华为 HiAi

- 基本使用

HiAI 使用时端+云的方式，官方提供的 SDK，包括三个：

1. hwhiai Engine SDK
2. 麒麟810/990芯片HiAI Foundation开发套件 （DDK V320版本）
3. 麒麟970/980芯片HiAI Foundation开发套件（DDK V200版本）

SDK下载列表

SDK名称	描述
DDK V320版本	麒麟810/990芯片HiAI Foundation开发套件
DDK V200及之前版本	麒麟970/980芯片HiAI Foundation开发套件
SDK下载	Huawei HiAI Engine SDK

注册一个新的应用接入端，用于测试 HiAi 接口能力集，了解基本的使用和大致的接口结构。

填写基本的信息后，就注册好了 App 的基础信息。

- ddk 包说明

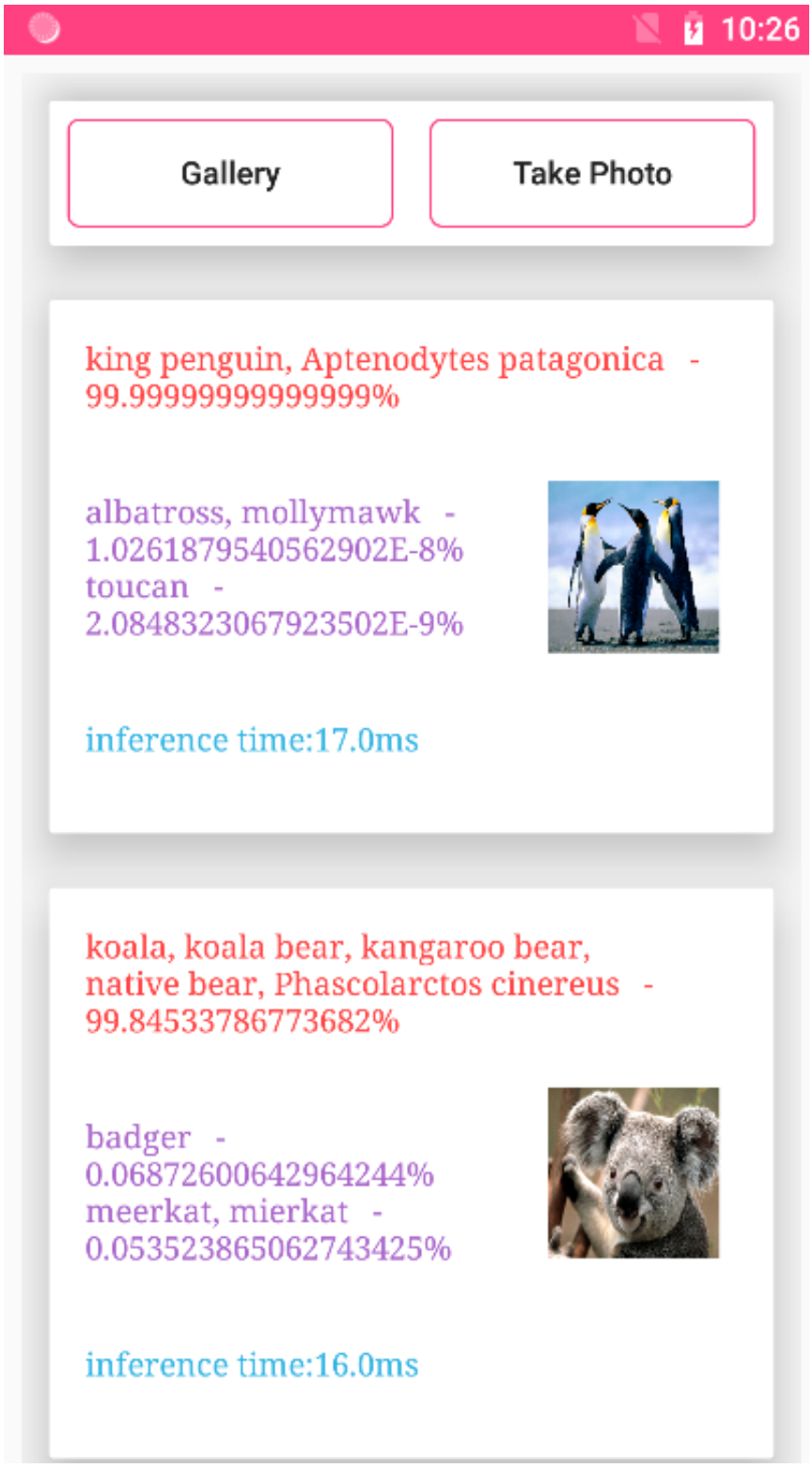
DDK(Device Development Kit)是 AI 开放的一个开发包，包含的是一个完整的 AI开发环境。包体包括以下几个部分：

