1. 系统概述

本文档描述了ReID多视频处理系统的后端集成方法。该系统支持行人检测、跟踪和重识别功能，可以同时处理多个视频源（本地视频文件或RTSP流），并提供行为分析（进店/出店/过店）等功能。

2. 核心类及对接方法

2.1 ReIDTracker 类

`ReIDTracker` 是系统核心类，负责单个视频流的处理。

初始化方法

```
tracker = ReIDTracker(log_system=None)
```

- `log_system`： 可选参数，日志系统实例，不提供则会创建新实例

关键方法

1. 设置处理环境

```
success = tracker.setup_processing(video_path="path/to/video.mp4", tempDataPath="config.json")
```

- `video_path`： 视频文件路径或RTSP URL
- `tempDataPath`： 包含ROI和边界信息的JSON配置文件路径
- 返回：设置是否成功

2. 设置视频写入器

```
success = tracker.setup_video_writer(video_path, output_dir="processed", suffix="_result")
```

- `video_path`： 源视频路径
- `output_dir`： 输出目录
- `suffix`： 输出文件名后缀
- 返回：设置是否成功

3. 处理单帧图像

```
output_frame, info = tracker.process_frame(frame=None, skip_frames=2, match_thresh=0.5, is_tracl
```

- `frame`： 要处理的帧，为None则从视频源读取
- `skip_frames`： 每隔多少帧处理一次
- `match_thresh`： ReID匹配阈值（0-1）
- `is_track`： 是否进行跟踪
- 返回：处理后的帧和包含处理信息的字典

4. 重新加载搜索引擎

```
tracker.re_load_search_engine()
```

当数据库中的特征发生变化时调用此方法刷新ReID特征库

5. 释放资源

```
tracker.release()
```

释放视频资源和窗口

6. 运行处理

```
tracker.run(video_path, tempDataPath="v1.json", skip_frames=2, match_thresh=0.5, is_track=True,
```

处理整个视频

2.2 StreamManager 类

`StreamManager` 管理多个视频流，提供配置管理、状态监控和重连机制。

初始化方法

```
manager = StreamManager(temp_dir=None, max_reconnect=10, mjpeg_server_port=8554)
```

- `temp_dir`： 临时文件目录
- `max_reconnect`： RTSP流最大重连次数
- `mjpeg_server_port`： MJPEG服务器端口

关键方法

1. 设置流

```
mjpeg_urls = manager.setup_streams(config_list, show_windows=True)
```

- `config_list`： 配置列表，每个元素包含URL和区域配置
- `show_windows`： 是否显示视频窗口
- 返回：原始URL和对应MJPEG URL列表的字典列表

2. 开始处理

```
success = manager.start_processing(skip_frames=4, match_thresh=0.15, is_track=True, save_video=I
```

开始处理所有设置的视频流

3. 停止处理

```
success = manager.stop_processing(timeout=5)
```

停止处理所有视频流

4. 获取流状态

```
info = manager.get_stream_info(index=None)
```

获取单个流或所有流的信息

5. 更新流状态

```
success = manager.update_stream_status(index, active=True, reconnect_increment=False)
```

更新流状态（活跃/不活跃）

2.3 多视频处理函数

处理多个视频

```
results = process_quad_videos(
    video_path1, video_path2, video_path3, video_path4,
    tempDataPath1="v1.json", tempDataPath2="v2.json", tempDataPath3="v3.json", tempDataPath4="v4
    skip_frames=2, match_thresh=0.15, is_track=True, save_video=True,
    stop_event=None, stream_manager=None, show_windows=[True, True, True, True]
)
```

- 处理1-4个视频（使用独立的ReIDTracker实例）
- 可以通过 `stop_event` 参数控制处理停止
- 通过 `show_windows` 参数控制是否显示窗口
- 返回包含各视频处理结果的列表

3. 数据库操作

系统使用SQLite数据库存储ReID特征。提供以下API：

```
# 初始化数据库
init_db(db_path)

# 添加特征
add_feature(db_path, db_name, feature, person_id, quality=0.0)

# 更新特征
update_feature(db_path, db_name, feature, person_id, quality=0.0)

# 删除特征
delete_feature(db_path, db_name, person_id)

# 加载特征
base_feat_lists, base_idx_lists = load_features_from_sqlite(db_path, db_name, dims=1280)

# 获取最大人员ID
max_id = get_max_person_id(db_path)

# 清空特征
clear_all_features(db_path)
```

