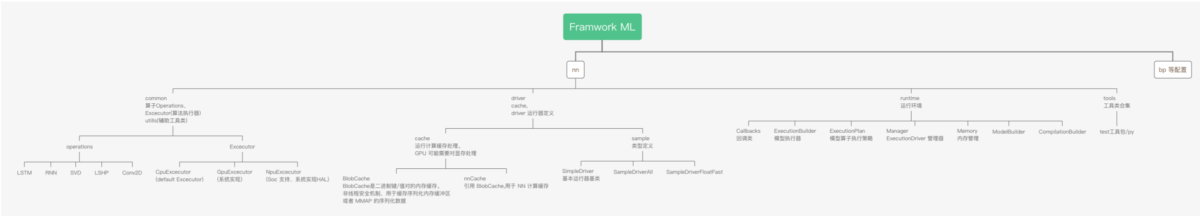
Android源码学习笔记--NNAPI

- ## 简介

NNAPI 是 Android 用于机器学习的 API 简称。主要定位机器学习、机器算法导出的模型，在 Android运行相应的 Op算子，并且选择不同的运行策略实现加速的逻辑推理的过程。整体的软件结构包括如下：



- ## Executor 与 Operations

所有硬件平台的算法固化模式，都是以 FPGA / DSP 固化代码实现硬件算子单元，以此达到加速运行算法的目的。比如Open CL / Open GL,上层业务方或者系统集成 SOC 方都仅仅是使用芯片提供的基础接口能力，实现自己的算法。这里的概念有两个，基本算子单元（Operations）和运行器（Executor）。

Executor：可以简单的理解是算法的执行器。

Operations: 换而言之是基本的算子集成计算单元。

一般的模型都会配置特殊的特征矩阵、计算的图、计算基本算子等等。机器学习的端执行主要是翻译模型中保存的计算图、向量、特征，并把输入的序列化数据，再放入到运行环境中的模型执行器中按图运算。最终得出计算结果的一个过程。

这里的 Executor 是一个算子执行器。

```
// Information we maintain about each operand during execution that
// may change during execution.
struct RunTimeOperandInfo {
    // TODO Storing the type here is redundant, as it won't change during
execution.
    OperandType type;
    // The type and dimensions of the operand.  The dimensions can
    // change at runtime.  We include the type because it's useful
    // to pass together with the dimension to the functions implementing
    // the operators.
    std::vector<uint32_t> dimensions;

    float scale;
    int32_t zeroPoint;
    // Where the operand's data is stored.  Check the corresponding
    // location information in the model to figure out if this points
    // to memory we have allocated for an temporary operand.
    uint8_t* buffer;
```

```
    // The length of the buffer.
    uint32_t length;
    // Whether this is a temporary variable, a model input, a constant,
etc.
    OperandLifeTime lifetime;
    // Keeps track of how many operations have yet to make use
    // of this temporary variable.  When the count is decremented to 0,
    // we free the buffer.  For non-temporary variables, this count is
    // always 0.
    uint32_t numberOfUsesLeft;

    Shape shape() const {
        return Shape{.type = type, .dimensions = dimensions, .scale =
scale, .offset = zeroPoint};
    }
};

// Used to keep a pointer to each of the memory pools.
//
// In the case of an "mmap_fd" pool, owns the mmap region
// returned by getBuffer() -- i.e., that region goes away
// when the RunTimePoolInfo is destroyed or is assigned to.
class RunTimePoolInfo {
public:
    // If "fail" is not nullptr, and construction fails, then set *fail =
true.
    // If construction succeeds, leave *fail unchanged.
    // getBuffer() == nullptr IFF construction fails.
    explicit RunTimePoolInfo(const hidl_memory& hidlMemory, bool* fail);

    explicit RunTimePoolInfo(uint8_t* buffer);

    // Implement move
    RunTimePoolInfo(RunTimePoolInfo&& other);
    RunTimePoolInfo& operator=(RunTimePoolInfo&& other);

    // Forbid copy
    RunTimePoolInfo(const RunTimePoolInfo&) = delete;
    RunTimePoolInfo& operator=(const RunTimePoolInfo&) = delete;

    ~RunTimePoolInfo() { release(); }

    uint8_t* getBuffer() const { return mBuffer; }

    bool update() const;

private:
    void release();
    void moveFrom(RunTimePoolInfo&& other);

    hidl_memory mHidlMemory;     // always used
```

