文件类型编号：

<基于GIS地图的可视化智慧安防系统>

**设计说明书**

<*V1.0*>

(适用于过程语言开发)

**文件编号：**

编写者： 徐祺翔，丁宗参，张浩东，唐鑫臣

编写部门： 异常检测部门

编写时间： 2025/5/29

西北工业大学软件工程实验

改版履历

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 改版履历 | | 文件名: | | | | 制定部门： |
| 版数 | 承认/日期 | | 查阅/日期 | 作者/日期 | 改版内容 | |
|  |  | |  |  |  | |

**目录**

[1. 概述 1](#_Toc199412786)

[1.1. 概述 1](#_Toc199412787)

[1.2. 命名规则 2](#_Toc199412788)

[1.3. 相关文档 4](#_Toc199412789)

[2. 数据设计 5](#_Toc199412790)

[2.1. 数据库结构 5](#_Toc199412791)

[2.2. 外部文件结构 7](#_Toc199412792)

[2.3. 外部文件结构 8](#_Toc199412793)

[2.4. 外部文件结构 9](#_Toc199412794)

[3. 程序结构 10](#_Toc199412795)

[3.1. 程序整体结构 10](#_Toc199412796)

[3.2. 模块一览表 11](#_Toc199412797)

[4. 逻辑设计 13](#_Toc199412798)

[4.1. 用户认证模块 13](#_Toc199412799)

[4.2. 视频异常检测模块 14](#_Toc199412800)

[4.3. GIS地图可视化模块 15](#_Toc199412801)

[5. 接口设计 16](#_Toc199412802)

[6. 其他事项 17](#_Toc199412803)

# 概述

## **概述**

本项目聚焦智慧安防领域，针对传统安防系统误报率高、模型检测结果可解释性不足，可视化水平低，泛化性能力差等问题，基于视觉语言大模型（VLM）和GIS地图技术开发“基于GIS地图的可视化智慧安防系统”。系统整合计算机视觉、视觉语言大模型、数据库管理，GIS地图可视化等技术，通过Qwen2-VL视觉语言模型、RAG检索增强生成技术及Agent任务分配机制，实现对视频流的分析、异常行为识别、检测结果的自然语言描述生成以及GIS可视化检测结果，解决传统方案在检测结果精确度、可解释性以及可视化上的瓶颈。

系统核心功能包括用户管理、视频管理、视频异常检测及自然语言描述生成和GIS可视化。基于视觉语言大模型强大的推理和跨模态生成能力，并结合RAG与Agent技术，系统可精准识别入侵、打斗、火灾等异常事件，生成包含行为类型、位置、逻辑依据的自然语言描述。实现检测结果可解释性的提升。同时基于视觉语言大模型强大的泛化能力，本系统可以实现免训练地适应新的检测场景和任务。

同时项目在Django框架下使用MySQL数据库来实现用户管理、视频存储、异常片段检索等功能。数据库设计涵盖用户表、监控表、源视频表及异常片段表，通过外键约束与级联删除确保数据完整性，支持多条件查询与可视化展示。在此基础之上，我们使用GIS地图技术，在监控设备的实际地图上实现对异常事件的精确标注，从而实现检测结果的高可视化展示。

## **命名规则**

**(1)文件名**

文件名应采用有意义的英文单词或缩写组合，避免使用无意义的字符或中文。如果文件名由多个单词组成，使用下划线（\_）进行分隔，以增强可读性。这样的命名方式能够清晰地表达文件的用途或内容，方便开发者在维护和扩展项目时快速定位和识别文件。同时，文件名应尽量简洁明了，避免过长的文件名导致在某些系统或环境中出现显示问题。对于与特定功能或模块相关的文件，文件名应能够反映出这种关联，如webuicore/forms.py表明该文件与 Web UI 核心的表单功能有关。

**(2)类名**

类名采用大驼峰命名法，即每个单词的首字母大写，其余字母小写。这种命名方式能够清晰地表示类的名称，并且与其他命名类型（如函数名、变量名）区分开来。类名应该能够准确描述该类的功能或用途，使开发者在阅读代码时能够快速理解类的作用。同时，类名应具有一定的抽象性，避免过于具体或与实现细节相关的命名，以便在不同的场景下复用和扩展。

**(3)变量名（public类型除外）**

非公共类型的变量名采用小驼峰命名法，即第一个单词的首字母小写，后续单词的首字母大写。这种命名方式能够使变量名具有一定的可读性，同时与公共类型的变量名区分开来。变量名应具有描述性，能够清晰地表达变量的用途或含义。避免使用过于简短或无意义的变量名，如a、b等，除非是在循环控制等小范围内使用。