4. 配置文件格式

配置文件使用JSON格式，包含以下信息：

```
{
  "points": [[x1, y1], [x2, y2], [x3, y3], [x4, y4]],  // ROI区域多边形顶点
  "b1": [[x1, y1], [x2, y2]],  // 进店线
  "b2": [[x1, y1], [x2, y2]],  // 边界线2
  "g2": [[x1, y1], [x2, y2]]   // 边界线3
}
```

其中b2和g2可以不用给出，算法会自动生成

5. 实际对接示例

5.1 单个视频处理

```
from reid import ReIDTracker

# 初始化跟踪器
tracker = ReIDTracker()

# 设置处理环境
tracker.setup_processing(video_path="camera1.mp4", tempDataPath="camera1_config.json")

# 设置视频写入
tracker.setup_video_writer("camera1.mp4", output_dir="results")

# 开始处理视频
tracker.run(video_path="camera1.mp4", save_video=True)

# 释放资源
tracker.release()
```

5.2 多视频处理

```python
from reid import StreamManager

# 创建流管理器
manager = StreamManager()

# 准备配置
config_list = [
    {
        "url": "rtsp://admin:pass@192.168.1.100/stream1",
        "points": [[100, 100], [500, 100], [500, 400], [100, 400]],
        "b1": [[100, 250], [500, 250]]
    },
    {
        "url": "path/to/video.mp4",
        "points": [[100, 100], [500, 100], [500, 400], [100, 400]],
        "b1": [[100, 250], [500, 250]]
    }
]

# 设置流
mjpeg_urls = manager.setup_streams(config_list)

# 开始处理
manager.start_processing(save_video=True)

# ... 应用运行中 ...

# 停止处理
manager.stop_processing()
```

5.3 处理结果获取

```python
# 通过StreamManager获取统计结果
stream_info = manager.get_stream_info()
for info in stream_info:
    print(f"Stream {info['url']} stats:")
    print(f"Enter count: {info['counts']['enter']}")
    print(f"Exit count: {info['counts']['exit']}")
    print(f"Pass count: {info['counts']['pass']}")
```

6. 注意事项

1. 首次使用需确保相关模型文件已正确配置
2. RTSP流处理可能需要处理网络连接中断问题
3. 高分辨率视频处理需要足够的GPU资源
4. ReID特征库随着识别的人数增加会影响匹配速度
5. 建议定期清理不活跃的特征数据

7. RTSP流使用实例

7.1 通过StreamManager处理RTSP流

```
from reid import StreamManager
import threading
import time

# 创建流管理器
stream_manager = StreamManager(max_reconnect=10, mjpeg_server_port=8554)

# 准备RTSP流配置
rtsp_configs = [
    {
        "url": "rtsp://admin:admin123@192.168.1.101:554/cam/realmonitor?channel=1&subtype=0",
        "points": [[200, 500], [600, 500], [600, 300], [200, 300]],
        "b1": [[200, 400], [600, 400]]
    },
    {
        "url": "rtsp://admin:admin123@192.168.1.102:554/cam/realmonitor?channel=1&subtype=0",
        "points": [[150, 450], [550, 450], [550, 250], [150, 250]],
        "b1": [[150, 350], [550, 350]]
    }
]

# 设置RTSP流
mjpeg_urls = stream_manager.setup_streams(rtsp_configs, show_windows=True)
print("MJPEG URLs for web display:", mjpeg_urls)

# 创建停止事件
stop_event = threading.Event()

# 启动MJPEG服务器（可选，用于Web显示）
stream_manager.start_mjpeg_server()

# 开始处理RTSP流
stream_manager.start_processing(skip_frames=2, match_thresh=0.15, save_video=False)

try:
    # 运行一段时间（例如10分钟）
    print("Processing RTSP streams. Press Ctrl+C to stop...")
    time.sleep(600)  # 运行10分钟
```

```
except KeyboardInterrupt:
    print("Stopping RTSP stream processing...")
finally:
    # 停止处理
    stream_manager.stop_processing()
    stream_manager.stop_mjpeg_server()

# 获取统计结果
results = stream_manager.get_stream_info()
for idx, info in enumerate(results):
    print(f"Camera {idx+1} stats:")
    print(f"  URL: {info['url']}")
    print(f"  Active: {info['active']}")
    print(f"  Enter count: {info['log_system'].get_business_count('enter')}")
    print(f"  Exit count: {info['log_system'].get_business_count('exit')}")
    print(f"  Pass count: {info['log_system'].get_business_count('pass')}")
    print(f"  Total unique people: {info['log_system'].get_total_unique_count()}")
```