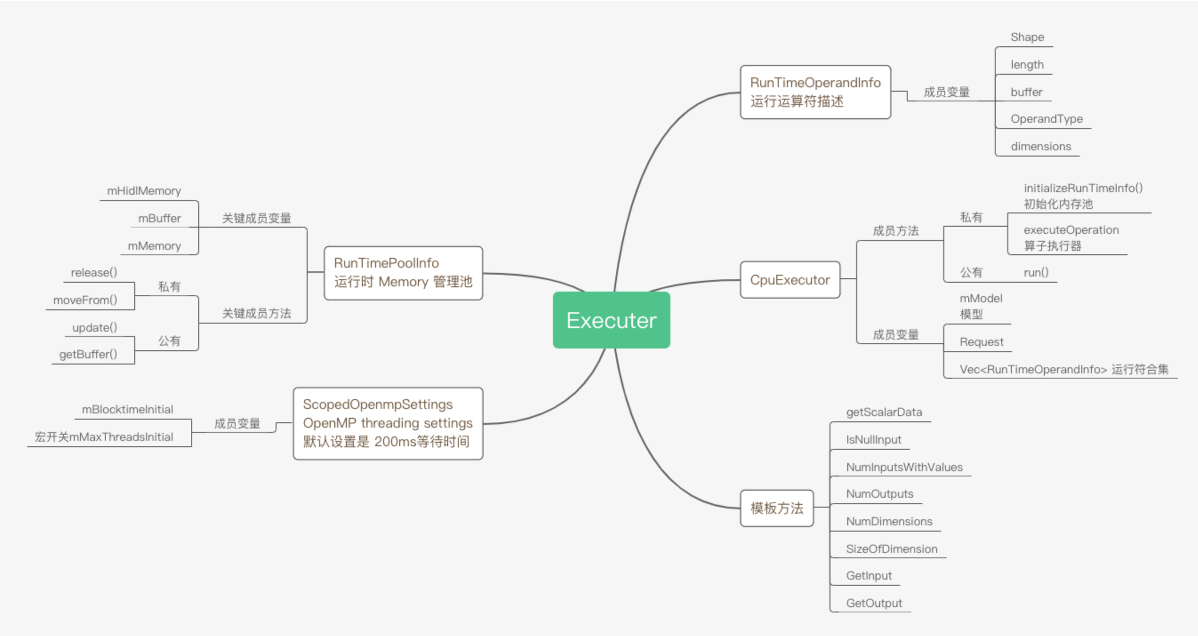
```
    uint8_t* mBuffer = nullptr;  // always used
    sp<IMemory> mMemory;         // only used when hidlMemory.name() ==
"ashmem"
};

bool setRunTimePoolInfosFromHidlMemories(std::vector<RunTimePoolInfo>*
poolInfos,
                                         const hidl_vec<hidl_memory>&
pools);

// This class is used to execute a model on the CPU.
class CpuExecutor {
public:
    // Executes the model. The results will be stored at the locations
    // specified in the constructor.
    // The model must outlive the executor.  We prevent it from being
modified
    // while this is executing.
    int run(const V1_0::Model& model, const Request& request,
            const std::vector<RunTimePoolInfo>& modelPoolInfos,
            const std::vector<RunTimePoolInfo>& requestPoolInfos);
    int run(const V1_1::Model& model, const Request& request,
            const std::vector<RunTimePoolInfo>& modelPoolInfos,
            const std::vector<RunTimePoolInfo>& requestPoolInfos);

private:
    bool initializeRunTimeInfo(const std::vector<RunTimePoolInfo>&
modelPoolInfos,
                               const std::vector<RunTimePoolInfo>&
requestPoolInfos);
    // Runs one operation of the graph.
    int executeOperation(const Operation& entry);
    // Decrement the usage count for the operands listed.  Frees the memory
    // allocated for any temporary variable with a count of zero.
    void freeNoLongerUsedOperands(const std::vector<uint32_t>& inputs);

    // The model and the request that we'll execute. Only valid while run()
    // is being executed.
    const Model* mModel = nullptr;
    const Request* mRequest = nullptr;

    // We're copying the list of all the dimensions from the model, as
    // these may be modified when we run the operatins.  Since we're
    // making a full copy, the indexes used in the operand description
    // stay valid.
    //    std::vector<uint32_t> mDimensions;
    // Runtime information about all the operands.
    std::vector<RunTimeOperandInfo> mOperands;
};

// Class for setting reasonable OpenMP threading settings. (OpenMP is used
```

```
by
// the Eigen matrix library.)
//
// Currently sets a low blocktime: the time OpenMP threads busy-wait for
more
// work before going to sleep. See b/79159165,
https://reviews.llvm.org/D18577.
// The default is 200ms, we set to 20ms here, see b/109645291. This keeps
the
// cores enabled throughout inference computation without too much extra
power
// consumption afterwards.
//
// The OpenMP settings are thread-local (applying only to worker threads
formed
// from that thread), see https://software.intel.com/en-us/node/522688 and
// http://lists.llvm.org/pipermail/openmp-dev/2016-July/001432.html. This
class
// ensures that within the scope in which an object is instantiated we use
the
// right settings (scopes may be nested), as long as no other library
changes
// them.  (Note that in current NNAPI usage only one instance is used in
the
// CpuExecutor thread).
//
// TODO(mikie): consider also setting the number of threads used. Using as
many
// threads as there are cores results in more variable performance: if we
don't
// get all cores for our threads, the latency is doubled as we wait for one
core
// to do twice the amount of work. Reality is complicated though as not all
// cores are the same. Decision to be based on benchmarking against a
// representative set of workloads and devices. I'm keeping the code here
for
// reference.
class ScopedOpenmpSettings {
public:
    ScopedOpenmpSettings();
    ~ScopedOpenmpSettings();
    DISALLOW_COPY_AND_ASSIGN(ScopedOpenmpSettings);
private:
    int mBlocktimeInitial;
#if NNAPI_LIMIT_CPU_THREADS
    int mMaxThreadsInitial;
#endif
};


namespace {
```

```
template <typename T>
T getScalarData(const RunTimeOperandInfo& info) {
  // TODO: Check buffer is at least as long as size of data.
  T* data = reinterpret_cast<T*>(info.buffer);
  return data[0];
}

inline bool IsNullInput(const RunTimeOperandInfo *input) {
    return input->lifetime == OperandLifeTime::NO_VALUE;
}

inline int NumInputsWithValues(const Operation &operation,
                               std::vector<RunTimeOperandInfo> &operands) {
  const std::vector<uint32_t> &inputs = operation.inputs;
  return std::count_if(inputs.begin(), inputs.end(),
                       [&operands](uint32_t i) {
                         return !IsNullInput(&operands[i]);
                       });
}

inline int NumOutputs(const Operation &operation) {
  return operation.outputs.size();
}

inline size_t NumDimensions(const RunTimeOperandInfo *operand) {
  return operand->shape().dimensions.size();
}

inline uint32_t SizeOfDimension(const RunTimeOperandInfo *operand, int i) {
  return operand->shape().dimensions[i];
}

inline RunTimeOperandInfo *GetInput(const Operation &operation,
                                    std::vector<RunTimeOperandInfo>
&operands,
                                    int index) {
  return &operands[operation.inputs[index]];
}

inline RunTimeOperandInfo *GetOutput(const Operation &operation,
                                     std::vector<RunTimeOperandInfo>
&operands,
                                     int index) {
  return &operands[operation.outputs[index]];
}

}  // anonymous namespace

} // namespace nn
} // namespace android
```

```
#endif // ANDROID_ML_NN_COMMON_CPU_EXECUTOR_H
```