**(4)变量名（public类型）**

公共类型的变量名也采用小驼峰命名法，但在命名时应更加注重其通用性和可理解性。公共变量通常会在多个模块或类中被访问和使用，因此其命名应该能够准确反映其代表的含义。例如，在项目中如果有公共的配置变量，如server\_url，采用小驼峰命名法能够使其他开发者快速理解该变量的用途。同时，公共变量名应避免使用过于具体或与实现细节相关的命名，以便在不同的环境和场景下使用。

**(5)小范围的计数器（循环控制等用）**

小范围的计数器通常使用简短的单字母或缩写来命名，如i、j、k等。这种命名方式在循环控制等小范围内使用时，能够简洁明了地表示计数器的作用，并且不会与其他变量名产生冲突。例如，在for循环中使用i作为计数器：for(let i = 0; i < filesData.length; i++)。使用简短的计数器名称可以提高代码的简洁性和可读性，同时也符合编程的习惯。

**(6)创建一个类的对象的方法**

创建一个类的对象的方法名采用New\_()或者Create\_()的规则。这种命名方式能够清晰地表明该方法的作用是创建一个新的对象。例如，如果有一个User类，可以定义一个 Create\_User()方法来创建User类的对象。使用这种命名规则可以使代码更加易于理解和维护，开发者在阅读代码时能够快速识别出创建对象的方法。

**(7)对象类型转换的方法**

对象类型转换的方法名采用To\_()的规则。这种命名方式能够明确地表示该方法的作用是将一个对象转换为另一种类型。例如，如果有一个String类型的对象，需要将其转换为Integer类型，可以定义一个To\_Integer()方法来实现这个转换。使用To\_()命名规则可以使代码更加清晰，开发者能够快速理解方法的功能。

**(8)返回值为Boolean类型的方法**

返回值为Boolean类型的方法名采用Is+形容词、Can+动词、Has+过去分词、动词+名词的规则。例如，bool IsEmpty()表示判断某个对象是否为空，bool CanGet()表示是否能够获取某个资源，bool HasChanged表示某个对象是否已经发生了变化，bool ContainsKey()表示某个对象是否包含某个键。这种命名方式能够直观地表达方法的返回值含义，使开发者在调用方法时能够快速理解其作用。

**(9)bool型的变量**

bool型的变量名应具有描述性，能够清晰地表达变量所代表的布尔状态。可以采用与返回值为Boolean类型的方法类似的命名规则，如isValid、canAccess、hasUpdated等。这样的命名方式能够使代码更加易于理解，开发者在阅读代码时能够快速识别出变量的含义。

**(10)其他**

在项目中，对于常量的命名，通常采用全大写字母，单词之间用下划线（\_）分隔的方式。例如，MAX\_LENGTH、DEFAULT\_VALUE等。这种命名方式能够清晰地表示常量的含义，并且与变量名区分开来。对于配置文件中的参数名，也应采用有意义的英文单词或缩写组合，使用下划线（）进行分隔，以提高可读性。同时，在编写代码时，应尽量避免使用全局变量，除非有必要，以减少代码的耦合度和复杂性。对于注释，应简洁明了，能够准确解释代码的功能和用途，帮助其他开发者理解代码。

