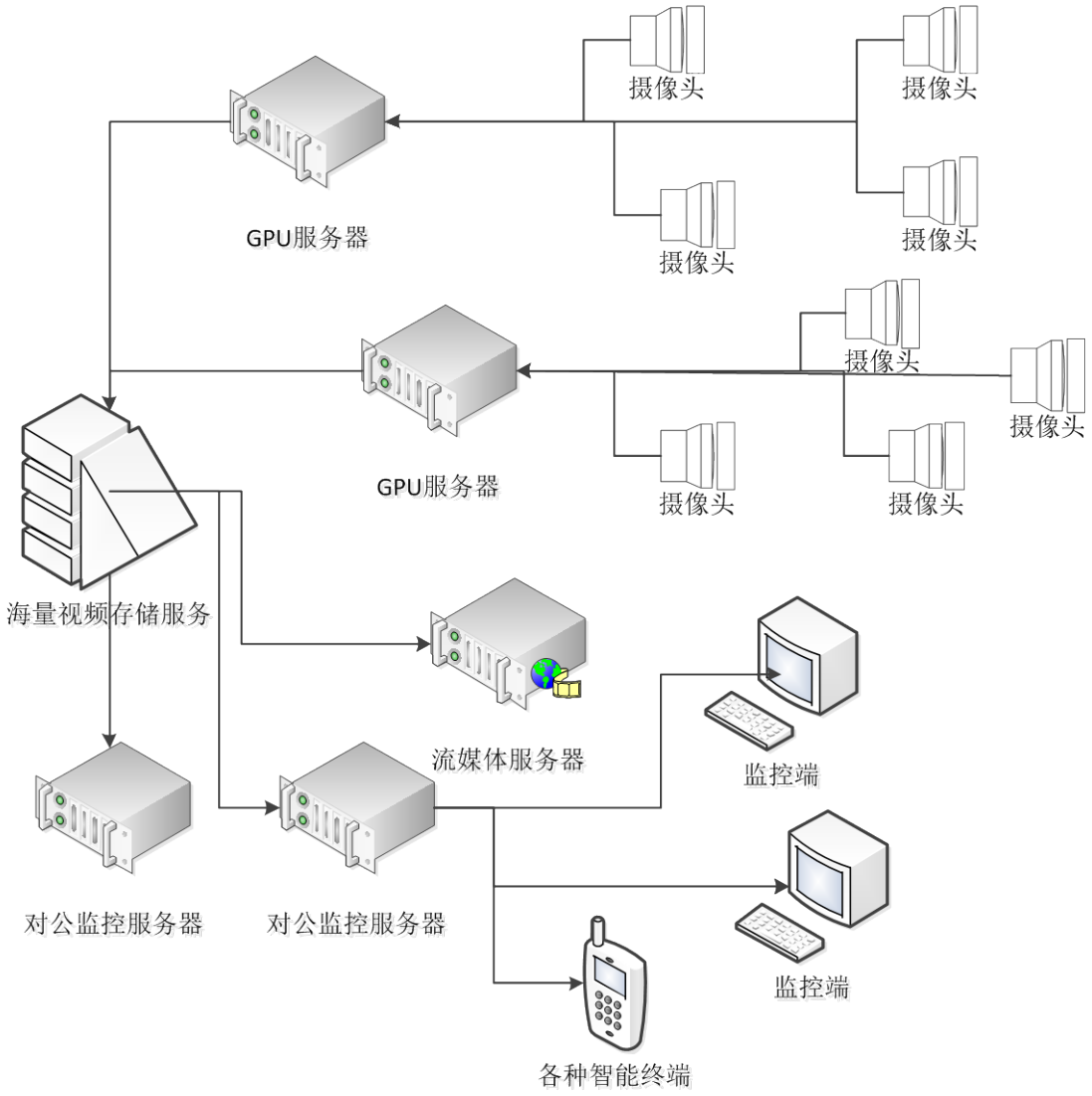
ZAI 监控系统使用指南

目录

系统架构

- ZAI 监控系统可以做到万物识别，同时它也具有准监控系统的功能
- ZAI 监控系统极易二次开发，最初设计以系统集成建设为主
- GPU 服务器可以驱动大规模摄像头，即时 1000 个 1080p 都可以带起来

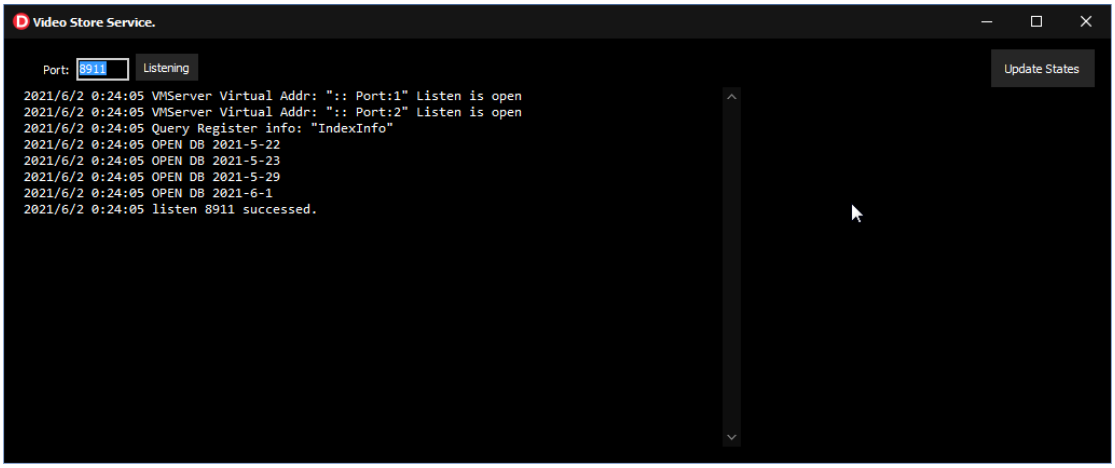


视频数据存储服务 VideoStoreService

简单介绍

- 存储服务可以支持 ARM Linux/Windows/IA32 Linux/AMD64 Linux 等等后台
- 存储服务二次开发和移植非常简单，凡是能运行 ZServer4D 的系统都能跑存储服务
- 运行存储服务不需要 GPU，普通阵列服务器即可，甚至可以使用低配 ddr3
- 存储服务按日分库，每日一个独立存储文件，该文件大小由接入的摄像头规模而定
- 存储服务的视频数据都是 h264 分片视频形式，尺度由参数控制
- 每一个视频分片包含：原始视频，AI 渲染视频，AI 识别结果，时间
- 存储服务的通讯协议使用 p2pVM 构建
- 存储服务不能直接对公，系统集成时需要关闭公网权限

存储服务在 Windows 下就是 UI 壳，无需任何配置脚本，打开 VideoStoreService.exe 即可
VideoStoreService 打开以后，会自动创建一个 VideoDatabase 子目录，所有的数据使用 ZDB1.0 存储，也就是后缀为.OX 的文件



外围功能不实现，交给维护环节去干：**存储服务不支持滚动存储，因此集成时，需要编写一些维护脚本，例如半夜监控可以不用全开，存储服务器定时重启，清理过去的监控数据等等。如果嫌麻烦，也可以根据目标场景通过二次开发做自动化维护**。
提示：存储服务器在构建时需要打开编译选项：**ZDB_PHYSICAL_FLUSH**，防止断电或则强制终止造成 ZDB1.0 的数据链条损坏。强制断电或则关闭程序，只会损失几个视频片段，不会发生数据丢失灾害。