Executor 中包含了几个定义：



这里有一个 RunTimePoolInfo 来管理内存，可能做过 Open CL和 Open GL的都清楚，在运算过程中，存在一步操作是将 CPU 的 Memory copy 到显存中，结果再从显示空间 copy 到内存。这样的过程主要是Android Kerrnl 和 user space 属于同一块 RAM 分区，显存实际用的物理地址也是外挂 RAM 的一个分区，GPU 工作所需的内存空间都会在显存空间中。同理，推理过程中，可能所需要的是不同的计算处理单元，也就由此设计了一个管理内存的工具便于管理模型执行过程中的内存。

原生的 Operations 支持以下几个，囊括了 简单经典的算子合集。如 RNN、SVD(奇异值分解算法)、LSTM(long short term memory(LSTM)，即LSTM算法全称)、Conv2D（2维数据卷积）等等。针对每个使用场景、一般而言，算法提供商都会有对应的算子实现容器。以 Conv2D cpu 版本为例，简单阐述下他具体的做法：

```
bool convFloat32(const float* inputData, const Shape& inputShape,
                 const float* filterData, const Shape& filterShape,
                 const float* biasData, const Shape& biasShape,
                 int32_t padding_left, int32_t padding_right,
                 int32_t padding_top, int32_t padding_bottom,
                 int32_t stride_width, int32_t stride_height,
                 int32_t activation,
                 float* outputData, const Shape& outputShape) {

    ANDROID_NN_CONV_PARAMETERS(float)

    float output_activation_min, output_activation_max;
    CalculateActivationRangeFloat(activation, &output_activation_min,
                                  &output_activation_max);
```

```
    // Prevent concurrent executions that may access the scratch buffer.
    std::unique_lock<std::mutex> lock(executionMutex);
    //调度算法 具体在 TensorFlow Lite Delegate Api 定义的 Conv 方法
    tflite::optimized_ops::Conv(
            inputData, convertShapeToDims(inputShape),
            filterData, convertShapeToDims(filterShape),
            biasData, convertShapeToDims(biasShape),
            stride_width, stride_height, paddingWidth, paddingHeight,
            output_activation_min, output_activation_max,
            outputData, convertShapeToDims(outputShape),
            im2colData, im2colDim);
    return true;
}
```

32位数据和 8位数据的卷积还是使用了 TensorFlow Lite Delegate API 所提供的 Conv 方法。

- ## BlobCache 与 nnCache

这两个 class 主要是 Cache管理类，首先看提供了那些 东西。
首先是操作内存的两个选择指令：

```
enum class Select {
        RANDOM,  // evict random entries
        LRU,     // evict least-recently-used entries

        DEFAULT = RANDOM,
    };
```