1. App_sample (Android demo 示例)
2. ddk（开放 SDK）
3. document (参考文档)
4. tools（Tensorflow/Caffe 等模型转换工具）

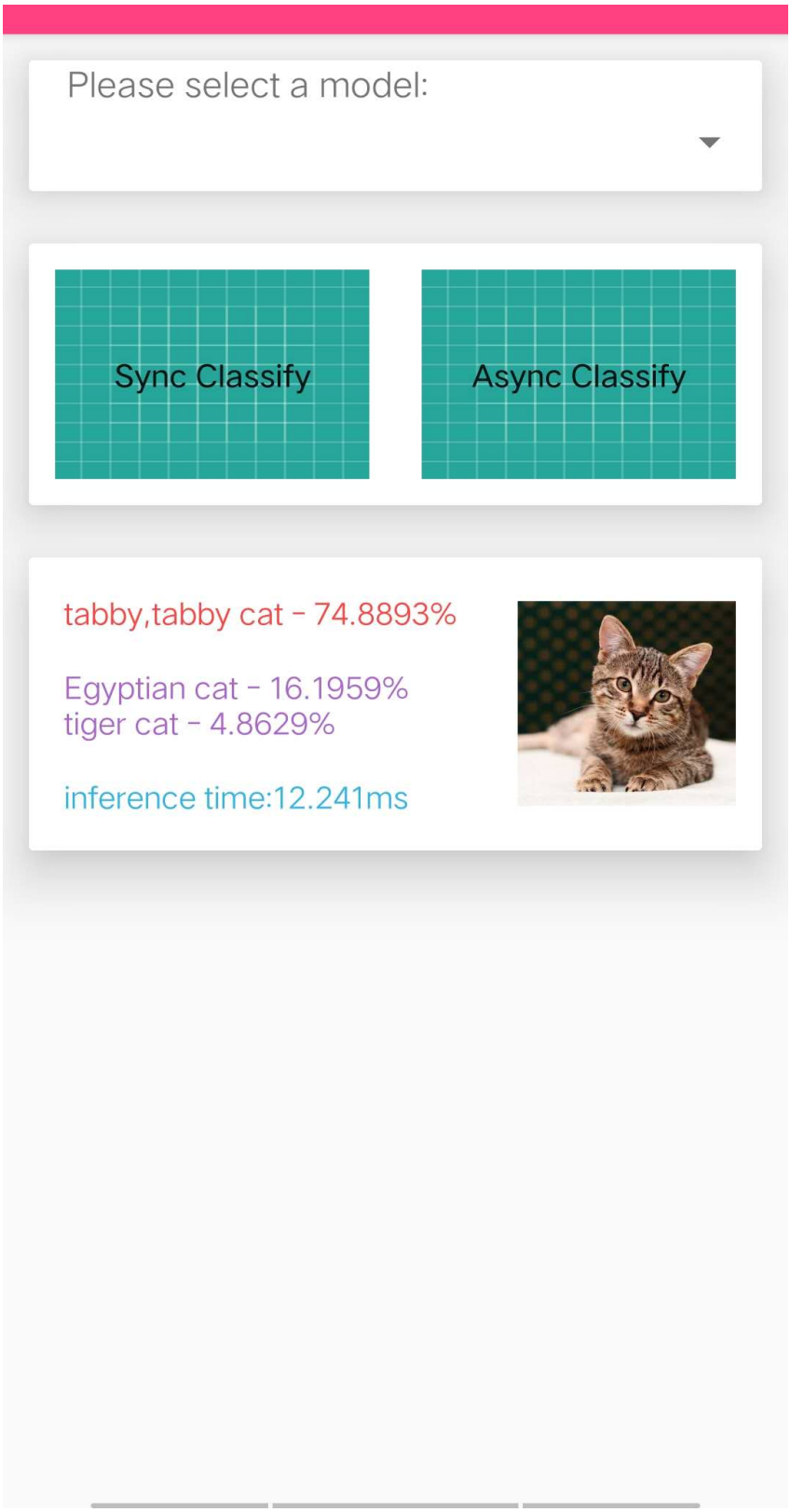
Ddk 工具是支持offline 模式的 AI 预测计算的。官方例子中介绍了使用的方法。

```
protected void initModels(){
    File dir =  getDir("models", Context.MODE_PRIVATE);
    String path = dir.getAbsolutePath() + File.separator;

    // add models to be loaded here
    /*
        AIPP is supported since v310.
        Run the aipp model, you need to remove the comment symbol from the
code block below.
    */
    /*
    ModelInfo model_1 = new ModelInfo();
    model_1.setModelSaveDir(path);
    model_1.setUseAIPP(true);
    model_1.setOfflineModel("hiai.om");
    model_1.setOfflineModelName("hiai");
    model_1.setOnlineModelLabel("labels_caffe.txt");
    demoModelList.add(model_1);
    */

    ModelInfo model_2 = new ModelInfo();
    model_2.setModelSaveDir(path);
    model_2.setUseAIPP(true);
    model_2.setOfflineModel("hiai_noaipp.om");
    model_2.setOfflineModelName("hiai_noaipp");
    model_2.setOnlineModelLabel("labels_caffe.txt");
    demoModelList.add(model_2);
}
```

