# ZAI 多语言与多平台业务对接系统

多语言对接系统的功能非常类似 RealTimeTester 系统，因为 RealTimeTester 系统把 ZAI 的所有 DNN 模型全部驱动了，导致整个系统比较复杂，一般人难以研究和阅读。

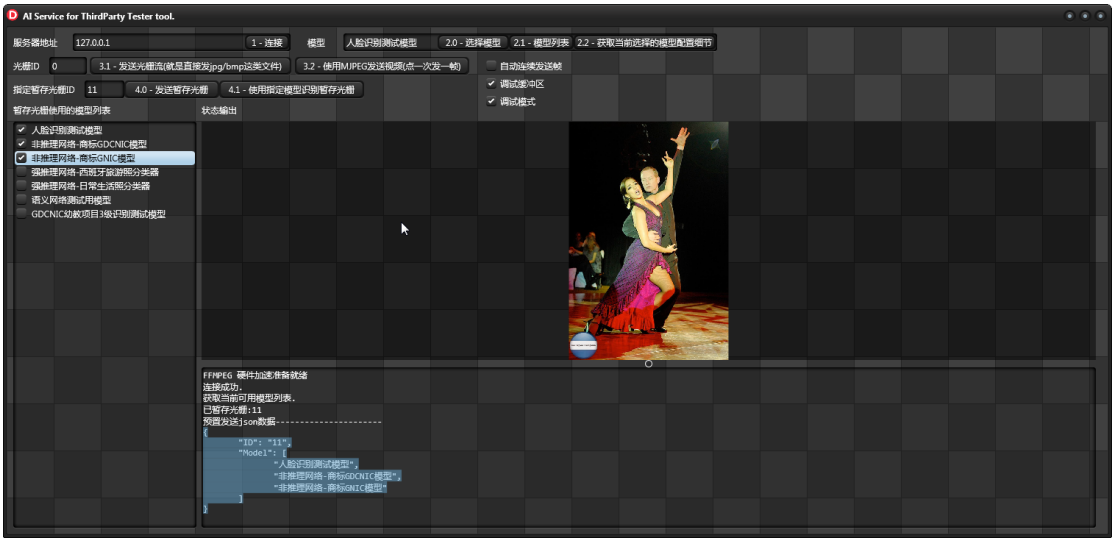
AIServiceForThirdParty 就是多语言对接系统的后台，这套系统采用 json 交互会话完成实时 AI 识别处理，二次开发的简便性优于 RealTimeTester 系统。

编程语言可以支持：VC++/VC#/VF#/GCC/CLang/XCodeOC/Swift/JAVA/PY/Delphi/FPC

**网路通讯模型必须为双工异步，不可以使用阻塞模型**

**AIServiceForThirdParty 是多语言多平台的接口后台，关闭 Debug 模式后可跑业务，可支持大规输入模队列的平滑识别**


**AIServiceForThirdPartyTesterTool 是帮助二次开发通讯协议分析和模型测试工具**




**附带提供一个 VC++的协议 demo**

# AI 后台通讯协议

## 目录

# 版本信息

通讯协议只会增加，不会在后续迭代时发生修改，尽量保证前端工作是一次性接入，避免反复接口。

## 当前版本 1.3

更新：详细介绍了 3 级模型和分类器返回结果

# 可接口的编程语言

支持：VC++/VC#/VF#/GCC/CLang/XCodeOC/Swift/JAVA/PY/Delphi/FPC
注意：不支持 JavaScript 脚本

# 接入步骤

1， 系统有实时性机制并且采用 MJPEG 硬件编码（使用 cuda 和加强指令 AVX2/AVX512/SSE4），接口前务必先开个测试项目走通 MJPEG 的编码流程。不要直接在公司的工程接入，独立开一个工程，独立接通，再接入公司主工程
2， 走通 MJPEG 硬件编码流程后，开始对接数据结构，也是本文档重点讲述内容
3， 走通数据结构以后，进入应用层开发和调试，配合服务器组和建模组即可完成工作
4， 迭代到一定时间后，开始准备内测公测这类试运营，这时候，我们需要重新梳理一次协议，协议加密，防 ddos/cc，压力测试等等工作
5， **系统分了 3 个识别等级，1 级表示输入数据直接识别返回，2 级表示识别后，再用别的识别第二次，然后返回，3 级则表示 3 个步骤的识别流程，这些模型在系统中是自由搭配的，通过配置脚本可以做出许多识别效果**。