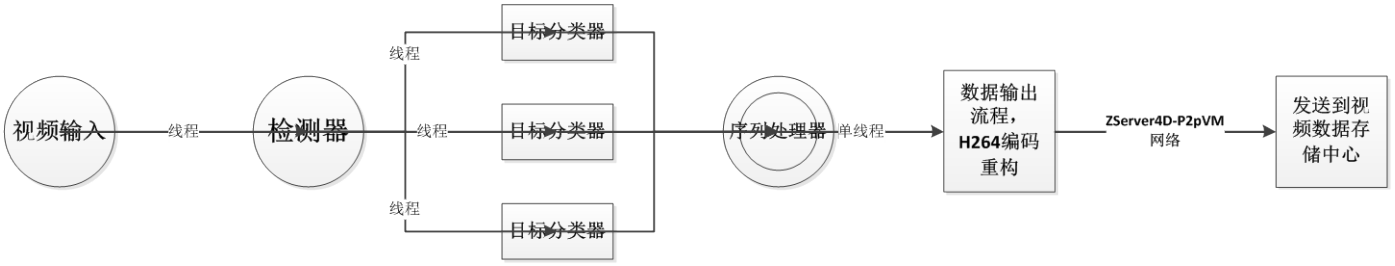
建议硬件配置：**直接买阵列服务器**，存储服务器不会对视频做任何计算，内存可以是 ddr3 体系，建议不低于 32G，磁盘 hdd 阵列插满就行，CPU 建议最低 2 核。云服器也是可以的。
这种阵列服务器配中高档也很便宜，主要看阵列 hdd 硬盘的数量。ups 可以选配。

分布式 GPU 视频处理服务器

简单介绍

- 视频处理服务同时充当了实时识别和监控转码服务，可在本地统一监控标准，可兼容多种网络摄像头
- 按准 HPC 构建，支持 1-16 张 GPU 的服务器/工作站，GPU 平均使用率可达 80%
- 支持分布式，需要实时识别摄像头太多时，直接堆 GPU 服务器
- 支持云服务器集成，可有效部署在阿里、腾讯，华为云中跑 AI 识别
- 支持实时推流，支持实时渲染（可视化 AI 识别渲染），支持自定义推流输出
- 监控处理大流程使用脚本控制，内置 2 种 GPU 性能模式，分别是多路 rome/xeon 模式，多 GPU 模式
- 撰写文本时已无重启稳定运行 30 天，监控端由于各种问题（wifi 断线，设备发烫，桥服务器宕机，摄像头和路由器每日定时重启），平均每个摄像头每天会重启 30 次，有时候某个摄像头会断网好几个小时候，当接上时，GPU 服务器会自动恢复工作
- 支持多种检测器和分类器模型，理论上 ZAI 支持的所有模型都能用
- GPU 视频处理服务器的识别内核是并入到 ZAI 核心库中的，未来它会持续被维护和升级
- 可视化性能调校，这是使用 UI 界面，根据 GPU/CPU 的处理能力实时监控硬件，然后使用脚本把不同的摄像头分配给不同的 GPU 处理，无需编程。

GPU 视频数据流程



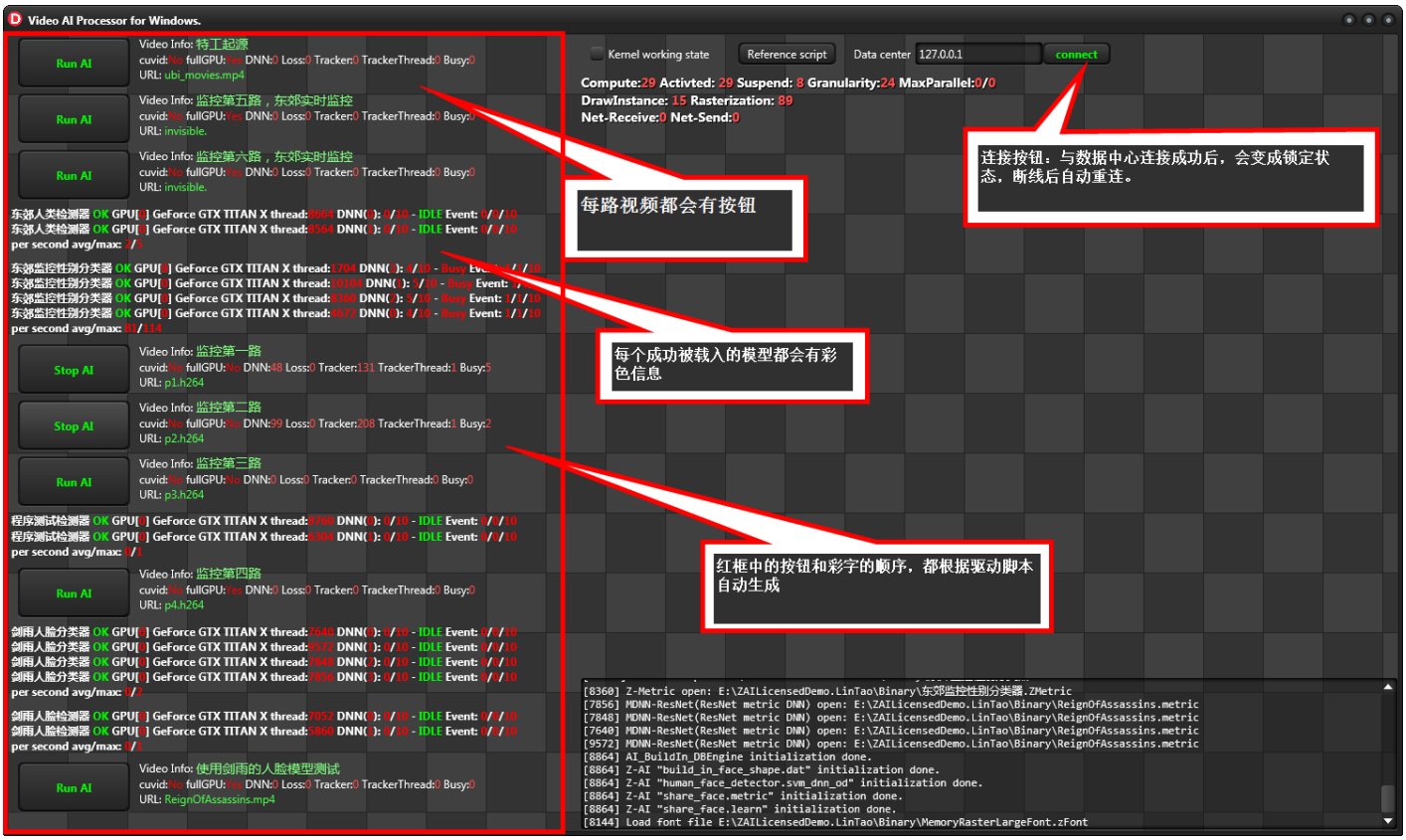
说明

- 视频输入为多路模式，具体承载规模根据 GPU 服务器配置而定，合理配置下，一张 GPU 可带动带 4-10 路，加摄像头就插 GPU，当 GPU 插满就买新服务器
- 视频输入支持：RTSP/RTMP/RTP/USB-Camera/File
- 检测器为全局 DNN-Thread 模式，既一个 GPU 会绑定 n 个物理线程，例如 4 张 GPU，每个 GPU 绑定 2 个线程，那么视频输入时会从 8 个线程中找出合理负载输入
- 目标分类器程序模型与检测器相同，都是 DNN-Thread，工作流程为，检测器完成检测，得到输出数据，再将数据以合理负载方式输入到检测器，最后得到识别结果
- 数据输出流程是将一个时间段的视频以 h264 打包，例如 15 秒片段，然后给片段标注时间并发送至存储服务器

GPU 视频处理服务器的界面

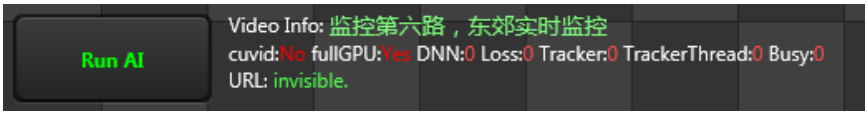
GPU 视频处理必须要界面，否则无法了解程序的运行状态，这会导致出现性能瓶颈，脚本写错误很难维护：只能靠猜。另外，在部署多 GPU 服务器时会需要看到 GPU 工作状态，这样可以挖掘更多的 GPU 潜力，否则 50%的 GPU 利用率等同于浪费投资。

提示：对于云服务器运营商，nvidia 对于数据中心有加密处理的，并不是所有的 GPU 都能被数据中心驾驭，一般来说，应该选择阿里/腾讯这种大公司提供的 GPU 服务器。小公司会时常遇到驱动安装不到位，nvsmi/ZAI 这种工具无法识别这种问题。

