数据采集模块开发文档

> - 参考项目：https://github.com//Evil0ctal/Douyin_TikTok_Download_API

核心目标：将开源项目的采集能力封装为微服务，接入 Nacos、Kafka 等中间件，按架构设计的「数据采集链路」实现功能。

一、先明确用户中心模块的核心定位与边界

1. 核心职责

> 对齐 ADR 定义

- 采集层：对接抖音 / 快手 / B 站等 UCG 平台官方 API 及合规爬虫，采集红人数据（粉丝量 / 互动率 / 领域标签）、视频数据（播放量 / 评论 / 无水印地址）；
- 处理层：对采集的原始数据进行标准化清洗、格式转换，适配 AI 算法服务及其他微服务的数据需求；
- 流转层：通过 Kafka 实现采集数据异步流转，支撑数据监控服务的实时分析；
- 服务层：为报价引擎、匹配系统等微服务提供统一的数据查询接口，注册到 Nacos 实现服务发现；
- 风控层：实现 API 调用限流、平台风控适配（如 X-Bogus/A-Bogus 算法），保障采集稳定性。

2. 模块边界

> 避免越权开发

- 不负责：报价生成、红人与品牌方匹配、数据监控分析（属于其他微服务职责）；
- 仅提供：数据采集、数据标准化、数据查询 / 推送接口，不存储业务核心数据（如合作订单、用户认证信息）。

二、开发前置准备（环境与工具对齐 ADR）

1. 本地环境适配（基于 WSL+Docker Desktop）

依赖组件	版本要求（ADR 指定）	部署方式（本地开发）
Python	3.10	WSL 内安装 Python3.10，配置虚拟环境（python -m ver
FastAPI	0.110.2	虚拟环境内安装：pip install fastapi==0.110.2
Uvicorn	0.24.0	接口服务启动依赖：pip install uvicorn==0.24.0
MySQL	8.0	Docker 启动 MySQL8.0 容器（映射 3306 端口，挂载数据 mysql/data:/var/lib/mysql）
Redis	7.0	Docker 启动 Redis7.0（单机版，用于限流计数与缓存：-v ./redis/data:/data）
Kafka	3.6	Docker 启动 Kafka+Zookeeper 集群（映射 9092 端口，zookeeper:2181）
Nacos	兼容 FastAPI 版本	Docker 启动 Nacos 单机版（映射 8848 端口，MODE=sta
SQLAlchemy	2.0.23	ORM 框架：pip install sqlalchemy==2.0.23
HTTPX	0.25.0	异步 HTTP 请求（复用开源项目依赖）：pip install httpx
Kafka-Python	2.0.2	Kafka 客户端：pip install kafka-python==2.0.2
Nacos-SDK-Python	1.4.0	Nacos 服务注册依赖：pip install nacos-sdk-python==1.

2. 工具准备

- 代码管理：Git（本地仓库 + 远程备份，关联开源项目分支）；
- 接口调试：Postman（测试 HTTP 接口）、Swagger（FastAPI 自动生成接口文档）；
- 数据库工具：DBeaver（连接 Docker 内 MySQL，管理采集相关表）；
- 容器管理：Docker Desktop（管理 MySQL/Redis/Kafka/Nacos 容器）；
- 日志调试：PyCharm/WSL 终端（查看服务日志、Kafka 消息流转）；
- 爬虫适配：Chrome 浏览器（获取 UCG 平台 Cookie，用于风控适配）。