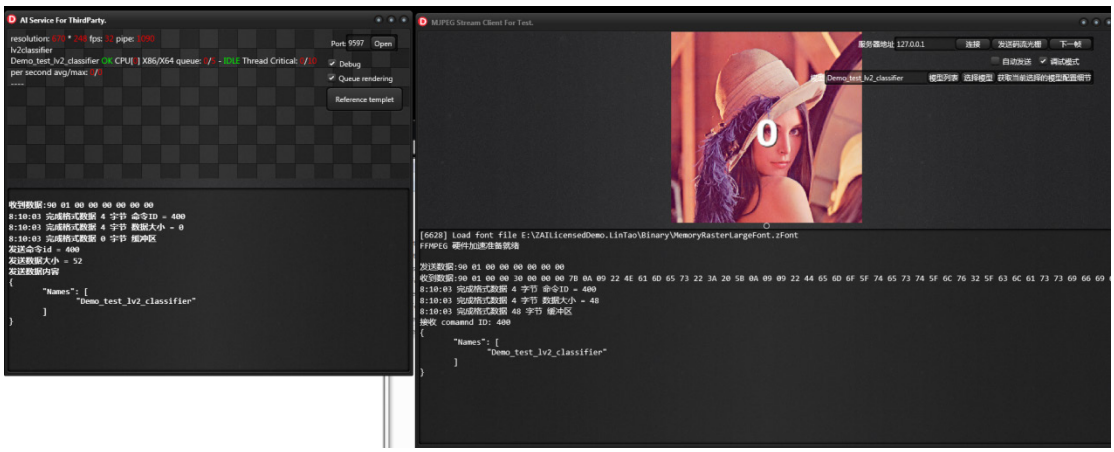
**提示：系统会把所有通讯数据都打印出来，在多数情况下这样的机制可以帮助接口开发。**
**提示，1/2/3 级识别结果会与模型中载入状态相吻合，下图中，模型如果支持 3 级识别，那么箭头所指都会是 true，如果 lv3Mode=true，那么识别的结果就以 id=603 进行处理即可**


# 通用协议

通用协议是指在本项目中的通过网络收发的公共数据结构。这套结构只有一个，如下

struct
{
(4byte)UInt32 :command_ID //命令 ID，用以区分不同的执行命令
(4Byte)UInt32: Size //数据大小
(根据 Size 而定的数据体)body...
}

上面就是一个数据块的公共协议，AI 后台所有的数据都用这套数据块
公共协议在测试用的 AI 服务器收发时均有提示



# 网络通讯机制

网络通讯机制采用单连接双工通信，既在一个连接中，同时收发，不可以直接使用阻塞类型的通讯框架，优先选择有 async 这类字样的异步网络通讯框架。

# 服务器命令

表示由客户端发送给服务器的格式数据

## 服务器命令 Command_ID=0

Command_ID=0 时，表示发送 MJPEG 推流，这时候，要求数据体前 4byte 位置为序列号 ID，后面为推流数据，推流的 Demo 范例参考，VS_MJPEGStreamClientForTest，这是使用 VC++编写的标准 Demo

## 服务器命令 Command_ID=1

Command_ID=1 时，表示发送 JPEG/PNG/BMP24/BMP32 文件，这时候，要求数据体前 4byte 位置为序列号 ID，后面为光栅数据。
光栅数据就是图片文件

## 服务器命令 Command_ID=2

Command_ID=2 时，表示发送 JPEG/PNG/BMP24/BMP32 文件，但是不会执行识别
要求数据体前 4byte 位置为序列号 ID，后面为光栅数据。
光栅数据就是图片文件