如果使用正常的华为手机，或者带有离线预测的手机，可以看到以下界面提示：



如果非华为系手机，则提示不可使用，只能在 CPU 中计算预测结果。



在 ddk 包中，主要包含了两个部分：

- 推理部分 依赖和头文件

目录文件	描述说明
ai_ddk_lib\lib64\libhiai.so	DDK模型推理依赖的动态库
ai_ddk_lib\include\HiAiModelManagerService.h	DDK对外提供C++接口

- 模型构建部分

目录文件	描述说明
ai_ddk_lib\lib64\libhiai_ir.so	IR算子定义 & 图构建依赖库
ai_ddk_lib\lib64\libhiai_ir_build.so	IR模型编译依赖库
ai_ddk_lib\include\hiai_ir_build.h	DDK IR API构建、算子

- 集成

离线模型转换

离线模型转换需要将Caffe或者TensorFlow模型转换为HiAI平台支持的模型格式，并可以按需进行AIPP操作、量化操作，使用场景及方法如下：

1.AIPP操作

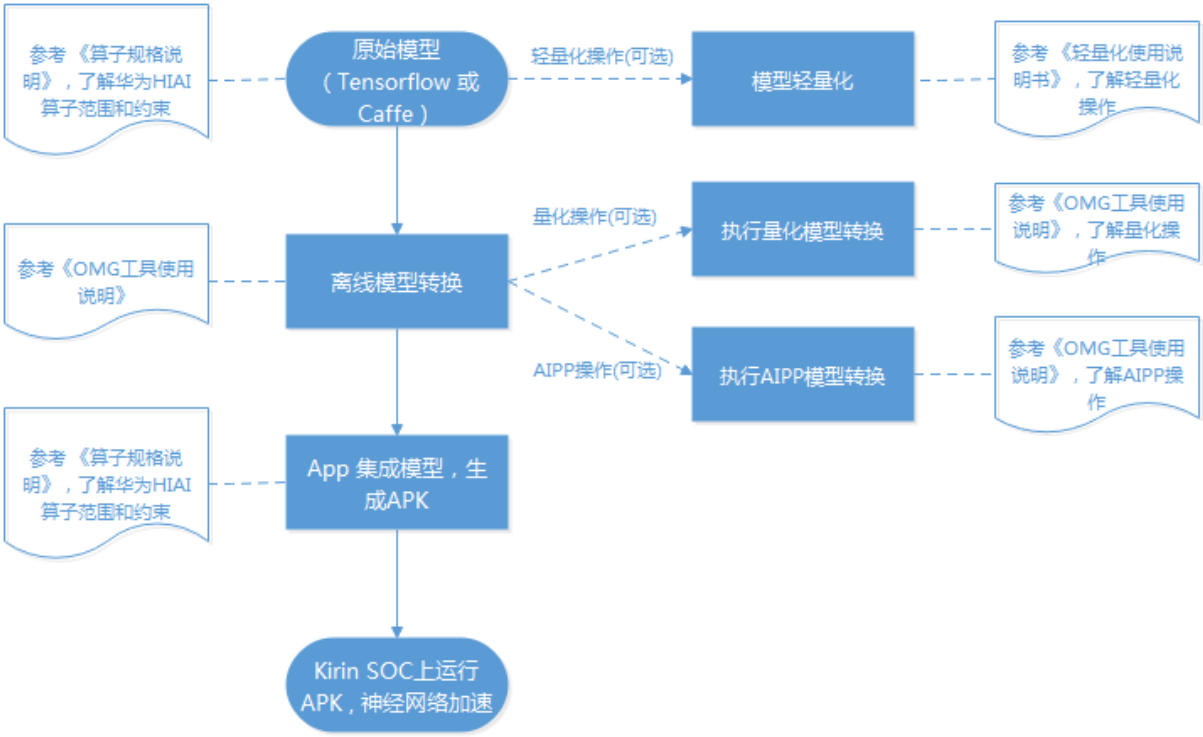
AIPP用于在硬件上完成图像预处理，包括改变图像尺寸、色域转换（转换图像格式）、减均值/乘系数（改变图像像素），运用后可避免重新训练匹配推理计算平台需要的数据格式，仅仅通过AIPP参数配置或者在软件层面上调用AIPP接口即可完成适配，同时由于硬件专用，可以获得较好的推理性能收益，具体操作可参考《华为HiAI_DDK_V320_OMG工具使用说明》中AIPP模型转换以及配置的操作指导。