7.2 处理RTSP断连情况

```
def rtsp_monitor_thread(stream_manager):
    """监控RTSP流状态并处理断连"""
    while not stop_event.is_set():
        # 获取所有流信息
        streams = stream_manager.get_stream_info()

        for idx, info in enumerate(streams):
            # 检查流是否活跃
            if not info['active'] and stream_manager.should_reconnect(idx):
                print(f"Camera {idx+1} disconnected, attempting to reconnect...")
                # 尝试重置流
                stream_manager.reset_reconnect_count(idx)

        # 每5秒检查一次
        time.sleep(5)

# 在主程序中启动监控线程
monitor_thread = threading.Thread(target=rtsp_monitor_thread, args=(stream_manager,))
monitor_thread.daemon = True
monitor_thread.start()
```

7.3 使用RTSP流进行分布式部署

```
from reid import StreamManager
import requests
import json
import time

# 定义中央服务器URL
central_server = "http://central-server-api.example.com/api/stats"

# 创建流管理器
stream_manager = StreamManager(mjpeg_server_port=8554)

# 加载配置文件中的RTSP流
def load_rtsp_config(config_file):
    with open(config_file, 'r') as f:
        return json.load(f)

# 加载RTSP配置
rtsp_configs = load_rtsp_config("rtsp_config.json")

# 设置RTSP流
stream_manager.setup_streams(rtsp_configs, show_windows=False)

# 启动MJPEG服务器以提供Web访问
stream_manager.start_mjpeg_server()

# 开始处理
stream_manager.start_processing(skip_frames=3, match_thresh=0.2)

# 定期将统计数据发送到中央服务器
def report_stats():
    while True:
        try:
            # 获取所有流的统计信息
            stats = []
            streams = stream_manager.get_stream_info()

            for idx, info in enumerate(streams):
                log_system = info['log_system']
                stats.append({
                    "camera_id": f"camera_{idx+1}",
                    "url": info["url"],
```

```
                    "timestamp": time.time(),
                    "enter_count": log_system.get_count("enter"),
                    "exit_count": log_system.get_count("exit"),
                    "pass_count": log_system.get_count("pass"),
                    "unique_count": log_system.get_unique_count(),
                    "active": info["active"]
                })

            # 发送到中央服务器
            requests.post(central_server, json={"stats": stats})
            print("Stats reported successfully")

        except Exception as e:
            print(f"Error reporting stats: {e}")

        # 每5分钟报告一次
        time.sleep(300)

# 启动报告线程
import threading
report_thread = threading.Thread(target=report_stats)
report_thread.daemon = True
report_thread.start()

# 主线程保持运行
try:
    while True:
        time.sleep(60)
except KeyboardInterrupt:
    print("Shutting down...")
    stream_manager.stop_processing()
    stream_manager.stop_mjpeg_server()
```

7.4 RTSP配置文件示例 (rtsp_config.json)

```
[
  {
    "url": "rtsp://admin:password@192.168.1.100:554/stream1",
    "points": [[100, 400], [500, 400], [500, 100], [100, 100]],
    "b1": [[100, 250], [500, 250]]
  },
  {
    "url": "rtsp://admin:password@192.168.1.101:554/stream1",
    "points": [[150, 450], [550, 450], [550, 150], [150, 150]],
    "b1": [[150, 300], [550, 300]]
  }
]
```