这里定义了两种内存操作方式，一种是 LRU 还有一种是随机的申请方式，默认是随机的使用方式。其次还有一个是 Capacity 枚举。

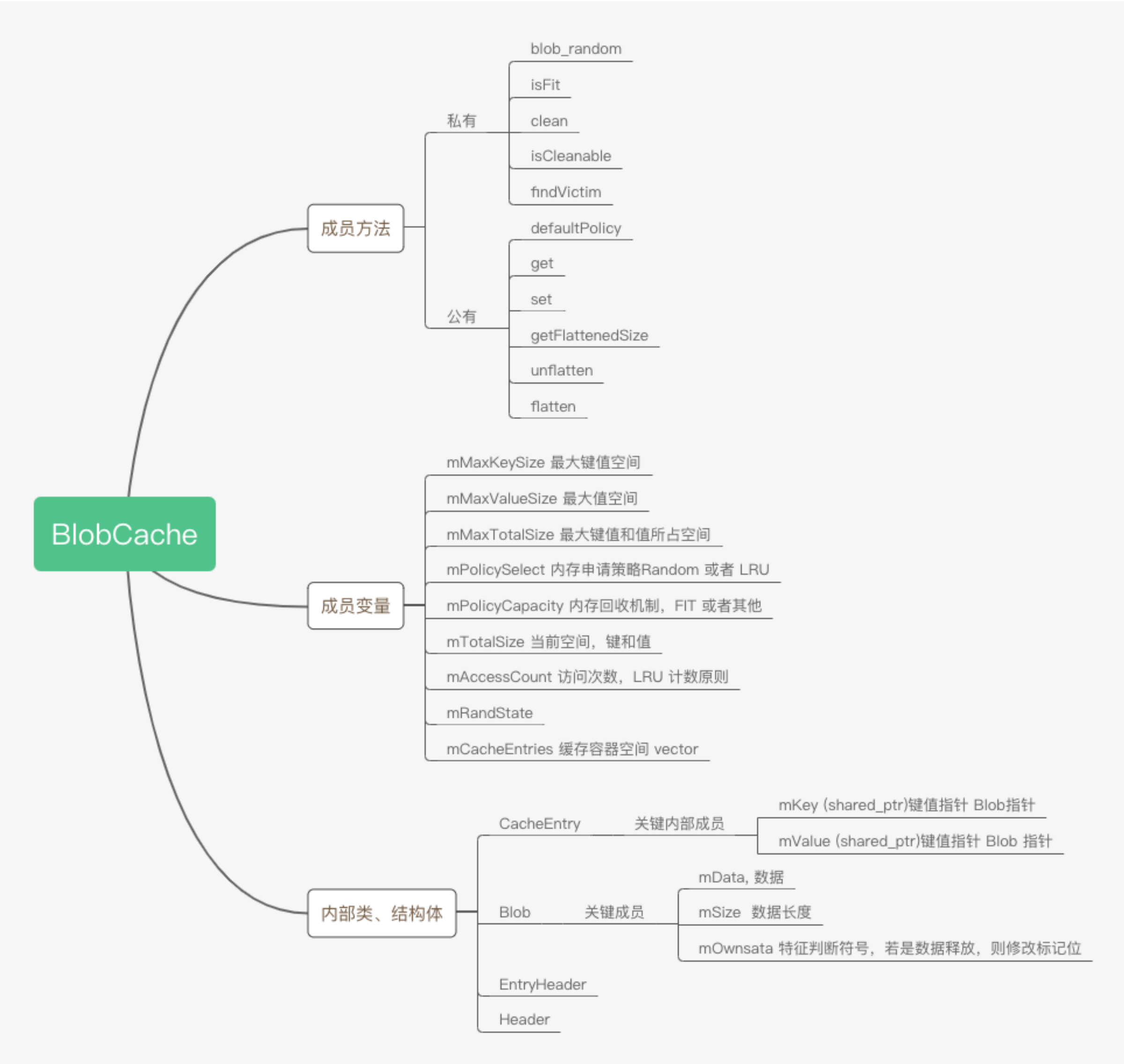
```
    enum class Capacity {
        // cut back to no more than half capacity; new/replacement
        // entry still might not fit
        HALVE,

        // cut back to whatever is necessary to fit new/replacement
        // entry
        FIT,

        // cut back to no more than half capacity and ensure that
        // there's enough space for new/replacement entry
        FIT_HALVE,

        DEFAULT = HALVE,
    };
```

这里是对内存申请的三种策略，砍掉一半内存还是看到非必要的内存，亦或者是砍掉

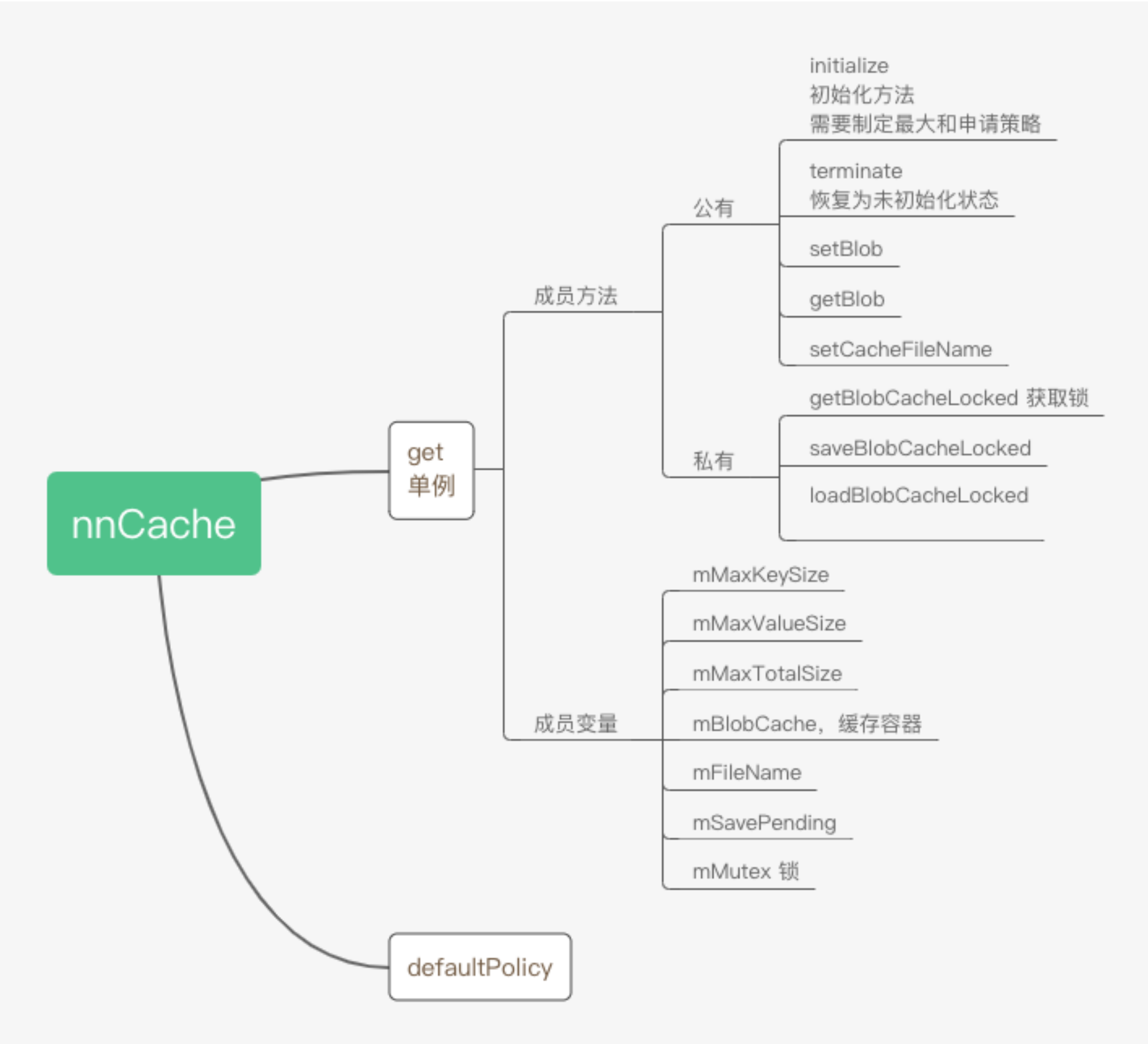
不超过一半的内存申请空间，以满足足够空间置换的方式。之后是一个内存清空的策略，在内存不足的情况下，如何清除缓存。

```
typedef std::pair<Select, Capacity> Policy;
static Policy defaultPolicy() { return Policy(Select::DEFAULT,
Capacity::DEFAULT); }
```