## **相关文档**

**doc/01-Plan.docx：**该文档包含了项目的整体规划，包括项目背景、项目目标、时间安排、资源分配等内容，为项目的启动和执行提供了宏观的指导框架。

**doc/02\_FDesign.docx：**该文档是项目整体的需求与功能设计规格书，详细描述了项目需要实现的各项功能及其具体要求。

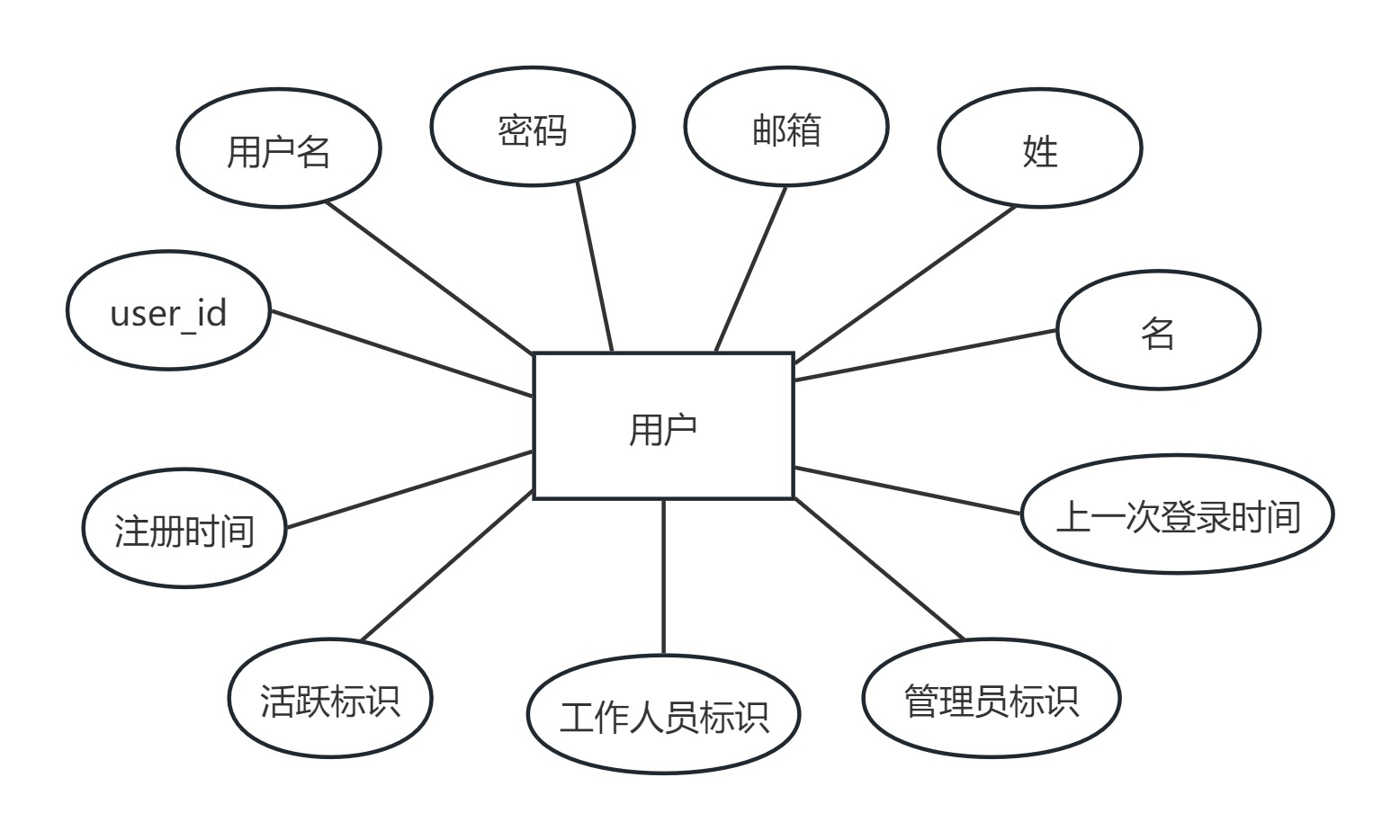
# 数据设计

## **数据库结构**

系统采用MySQL数据库，并基于Django ORM进行建模，实现数据表的自动迁移与管理。数据库共设计四张核心数据表，分别为：

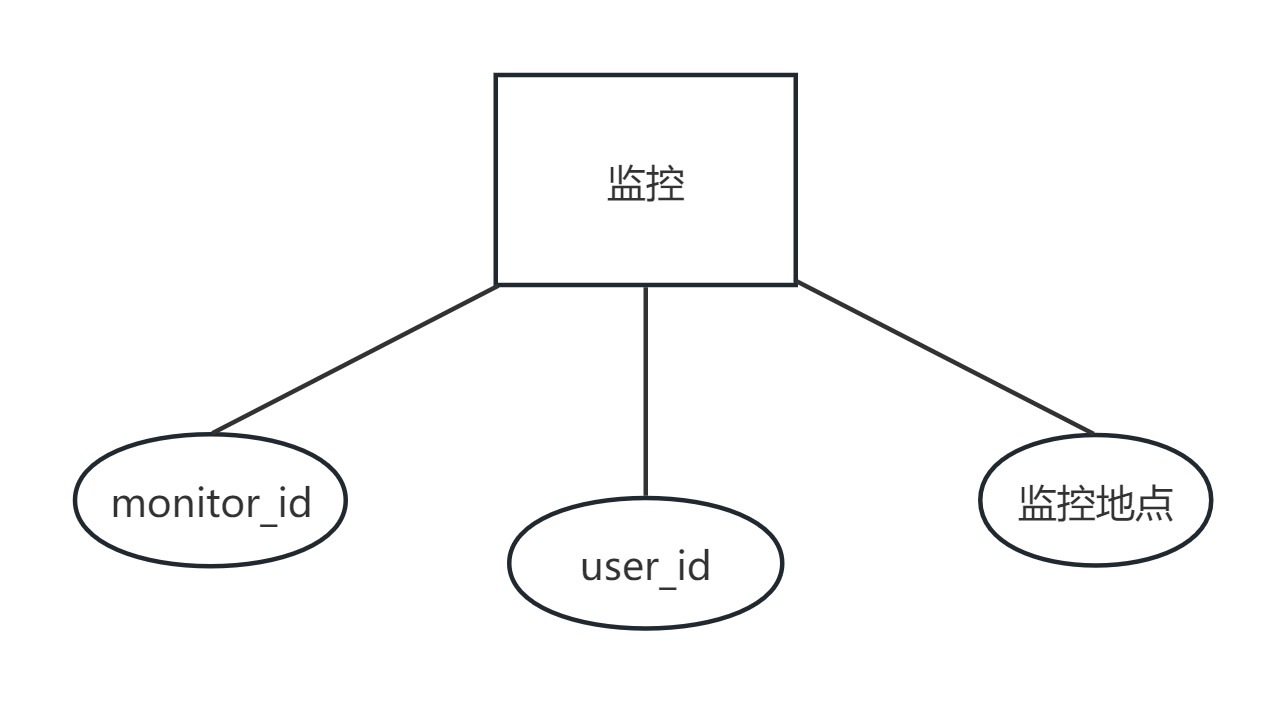
**(1)用户表（User）**

用于存储系统用户的基本信息，如用户名、密码、权限、注册时间等等。该表继承Django的AbstractUser模型，字段包括user\_id（主键）、username、password、email、is\_superuser等。