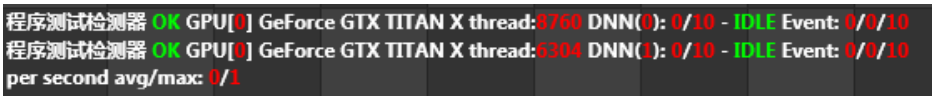


界面中的红字解读

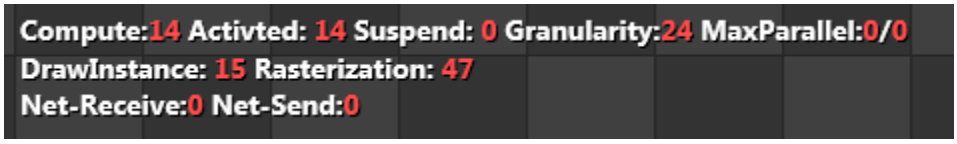
读懂红字就是看懂 GPU 的工作状态，对于性能优化，AI 计算策略这些工作来说是不可无视的必须工作



- cuvid：脚本如果打开了 cuvid，这里会是 yes，视频解码将会使用 gpu，由于 nv 处于策略考虑，gpu 都限制了解码规模，一般的 20xx/tesla，单卡只能支持 1-2 路视频同时 gpu 解码。另外，解码器内存传递会吃掉不少带宽，在高负载环境下，gpu 解码并不好用。笔者在这里都是使用 cpu 解码，现在 xeon 8280 也在降价，对于 cpu 解码，这是非常不错的。
- fullGPU：这里如果为 yes，程序会使用纯 GPU 工作模式，数据队列不会优化处理，全部扔给 gpu 计算，如果 No，程序会在 gpu 忙碌时使用相关性方法（无人机追踪，定位追踪）优化的来处理衔接视频帧。
- DNN：已经处理完成的视频帧数
- Loss：当 GPU 达到满负荷数据会处理不过来，这时候就会使用 loss 机制，这里的值表示 loss 的数量，在调试性能时，loss 应该是我们重点关注的数值
- Tracker：当 fullGPU 为 No，这里为正在排队准备进行相关性处理的数据长度，如果 fullGPU 为 Yes，那么 Tracker 就会为 0
- TrackerThread：处于规避高清视频性能先进，处理程序并不会直接使用并行，即使给 200 核的 cpu，并行也不够用，因此使用帧切片的方法来处理相关性计算，这里的线程是通过脚本配置的
- Busy：表示正处于队列中等待处理的实时视频帧数



- GPU[0]：这里的红 0 表示 GPU 的硬件 ID，该 ID 可以通过 GPU-Z，NVSMI 等等工具获得
- 8760：这个红字是系统线程 id
- DNN(0)：这里的 0 表示线程池 ID，该 ID 在 DNNThread 初始化时生成
- 0/10：这里的数字表示输入状态，0 表示当前 GPU 的输入队列状态，10 表示最大支持的输入队列，例如我们输入的图片有 20 张，但是最大队列只能到 10 张，这时候会发生卡线程动作，等待队列前面的图片处理完成。
- Event:5/100/10：这里的数值表示执行完识别后的并发输出状态，也是 cpu 设备执行线程的状态。5 表示当前正在并发执行的事件有 5 个，100 表示并行输出事件的队列长度，当输出事件达上限 10，系统的输出事件就会开始排队，这里的 100 就是当前正在排队的长度。GPU 处理数据非常快，系统会出现 cpu 跟不上 gpu 的情况，这时候，就会往输出事件队列里面堆积数据。



- Compute：线程总数 Activted：正在运行的线程 Suspend：挂起的线程
- Granularity：并行粒度，就是一个 for 最大可以有多少线程数量来处理
- MaxParallel(0/0)：第一个 0 表示当前的并行 for 数量，第二个 0 表示最大并行数量，如果第二个数值为 0，表示不限制并行数量。
- DrawInstance：15，DrawEngine 的实例数量有 15 个
- Rasterization：47，在内存创建的光栅有 47 个
- Net-Receive：0，网络数据的接收状态
- Net-Send：0，网络数据的发送状态

GPU 服务器脚本配置指南

脚本文件名必须固定为 **VideoAIProcessor.ini** 并且要与 **VideoAIProcessor.exe** 放在同一目录
监控 AI 处理器的设计思路，**视频采集->检测器模型->分类器模型**
我建议先准备好需要的模型，包括检测器，分类器，规范下文件命名，然后再编写脚本大纲，最后再去做细节和调试

- 检测器，只有两种，MMOD6L/MMOD3L
- 分类器，只有两种，ZMetric/Metric
- 视频输入，有两种设备模式，侧重于 rome/xeon 设备和侧重于 GPU 设备

监控脚本演示

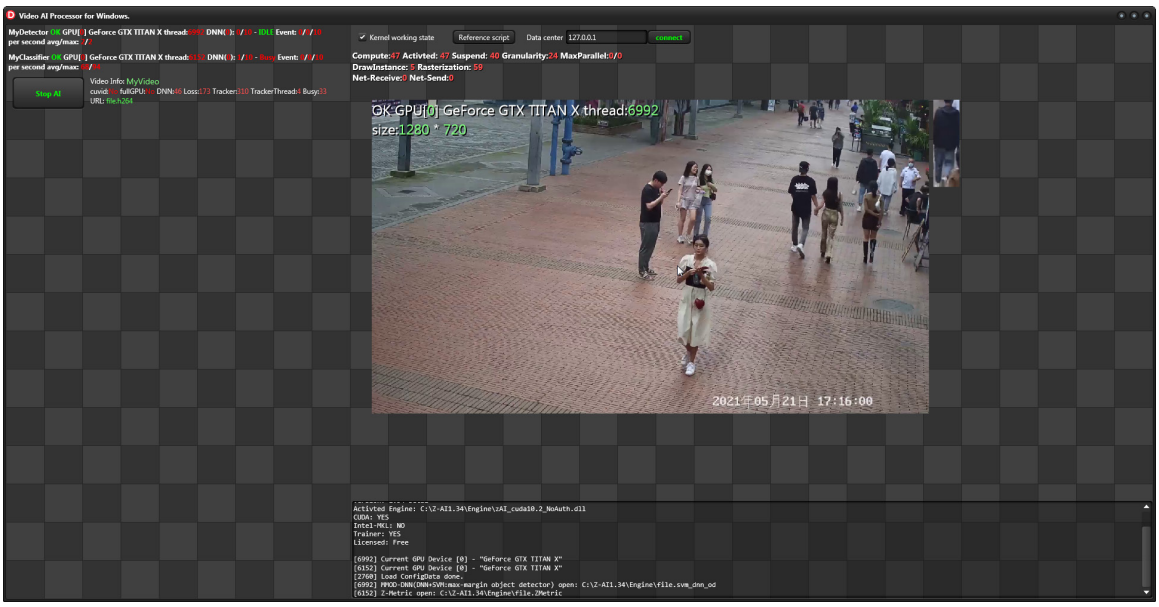
```
[MyDetector]
Type=MMOD6L
Model=file.svm_dnn_od
ClassifierProcessor= MyClassifier

[MyClassifier]
Type=ZMetric
Learn=file.learn
Model=file.ZMetric

[MyVideo]
Type=Video
VideoSource=file.h264
ODProcessor=MyDetector
```