之后是 Blob 类和 CacheEntry以及Header之间的关系。



在 BlobCache类上层，是一个调用者NNCache，为了便于管理，这里被设计成为了一个单例类。里面定义了一些 BlobCache 的引用，提供申请内存、释放策略等接口函数。



- **sample 定义类**

重点看下 SampleDriver 这个类，在对比其他几个和这个的区别。SampleDriver 继承于 IDevice 类。内部提供了每个 Device 的基础OP算子支持函数列表。接下来过一遍：

```
// Driver 定义
class SampleDriver : public IDevice {
public:
    SampleDriver(const char* name) : mName(name) {}
    ~SampleDriver() override {}
    //获取
    Return<void> getCapabilities(getCapabilities_cb cb) override;
    Return<void> getSupportedOperations(const V1_0::Model& model,
                                        getSupportedOperations_cb cb)
override;
    Return<ErrorStatus> prepareModel(const V1_0::Model& model,
                                     const sp<IPreparedModelCallback>&
```

```
callback) override;
    Return<ErrorStatus> prepareModel_1_1(const V1_1::Model& model,
ExecutionPreference preference,
                                         const sp<IPreparedModelCallback>&
callback) override;
    Return<DeviceStatus> getStatus() override;

    // Starts and runs the driver service.  Typically called from main().
    // This will return only once the service shuts down.
    int run();
protected:
    std::string mName;
};

// fixme 模型定义
class SamplePreparedModel : public IPreparedModel {
public:
    SamplePreparedModel(const Model& model) : mModel(model) {}
    ~SamplePreparedModel() override {}
    bool initialize();
    Return<ErrorStatus> execute(const Request& request,
                                const sp<IExecutionCallback>& callback)
override;

private:
    void asyncExecute(const Request& request, const sp<IExecutionCallback>&
callback);

    Model mModel;
    std::vector<RunTimePoolInfo> mPoolInfos;
};

} // namespace sample_driver
} // namespace nn
} // namespace android
```

可以从上面的头文件发现，可以获取相应的支持算子列表、导入模型、运行函数的基本函数。并且模型预测函数的代码看，这里是一个异步的设计，和 CL 的调用方式大同小异。

```
Return<ErrorStatus> SamplePreparedModel::execute(const Request& request,
                                                 const
sp<IExecutionCallback>& callback) {
    VLOG(DRIVER) << "execute(" << SHOW_IF_DEBUG(toString(request)) << ")";
    if (callback.get() == nullptr) {
        LOG(ERROR) << "invalid callback passed to execute";
        return ErrorStatus::INVALID_ARGUMENT;
    }
    if (!validateRequest(request, mModel)) {
        callback->notify(ErrorStatus::INVALID_ARGUMENT);
```

```
        return ErrorStatus::INVALID_ARGUMENT;
    }

    // This thread is intentionally detached because the sample driver
service
    // is expected to live forever.
    std::thread([this, request, callback]{ asyncExecute(request, callback);
}).detach();

    return ErrorStatus::NONE;
}
```

SampleDriverAll / SampleDriverFloatFast / SampleDriverFloatSlow 等等都继承于 SampleDriver， 但是他们挂钩底层 kernel 的服务是存在不同的，举例 Fast和 Slow两个 Driver.

```
service neuralnetworks_hal_service_sample_float_fast
/vendor/bin/hw/android.hardware.neuralnetworks@1.1-service-sample-float-
fast
    class hal
    user system
    group system


service neuralnetworks_hal_service_sample_float_slow
/vendor/bin/hw/android.hardware.neuralnetworks@1.1-service-sample-float-
slow
    class hal
    user system
    group system
```