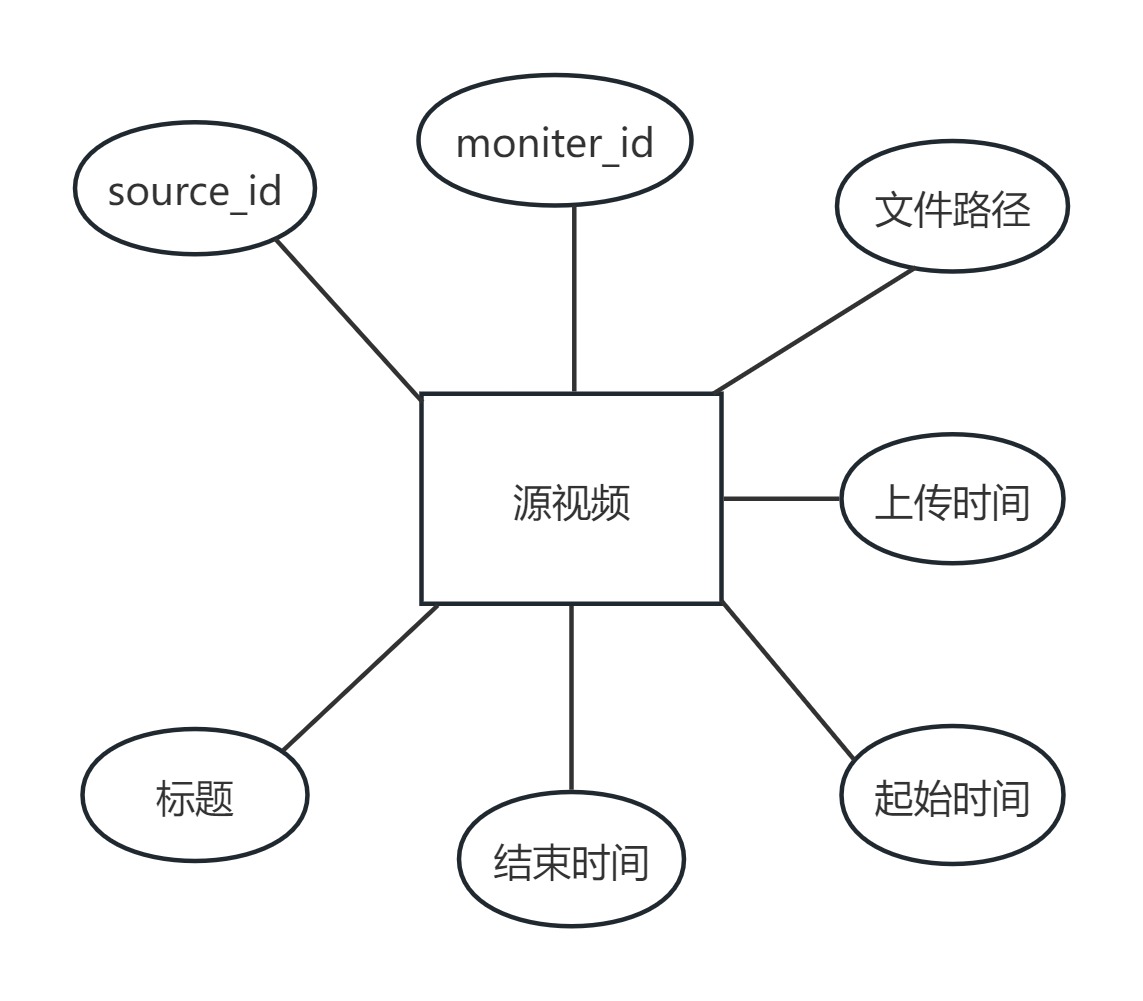
**(2)监控表（Monitor）**

记录各监控设备的信息及其所属用户。字段包括monitor\_id（主键）、user\_id（外键，关联User）、location。



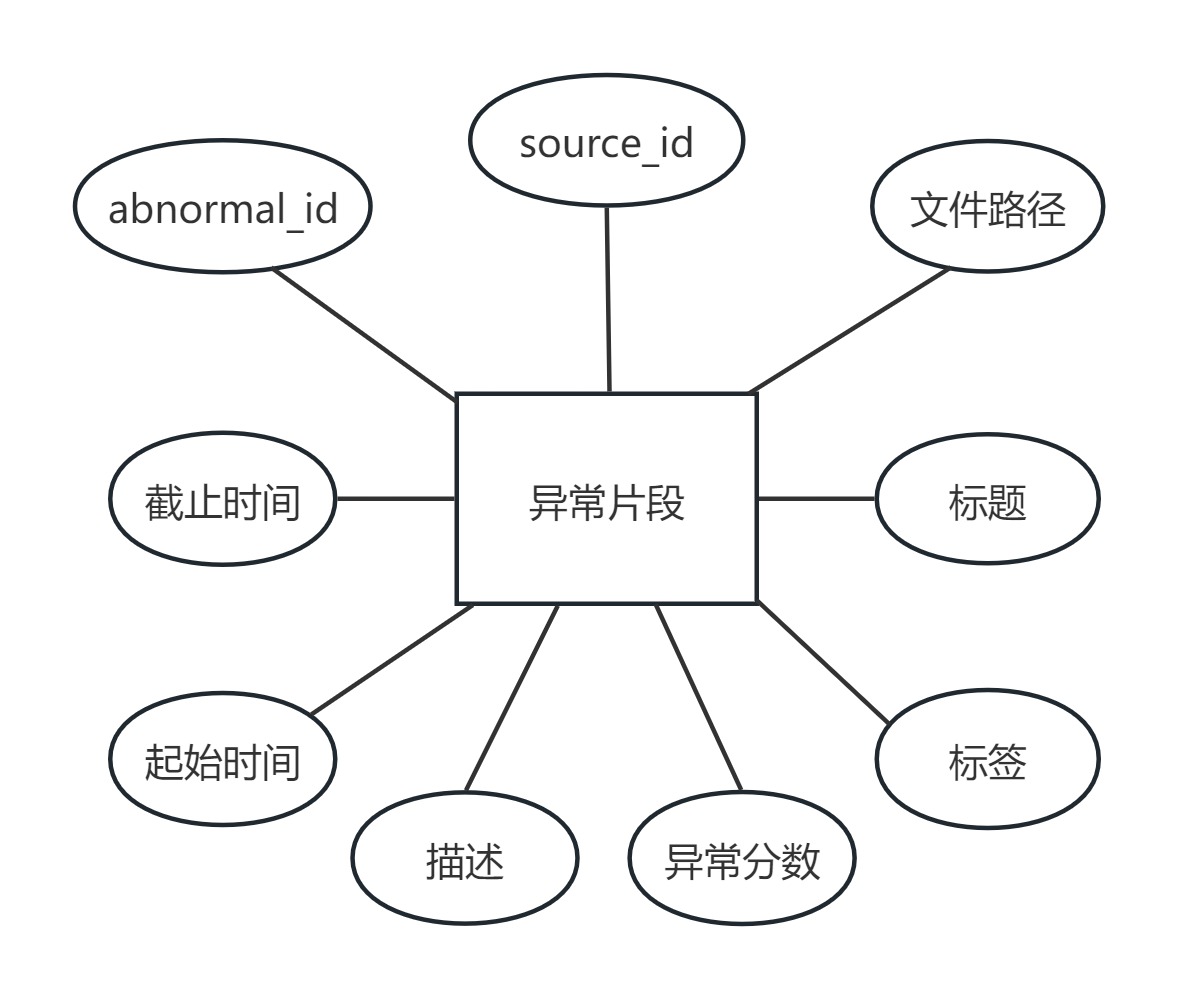
**(3)源视频表（SourceVideo）**

保存用户上传的原始监控视频的元数据，字段包括source\_id（主键）、monitor\_id（外键，关联Monitor）、title、save\_path、upload\_time、start\_time、end\_time。