三、核心功能拆解与开发优先级

按「基础采集→多平台适配→中间件集成→风控优化→微服务联调」的顺序开发，优先保障单平台数据采集与数据流转，再补充复杂功能。

开发阶段	核心功能模块	具体需求（对齐产品文档）	
阶段 1	单平台基础采集	1. 复用开源项目 /crawlers 模块，实现b站视频数据采集（播放量 / 评论 / 无水印地址）； 2. 数据标准化解析（统一字段格式，如粉丝量转数字、时间戳格式化）； 3. 基础 HTTP 接口开发（单视频链接采集）	P
阶段 2	中间件集成	1. 服务注册到 Nacos（支持其他微服务发现）； 2. 集成 Kafka，实现采集数据异步推送； 3. Redis 缓存高频采集数据（如红人基础信息，过期 1 小时）	P
阶段 3	风控与限流	1. 实现 API 调用限流（基于 Redis 计数器，按平台配置阈值）； 2. 适配 UCG 平台风控（复用开源项目 X-Bogus/A-Bogus 算法）； 3. 多 Cookie 轮询（Nacos 配置，规避单账号限流）	P
阶段 4	微服务接口适配	1. 提供红人数据查询接口（供报价引擎调用）； 2. 提供视频数据推送接口（供数据监控服务消费）； 3. 接口鉴权（内部服务调用验签）	P
阶段 5	多平台适配与批量采集	1. 扩展 TikTok/抖音采集能力（复用开源项目对应爬虫逻辑）； 2. 支持批量链接采集（抖音 / TikTok 混合输入）； 3. 采集结果本地存储（MySQL）	P

四、数据库设计（基于 MySQL8.0，对齐 ADR）

数据采集服务仅存储采集任务、采集记录、平台配置等结构化数据，设计如下（核心表 + 核心字段）：

1. 核心表结构（SQLAlchemy 映射）

代码块

```
# 1. 采集任务表（crawler_task）- 存储批量/单条采集任务信息
class CrawlerTask(Base):
    __tablename__ = "crawler_task"
    id = Column(BIGINT, primary_key=True, comment="任务唯一ID")
    user_id = Column(BIGINT, nullable=False, comment="关联用户中心user_base.ID（发起采集的用户）")
    task_type = Column(SmallInteger, nullable=False, comment="任务类型：1=单视频采集，2=批量视频采集，3=红人数据采集")
    platform_type = Column(SmallInteger, nullable=False, comment="UCG平台类型：1=抖音，2=TikTok，3=B站")
```

```
    input_content = Column(String(1000), nullable=False, comment="输入内容（视频链接/红人ID，批量用逗号分隔）")
    task_status = Column(SmallInteger, default=0, comment="任务状态：0=待执行，1=执行中，2=成功，3=失败")
    total_count = Column(Integer, default=0, comment="采集总数")
    success_count = Column(Integer, default=0, comment="采集成功数")
    create_time = Column(DateTime, default=datetime.now, comment="创建时间")
    update_time = Column(DateTime, default=datetime.now, onupdate=datetime.now, comment="更新时间")

# 2. 采集记录表（crawler_record）- 存储单条采集结果
class CrawlerRecord(Base):
    __tablename__ = "crawler_record"
    id = Column(BIGINT, primary_key=True, comment="记录唯一ID")
    task_id = Column(BIGINT, nullable=False, comment="关联crawler_task.id")
    resource_id = Column(String(50), comment="资源ID（视频ID/红人ID）")
    resource_type = Column(SmallInteger, comment="资源类型：1=视频，2=红人")
    raw_data = Column(Text, comment="采集原始数据（JSON字符串）")
    parsed_data = Column(Text, comment="标准化后的数据（JSON字符串）")
    error_msg = Column(String(500), comment="采集失败原因")
    create_time = Column(DateTime, default=datetime.now, comment="采集时间")

# 3. UCG平台配置表（ucg_platform_config）- 存储平台API配置、Cookie
class UcgPlatformConfig(Base):
    __tablename__ = "ucg_platform_config"
    id = Column(BIGINT, primary_key=True, comment="配置ID")
    platform_type = Column(SmallInteger, unique=True, nullable=False, comment="平台类型：1=抖音，2=TikTok，3=B站")
    api_url = Column(String(255), comment="平台API基础地址")
    cookie = Column(Text, comment="平台登录Cookie（多个用逗号分隔，轮询使用）")
    x_bogus_secret = Column(String(100), comment="X-Bogus算法密钥")
    rate_limit = Column(Integer, default=10, comment="每秒最大调用次数（限流阈值）")
    status = Column(SmallInteger, default=1, comment="配置状态：0=禁用，1=启用")
    update_time = Column(DateTime, default=datetime.now, onupdate=datetime.now, comment="更新时间")

# 4. 限流规则表（rate_limit_rule）- 存储API限流配置
class RateLimitRule(Base):
    __tablename__ = "rate_limit_rule"
    id = Column(BIGINT, primary_key=True, comment="规则ID")
    rule_key = Column(String(50), unique=True, nullable=False, comment="限流键（如platform:douyin）")
    limit_count = Column(Integer, nullable=False, comment="限制次数")
    limit_seconds = Column(Integer, nullable=False, comment="限制时间（秒）")
    create_time = Column(DateTime, default=datetime.now, comment="创建时间")
```