2.量化操作

量化是一种可以把fp32模型转化为低bit模型的操作，以节约网络存储空间、降低传输时延以及提高运算执行效率，量化操作可参考《华为HiAI_DDK_V320_OMG工具使用说明》中量化模型转换的操作指导。

APP集成

APP集成流程包含模型预处理、加载模型、运行模型、模型后处理。



- 透出接口

关于 Model 的接口类都在 ModelManager.java 中定义，首先可以看下工程里面有的

```
/**
* 方法描述：
* 运行模型
* @version
*/
public static native ArrayList<float[]> runModelSync (ModelInfo modelInfo,
                                                       ArrayList<byte[]>
buf);
/**
* 方法描述：
* 取得使用时间
* @version
*/
public static native long GetTimeUseSync ();
/**
* 方法描述：
* 运行模型
* @version
*/
public static native void runModelAsync (ModelInfo modelInfo,
ArrayList<byte[]> buf,
                                         ModelManagerListener listener);
```

```
/**
* 方法描述:
* 导入模型
* @version
*/
public static native ArrayList<ModelInfo> loadModelAsync
(ArrayList<ModelInfo> modelInfo);

/**
* 方法描述:
* 导入模型
* @version
*/
public static native ArrayList<ModelInfo> loadModelSync
(ArrayList<ModelInfo> modelInfo);

/**
* @param offlinemodelpath /xxx/xxx/xxx/xx.om
* @return ture : it can run on NPU
* false: it should run on CPU
*/
public static native boolean modelCompatibilityProcessFromFile (String
offlinemodelpath);

//public static native boolean modelCompatibilityProcessFromBuffer(byte[]
onlinemodelbuffer,
// byte[] modelparabuffer,String framework,String offlinemodelpath);
```

按照 sourceDemo 的示例，可以发现，首先做的是把模型从 asset 中 copy 到指定路径中。然后离线模型会使用 loadModelSync 方法导入模型信息。

```
extern "C"
JNIEXPORT jobject JNICALL
Java_com_huawei_hiaidemo_utils_ModelManager_loadModelSync(JNIEnv *env,
jclass type,jobject modelInfo){

    jclass classList = env->GetObjectClass(modelInfo);
    if(classList == nullptr){
        LOGE("[HIAI_DEMO_SYNC] can not find List class.");
    }

    //解析mediainfo 内容 存入 vector 容器
    jmethodID listGet = env->GetMethodID(classList, "get", "
(I)Ljava/lang/Object;");
    jmethodID listSize = env->GetMethodID(classList, "size", "()I");
    int len = static_cast<int>(env->CallIntMethod(modelInfo, listSize));

    vector<string> names, modelPaths;
    vector<bool> aipps;
    for(int i = 0;i < len ;i++){
        jobject modelInfoObj = env->CallObjectMethod(modelInfo, listGet,
```

```
i);
        jclass modelInfoClass = env->GetObjectClass(modelInfoObj);
        jmethodID getOfflineModelName = env-
>GetMethodID(modelInfoClass,"getOfflineModelName","()Ljava/lang/String;");
        ……

        LOGE("[HIAI_DEMO_SYNC] useaipp is %d.", bool(useaipp==JNI_TRUE));
        aipps.push_back(bool(useaipp==JNI_TRUE));
        names.push_back(string(modelName));
        modelPaths.push_back(string(modelPath));
    }

    // load 模型
    if (!g_clientSync)
    {
        g_clientSync = LoadModelSync(names, modelPaths,aipps);
        if (g_clientSync == nullptr)
        {
            LOGE("[HIAI_DEMO_SYNC] g_clientSync loadModel is nullptr.");
            return nullptr;
        }
    }

    // load model 解析 N C W H 四维特征向量
    LOGI("[HIAI_DEMO_SYNC] sync load model INPUT NCHW : %d %d %d %d." ,
inputDimension[0][0].GetNumber(), inputDimension[0][0].GetChannel(),
inputDimension[0][0].GetHeight(), inputDimension[0][0].GetWidth());
    LOGI("[HIAI_DEMO_SYNC] sync load model OUTPUT NCHW : %d %d %d %d." ,
outputDimension[0][0].GetNumber(), outputDimension[0][0].GetChannel(),
outputDimension[0][0].GetHeight(), outputDimension[0][0].GetWidth());

    for(int i = 0;i < len ;i++){
        jobject modelInfoObj = env->CallObjectMethod(modelInfo, listGet,
i);
        jclass modelInfoClass = env->GetObjectClass(modelInfoObj);
        jfieldID input_n_id = env-
>GetFieldID(modelInfoClass,"input_N","I”);

        ……

        jfieldID input_Number = env-
>GetFieldID(modelInfoClass,"input_Number","I");
        env->SetIntField(modelInfoObj,input_n_id,inputDimension[i]
[0].GetNumber());
        env->SetIntField(modelInfoObj,input_c_id,inputDimension[i]
[0].GetChannel());

        ……

        jfieldID output_n_id = env-
>GetFieldID(modelInfoClass,"output_N","I”);
```