**(4)异常片段表（AbnormalClip）**

存储从原始视频中分析出的异常片段及其标签、描述、评分等信息。字段包括abnormal\_id（主键）、source\_id（外键，关联SourceVideo）、save\_path、title、label、caption、score、start\_time、end\_time。



上述数据库结构通过外键实现层级关联，确保每个异常片段可溯源至对应的视频及监控设备，最终归属于具体用户，构成数据的完整闭环。

## **外部文件结构**

系统中视频数据与异常片段视频不直接存储于数据库，而是以文件形式存储在服务器的文件系统中：

**(1)原始视频文件**

通过用户上传功能保存至指定目录，数据库中仅记录其存储路径（save\_path字段）。

**(2)异常片段文件**

由异常检测模块生成并保存为独立的视频文件，同样仅记录其路径至数据库。

该设计减轻了数据库存储压力，便于视频文件的独立管理和大容量扩展。

## **外部文件结构**

后端采用Django的数据模型（models）定义数据结构，前端页面通过表单与视图层进行数据交互，核心数据结构包括：

**(1)用户结构体（User）**

封装用户名、密码、权限标识、注册时间等。

**(2)视频结构体（SourceVideo）**

包含标题、时间戳、路径、关联监控ID。

**(3)异常片段结构体（AbnormalClip）**

包括异常标签、描述、时间范围、分数、路径等字段，供前端展示与用户筛查。

前后端之间通过JSON进行数据传输，所有结构体字段均具备明确的字段类型与取值范围。

## **外部文件结构**

多个对象之间通过ID字段进行交叉引用，以构建完整的数据逻辑网：

**(1)User.id↔Monitor.user\_id**

用户与监控设备的管理关系；

**(2)Monitor.monitor\_id↔SourceVideo.monitor\_id**

监控设备与上传视频之间的物理关联；

**(3)SourceVideo.source\_id ↔ AbnormalClip.source\_id**

源视频与检测出的异常片段的内容衍生关系；

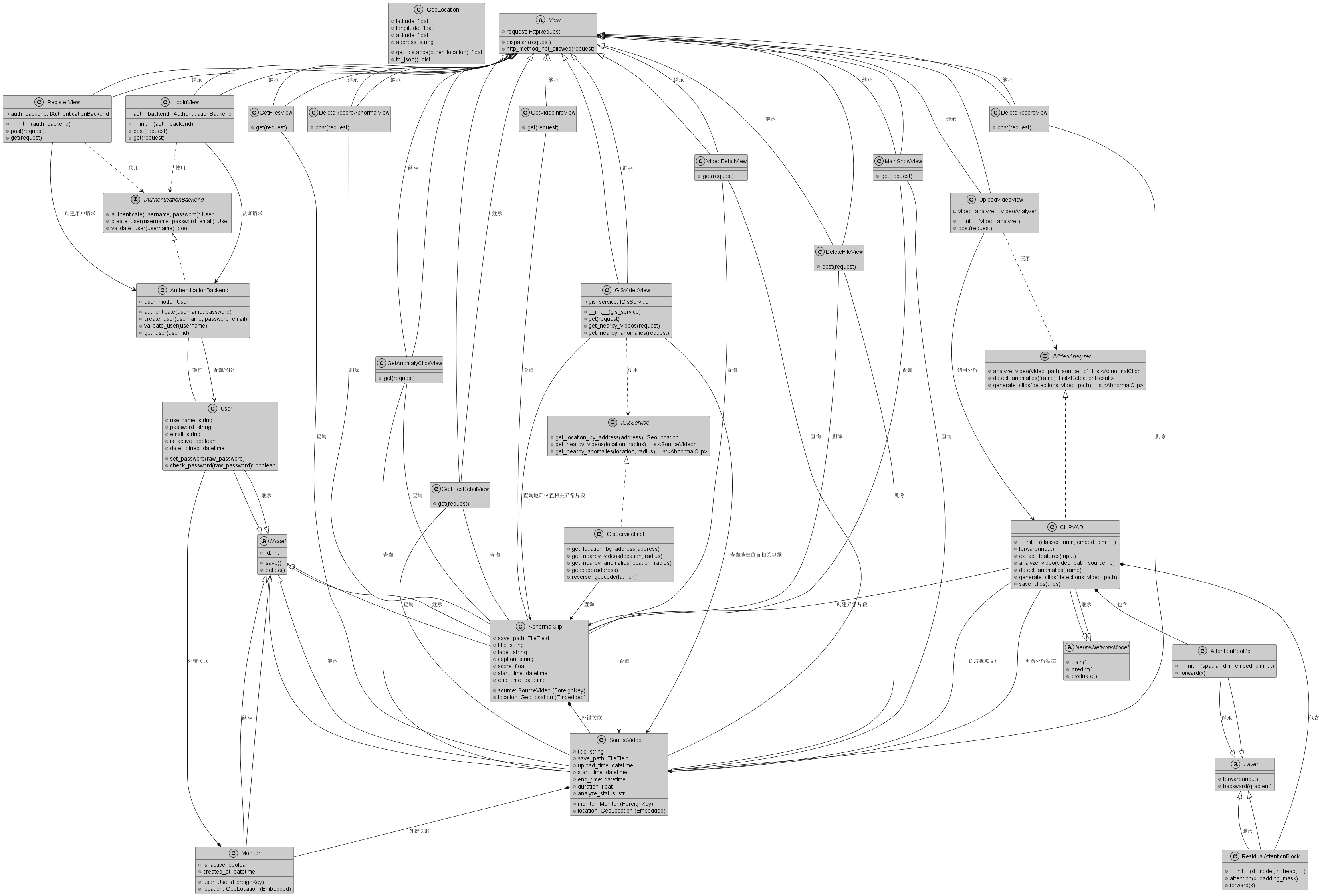
**(4)save\_path字段**

贯穿SourceVideo和AbnormalClip，分别对应原视频和异常片段的存储位置，实现视频文件的实际访问。

这种基于主外键加路径引用的设计，兼顾了数据的逻辑一致性与文件系统的高效扩展能力。

# 程序结构

## **程序整体结构**



## **模块一览表**

|  |  |  |  |
| --- | --- | --- | --- |
| 源程序文件名 | **类名** | **功能概要** | **使用语言** |
| authentication/  backends.py | AuthenticationBackend | 实现用户认证和注册的业务逻辑，作为用户数据的唯一访问入口 | Python |
