

基于 Wagtail 的现代前后端分离 CMS 设计方案

1. 项目目标

设计一套基于 Python + Wagtail 的现代 Headless CMS（前后端分离内容管理系统），满足以下目标：

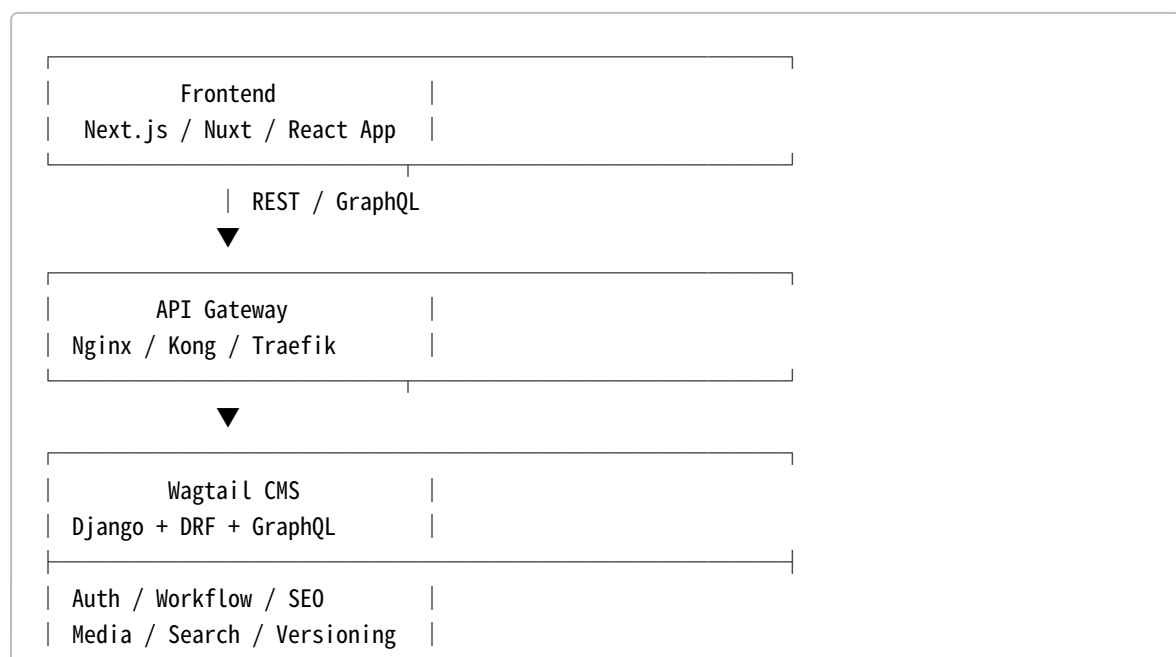
- 支持多站点、多语言、多租户
- 提供 REST / GraphQL API
- 前后端彻底解耦
- 支持内容审核与发布流
- 支持 SEO、媒体管理、版本管理
- 支持高并发与云原生部署
- 可扩展的插件化架构
- 面向企业级内容平台

