.设置

名称：默认仓库名

默认分支：选主分支

语言：该项目的文档所呈现的语言

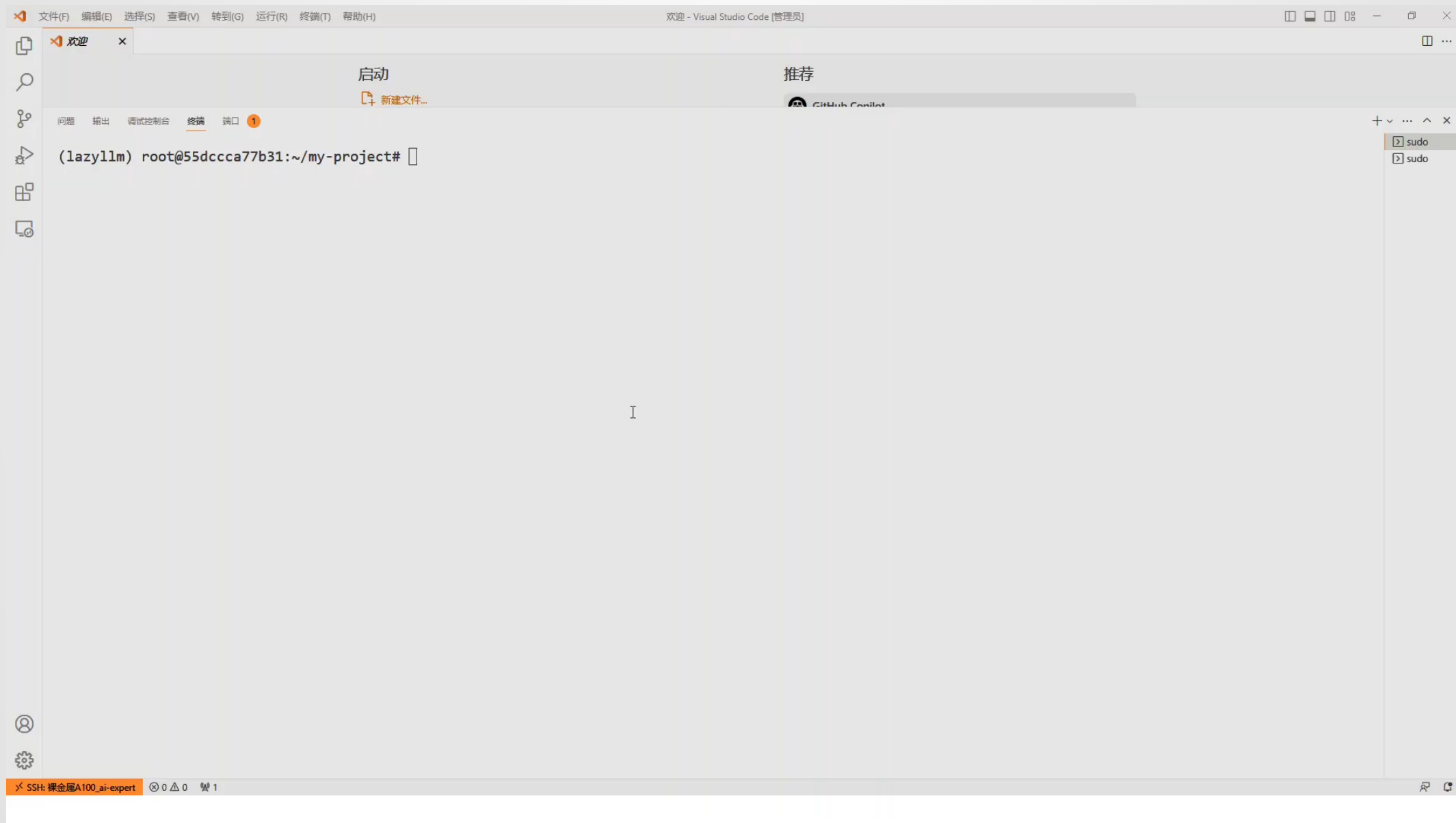
7.点击 **下一页**，系统会自动构建文档。

8.Read the Docs 会自动构建并托管文档

9.查看文档 <https://<your-project>.readthedocs.io/> <your-project> 换成自己的项目名称



托管文档



制品(Artifacts)



在 Python 项目开发过程中，除了编写代码，我们通常还需要将项目打包成 **可发布、可安装、可复现** 的格式，这些被称为 **制品 (Artifacts)**。制品可以是 **Python 包 (wheel、sdist)**、**Docker 镜像**、**二进制文件**、**可执行程序**。制作制品的优势是**便于发布与分发，确保环境一致性并便于测试**。

whl包制品

为什么要做whl包制品

.whl (Wheel) 是一种 **Python 包的二进制分发格式**，相比源码安装更高效，广泛用于发布和部署 Python 项目。

- **提高安装速度**: 打成whl包可以直接通过 `pip install my-project.whl` 比源码安装更快
- **行业共识**: 现在python制品通常都是做成whl包，并上传官方pypi仓库，方便更好的传播与使用

镜像制品

为什么要做Docker镜像?