| videos/models.py | User | 表示系统用户，存储用户基本信息和认证相关字段 | Python |
| videos/models.py | Monitor | 表示监控设备，关联到特定用户，记录设备位置和状态 | Python |
| videos/models.py | SourceVideo | 表示上传的源视频文件，关联到监控设备，记录视频元数据 | Python |
| videos/models.py | AbnormalClip | 表示从源视频中检测出的异常片段，关联到具体源视频 | Python |
| videos/analyzers/  clipvad.py | CLIPVAD | 基于深度学习的视频异常检测模型，实现视频分析和异常片段生成 | Python |
| videos/analyzers/  layers.py | AttentionPool2d | 神经网络中的注意力池化层，用于特征提取和降维 | Python |
| videos/analyzers/  layers.py | ResidualAttentionBlock | 神经网络中的残差注意力块，用于特征增强和信息保留 | Python |
| videos/views/  auth.py | LoginView | 处理用户登录请求，依赖认证后端完成身份验证 | Python |
| videos/views/  auth.py | RegisterView | 处理用户注册请求，依赖认证后端创建新用户 | Python |
| videos/views/  main.py | MainShowView | 显示系统主界面，提供视频列表和统计信息 | Python |
| videos/views/  upload.py | UploadVideoView | 处理视频上传请求，调用视频分析器进行异常检测 | Python |
| videos/views/  files.py | GetFilesView | 返回视频文件列表 | Python |
| videos/views/  files.py | GetFilesDetailView | 返回特定视频的详细信息 | Python |
| videos/views/  anomalies.py | GetAnomalyClipsView | 返回异常片段列表 | Python |
| videos/views/  delete.py | DeleteRecordView | 处理视频记录删除请求 | Python |
| videos/views/  delete.py | DeleteRecordAbnormalView | 处理异常片段记录删除请求 | Python |
| videos/views/  delete.py | DeleteFileView | 处理物理文件删除请求 | Python |
| videos/views/  detail.py | VideoDetailView | 返回视频详细信息和关联的异常片段 | Python |