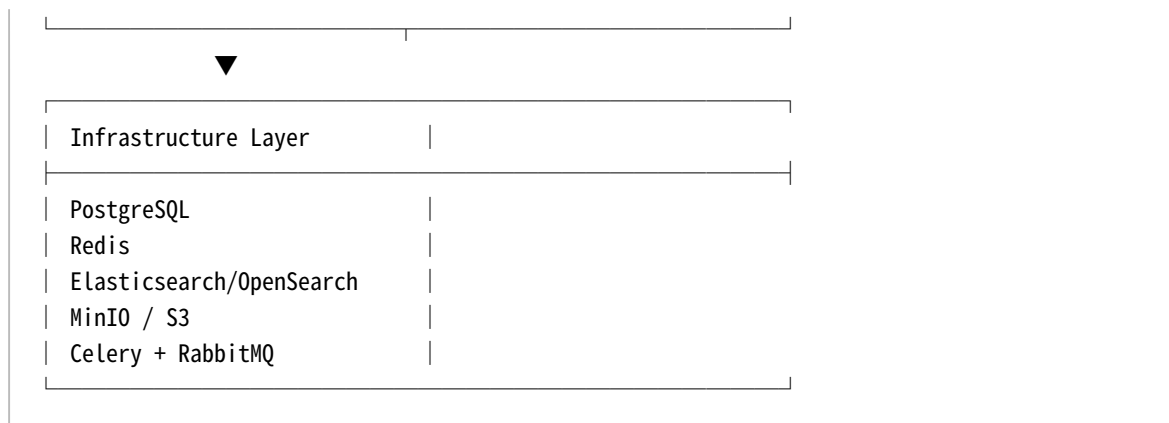
适用场景：

- 企业官网
- 新闻门户
- 内容运营平台
- 教育平台
- SaaS CMS 平台
- AI 内容生成平台
- 电商内容中台

2. 技术架构总览

整体架构





3. 技术栈设计

后端技术栈

模块	技术
Web Framework	Django
CMS Core	Wagtail
API	Django REST Framework
GraphQL	Strawberry / Graphene
Auth	JWT + OAuth2
Database	PostgreSQL
Cache	Redis
Search	Elasticsearch / OpenSearch
Async Task	Celery
Message Queue	RabbitMQ / Redis
Object Storage	MinIO / AWS S3
Container	Docker
Orchestration	Kubernetes
CI/CD	GitHub Actions / GitLab CI

前端技术栈

模块	技术
SSR Framework	Next.js

模块	技术
UI	React
State	Zustand / Redux Toolkit
Data Fetching	React Query / SWR
Styling	Tailwind CSS
CMS Rendering	Dynamic Block Renderer
i18n	next-intl
SEO	Next SEO

4. Wagtail 核心架构设计

核心模块划分

```
cms/  
├── apps/  
│   ├── core/  
│   ├── accounts/  
│   ├── pages/  
│   ├── media/  
│   ├── seo/  
│   ├── workflow/  
│   ├── search/  
│   ├── analytics/  
│   ├── forms/  
│   ├── comments/  
│   ├── ai/  
│   └── api/  
├── config/  
├── requirements/  
├── docker/  
└── deployment/
```

5. 内容模型设计

Page Model 设计

Wagtail 最大优势是 Page Tree。

推荐采用：

- Page + StreamField
- 组件化内容建模
- 动态 Block 系统

示例：

```
from wagtail.models import Page
from wagtail.fields import StreamField
from wagtail import blocks
from wagtail.admin.panels import FieldPanel

class BlogPage(Page):
    body = StreamField([
        ('heading', blocks.CharBlock()),
        ('paragraph', blocks.RichTextBlock()),
        ('image', ImageChooserBlock()),
    ], use_json_field=True)

    content_panels = Page.content_panels + [
        FieldPanel('body')
    ]
```

推荐内容模型结构

```
HomePage
├── ArticleIndexPage
│   ├── ArticlePage
│   └── ArticlePage
├── ProductIndexPage
│   └── ProductPage
├── LandingPage
├── MarketingPage
└── DocsPage
```

6. Headless API 设计

REST API 设计

推荐使用：

- DRF
- Wagtail API v2
- 自定义 Serializer

API 示例：

```
/api/v1/pages/  
/api/v1/articles/  
/api/v1/products/  
/api/v1/navigation/  
/api/v1/search/  
/api/v1/auth/
```

GraphQL 设计

推荐：

- Strawberry GraphQL
- 自动 Schema
- 支持内容聚合查询

示例：

```
query {  
  articles {  
    title  
    slug  
    body  
  }  
}
```

7. 前后端通信设计

推荐模式

SSR + ISR

Next.js:

- SSR: 动态内容
- ISR: 营销页面
- SSG: 静态文档

数据获取

```
Frontend
  ↓
API Gateway
  ↓
Wagtail API
  ↓
PostgreSQL
```

缓存层:

```
Frontend Cache
  ↓
CDN
  ↓
Redis
```

8. StreamField 组件化体系

推荐 Block 体系

```
blocks/
├── hero.py
├── richtext.py
├── faq.py
├── gallery.py
├── pricing.py
├── cta.py
```

```
|—— video.py
|—— embed.py
```

Block 示例

```
class HeroBlock(blocks.StructBlock):
    title = blocks.CharBlock()
    subtitle = blocks.TextBlock()
    image = ImageChooserBlock()

class Meta:
    icon = 'image'
```