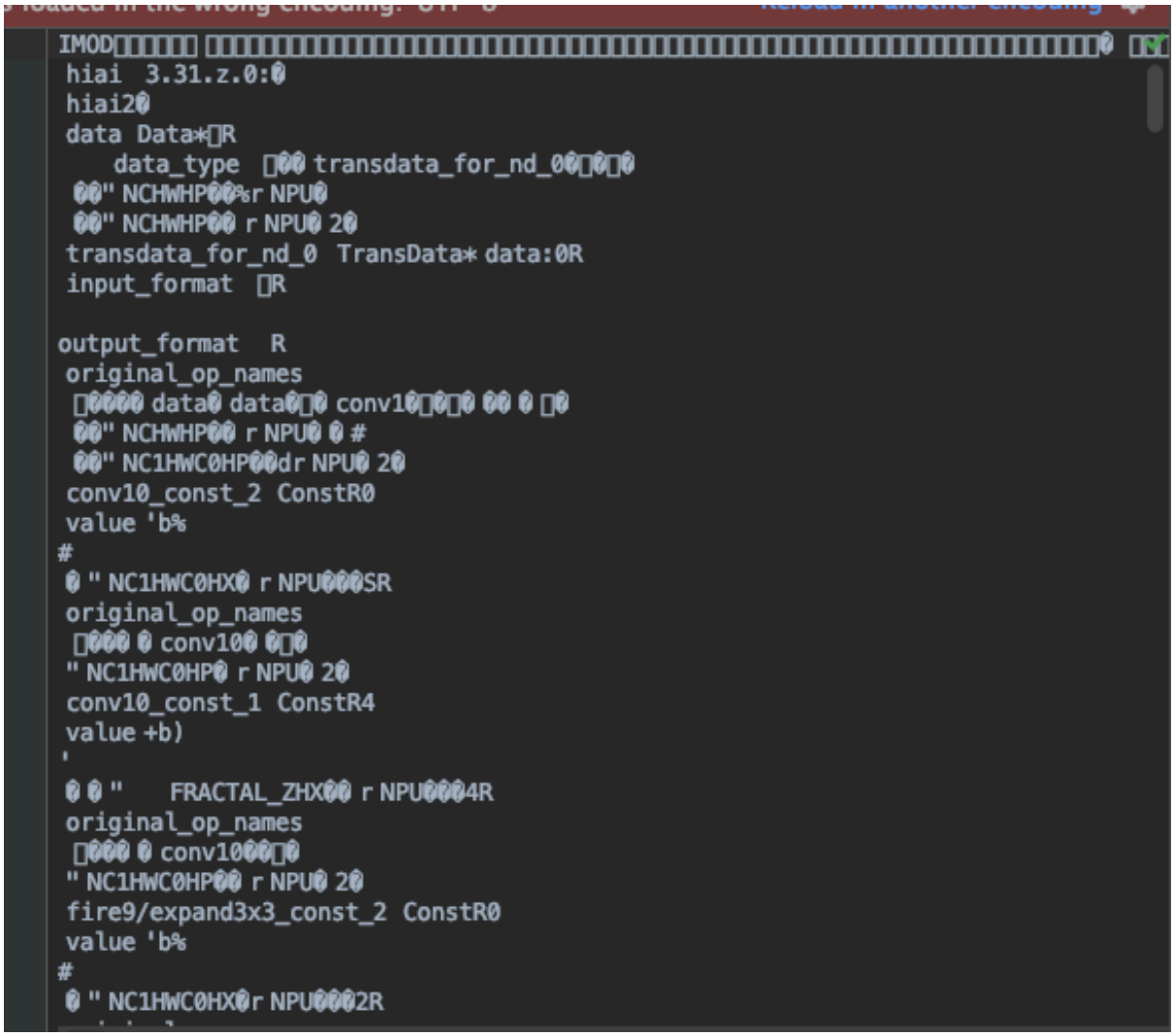
```
        ……..

        env->SetIntField(modelInfoObj,output_n_id,outputDimension[i]
[0].GetNumber());
    }
    return modelInfo;
}
```