检测器

分类器

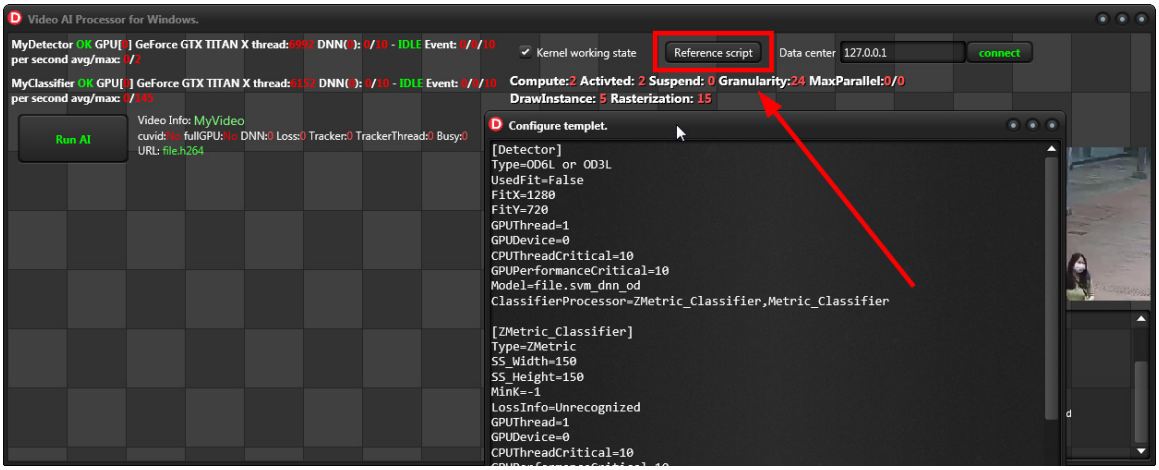
视频输入

上述脚本在运行以后

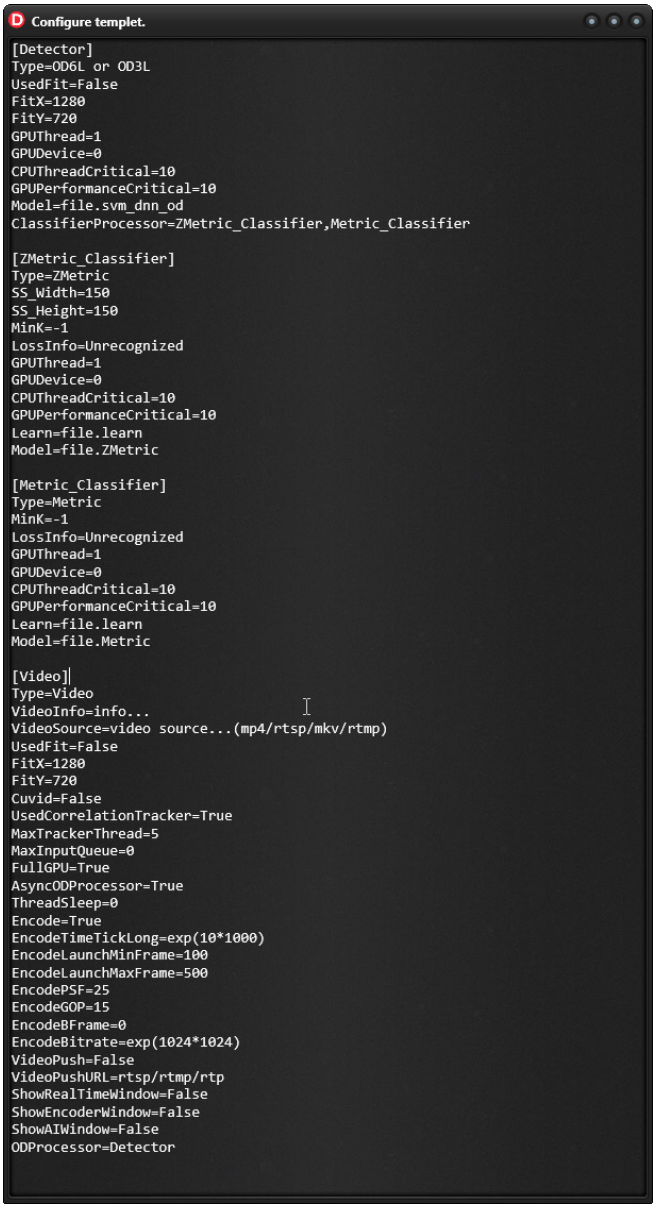


脚本定义

GPU 服务器内置的脚本定义都可以使用 Refrence script 按钮来得到它



通过 Configure templet 窗口我们可以得到 4 种程序模块配置，分别是 Detector，ZMetric，Metric，video



Detector

Type=OD6L or OD3L
指定程序类型，只能是 MMOD6L/MMOD3L，OD6L/OD3L 是一种简写，具体细节去阅读代码

UsedFit=False
FitX=1280
FitY=720
检测器 Fit 是一种软件缩放，功能是让输入光栅符合建模环境的尺度，检测器中的 Fit 不应该打开，它对视频的实时性远不如硬件 Fit，而硬件 Fit 是在 Video 程序中实现的

GPUDevice=0
指定检测器使用的 GPU 设备，这里的 0 是设备 ID，该 ID 可以通过 NVSMI or GPU-Z 这类工具来获取。一般来说如果 HPC 插有多张同类卡，该 ID 可按插卡顺序数数，第一张卡从 0 开始

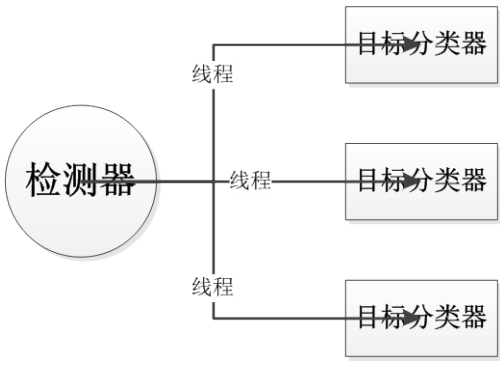
GPUThread=1
每个 GPU 使用多少条工作线程，默认为 1。因为 GPU 的数据 copy 机制是阻塞的，视频光栅需要从北桥在传输到 GPU，然后才能跑识别，即使有多条线程工作，也会出现明显阻塞，一般来说，2-3 个线程就足够了。在多卡系统，北桥带宽可能出现不够用的情况。

GPUPerformanceCritical=10
GPU 的性能临界，这个参数是限制每个 GPU 线程最大的等待队列。10 表示每个线程最大只能有 10 张光栅等待识别，该参数可大可小，根据内存规模而定。注意：如果 GPU 性能跟不上，输入会发生阻塞，例如从 RTMP 读取光栅被临界阻塞，这时候会使用临时缓冲区暂存，缓冲区堆满以后就发生断线，然后进入拉流重连程序。

CPUThreadCritical=10
当检测器处理完成后，会抛出一个并发形式的识别输出线程事件，该参数是限定最大并发线程数量。如果达到该数量，程序会暂存到触发队列中

Model=file.svm_dnn_od
根据 Type 参数而定，如果为 OD6L/MMOD6L 那么 Model 是后缀.svm_dnn_od 模型文件。如果为 OD3L/MMOD3L 那么 Model 是.svm_dnn_od_3L 模型文件

ClassifierProcessor=分类器 1，分类器 2，分类器 3，分类器 4
分类器流程，可以是 ZMetric 分类器也可以是 Metric 分类器，如果有多个分类器，用逗号隔开



ZMetric

Type=ZMetric
指定程序类型为分类器，依靠于 Z-AI 强大的建模能力，ZMetric 模型很容易制作，一般来说，有数据就可以有模型
ZMetric 有能力替代 Z-AI 中的所有分类器模型，包括 RNIC/LRNIC/Metric/LMetric
ZMetric 是 Z-AI 在 1.3 版本新出的一种卷积网络，具体细节见 Demo 和更新日志

SS_Width=150
SS_Height=150
输入尺度，这里需要跟 Z-AI 建模时的定义相同，提示一下，一般来说 ZMetric 的尺度不要低于 60

MinK=-1
LossInfo=Unrecognized
这是一个条件参数，当 MinK 低于 1，那么才输出标签，否则返回 LossInfo

GPUThread=1
GPUPerformanceCritical=10
每个 GPU 使用多少条工作线程，默认为 1。因为 GPU 的数据 copy 机制是阻塞的，视频光栅需要从北桥在传输到 GPU，然后才能跑识别，即使有多条线程工作，也会出现明显阻塞，一般来说，2-3 个线程就足够了。在多卡系统，北桥带宽可能出现不够用的情况。

GPUDevice=0
指定检测器使用的 GPU 设备，这里的 0 是设备 ID，该 ID 可以通过 NVSMI or GPU-Z 这类工具来获取。一般来说如果 HPC 插有多张同类卡，该 ID 可按插卡顺序数数，第一张卡从 0 开始

CPUThreadCritical=10
当检测器处理完成后，会抛出一个并发形式的识别输出线程事件，该参数是限定最大并发线程数量。如果达到该数量，程序会暂存到触发队列中