前端渲染器

```
StreamField JSON
    ↓
Frontend Renderer
    ↓
React Components
```

例如：

```
const components = {
    hero: HeroComponent,
    faq: FAQComponent,
    gallery: GalleryComponent,
}
```

9. 用户权限与认证

推荐认证方案

CMS 后台

- Django Session Auth
- Wagtail Permission

前端用户

- JWT

- OAuth2
- SSO

推荐：

- Keycloak
- Auth0
- 自建 OAuth Server

RBAC 权限模型

```
graph TD
    SA[Super Admin] --- TA[Tenant Admin]
    TA --- Editor
    TA --- Reviewer
    TA --- Author
    TA --- Viewer
```

10. workflow与内容审核

Wagtail 原生支持：

- Workflow
- Moderation
- Revision
- Scheduled Publish

推荐流程：

```
graph TD
    Draft --> Review
    Review --> Approved
    Approved --> Published
```

增强建议：

- Slack 通知
 - 企业微信通知
 - 自动审校
 - AI 内容检测
-

11. 搜索架构

推荐架构

```
graph TD;
  Wagtail --> Elasticsearch/OpenSearch;
  Elasticsearch/OpenSearch --> Frontend_Search_UI[Frontend Search UI];
```

搜索功能

支持：

- 全文搜索
- 拼音搜索
- 分词搜索
- 自动补全
- 标签搜索
- AI Semantic Search

12. SEO 体系

SEO 模块设计

每个 Page：

```
seo_title
seo_description
og_image
canonical_url
schema_json
```

前端 SEO

Next.js 动态注入：

- Meta
- OpenGraph
- JSON-LD
- Sitemap

- Robots

13. 多语言设计

推荐方案

Wagtail i18n:

```
WAGTAIL_I18N_ENABLED = True
```

支持:

- 中文
- 英文
- 日文
- 多区域站点

URL 设计

```
/zh/articles/  
/en/articles/  
/ja/articles/
```

14. 多站点与多租户

多站点

Wagtail 原生支持:

```
Site A  
Site B  
Site C
```

多租户推荐方案

方案一：共享数据库

tenant_id

优点：

- 成本低
- 易扩展

缺点：

- 数据隔离较弱

方案二：Schema 隔离

推荐：

- django-tenants

优点：

- 隔离性强

缺点：

- 运维复杂

企业级推荐：

Wagtail + django-tenants

15. 媒体资源体系

推荐架构

```
Editor Upload
  ↓
MinIO/S3
  ↓
CDN
  ↓
Frontend
```

图片处理

推荐:

- Wagtail Renditions
- WebP
- AVIF
- 自动压缩

视频处理

推荐:

- Mux
- Cloudflare Stream
- OSS 视频点播

16. AI 能力扩展

AI 模块设计

AI Service

- └── SEO Rewrite
- └── Content Summary
- └── Auto Translation
- └── Tag Generation
- └── Semantic Search
- └── RAG Knowledge Base

推荐 AI 架构

```
graph TD;
  Wagtail --> CeleryTask[Celery Task];
  CeleryTask --> LLMService[LLM Service];
  LLMService --> StoreResult[Store Result];
```

推荐模型：

- OpenAI
- Claude
- DeepSeek
- Qwen

17. 缓存与性能优化

缓存层设计

```
graph TD; A[Browser Cache] --> B[CDN]; B --> C[Nginx Cache]; C --> D[Redis]
```