|  |  |  |  |
| --- | --- | --- | --- |
| 源程序文件名 | **类名** | **功能概要** | **使用语言** |
| videos/views/info.py | GetVideoInfoView | 返回视频相关信息和分析结果 | Python |
| core/views.py | View | 所有视图类的抽象基类，定义 HTTP 请求处理的基本接口 | Python |
| core/models.py | Model | 所有数据库模型的抽象基类，定义基本的数据操作方法 | Python |
| authentication/  interfaces.py | IAuthenticationBackend | 定义用户认证后端的接口规范 | Python |
| videos/analyzers/  interfaces.py | IVideoAnalyzer | 定义视频分析器的接口规范 | Python |
| videos/analyzers/  base.py | NeuralNetworkModel | 所有神经网络模型的抽象基类 | Python |
| videos/analyzers/  base.py | Layer | 所有神经网络层的抽象基类 | Python |
| gis/models.py | GeoLocation | 表示地理位置信息，包含经纬度、海拔和地址等属性 | Python |
| gis/services.py | IGisService | 定义 GIS 服务的接口规范，提供地理编码、位置查询等功能 | Python |
| gis/services.py | GisServiceImpl | GIS 服务接口的具体实现，处理地理空间查询和分析 | Python |
| videos/views/gis.py | GISVideoView | 处理基于 GIS 的视频和异常片段查询请求，提供地图可视化接口 | Python |

# 逻辑设计

## **用户认证模块**

|  |  |
| --- | --- |
| **模块名**：用户认证模块 | **功能概要：**基于Django RBAC模型，实现用户注册、登录、权限控制及操作审计功能。支持OAuth2 第三方认证集成与多因子认证扩展。 |
| **输入/输出条件说明：**  1.输入条件  (1)用户名/密码（注册/登录时）  (2)认证令牌（OAuth 模式）  (3)权限配置参数（管理员设置）  2.输出条件  (1)登录状态令牌（JWT/SESSION）  (2)用户权限列表  (3)操作审计日志 | |
| **注释/约束/制限：**  1.密码存储需使用PBKDF2+HMAC算法  2.登录失败5次后锁定账户30分钟  3.管理员权限需二次验证（如短信验证码）  4.跨域认证需配置CORS中间件 | |
| **逻辑处理图：** | |

## **视频异常检测模块**

|  |  |
| --- | --- |
| **模块名**：视频异常检测模块 | **功能概要：**基于视觉语言大模型（Qwen2-VL）与CLIP特征提取技术，实现对视频流的逐帧分析，识别入侵、打斗、火灾等异常行为，生成异常片段及其自然语言描述。该模块支持免训练适应新场景，并通过RAG检索增强生成技术提升检测结果的可解释性。 |
| **输入/输出条件说明：**  1.源视频文件路径（MP4/AVI格式，单文件≤2GB）  2.检测模式参数（实时模式/批量模式）  3.异常事件类型字典（可配置，如“入侵”“火灾”“聚集”等） | |
| **注释/约束/制限：**  1.依赖NVIDIA GPU及CUDA环境（RTX 3060及以上）  2.单GPU同时处理视频流≤16路（1080P@25fps）  3.视频编码需为H.264/H.265，音频流需为AAC  4.模型首次运行需下载约 10GB 预训练权重 | |
| **逻辑处理图：** | |

## **GIS地图可视化模块**

|  |  |
| --- | --- |
| **模块名：**GIS地图可视化模块 | **功能概要：**基于Leaflet.js/Mapbox GL JS引擎，实现监控设备地理标注、异常事件空间可视化、热力图分析及路径规划功能。支持图层管理、空间查询与设备信息联动展示。 |
| **输入/输出条件说明：**  1.输入条件  (1)监控设备坐标数据（经纬度，WGS84坐标系）  (2)异常事件位置及时间范围  (3)用户地图操作指令（缩放、拖拽、图层切换）  2.输出条件  (1)地图可视化界面（含设备标注、异常点位）  (2)空间分析结果（热力图、路径规划路线）  (3)设备信息弹窗（含实时视频链接） | |
| **注释/约束/制限：**  1.需申请地图服务API密钥（如Mapbox Token）  2.大规模设备标注时需启用聚类算法（如Leaflet.markercluster）  3.空间分析依赖后端GeoDjango或PostGIS支持  4.移动端适配需单独开发触控交互逻辑 | |
| **逻辑处理图：** | |

# 接口设计

已在02-FDesign.docx中有了详细说明，这里没有特别补充，略。

# 其他事项

没有。

保护页