Learn=file.learn
Model=file.ZMetric
使用 Z-AI 制作 ZMetric 模型会得到.Learn+.ZMetric 两个模型数据，在这里指定文件名即可

Metric

Type=Metric
固定 Metric 程序类型
其它参数与 ZMetric 一样，Metric 是固定 150*150 尺度，模型训练推理略强于 ZMetric

Video

Type=Video
固定 Video 程序类型

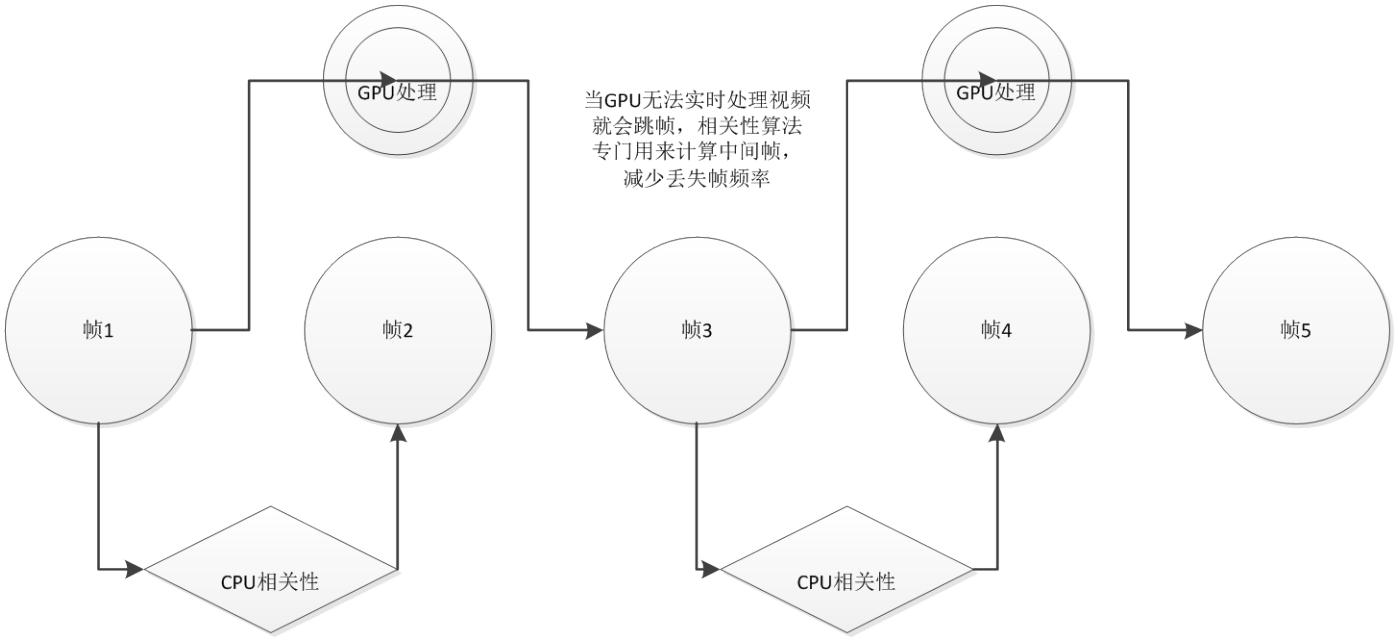
VideoInfo=info...
输入视频信息，例如给一个有辨识度的信息，XX 街道第 N 路监控

VideoSource=video source...(mp4/rtsp/mkv/rtmp)
视频输入源，这里的输入源可以是文件，而文件可以是.mp4/.mkv/.h264
如果是摄像头，可以是 rtsp/rtmp/usb-Device，如果是网络摄像头具体都在摄像头端配置好协议，通过 DNS/IP 给出地址即可输入到 Z-AI 的监控系统中来

UsedFit=False
FitX=1280
FitY=720
重要参数，硬件级 Fit 缩放，几乎无延迟。当使用 Z-AI 建模时输入尺度必须有所限制，小于 2k 或则小于 1080p，光栅不符合规范容易出现误检测或则找不到，这时候 Fit 作为规范视频输入尺度来统一处理。
Fit 缩放参数在打开了 cuvid 以后会使用 GPU 进行计算，几乎不消耗宝贵的 cpu 资源

Cuvid=False
是否使用 GPU 对输入视频进行解码。注意：大多数 GPU 对解码都有限定，一般来说，1-2 路，另外解码后，光栅的传输需要消耗北桥带宽，假如监控视频大于 2 路，这个参数建议给 False

UsedCorrelationTracker=False
使用相关性算法处理间隔帧，如果打开，就是相关性追踪性能模式，该模式可以大幅降低 GPU 检测器开销，而使用 cpu 进行相关性计算。**相关性适合 GPU 偏低+CPU 偏高的服务器，不建议个人 PC 运行该算法**。



注意：如果要打开相关性模式，FullGPU 需要是 false，否则相关性将不会起作用

MaxTrackerThread=5
最大相关性并发线程数量。如果给 0 表示无限制。
注意：相关性内部不会使用并行 for 处理间隔帧，因为间隔帧可能拉数十或则上百帧，使用并行会导致 hpc 整体性

能严重下降。

MaxInputQueue=0
0 表示不限制最大输入的等待队列。这个值也是一种 GPU 的最后性能防线：GPU 有很多设备号，数据队列这类机制，当这类机制无法满足巨大的实时视频流时，等待处理的视频数据就会开始拥堵，一旦达到了 MaxInputQueue 防线，那么视频帧将会按照 Lose 方式进行处理。
如果需要后台稳定，建议这个值给 200-500 间，根据内存大小而定。

FullGPU=True
打开这个选项以后，所有帧都会往 gpu 输入，这些输入都是排队形式。

注意：FullGPU 一旦打开，程序会进入纯 GPU 计算模式，UsedCorrelationTracker 将会失效。
FullGPU 可以让我们更简单暴力的通过堆 GPU 堆 HPC 来解决多路监控的识别问题。

AsyncODProcessor=True
该选项是在读取监控 rtsp/rtmp 时，希望达到即时处理下一帧的目的，而让线程开一个 Post 任务去把数据扔给检测器，默认都是打开的。配置时可以忽视。

ThreadSleep=0
每次从 rtsp/rtmp/mp4/mkv/h264 这些输入源读取视频时，都会执行一次线程 sleep。
监控因为都有帧数限制，我们可以忽视这个参数。但是输入源如果是文件，那么就是满负荷读取，ThreadSleep 则是让满负荷读取变的有限制。
0 表示不延迟，其它值表示每次读取视频源的接收延迟的毫秒。

Encode=True
是否重构视频片段，该参数会对输入视频做 h264 重构，并且会将识别结果渲染到重构视频，最后它还会存储到数据中心。重构视频片段在分线程中进行，不会影响输入线程和识别线程，简单来说，是在一条分线程中干完 h264 重构+AI 视觉渲染。

EncodeTimeTickLong=exp(10*1000)
重构视频的时间刻度，单位是毫秒，exp(10*1000)，含义是使用表达式，计算 10*1000 的整数，也就是大约每 10 秒物理时间就会做一个片段重构，然后发给数据中心

EncodeLaunchMinFrame=100
该参数是限制每次重构不能低于 100 帧

EncodeLaunchMaxFrame=500
该参数是限制每次重构不能高于 500 帧

EncodePSF=25
PSF，每秒固定帧数，具体详见 FFMPEG 的编码器相关技术信息，搜一下
这些信息建议与监控参数对号

EncodeGOP=15
GOP，相当于对齐帧，也可以说 I 帧，等同于数据规格为 fullsize 的帧，其它帧的数据规格可以理解成 motion 数据帧。，
具体详见 FFMPEG 的编码器相关技术信息，搜一下
这些信息建议与监控参数对号