重点是上面标红的代码段，这段代码是 load 整个模型，并且解析特征 OP 所使用的。具体跟踪一下代码：

```
shared_ptr<AiModelMngerClient> LoadModelSync(vector<string> names,
vector<string> modelPaths, vector<bool> Aipps)
{
    //创建一个 model 客户端
    shared_ptr<AiModelMngerClient> clientSync =
make_shared<AiModelMngerClient>();
    if (clientSync == nullptr)
    {
        LOGE("[HIAI_DEMO_SYNC] Model Manager Client make_shared error.");
        return nullptr;
    }
    // 初始化 客户端
    int ret = clientSync->Init(nullptr);
    if (ret != 0) {
        LOGE("[HIAI_DEMO_SYNC] Model Manager Init Failed.");
        return nullptr;
    }

    //创建一个模型 build 并且导入模型文件中的 op
    shared_ptr<AiModelBuilder> mcbuilder = make_shared<AiModelBuilder>
(clientSync);

    vector<shared_ptr<AiModelDescription>> model_desc;
    vector<MemBuffer*> resource;
    for (size_t i = 0; i < modelPaths.size(); ++i){
        string modelPath = modelPaths[I];

        ……..

        LOGE("[HIAI_DEMO_SYNC] loadModel %s IO Tensor.", desc-
>GetName().c_str());
        model_desc.push_back(desc);
    }
    // 讲导出的描述符塞入到 客户端中
    ret = clientSync->Load(model_desc);
    for (auto tmpBuffer : resource){
        mcbuilder->MemBufferDestroy(tmpBuffer);
    }
```

```
    inputDimension.clear();
    outputDimension.clear();
    input_tensor.clear();
    output_tensor.clear();
    // 解析 OP DIMS 到容器。
    for (size_t i = 0; i < names.size(); ++i) {
        string modelName = names[i];
        bool isUseAipp = Aipps[i];
        LOGI("[HIAI_DEMO_SYNC] Get model %s IO Tensor. Use AIPP %d",
modelName.c_str(), isUseAipp);
        vector<TensorDimension> inputDims, outputDims;

        ……

        inputDimension.push_back(inputDims);
        outputDimension.push_back(outputDims);

        ……

        input_tensor.push_back(inputTensors);

        if (input_tensor.size() == 0) {
            LOGE("[HIAI_DEMO_SYNC] input_tensor.size() == 0");
            return nullptr;
        }

        for (auto out_dim : outputDims) {
            shared_ptr<AiTensor> output = make_shared<AiTensor>();
            ret = output->Init(&out_dim);
            if(ret != 0){
                LOGE("[HIAI_DEMO_SYNC] model %s AiTensor Init
failed(output).", modelName.c_str());
                return nullptr;
            }
            outputTensors.push_back(output);
        }

        output_tensor.push_back(outputTensors);

        if (output_tensor.size() == 0) {
            LOGE("[HIAI_DEMO_SYNC] output_tensor.size() == 0");
            return nullptr;
        }
    }

    return clientSync;
}
```

打开模型文件，以字符来看，可以隐约看到一些内容。



基本上模型有hiai 版本号等信息，还有 opname 的说明。上层还暴露了一个接口是`runModelAsync` 方法，这个方法的输入参数一个是对应从JNI 层解析好并存储于 Java 的 mediainfo list，另一个是输入的 bytes，以及一个回调接口。还是看 JNI 的代码：

```
extern "C"
JNIEXPORT void JNICALL
Java_com_huawei_hiaidemo_utils_ModelManager_runModelAsync(JNIEnv *env,
jclass type, jobject modelInfo, jobject bufList, jobject callbacks)
{
    // 解析模型
    // check params
    if(env == nullptr)
    {
        LOGE("[HIAI_DEMO_ASYNC] runModelAsync env is null");
        return;
    }
    jclass ModelInfo = env->GetObjectClass(modelInfo);


    ` ` ` `
```

```
    //run
    LOGI("[HIAI_DEMO_ASYNC] INPUT NCHW : %d %d %d %d." , inputDimension[0]
[0].GetNumber(), inputDimension[0][0].GetChannel(), inputDimension[0]
[0].GetHeight(), inputDimension[0][0].GetWidth());
    LOGI("[HIAI_DEMO_ASYNC] OUTPUT NCHW : %d %d %d %d." ,
outputDimension[0][0].GetNumber(), outputDimension[0][0].GetChannel(),
outputDimension[0][0].GetHeight(), outputDimension[0][0].GetWidth());

    auto input_tensor0 = findInputTensor(vecIndex);

    for(int i = 0;i < listLength;i++){
        jbyteArray buf_ = (jbyteArray)(env->CallObjectMethod(bufList,
listGet, i));
        if (buf_ == nullptr)
        {
            LOGE("[HIAI_DEMO_ASYNC] buf_ is nullptr.");
            return;
        }
        jbyte *dataBuff = nullptr;
        int databuffsize = 0;
        dataBuff = env->GetByteArrayElements(buf_, nullptr);
        databuffsize = env->GetArrayLength(buf_);

        if(((*input_tensor0)[i]->GetSize() != databuffsize))
        {
            LOGE("[HIAI_DEMO_ASYNC] input->GetSize(%d) != databuffsize(%d)
",(*input_tensor0)[i]->GetSize(),databuffsize);
            return ;
        }
        memmove((*input_tensor0)[i]->GetBuffer(), dataBuff,
(size_t)databuffsize);
        env->ReleaseByteArrayElements(buf_, dataBuff, 0);
    }

    // 创建 AI Context
    AiContext context;
    string key = "model_name";
    string value = modelName;
    value += ".om";
    context.AddPara(key, value);
    LOGI("[HIAI_DEMO_ASYNC] JNI runModel modelname:%s", value.c_str());

    // AI client Process 模型
    int istamp = 0;
    gettimeofday(&tpstart, nullptr);
    int ret = mclientAsync->Process(context, *input_tensor0,
output_tensor_vec[vecIndex], 300, istamp);
    if (ret != 0)
    {
        LOGE("[HIAI_DEMO_ASYNC] Runmodel Failed! ret=%d.",ret);
```

```
        return ;
    }
    LOGE("[HIAI_DEMO_ASYNC] Runmodel Succ! istamp=%d.",istamp);

    // 输出结果
    std::unique_lock<std::mutex> lock(mutex_map);
    map_input_tensor.insert(pair<int32_t,vector<shared_ptr<AiTensor>>*>
(istamp, input_tensor0));
    condition_.notify_all();

    env->ReleaseStringUTFChars(modelname, modelName);
}
```