## 服务器命令 Command_ID=3

Command_ID=3 时，表示是立即根据序列号 ID 和指定的多个模型执行识别

```
{
    "ID": "9999", 9999 表示序列号 ID
    "Model": [
        "人脸识别测试模型",
        "非推理网络-商标 GDCNIC 模型",
        "非推理网络-商标 GNIC 模型"
    ]
}
```

# 服务器命令 Command_ID=100

重新初始化 MJPEG 的解码参数，该命令主要解决推流的光栅发生尺寸变化问题，例如，前端的分辨率从 480*640 编程 768*1024，这时候，需要发这条命令给服务器，然后再推流
**该命令没有数据**

# 服务器命令 Command_ID=200

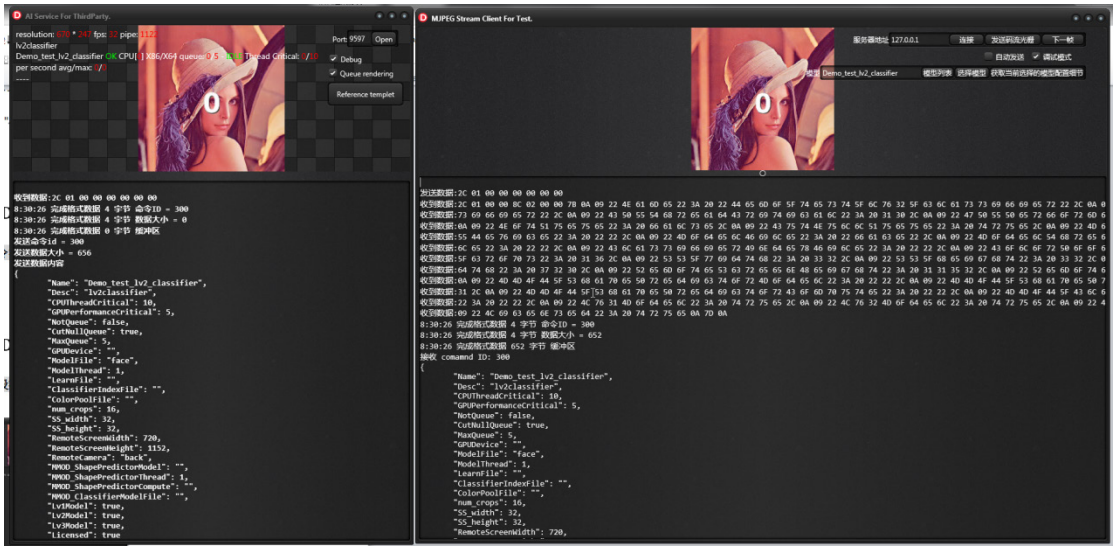
选择模型，数据体要求使用 utf8 编码的 json，格式如下

```
{
    "Model": "Demo_test_lv2_classifier"
}
```