其余接口和功能方法都类似。

- **runtime**

runtime 部分可以拆分几个部分，第一是 Execution和 Model 部分，第二个是 Memory 部分，之后是 Manager 部分。

- **Runtime--Execution 和 Model.**

首先是 ExecutionBuilder 类，先屡下整个类结构，ExecutionBuilder 类内部声明了结构体，ModelArgumentInfo / ExecutionBuilder / StepExecutor 三个部分。

```
// TODO move length out of DataLocation
struct ModelArgumentInfo {
    // Whether the argument was specified as being in a Memory, as a
pointer,
    // has no value, or has not been specified.
    // If POINTER then:
```

```
    //   locationAndLength.length is valid.
    //   dimensions is valid.
    //   buffer is valid
    // If MEMORY then:
    //   locationAndLength.{poolIndex, offset, length} is valid.
    //   dimensions is valid.
    enum { POINTER, MEMORY, HAS_NO_VALUE, UNSPECIFIED } state =
UNSPECIFIED;
    DataLocation locationAndLength;
    std::vector<uint32_t> dimensions;
    void* buffer;

    int setFromPointer(const Operand& operand, const
ANeuralNetworksOperandType* type, void* buffer,
                       uint32_t length);
    int setFromMemory(const Operand& operand, const
ANeuralNetworksOperandType* type,
                      uint32_t poolIndex, uint32_t offset, uint32_t
length);
    int setFromTemporaryMemory(const Operand& operand, uint32_t poolIndex,
uint32_t offset);
    int updateDimensionInfo(const Operand& operand, const
ANeuralNetworksOperandType* newType);
};
```

ModelArgumentInfo 是一个模型容器类，里面包括了数据的出处和算法 Operand 类型。接下来是 ExecutionBuilder主类。

```
class ExecutionBuilder {
    friend class StepExecutor;
public:
    ExecutionBuilder(const CompilationBuilder* compilation);

    int setInput(uint32_t index, const ANeuralNetworksOperandType* type,
const void* buffer,
                 size_t length);
    int setInputFromMemory(uint32_t index, const
ANeuralNetworksOperandType* type,
                           const Memory* memory, size_t offset, size_t
length);
    int setOutput(uint32_t index, const ANeuralNetworksOperandType* type,
void* buffer,
                  size_t length);
    int setOutputFromMemory(uint32_t index, const
ANeuralNetworksOperandType* type,
                            const Memory* memory, size_t offset, size_t
length);
    int startCompute(sp<ExecutionCallback>* synchronizationCallback);

    const ModelBuilder* getModel() const { return mModel; }

private:
```

```
    const ModelBuilder* mModel;
    const ExecutionPlan* mPlan;

    // This is a DeviceManager::kPartitioning* value captured from
    // CompilationBuilder when the ExecutionBuilder is constructed.
    uint32_t mPartitioning;

    // The information we'll send to the driver about the inputs and
outputs.
    // Note that we build this in two steps:
    // 1. As the arguments are specified, set the corresponding mInputs or
mOutputs element.
    //    If set from a pointer, don't set the location in the
RequestArgument but store it
    //    instead in mInputBuffers or mOutputBuffers.
    // 2. Once we have all the inputs and outputs, if needed, allocate
shared memory for
    //    the m*Buffers entries.  Copy the input values into the shared
memory.
    // We do this to avoid creating a lot of shared memory objects if we
have a lot of
    // parameters specified via pointers.  We also avoid copying in the
case where
    // some of the nodes will interpreted on the CPU anyway.
    std::vector<ModelArgumentInfo> mInputs;
    std::vector<ModelArgumentInfo> mOutputs;
    MemoryTracker mMemories;
};
```

内部定义了输入和输出的函数方法，并且有一个开始动作的方法。存储了两个容器，一个是输入 ModelArgumentInfo容器，还有一个是输出的 ModelArgumentInfo 容器。最后是定义了执行器的策略计划类和模型 build 类。这里重点看下 startCompute 方法。首先是检查输入参数和输出参数是否合法。

```

int ExecutionBuilder::startCompute(sp<ExecutionCallback>*
synchronizationCallback) {
    *synchronizationCallback = nullptr;

    // TODO validate that we have full types for all inputs and outputs,
    // that the graph is not cyclic,

    for (auto& p : mInputs) {
        if (p.state == ModelArgumentInfo::UNSPECIFIED) {
            LOG(ERROR) << "ANeuralNetworksExecution_startCompute not all
inputs specified";
            return ANEURALNETWORKS_BAD_DATA;
        }
    }
    for (auto& p : mOutputs) {
```

```
        if (p.state == ModelArgumentInfo::UNSPECIFIED) {
            LOG(ERROR) << "ANeuralNetworksExecution_startCompute not all
outputs specified";
            return ANEURALNETWORKS_BAD_DATA;
        }
    }
```

之后分两个路线，如果定义了禁用分区执行选项，则使用 CPU，否则利用策略执行。

```
#ifndef DISABLE_PARTITIONED_EXECUTION
    {
        // TODO: Remove the non-plan-based path once we've fully integrated
ExecutionPlan
        // with the compilation and execution phases of the NN API?  Or
retain that path
        // as a fallback in the case of partitioning failure?
        //
        // TODO: Entire plan-based-path should run in an asynchronous
thread --
        // take the asynchronous thread logic out of startComputeOnCpu()
and use
        // it to wrap the plan-based-path.
        if (mPartitioning > 0) {
            const bool allowFallback =
DeviceManager::partitioningAllowsFallback(mPartitioning);
            std::shared_ptr<ExecutionPlan::Controller> controller = mPlan-
>makeController(this);
            if (controller == nullptr) {
                if (!allowFallback) {
                    return ANEURALNETWORKS_OP_FAILED;
                }
            } else {
                // TODO: use a thread pool

                // Prepare the callback for asynchronous execution.
                // sp<ExecutionCallback> object is returned when the
                // execution has been successfully launched, otherwise a
                // nullptr is returned.  The executionCallback is
                // abstracted in the NN API as an "event".
                sp<ExecutionCallback> executionCallback = new
ExecutionCallback();
                std::thread thread(asyncStartComputePartitioned, this,
mPlan, controller,
                                   allowFallback,
                                   executionCallback);
                executionCallback->bind_thread(std::move(thread));
                *synchronizationCallback = executionCallback;
                return ANEURALNETWORKS_NO_ERROR;
            }
        }
```

```
    }

    //过滤

    // Run on the CPU.
    VLOG(EXECUTION) << "ExecutionBuilder::startCompute (without plan) on
CPU";
    StepExecutor executor(this, mModel,
                          nullptr /* no VersionedIDevice, so CPU */,
                          nullptr /* no IPreparedModel */);
    executor.mapInputsAndOutputsTrivially();
    return executor.startCompute(synchronizationCallback);
```