7.5 处理多个RTSP流的性能优化

```
from reid import StreamManager
import psutil
import time

# 创建流管理器
stream_manager = StreamManager()

# 加载RTSP配置
rtsp_configs = [...]  # 从配置文件或其他来源加载

# 设置性能监控
def monitor_performance():
    while True:
        cpu_percent = psutil.cpu_percent()
        memory_percent = psutil.virtual_memory().percent

        print(f"System Load - CPU: {cpu_percent}%, Memory: {memory_percent}%")

        # 如果系统负载过高，增加跳帧数
        if cpu_percent > 80:
            print("High CPU load detected, increasing frame skipping...")
            # 停止当前处理
            stream_manager.stop_processing()
            # 使用更高的skip_frames重新启动
            stream_manager.start_processing(skip_frames=8)

        time.sleep(30)  # 每30秒检查一次

# 启动性能监控
import threading
monitor_thread = threading.Thread(target=monitor_performance)
monitor_thread.daemon = True
monitor_thread.start()

# 设置RTSP流并开始处理
stream_manager.setup_streams(rtsp_configs)
stream_manager.start_processing(skip_frames=4)

# 主程序保持运行
try:
    while True:
```

```
            time.sleep(1)
    except KeyboardInterrupt:
        stream_manager.stop_processing()
```

输入待处理的流地址和配置信息：

```
[
   {
   "b1": 500, 600], [750, 600,
   "points": 500, 600], [750, 600], [750, 400], [500, 400,
   "rtsp_url":"rtsp://localhost:8554/live"
}, {
   "b1": 500, 600], [750, 600,
   "points": 500, 600], [750, 600], [750, 400], [500, 400,
   "rtsp_url":"rtsp://localhost:8554/live"
}

]
```

输出：

mjpeg流列表

```
[{
"rtsp_url":"rtsp://localhost:8554/live",
"mepeg":"http://localhost:8554/live",
},
{
"rtsp_url":"rtsp://localhost:8554/live",
"mepeg":"http://localhost:8554/live",
}
]
```

8. RTSP接入与MJPEG转换

8.1 RTSP接入流程

StreamManager 类提供了简便的RTSP流接入和管理功能，还支持将RTSP流转换为MJPEG流以便于Web端展示。以下是接入RTSP流并获取对应MJPEG流的详细步骤：

8.1.1 接入RTSP流的API

```
from reid import StreamManager

# 初始化流管理器，指定MJPEG服务器端口
stream_manager = StreamManager(mjpeg_server_port=8554)

# 定义RTSP流配置列表
rtsp_config_list = [
    {
        "rtsp_url": "rtsp://localhost:8554/live",
        "points": [[500, 600], [750, 600], [750, 400], [500, 400]],
        "b1": [[500, 600], [750, 600]]
    },
    {
        "rtsp_url": "rtsp://localhost:8554/live2",
        "points": [[500, 600], [750, 600], [750, 400], [500, 400]],
        "b1": [[500, 600], [750, 600]]
    }
]

# 启动MJPEG服务器（必须在setup_streams前调用）
stream_manager.start_mjpeg_server()

# 设置RTSP流，同时生成对应的MJPEG URL
mjpeg_urls = stream_manager.setup_streams(rtsp_config_list)

# 开始处理视频流
stream_manager.start_processing(skip_frames=2, save_video=False)

# 打印MJPEG URL列表
print(mjpeg_urls)
```

8.1.2 MJPEG URL返回格式

```
[
    {
        "rtsp_url": "rtsp://localhost:8554/live",
        "mjpeg": "http://localhost:8554/live"
    },
    {
        "rtsp_url": "rtsp://localhost:8554/live2",
        "mjpeg": "http://localhost:8554/live2"
    }
]
```

8.2 StreamManager对RTSP进行转换的实现

StreamManager类内部实现了RTSP到MJPEG的转换功能，主要通过以下几个方法：

```
def generate_mjpeg_url(self, url: str, index: int) -> str:
    """
    根据原始URL生成MJPEG URL

    Args:
        url: 原始RTSP URL
        index: 流索引

    Returns:
        对应的MJPEG URL
    """
    # 从URL中提取路径部分作为MJPEG流名称
    parsed = urlparse(url)
    path = parsed.path.strip('/')

    if not path:
        path = f"stream_{index}"

    # 返回MJPEG服务器URL
    return f"http://localhost:{self.mjpeg_server_port}/{path}"

def start_mjpeg_server(self) -> bool:
    """启动MJPEG服务器"""
    try:
        # 实例化MJPEG服务器
        self.mjpeg_server = MJPEGServer(port=self.mjpeg_server_port)
        self.mjpeg_server.start()
        return True
    except Exception as e:
        print(f"Error starting MJPEG server: {e}")
        return False
```