2. 索引设计

- 高频查询字段：`crawler_task.user_id`（用户关联查询）、`crawler_record.task_id`（任务关联查询）、`ucg_plat[illegible]`（平台配置查询）；
- 唯一索引：`ucg_platform_config.platform_type`（避免重复配置）、`rate_limit_rule.rule_key`（限流规则唯一）。

五、核心接口设计（RESTful API，基于 FastAPI）

按「用户端操作接口 + 微服务调用接口」分类，统一前缀 `/api/v1/crawler`，通过 FastAPI 自动生成 Swagger 文档（访问 `/docs`）。

1. 用户端操作接口（红人 / 品牌方调用，需 自定义token 鉴权）

接口路径	方法	功能描述	请求参数示例

/task/create	POST	创建采集任务（单条 / 批量）	Header: Authorization: Bearer {token}
			Body: {
			"task_type":1,
			"platform_type":1,
			"input_content":"https://v.douyin.com/L5pbfdP/"
			}

/task/status	GET	查询任务状态	Header: Authorization: Bearer {token} Query: task_id=123
/record/detail	GET	查询采集结果详情	Header: Authorization: Bearer {token} Query: task_id=123&resource_id=7156033831819037994

2. 微服务调用接口（内部模块调用，需服务鉴权）

接口路径	方法	功能描述	调用方	请
/inner/kol/data	GET	查询红人基础数据（供报价引擎）	报价引擎服务	Query: user_id=123&
/inner/video/data	GET	查询视频数据（供数据监控服务）	数据监控服务	Query: video_id=xxx
/inner/platform/config	GET	获取平台配置（供自身采集使用）	数据采集服务自身	Query: platform_typ

六、关键技术实现（对齐 ADR 技术栈）

1. 服务注册与发现（Nacos 集成）

- 核心逻辑：服务启动时注册到 Nacos，提供服务名称、IP、端口，支持其他微服务通过 Nacos 发现并调用接口；
- 代码示例：

代码块

```
from nacos import NacosClient
from fastapi import FastAPI

app = FastAPI(title="Data Collection Service")

# Nacos连接配置（从环境变量读取，生产环境避免硬编码）
NACOS_IP = "localhost"
NACOS_PORT = 8848
SERVICE_NAME = "data-collection-service"
SERVICE_IP = "localhost"
SERVICE_PORT = 8000

# 注册服务到Nacos
@app.on_event("startup")
async def register_to_nacos():
```

```
    nacos_client = NacosClient(f"{NACOS_IP}:{NACOS_PORT}")
    nacos_client.add_naming_instance(
        service_name=SERVICE_NAME,
        ip=SERVICE_IP,
        port=SERVICE_PORT,
        cluster_name="DEFAULT"
    )
```

2. 多平台数据采集（复用开源项目爬虫）

- 核心逻辑：复用 Douyin_TikTok_Download_API 的 `/crawlers` 模块，剥离 PyWebIO 依赖，适配微服务的配置化调用；
- 代码示例（抖音视频采集）：

代码块

```python
from crawlers.douyin.web.scraper import DouyinWebScraper
from sqlalchemy.orm import Session
from models import ucg_platform_config

def crawl_douyin_video(db: Session, video_url: str) -> dict:
    # 从数据库获取抖音配置（Cookie、X-Bogus密钥）
    config = db.query(ucg_platform_config.UcgPlatformConfig).filter(
        ucg_platform_config.UcgPlatformConfig.platform_type == 1,
        ucg_platform_config.UcgPlatformConfig.status == 1
    ).first()

    # 初始化开源项目的爬虫实例
    scraper = DouyinWebScraper(cookie=config.cookie)
    # 调用开源项目的解析方法
    raw_data = scraper.hybrid_parsing(video_url)
    # 标准化数据（统一字段格式）
    parsed_data = {
        "video_id": raw_data.get("aweme_id"),
        "play_count": raw_data.get("statistics", {}).get("play_count", 0),
        "comment_count": raw_data.get("statistics", {}).get("comment_count", 0),
        "no_watermark_url": raw_data.get("video", {}).get("play_addr", {}).get("url_list", [])[0],
        "author_id": raw_data.get("author", {}).get("uid")
    }
    return raw_data, parsed_data
```