否则执行按照分区策略走，代码：

```
#else
    {
        // Find a driver that can handle all the operations.
        // TODO: Does not handle CPU fallback (which is tricky because
        //       StepExecutor::startCompute() is designed as
        //       asynchronous).
        // TODO: Does not actually behave asynchronously (because
        //       StepExecutor::startCompute() isn't actually asynchronous
        //       on a device as opposed to a CPU).
        Model hidlModel;
        mModel->setHidlModel(&hidlModel);
        const std::vector<std::shared_ptr<Device>>& devices =
DeviceManager::get()->getDrivers();
        for (const auto& device : devices) {
            hidl_vec<bool> supports;
            VLOG(EXECUTION) << "Checking " << device->getName();
            device->getSupportedOperations(hidlModel, &supports);
            if (std::find(supports.begin(), supports.end(), false) ==
supports.end()) {
                VLOG(EXECUTION) << "ExecutionBuilder::startCompute (without
plan) on " << device->getName();
                StepExecutor executor(this, mModel, device->getInterface(),
                                      nullptr /* no IPreparedModel, so
compile */);
                executor.mapInputsAndOutputsTrivially();
                return executor.startCompute(synchronizationCallback);
            }
        }
    }
```

这里我们开启一个新的类，也就是 ExecutionBuilder 中的另一个成员类方法。StepExecutor类，看到 StepExecutor 中的 startCompute 方法。

```
int StepExecutor::startCompute(sp<ExecutionCallback>*
synchronizationCallback) {
    if (VLOG_IS_ON(EXECUTION)) {
```

```
        logArguments("input", mInputs);
        logArguments("output", mOutputs);
    }
    if (mDriver == nullptr) {
        return startComputeOnCpu(synchronizationCallback);
    } else {
        return startComputeOnDevice(synchronizationCallback);
    }
}
```

也就是说默认走的是 CPU，如果有其它逻辑计算单元器件接入到系统，则会使用 startComputerOnDevice 方法。分开走进这两个方法，以 CPU 为例：

```
int StepExecutor::startComputeOnCpu(sp<ExecutionCallback>*
synchronizationCallback) {
    // TODO: use a thread pool
    //设置一个空的模型容器
    Model model;
    mModel->setHidlModel(&model);

    // Prepare the callback for asynchronous execution.
sp<ExecutionCallback>
    // object is returned when the execution has been successfully
launched,
    // otherwise a nullptr is returned. The executionCallback is abstracted
in
    // the NN API as an "event".
    sp<ExecutionCallback> executionCallback = new ExecutionCallback();
    *synchronizationCallback = nullptr;

    std::vector<RunTimePoolInfo> modelPoolInfos;
    if (!setRunTimePoolInfosFromHidlMemories(&modelPoolInfos, model.pools))
{
        return ANEURALNETWORKS_UNMAPPABLE;
    }
    // 创建内存
    std::vector<RunTimePoolInfo> requestPoolInfos;
    requestPoolInfos.reserve(mMemories.size());
    bool fail = false;
    for (const Memory* mem : mMemories) {
        requestPoolInfos.emplace_back(mem->getHidlMemory(), &fail);
    }
    if (fail) {
        return ANEURALNETWORKS_UNMAPPABLE;
    }
    // Create as many pools as there are input / output.
    auto fixPointerArguments = [&requestPoolInfos]
(std::vector<ModelArgumentInfo>& argumentInfos) {
        for (ModelArgumentInfo& argumentInfo : argumentInfos) {
            if (argumentInfo.state == ModelArgumentInfo::POINTER) {
```

```
                argumentInfo.locationAndLength.poolIndex =
                            static_cast<uint32_t>(requestPoolInfos.size());
                argumentInfo.locationAndLength.offset = 0;
                requestPoolInfos.emplace_back(static_cast<uint8_t*>
(argumentInfo.buffer));
            }
        }
    };
    // 解析算法
    fixPointerArguments(mInputs);
    fixPointerArguments(mOutputs);

    Request request;
    setRequestArgumentArray(mInputs, &request.inputs);
    setRequestArgumentArray(mOutputs, &request.outputs);

    // 执行
    // TODO: should model be moved with a std::cref?
    std::thread thread(asyncStartComputeOnCpu, model, std::move(request),
                       std::move(modelPoolInfos),
std::move(requestPoolInfos),
                       executionCallback);
    executionCallback->bind_thread(std::move(thread));

    *synchronizationCallback = executionCallback;
    return ANEURALNETWORKS_NO_ERROR;
}
```

接着走进 asyncStartComputerOnCpu方法。

```
static void asyncStartComputeOnCpu(const Model& model, const Request&
request,
                                   const std::vector<RunTimePoolInfo>&
modelPoolInfos,
                                   const std::vector<RunTimePoolInfo>&
requestPoolInfos,
                                   const sp<IExecutionCallback>&
executionCallback) {
    CpuExecutor executor;
    int err = executor.run(model, request, modelPoolInfos,
requestPoolInfos);
    executionCallback->notify(convertResultCodeToErrorStatus(err));
}
```