上述代码分析，其主要的核心代码在`static shared_ptr<AiModelMngerClient> mclientAsync`这个类中。这个类主要定义在 HiAIModelManagerService.h 中。

这个头文件中，主要定义了两个类，一个是 AIModelBuilder，一个是 AiModelMngerClient。接口方法如下：

```
/*
*@file HiAiModelManagerService.h
*
* Copyright (C) 2019. Huawei Technologies Co., Ltd. All rights reserved.
*
*/
#ifndef __AI_MODEL_MANGER_SERVICE_H__
#define __AI_MODEL_MANGER_SERVICE_H__

#include <string>
#include <vector>
#include <map>
#include <mutex>
#include "HiAiModelManagerType.h"
#include "HiAiAippPara.h"

namespace hiai {
class AiModelMngerClientImpl;
class AiModelMngerClient;
class AiModelBuilderImpl;
class AiModelBuilder {
public:
    AiModelBuilder(std::shared_ptr<AiModelMngerClient> client = nullptr);

    virtual ~AiModelBuilder();

    /*
    * @brief OM离线模型在线编译接口
    * @param [in] pinputMemBuffer 输入的OM离线模型buffer
    * @param [in] poutputModelBuffer 输出模型buffer
    * @param [out] poutputModelSize 输出模型大小
    * @return AIStatus::AI_SUCCESS 成功
```

```
    * @return Others 失败
    */
    AIStatus BuildModel(const std::vector<MemBuffer *> &pinputMemBuffer,
MemBuffer *poutputModelBuffer, uint32_t &poutputModelSize);

    /*
    * @brief 从文件读取OM离线模型proto信息
    * @param [in] path, 模型文件路径
    * @return MemBuffer * proto信息存储地址
    * @return nullptr 获取失败
    */
    MemBuffer* ReadBinaryProto(const std::string path);

    /*
    * @brief 从内存读取OM离线模型proto信息
    * @param [in] data, OM离线模型内存地址
    * @param [in] size, OM离线模型内存存储大小
    * @return MemBuffer * proto信息存储地址
    * @return nullptr 获取失败
    */
    MemBuffer* ReadBinaryProto(void* data, uint32_t size);

    /*
    * @brief 为输入OM离线模型用户内存buffer创建通用MemBuffer
    * @param [in] data, 模型用户内存地址
    * @param [in] size, 模型内存存储大小
    * @return MemBuffer * proto信息存储地址
    * @return nullptr 获取失败
    */
    MemBuffer* InputMemBufferCreate(void *data, uint32_t size);

    /*
    * @brief 为输入OM离线模型从文件创建MemBuffer
    * @param [in] path 文件路径
    * @return MemBuffer * 创建的输入MemBuffer内存指针
    * @return nullptr 创建失败
    */
    MemBuffer* InputMemBufferCreate(const std::string path);

    /*
    * @brief 为在线编译输出模型创建MemBuffer
    * @param [in] framework 模型平台类型
    * @param [in] pinputMemBuffer 输入的OM离线模型buffer
    * @return MemBuffer * 创建的输出模型MemBuffer内存指针
    * @return nullptr 创建失败
    */
    MemBuffer* OutputMemBufferCreate(const int32_t framework, const
std::vector<MemBuffer *> &pinputMemBuffer);
    /*
    * @brief 注销MemBuffer内存，通过上述MemBuffer申请的内存最终都需要由此接口进行释
```

```
放
    * @param [in] membuf, 创建的MemBuffer内存
    * @return void
    */
    void MemBufferDestroy(MemBuffer *membuf);

    /*
    * @brief 将模型buffer导出到文件
    * @param [in] membuf, 存储离线模型信息内存指针
    * @param [in] pbuildSize, 离线模型大小
    * @param [in] pbuildPath, 离线模型导出文件存储路径
    * @return AIStatus::AI_SUCCESS 导出成功
    * @return Others 导出失败
    */
    AIStatus MemBufferExportFile(MemBuffer *membuf, const uint32_t
pbuildSize, const std::string pbuildPath);

private:
    std::shared_ptr<AiModelBuilderImpl> builderImpl_;
};

class AiModelMngerClient {