8.3 完整接入示例

以下是一个完整的接入示例，包括错误处理和状态监控：

```python
from reid import StreamManager
import json
import time
import threading

# 读取RTSP配置
def read_rtsp_config(config_file):
    with open(config_file, 'r') as f:
        return json.load(f)

# 从配置文件加载RTSP流信息
rtsp_configs = read_rtsp_config("rtsp_config.json")

# 创建流管理器
stream_manager = StreamManager(mjpeg_server_port=8554, max_reconnect=5)

# 启动MJPEG服务器
if not stream_manager.start_mjpeg_server():
    print("Failed to start MJPEG server, exiting...")
    exit(1)

# 设置RTSP流，获取对应的MJPEG URLs
try:
    mjpeg_urls = stream_manager.setup_streams(rtsp_configs)
    # 将MJPEG URLs保存到文件，供前端使用
    with open('mjpeg_urls.json', 'w') as f:
        json.dump(mjpeg_urls, f, indent=2)
    print(f"MJPEG URLs saved to mjpeg_urls.json")
except Exception as e:
    print(f"Error setting up streams: {e}")
    stream_manager.stop_mjpeg_server()
    exit(1)

# 启动流处理
if not stream_manager.start_processing(skip_frames=2, match_thresh=0.2):
    print("Failed to start processing, exiting...")
    stream_manager.stop_mjpeg_server()
    exit(1)

# 监控RTSP流状态的线程
def monitor_streams():
    while True:
        # 获取所有流信息
```

```
        streams = stream_manager.get_stream_info()

        for idx, info in enumerate(streams):
            # 检查流是否活跃
            if not info['active']:
                print(f"Stream {idx} ({info['url']}) is inactive")
                if stream_manager.should_reconnect(idx):
                    print(f"Attempting to reconnect stream {idx}...")
                    stream_manager.reset_reconnect_count(idx)

        # 每10秒检查一次
        time.sleep(10)

# 启动监控线程
monitor_thread = threading.Thread(target=monitor_streams)
monitor_thread.daemon = True
monitor_thread.start()

try:
    # 主线程保持运行
    while True:
        # 每分钟获取一次统计数据
        stats = {}
        streams = stream_manager.get_stream_info()

        for idx, info in enumerate(streams):
            log_system = info['log_system']
            stats[f"camera_{idx}"] = {
                "url": info['url'],
                "active": info['active'],
                "enter_count": log_system.get_count("enter"),
                "exit_count": log_system.get_count("exit"),
                "pass_count": log_system.get_count("pass")
            }

        # 保存统计数据到文件
        with open('stream_stats.json', 'w') as f:
            json.dump(stats, f, indent=2)

        time.sleep(60)

except KeyboardInterrupt:
    print("Stopping application...")
```

```
    stream_manager.stop_processing()
    stream_manager.stop_mjpeg_server()
```

8.4 RTSP配置文件格式 (rtsp_config.json)

```
[
    {
        "rtsp_url": "rtsp://localhost:8554/live",
        "points": [[500, 600], [750, 600], [750, 400], [500, 400]],
        "b1": [[500, 600], [750, 600]]
    },
    {
        "rtsp_url": "rtsp://admin:password@192.168.1.100:554/stream",
        "points": [[100, 500], [600, 500], [600, 100], [100, 100]],
        "b1": [[100, 300], [600, 300]]

    }
]
```

8.5 注意事项

1. 请确保RTSP URL格式正确，包含必要的认证信息
2. MJPEG服务器端口需要未被占用
3. 转换后的MJPEG流可能会降低图像质量，但提高了兼容性
4. 处理多个高清RTSP流需要较高的系统资源
5. 建议定期检查RTSP连接状态，实现自动重连机制
6. 对于公网环境，建议对MJPEG流添加安全认证

通过以上方法，您可以轻松接入RTSP流并获得对应的MJPEG流URLs，便于在Web端展示视频内容并进行行人检测与跟踪分析。