3. 异步数据流转（Kafka 集成）

- 核心逻辑：采集成功后，将标准化数据推送到 Kafka 指定 Topic，供数据监控服务消费；
- 代码示例：

代码块

```python
from kafka import KafkaProducer
import json
from datetime import datetime

# Kafka配置（从Nacos读取）
KAFKA_BOOTSTRAP_SERVERS = "localhost:9092"
KAFKA_TOPIC = "crawler_video_data"

# 初始化Kafka生产者
producer = KafkaProducer(
    bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS,
    value_serializer=lambda v: json.dumps(v).encode("utf-8")
)

def send_to_kafka(platform_type: int, resource_type: int, parsed_data: dict):
    """推送采集数据到Kafka"""
    message = {
        "platform_type": platform_type,
        "resource_type": resource_type,
        "parsed_data": parsed_data,
        "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    }
    producer.send(KAFKA_TOPIC, value=message)
    producer.flush()  # 确保消息发送成功
```

4. API 限流控制（Redis 实现）

- 核心逻辑：基于 Redis 计数器实现限流，按平台配置阈值限制 API 调用频率，避免触发 UCG 平台限流；
- 代码示例：

代码块

```python
import redis
from fastapi import HTTPException, Header
from sqlalchemy.orm import Session
from models import rate_limit_rule

# Redis连接配置
redis_client = redis.Redis(host="localhost", port=6379, db=0)

```

```python
def rate_limit_check(db: Session, platform_type: int, token: str = Header(...)):
    """限流校验中间件"""
    # 获取限流规则（如抖音平台：platform:1）
    rule_key = f"platform:{platform_type}"
    rule = db.query(rate_limit_rule.RateLimitRule).filter(
        rate_limit_rule.RateLimitRule.rule_key == rule_key
    ).first()
    if not rule:
        return  # 无规则则不限流

    # Redis计数器：key=rule_key:token，value=调用次数，过期时间=rule.limit_seconds
    redis_key = f"{rule_key}:{token}"
    current_count = redis_client.incr(redis_key)
    if current_count == 1:
        redis_client.expire(redis_key, rule.limit_seconds)

    # 超过阈值返回429
    if current_count > rule.limit_count:
        raise HTTPException(
            status_code=429,
            detail=f"API调用过于频繁，请{rule.limit_seconds}秒后再试"
        )
```

5. 风控适配（复用开源项目算法）

- 核心逻辑：复用开源项目的 X-Bogus/A-Bogus 算法，生成平台 API 所需的校验参数，规避风控拦截；
- 关键处理：从 Nacos / 数据库动态读取 Cookie，支持多 Cookie 轮询（当一个 Cookie 失效时自动切换下一个）。

6. 接口鉴权（自定义token+ 服务验签）

- 用户端接口：通过 自定义token 解析用户 ID，校验登录态（自定义token 从用户中心服务获取，Redis 缓存黑名单）；
- 微服务接口：内部服务调用时，通过 Nacos 配置的服务密钥验签（避免外部非法访问）。

七、开发与测试流程（适配本地 WSL+Docker 环境）

1. 开发步骤

1. 初始化项目：
    - 创建项目结构（参考如下），初始化 Git 仓库；

```
data-collection-service/
├── app/
│   ├── api/            # 接口层（endpoints/schemas）
│   ├── core/           # 核心配置（Nacos/Kafka/Redis连接）
│   ├── db/             # 数据库（models/session.py）
│   ├── service/        # 业务层（crawlers/parser/kafka_producer）
│   └── main.py         # 服务入口（FastAPI启动）
├── configs/            # 配置文件（本地开发用）
├── sql/                # 数据库脚本（表创建SQL）
├── Dockerfile          # 容器化打包
└── requirements.txt    # 依赖清单
```