Model 后的字符串为模型名字

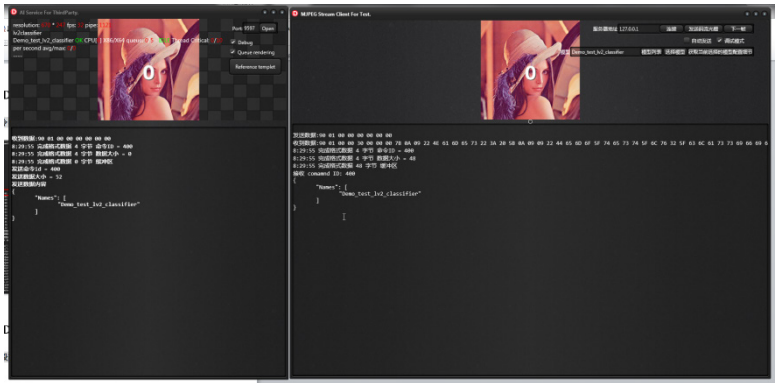
# 服务器命令 Command_ID=300

获取当前选择模型的详细信息，**该命令没有数据**，服务器会通过接收信道发送一个 json 的数据块回来，参考 客户端命令，Command_ID=300



# 服务器命令 Command_ID=400

获取服务器支持的模型列表，**该命令没有数据**，服务器会通过接收信道发送一个 json 的数据块回来，参考 客户端命令，Command_ID=400

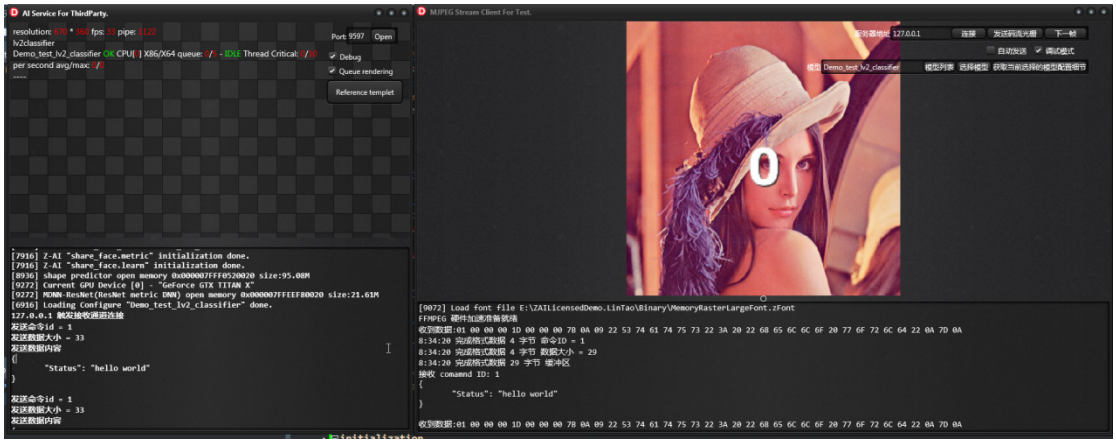


# 客户端命令

表示由服务器主动向客户端发出的命令，既客户端收到的命令

## 客户端命令，Command_ID=1

显示状态信息，数据体为 utf8 格式的 json 文本



## 客户端命令，Command_ID=300

这条命令，必须先往服务器发个 command_id 是 300 的请求，然后服务器会立即返回当前的模型信息，这些信息也表示客户端的需要进行的工作模式，数据体为 utf8 格式的 json 文本。
参考 服务器命令 Command_ID=300

## 客户端命令，Command_ID=400

这条命令，必须先往服务器发个 command_id 是 400 的请求，然后服务器会立即返回当前支持的模型列表信息，数据体为 utf8 格式的 json 文本。
参考 服务器命令 Command_ID=400

## 客户端命令，Command_ID=501

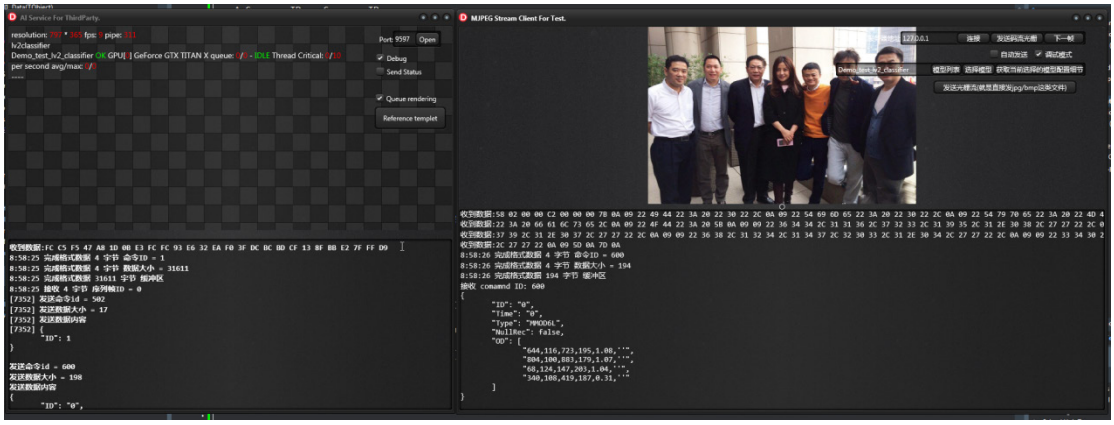
输入的推流/光栅数据被服务器拒绝：当 GPU 服务器忙碌时，会丢弃一部分数据，收到这条命令，可以做一些内存释放之类操作