EncodeBFrame=0
B 帧。具体详见 FFMPEG 的编码器相关技术信息，搜一下
这些信息建议与监控参数对号

EncodeBitrate=exp(1024*1024)
h264 码率，这些信息建议与监控参数对号

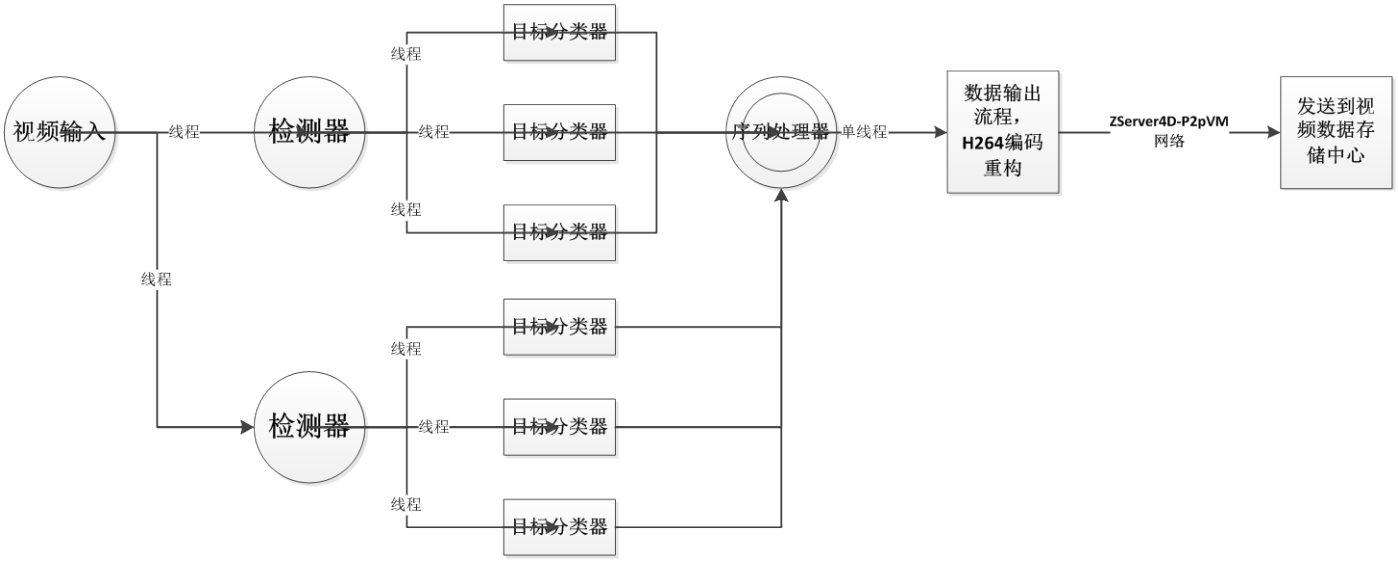
VideoPush=False
VideoPushURL=rtsp/rtmp/rtp
在读取视频时，同时推流到目标服务器，

ShowRealTimeWindow=False
显示实时窗口，与使用 ffplay/vlc 播放 rtsp 一致，无 GPU/CPU 处理延迟

ShowEncoderWindow=False
显示编码窗口，这是在读取完 rtsp/rtmp 这类视频以后，重构 h264 的窗口

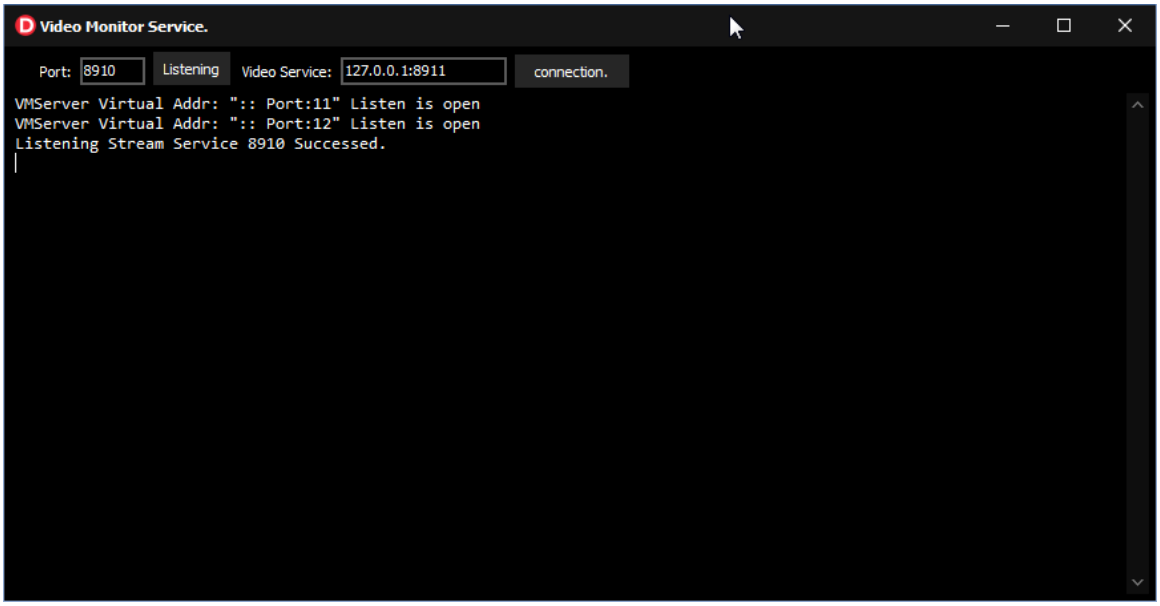
ShowAIWindow=False
显示编码窗口，这是在异步执行 AI 处理，然后再将处理结果渲染成光栅，最后再编码 h264 的窗口

ODProcessor=检测器程序块 1, 检测器程序块 2, 检测器程序块 3
视频会输入给那些检测器，可以通过逗号，给多个检测器



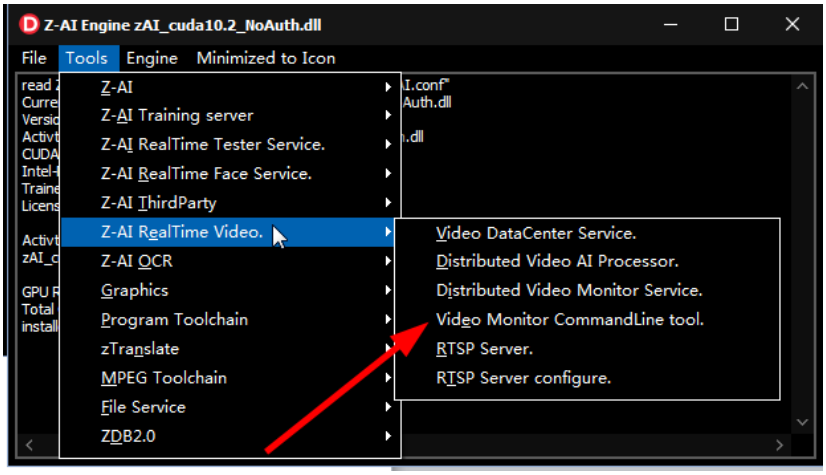
分布式监控服务器

Monitor Service 它侧重于对监控端提供查询和下载视频的服务，属于轻量级服务器
使用 Monitor Service 只需要在打开以后，连接到数据存储中心就不用再管它了（自动化断线重连）
Monitor Service 可以根据不同的网络架构进行分布式部署



监控客户端

Z-AI 的监控客户端采用命令行工具形式提供，可以通过工具链系统启动它



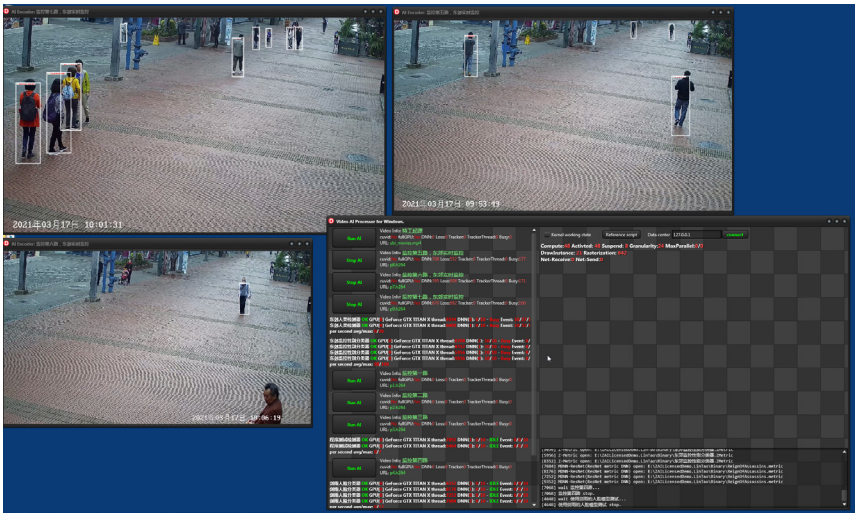
监控客户端是按阻塞网络方式编写的。我们需要自己设计监控端 UI，然后使用命令行来查询和下载监控视频。
具体使用发方法在命令行窗口 VideoMonitor.exe –help 即可