Docker 是一种开源的容器化平台，用于自动化应用部署。它通过容器技术将应用程序及其依赖打包成一个 轻量、可移植的单元，能在不同环境中一致运行。

使用Docker制作镜像制品有下面好处：

- 环境一致性：避免“在我机器上能跑，但在服务器上不行”的问题。保证在 开发、测试、生产 环境，代码和依赖都一样。
- 便于部署和扩展：一次构建，随处运行，可以在 服务器、Kubernetes、云环境 运行。
- 行业影响力：Docker在业内使用广泛，有自己的官网，基于此有利于更好的传播与使用。



使用 setuptools 进行打包 在 setup.py 中定义：

```
1.from setuptools import setup, find_packages

2.setup(
    name="my_project",
    version="0.1.0",
    packages=find_packages(),
    install_requires=["lazyllm"],
    author="Your Name",
    author_email="your@email.com",
    description="A simple RAG retriever package",
    long_description=open("README.md").read(),
    long_description_content_type="text/markdown",
    python_requires=">=3.6",
)
```

生成 .whl 包：

```
$ python setup.py bdist_wheel
running bdist_wheel
running build
running build_py
creating build
creating build/lib
creating build/lib/my_project

....

adding 'my_project-0.1.0.dist-info/WHEEL'
adding 'my_project-0.1.0.dist-info/top_level.txt'
adding 'my_project-0.1.0.dist-info/RECORD'
removing build/bdist.linux-x86_64/wheel
>>> ls dist/
my_project-0.1.0-py3-none-any.whl
```



上传whl包到pypi



注册pypi账号

1. 访问 PyPI 官网 创建账号并登录。
2. 点击右上角 "Account settings" 进入设置页面。
3. 在 "API tokens" (API 令牌) 部分, 点击 "Add API token" (添加 API 令牌) 。
4. 配置 Token:
 - 名称 (Name) : 例如 INDEX_PYPI_TOKEN
 - 作用范围 (Scope) :
 - Entire account (整个账户) : 允许管理所有 PyPI 项目 (不推荐)
 - Specific project (指定项目) : 建议选择你的项目名称 (更安全)
5. 点击 Create token (创建令牌) 。
6. 复制生成的 API Token (仅显示一次, 注意: 不要泄露此令牌!)

制作并推送whl包

7. 本地添加 ~/.pypirc 文件

```
[pypi]
username = __token__
password = pypi-
xxxxxxxxxxxxxxxxxxxxYjNjNS0xMDExNW
MwMzhlNDMiXQAABiDpxiNjoqIT3SJDN
rQPP-BJI_AhO7pHErgKvOnS4jzNrQ
```

8. 安装工具及上传制品

```
>>> pip install twine
>>> twine upload dist/*
```



whl包的自动化构建



通过持续部署来发布应用

通过集成github action实现自动化发布，是github项目经常使用的一种方式，不用人为手动构建，可以在设定情况下构建包并上传到官方pypi仓库（比如 打tag）

1. 进入你的 GitHub 仓库。
2. 点击 "Settings" → "Secrets and variables" → "Actions".
3. 在 "Secrets" 部分点击 "New repository secret".
4. 填写：
 - Name (名称)：如 INDEX_PYPI_TOKEN
 - Value (值)：粘贴你的 API Token
5. 点击 "Add secret".
6. 仓库中添加.github/workflows/publish-to-pypi.yaml文件

name: Publish to PyPI

on:

push:

tags:

- "v*" # 仅在创建 tag (如 v1.0.0) 时触发

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v3

- name: Setup Python

uses: actions/setup-python@v4

with:

python-version: "3.x"

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install build twine

- name: Build package

run: python -m build

- name: Upload to PyPI

env:

INDEX_PYPI_TOKEN: \${ secrets.INDEX_PYPI_TOKEN }

run: |

twine upload --username __token__ --password \$INDEX_PYPI_TOKEN dist/*



1. 注册 Docker Hub 账号

- 访问 Docker Hub 官网 并注册一个账号。注册的账号名就是 命名空间 (Namespace)
- 注册成功后，登录你的 Docker Hub 账户。

2. 创建仓库 (Repository)

1. 点击右上角的 Create a repository (创建仓库)。

2. 填写仓库信息：

• Repository Name (仓库名称)：如 my-project

• Visibility (可见性)：

- Public (公开)：任何人都可以拉取你的镜像
- Private (私有)：只有你或授权用户可以访问

3. 点击 Create (创建)。

3. 登录 Docker Hub

- 在终端或命令行运行：

>>> docker login

- 然后浏览器打开<https://login.docker.com/activate>，并输入终端对应的code SPQK-WMDJ。如果登录成功，会显示：Login Succeeded

```
root@ubuntu:~# docker login
USING WEB-BASED LOGIN
Info → To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: SPQK-WMDJ
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser...

WARNING! Your credentials are stored unencrypted in '/root/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
```



镜像发布



构建并推送镜像

1. 编辑Dockerfile

1. 选择基础镜像

FROM python:3.10

2. 设置环境变量

ENV PYTHONUNBUFFERED=1

3. 创建应用文件夹

RUN mkdir /app

4. 拷贝源码

COPY my_project /app/my_project

COPY requirements.txt /tmp/requirements.txt

ENV PYTHONPATH="/app:\${PYTHONPATH}"

5. 安装依赖

RUN pip install -r /tmp/requirements.txt \

&& rm -rf /tmp/requirements.txt

6. 创建非 root 用户, 提升安全性

RUN useradd -m myuser

USER myuser

7. 设置默认启动命令

CMD ["/bin/bash"]

2. 构建镜像

```
>>> docker build -t username/my-project:0.1.0 .
```

3. 发布镜像

```
>>> docker push username/my-project:0.1.0
```

镜像使用

1. 拉取镜像

```
>>> docker pull username/my-project:0.1.0
```

2. 起容器使用

```
>>> docker run -it --name my-project username/my-project:0.1.0
```



感谢聆听
Thanks for Listening