## 客户端命令，Command_ID=502

输入的推流/光栅数据被服务器所接受，收到这条命令，表示服务器正在计算，在若干毫秒后会就会收到一条识别结果了

# 客户端命令，Command_ID=600

1 级识别结果



在识别结果中 ID，表示发送时输入序列帧，这个序列帧号与发送推流或则光栅时给出的序列帧 ID 是对应的。
NullRec，表示成功识别
Type，数据结构类型
数据类型有：Metric/LMetric/MMOD6L/MMOD3L/GDCNIC/GNIC/SS

识别结果格式 Metric/LMetric 时，表示度量化网络，会包含 K 值和标签，格式如下
"k": 0.094610330503952
"label": "George_W_Bush"
k 值越小表示越接近样本库某个人，换算方式为(1-k)*100=91%的相似性
标签是建模时输入的分类字符串

识别结果格式如果为 MMOD6L/MMOD3L，会包含 OD 坐标，可信度，OD 标签，格式如下

```
"Result": {
    "OD": [
        {
            "left": 140,
            "top": 63,
            "right": 337,
            "bottom": 260,
            "confidence": 1.08152258396149,
            "label": ""
        }
    ]
```

处理 OD 坐标时，坐标值都是输入图像的绝对值。confidence 表示与标注样本的贴近度，越小越贴近，多数情况下，不需要管这个值，假如出现了很多误检框体，那么可以根据该值做剔除处理。标签一般用于大分类，例如，汽车，行人，卡片，由于 OD 神经网络是凸方式收敛，导致了标签只能做一些粗线条分类，无法像 Metric 那样可以做精确局部分类。

识别结果格式如果为 RNIC/LRNIC/GDCNIC/GNIC，会包含所有分类相似性，由大大到小排序


```
"result": {
    "classifier": [
        {
            "k": 0.999914288520813,
            "label": "商标 1.JPG"
        },
        {
            "k": 8.48698618938215E-5,
            "label": "商标 2.JPG"
        },
        {
            "k": 8.0440190686204E-7,
            "label": "商标 3.JPG"
        }
    ]
```


k 值越大表示越接近标签目标

label 值是训练时定义的标签

识别结果如果为 SS，表示语义分割模型，该模型主要用于数据科学领域，例如制作大数据，例如生成特朗普的电视讲话，格式如下


```
"result": {
    "SS": [
        {
            "color": "FFD79ADA",
            "label": "人类"
        }
    ],
    "png-alpha": "",
    "png-data": ""
```


SS 中的 color 按 A,R,G,B，小序列 CPU 方式进行对齐

label 为 SS 检测到的目标像素

png-alpha，以 base64 编码的 png 存储格式，视觉效果数据，这些效果都按凸包封闭线绘制

png-data，以 base64 编码的 png 存储格式，分割像素数据，这些像素会与 color 值所对应

# 客户端命令，Command_ID=602

2 级识别结果，在 2 级识别结果中会包含 1 级识别的值，也就是说，假如我们收到了 2 级识别结果，我们就不需要再管 1 级结果了，直接按 2 级识别结果处理它即可

# 客户端命令，Command_ID=603

3 级识别结果，3 级识别结果会自动把 1、2 级识别结果都包含进来，
例如，
我们的模型按检测器->SP+投影重构->分类器走
结果会依次收到 600/602/603
600 表示检测器结果
602 表示 SP 坐标结果
603 表示分类器识别出的度量化网络/分类器目标结果
这时候，我们不需要管 600/602，只需要处理 603 里面的数据即可

全文完
by.qq600585