制作 AI 渲染大屏幕

首先在 Video 程序配置块打开 ShowAIWindow

```
[监控第五路，东郊实时监控]
Type=Video
VideoInfo=监控第五路，东郊实时监控
VideoSource=p8.h264
ThreadSleep=100
Cuvid=False
ODProcessor=东郊人类检测器
UsedFit=True
FitX=1280
FitY=720
FullGPU=True
UsedCorrelationTracker=False
MaxTrackerThread=2
Encode=True
EncodeTimeTickLong=exp(10*1000)
EncodePSF=10
EncodeGOP=5
EncodeBFrame=0
EncodeBitrate=exp(1024*1024*4)
ShowRealTimeWindow=False
ShowEncoderWindow=False
ShowAIWindow=True
```

在 UI 中 Run AI，得到带有框体和标注的弹出窗口



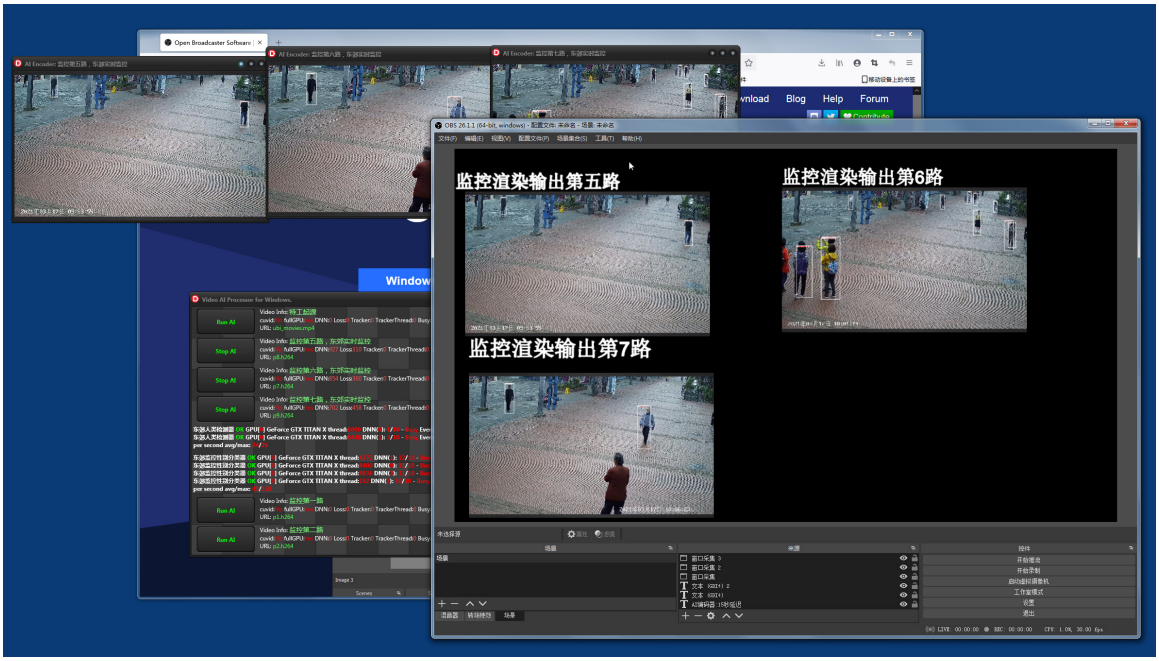
准备好 OBS 工具

https://obsproject.com/

设计一下场景布局

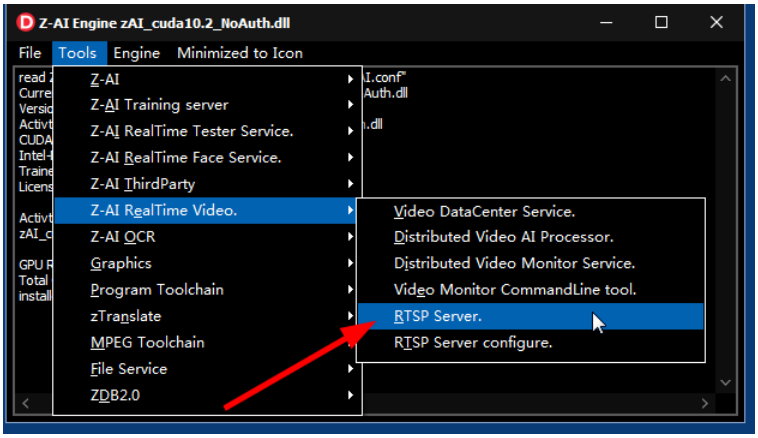
在 obs 中设置捕获窗口

GPU 视频视频处理服务不会停，即时断线也不会停，obs 可以长时间开启



打开监控推流服务

Z-AI 内置的 RTSP Server 是傻瓜化的单路推流，打开就可以用，如果要定义多路推流，就自己修改配置



然后再使用 obs 往里面面推流

这时候再通过 TV/手机/电脑即可监视实时视频

如何实时监控

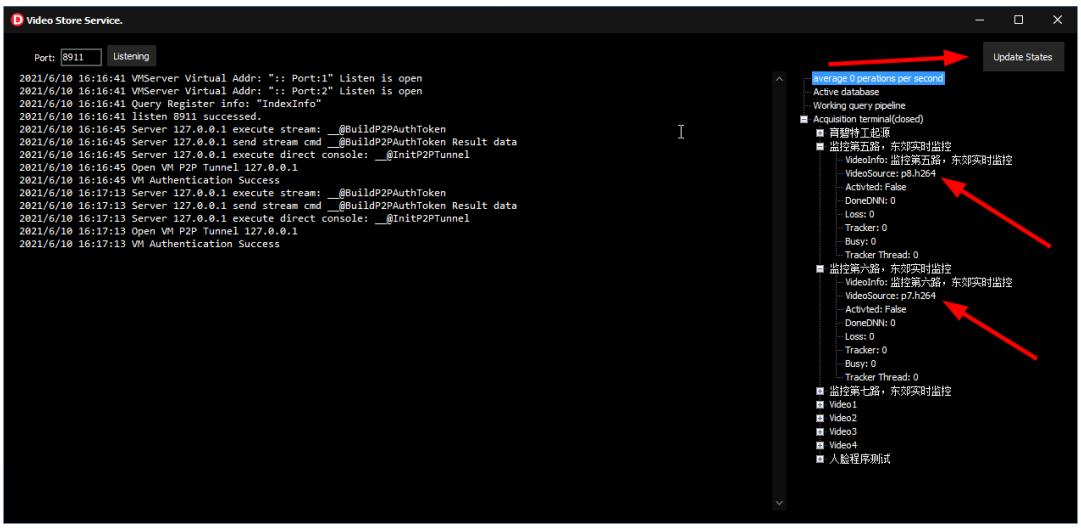
在需要监控的手机端或则电脑，安排一个播放器，可以自己写也可以用开源，或则直接使用 Z-AI 内置的 ffplay 都可以实时监控。另外，实时监控还需要一个推流的地址，下面介绍 2 种推流办法