public:
    AiModelMngerClient();
    virtual ~AiModelMngerClient();

    /*
    * @brief 初始化接口
    * @param [in] listener 监听接口, 设置为nullptr时为同步调用, 否则为异步
    * @return AIStatus::AI_SUCCESS 成功
    * @return Others 失败
    */
    AIStatus Init(std::shared_ptr<AiModelManagerClientListener> listener);

    /*
    * @brief 加载模型
    * @param [in] pmodelDesc 模型信息
    * @return AIStatus::AI_SUCCESS 成功
    * @return AIStatus::AI_INVALID_API 失败, 表示设备不支持NPU
    * @return Others 失败
    */
    AIStatus Load(std::vector<std::shared_ptr<AiModelDescription>>
&pmodelDesc);

    /*
    * @brief 模型处理接口, 运行加载模型的模型推理
    * @param [in] context, 模型运行上下文, 必须带model_name字段
    * @param [in] pinputTensor, 模型输入节点tensor信息
    * @param [in] poutputTensor, 模型输出节点tensor信息
    * @param [in] timeout, 推理超时时间
```

```
    * @param [in] piStamp 异步返回标识，基于该标识和模型名称做回调索引
    * @return AIStatus::AI_SUCCESS 成功
    * @return Others 失败
    */
    AIStatus Process(AiContext &context,
std::vector<std::shared_ptr<AiTensor>> &pinputTensor,
std::vector<std::shared_ptr<AiTensor>> &poutputTensor, uint32_t timeout,
int32_t &piStamp);

    /*
    * @brief 模型兼容性检查接口
    * @param [in] pmodelDesc，模型描述
    * @param [out] pisModelCompatibility，兼容性检查标识
    * @return AIStatus::AI_SUCCESS 兼容性检查通过
    * @return Others 兼容性检查失败
    */
    AIStatus CheckModelCompatibility(AiModelDescription &pmodelDesc, bool
&pisModelCompatibility);

    /*
    * @brief 获取模型输入输出tensor信息
    * @param [in] pmodelName，模型名
    * @param [out] pinputTensor 输出参数，存储模型输入节点tensor信息
    * @param [out] poutputTensor 输出参数，存储模型输出节点tensor信息
    * @return AIStatus::AI_SUCCESS 获取成功
    * @return Others 获取失败
    */
    AIStatus GetModelIOTensorDim(const std::string& pmodelName,
std::vector<TensorDimension>& pinputTensor, std::vector<TensorDimension>&
poutputTensor);

    /*
    * @brief 获取模型AIPP 配置信息
    * @param [in] pmodelName，模型名
    * @param [out] aippPara 输出参数，模型中的AIPP配置参数
    * @return AIStatus::AI_SUCCESS 获取成功
    * @return Others 获取失败
    */
    AIStatus GetModelAippPara(const std::string& modelName,
std::vector<std::shared_ptr<AippPara>>& aippPara);

    /*
    * @brief 获取模型对应输入的AIPP 配置信息
    * @param [in] pmodelName，模型名
    * @param [in] index，输入下标
    * @param [out] aippPara 输出参数，模型对应输入的AIPP配置参数
    * @return AIStatus::AI_SUCCESS 获取成功
    * @return Others 获取失败
    */
    AIStatus GetModelAippPara(const std::string& modelName, uint32_t index,
```

```
std::vector<std::shared_ptr<AippPara>>& aippPara);

    /*
    * @brief 获取HiAI版本
    * @return char * HiAI版本
    * @return nullptr 获取失败
    */
    char * GetVersion();

    /*
    * @brief 卸载模型
    * @return AIStatus::AI_SUCCESS 卸载成功
    * @return Others 卸载失败
    */
    AIStatus UnLoadModel();
private:
    friend class AiModelBuilderImpl;
    std::shared_ptr<AiModelMngerClientImpl> clientImpl_;
};

} //end namespace hiai

#endif
```

主体的使用方法是 AiModelMngerClient ，这个类提供了大部分需要用到的接口，比如 load 模型，预测等方法。

其余的提供了一个 Type 的定义，里面包含了 Tensor 的模型功能接口，还有一个图形、OP、运算符的相关接口。感兴趣可以看下。