优化策略

API

- Redis Cache
- Query Optimization
- Select Related
- Prefetch Related

Frontend

- ISR
- Edge Cache
- Lazy Loading
- Code Splitting

18. 事件驱动架构

推荐事件系统

```
graph TD; A[Content Published] --> B[Event Bus]; B --> C[ ]
```

Webhook
↓
Frontend Revalidate

推荐用途

- CDN 刷新
- 搜索同步
- AI 分析
- Slack 通知
- 数据同步

19. DevOps 与部署

Docker 架构

nginx
frontend
wagtail
postgres
redis
rabbitmq
elasticsearch
minio

Kubernetes 推荐部署

Ingress
↓
Frontend Pods
Backend Pods
Worker Pods

推荐 CI/CD

Git Push
↓
GitHub Actions
↓
Docker Build

↓
K8S Deploy

20. 安全设计

必须实现

- HTTPS
- CSP
- JWT Rotation
- Rate Limit
- WAF
- CSRF 防护
- SQL 注入防护
- XSS 防护

审计日志

User Login
Content Edit
Publish Action
Permission Change

21. 推荐目录结构（企业级）

```
project/
├── backend/
│   ├── apps/
│   ├── config/
│   ├── api/
│   └── requirements/
├── frontend/
│   ├── app/
│   ├── components/
│   ├── lib/
│   └── services/
├── infrastructure/
│   ├── docker/
│   ├── k8s/
│   └── terraform/
```

```
|  
└── docs/
```

22. 推荐前端架构（Next.js）

```
frontend/  
├── app/  
├── components/  
├── blocks/  
├── layouts/  
├── services/  
├── hooks/  
├── stores/  
├── styles/  
└── utils/
```

23. 推荐 API 规范

REST 风格

```
GET /api/v1/articles  
GET /api/v1/articles/:id  
POST /api/v1/articles  
PUT /api/v1/articles/:id  
DELETE /api/v1/articles/:id
```

API Versioning

```
/api/v1/  
/api/v2/
```

24. 企业级扩展能力

推荐插件体系

```
plugins/  
├── comments/
```


└── ecommerce/
└── analytics/
└── ai/
└── crm/
└── marketing/

推荐扩展方式

- Django App
- Signals
- Hook System
- Middleware
- Webhook

25. 可观测性体系

推荐组件

功能	技术
Logging	ELK
Metrics	Prometheus
Dashboard	Grafana
Tracing	OpenTelemetry
Error Monitor	Sentry

26. 推荐开发阶段

Phase 1

- Wagtail 基础 CMS
- REST API
- Next.js 前端
- 用户系统

Phase 2

- GraphQL
- 搜索系统
- SEO 系统

- 工作流
-

Phase 3

- 多租户
 - AI 能力
 - 微服务拆分
 - SaaS 化
-

27. 推荐的最终企业架构

中小型项目

```
Wagtail Monolith
+ Next.js
+ PostgreSQL
+ Redis
```

中大型项目

```
Wagtail CMS
+ API Gateway
+ Next.js
+ Elasticsearch
+ Redis
+ Celery
+ S3
+ Kubernetes
```

SaaS CMS 平台

```
Wagtail
+ Multi-Tenant
+ Event Bus
+ AI Service
+ Microservices
+ Edge CDN
```

28. 为什么选择 Wagtail

相比传统 CMS:

能力	Wagtail	WordPress	Drupal
Python 生态	强	弱	弱
Headless	强	中	强
开发体验	优秀	一般	较复杂
StreamField	极强	一般	中
企业扩展	强	中	强
AI 集成	容易	一般	一般
前后端分离	非常适合	一般	适合

29. 推荐最终技术选型

最佳实践组合

Backend

- Django 5
- Wagtail 7
- DRF
- Strawberry GraphQL
- PostgreSQL
- Redis
- Celery

Frontend

- Next.js 15
- React 19
- TailwindCSS
- React Query

Infra

- Docker
 - Kubernetes
 - OpenSearch
 - MinIO
 - Cloudflare CDN
-

30. 总结

这套基于 Wagtail 的现代 CMS 架构具有以下特点：

- 真正前后端分离
- 强大的内容建模能力
- 优秀的编辑体验
- 企业级扩展能力
- 多租户与 SaaS 化能力
- AI Ready
- 云原生部署
- 高性能与高可维护性

适合构建：

- 企业内容平台
- Headless CMS
- AI CMS
- 多站点平台
- SaaS CMS
- 国际化内容平台

如果需要，我还可以继续帮你设计：

1. Wagtail Headless CMS 完整数据库设计
2. Wagtail + Next.js 完整项目脚手架
3. 企业级权限系统设计
4. StreamField 组件库设计
5. GraphQL Schema 设计
6. Kubernetes 部署方案
7. Docker Compose 开发环境
8. 多租户 SaaS 架构
9. AI 内容生成架构
10. 微服务拆分方案
11. 高并发优化方案
12. 企业级目录结构模板
13. Wagtail 插件体系
14. 完整 API 设计规范
15. 前端动态页面渲染引擎