2. 核心功能开发：
   - 按「阶段 1→阶段 5」顺序开发，优先实现单平台采集与接口，再集成中间件；
   - 复用开源项目 `/crawlers` 目录，改造配置读取方式（从数据库 / Nacos 读取，而非本地 config.yaml）。
3. 接口测试：
   - 启动服务：`uvicorn app.main:app --reload --host 0.0.0.0 --port 8000`；
   - 访问 `http://localhost:8000/docs`，用 Swagger 测试接口（如创建采集任务、查询结果）；
   - 验证数据流转：采集成功后，通过 Kafka 工具查看消息是否推送，MySQL 是否存储记录。
4. 微服务联调：
   - 本地启动 Nacos、用户中心服务，模拟报价引擎调用 `/inner/kol/data` 接口，验证数据返回正确性；
   - 测试限流功能：高频调用接口，确认超过阈值返回 429。
5. 容器化打包：
   - 编写 Dockerfile，构建镜像并启动容器，验证容器内服务能正常连接 Docker 中的中间件。

2. Dockerfile 示例（适配 Python+FastAPI）

代码块

```
# 构建阶段
FROM python:3.10-slim AS builder

WORKDIR /app
COPY requirements.txt .
# 安装依赖
RUN pip install --no-cache-dir -r requirements.txt -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

```
8
9   # 复制项目文件
10  COPY . .
11
12  # 运行阶段
13  FROM python:3.10-slim
14
15  WORKDIR /app
16  # 复制依赖和项目文件
17  COPY --from=builder /usr/local/lib/python3.10/site-packages
    /usr/local/lib/python3.10/site-packages
18  COPY --from=builder /app .
19
20  # 暴露端口
21  EXPOSE 8000
22
23  # 启动服务
24  CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

3. 本地测试命令

- 启动依赖容器：`docker-compose up -d mysql redis kafka nacos`（编写 docker-compose.yml 管理多容器）；
- 初始化数据库：执行 `sql/init_table.sql` 创建表结构；
- 启动服务：`source venv/bin/activate && uvicorn app.main:app --reload`；
- 接口测试：Postman 访问 `http://localhost:8000/api/v1/crawler/task/create`。

八、风险与应对（对齐 ADR 风险评估）

风险点	影响程度	应对方案
UCG 平台 API 限流 / 风控拦截	高	1. 基于 Redis 实现按平台限流，阈值参考平台配额； 2. 多 Cookie 轮询（Nacos 配置多个有效 Cookie）； 3. 降级策略：限流时返回 Redis 缓存的历史数据，同 4. 复用开源项目 X-Bogus/A-Bogus 算法，动态生成核
采集数据格式不一致（多平台差异）	中	1. 制定统一的数据标准化规则（如视频 ID、播放量字 2. 每个平台编写专属解析函数，适配字段差异； 3. 数据库存储原始数据，便于问题排查。
Kafka 消息丢失	中	1. 启用 Kafka 生产者确认机制（acks=1）； 2. 关键采集数据本地数据库备份，定期校验 Kafka 消 3. 实现消息重试机制，失败消息存入死信队列。
本地 Docker 容器通信失败	中	1. 配置 Docker 网络为桥接模式，确保 WSL 内的服务 2. 容器启动时添加--link参数（如--link nacos:nacos） 3. 本地开发时使用localhost作为中间件地址，避免硬
开源项目爬虫逻辑失效（平台 API 变更）	高	1. 监控采集成功率，当失败率超过阈值时告警； 2. 预留扩展接口，支持快速替换爬虫逻辑； 3. 优先对接平台官方 API，减少对爬虫的依赖。

九、输出物清单（个人项目归档）

1. 代码仓库：完整的 Python 项目代码（含注释、配置文件、开源项目复用模块改造记录）；
2. 接口文档：Swagger 生成的 HTML 文档（放在项目 `docs` 目录）；
3. 数据库脚本：MySQL 表创建 SQL 脚本（`sql/init_table.sql`）、测试数据脚本；
4. Docker 镜像：数据采集服务的 Docker 镜像（本地镜像或推送到镜像仓库）；
5. 测试报告：核心功能测试用例（如单平台采集、批量采集、限流、降级）及测试结果；
6. 配置说明：Nacos/Kafka/Redis 等中间件配置手册，Cookie 获取与更新指南。