通过 VideoPush 开关实时推流

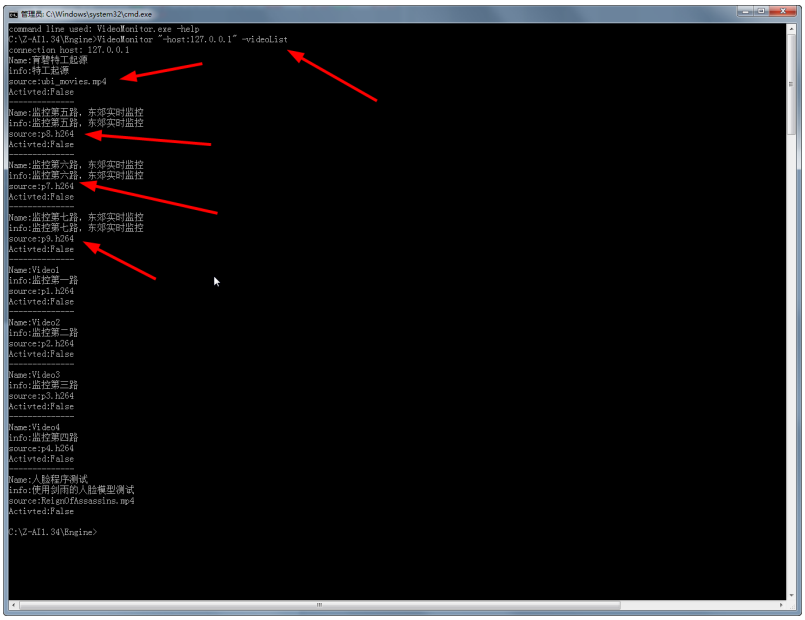
VideoPush=False
VideoPushURL=RTSP 服务器推流地址，这些服务器可以用云也可以自己搭建，Z-AI 内置有一个 RTSP 服务器
然后访问这个 RTSP 服务器就可以推流了

直接访问摄像头获取推流图像

通过数据中心服务器的状态信息获取推流地址



通过 videoMoitor 的命令行获取推流地址
Source 就是推流地址



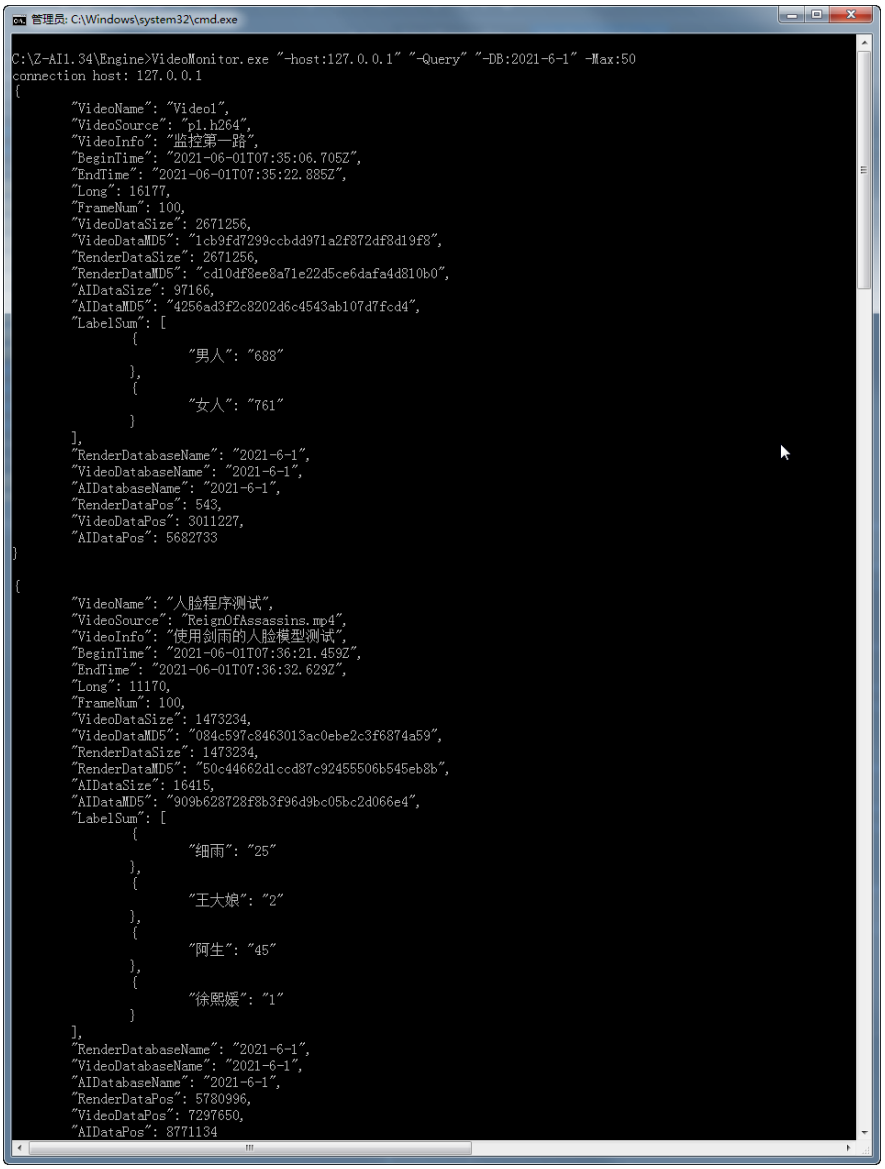
如何开发统计算法

一般的统计直接使用命令行工具就可以了，例如

从数据中心服务器查询一个时间范围

VideoMonitor.exe "-host:127.0.0.1" "-Query:BeginTime=2021-5-31 15:00:00,EndTime=2021-5-31 16:00:00" "DB:2021-5-31"

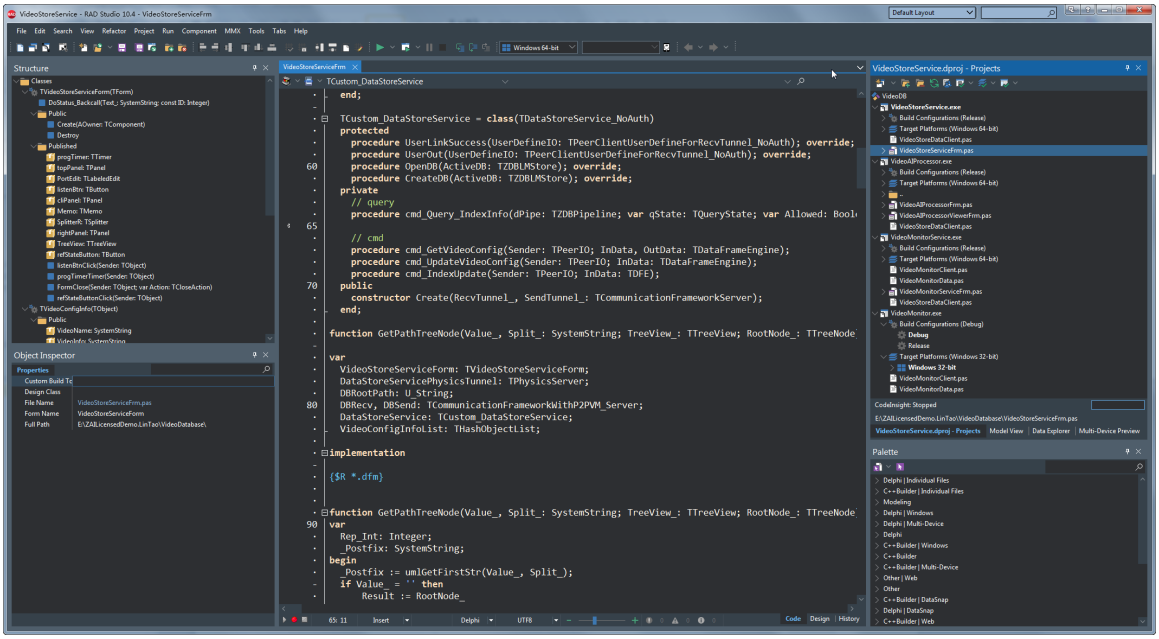
这时候就会得到结果，分析一下返回值就可以了



二次开发

Z-AI 的监控系统核心库都包含在 Z-AI 内核中的，未来会随 Z-AI 系统升级更新和优化，一般来说这一部分主要都是 AI 识别处理的代码，和主架构关系不大。

二次开发多用于包装和修改，这类开发直接打开工程组修改编译即可



二次开发如果做端，那些手机端，pc 端，或则包一下 IOT 端之类
这时候主要参考 VideoMonitor.dproj，参考一下它的通讯协议，数据流程，整合到目标端就可以了

by.qq600585
2021-6