自此，就完成了整个 Cpu 计算模型逻辑推导的全部过程。换额外 计算单元的矢量看看。

```
int StepExecutor::startComputeOnDevice(sp<ExecutionCallback>*
synchronizationCallback) {
    nnAssert(mDriver != nullptr);
```

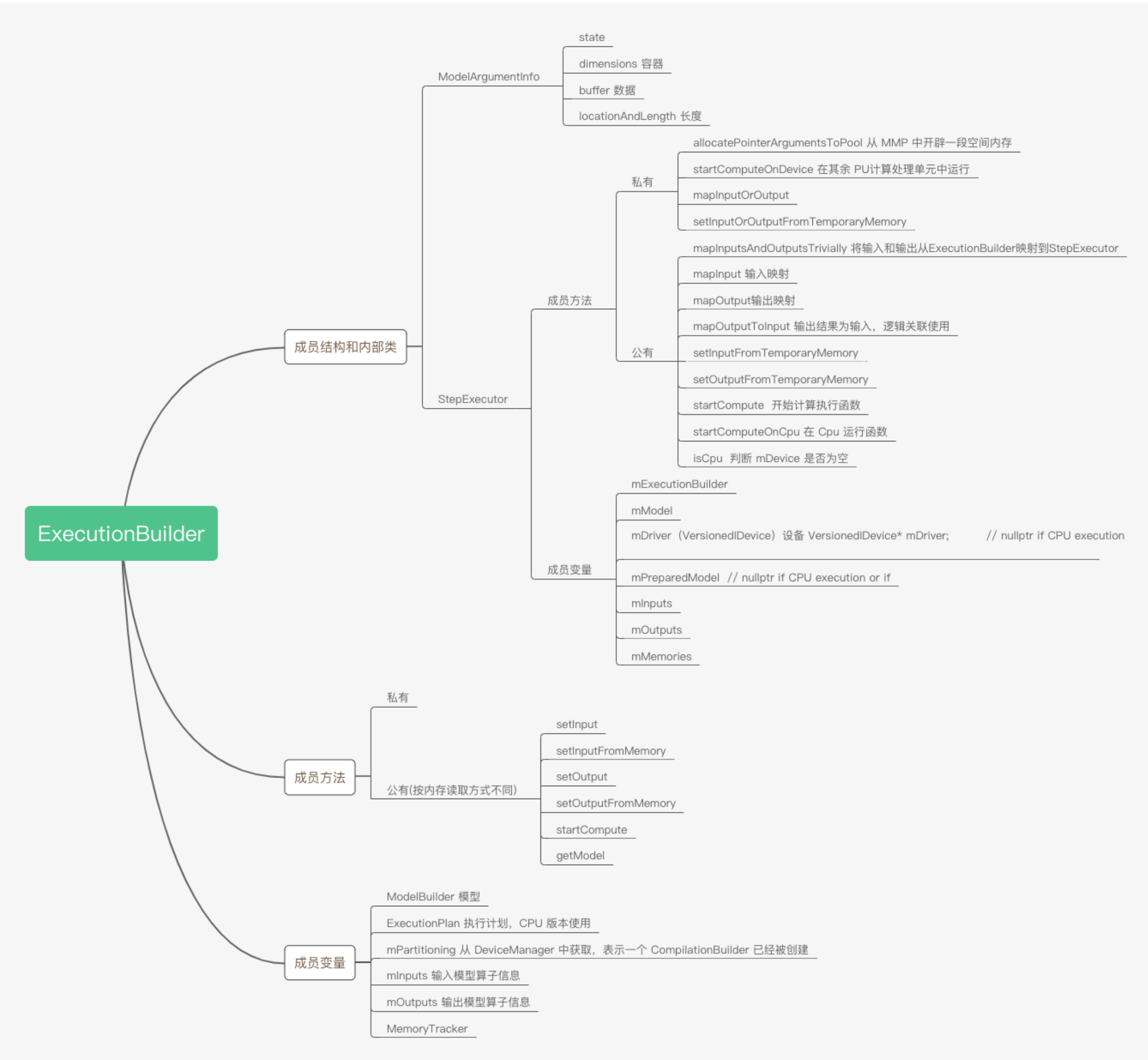
```
    *synchronizationCallback = nullptr;

    // TODO: Remove the mPreparedModel == nullptr case once we've fully
integrated
    // ExecutionPlan with the compilation and execution phases of the NN
API
    if (mPreparedModel == nullptr) {
        Model model;
        mModel->setHidlModel(&model);

        // TODO Dangerous!  In async, the model will outlive it here. Safe
for now
        sp<PreparedModelCallback> preparedModelCallback = new
PreparedModelCallback();

        // TODO(butlermichael): Propagate user preference to this point
instead of
        // using default value of
ANEURALNETWORKS_PREFER_FAST_SINGLE_ANSWER, or
        // remove this entire block of code since it is a stale path that
is only
        // encountered on an #if-removed code.
        ExecutionPreference preference =
            static_cast<ExecutionPreference>
(ANEURALNETWORKS_PREFER_FAST_SINGLE_ANSWER);
        // 配置模型参数和模型
        ErrorStatus prepareLaunchStatus = mDriver->prepareModel(model,
preference,
                                                                 preparedMod
elCallback);
        if (prepareLaunchStatus != ErrorStatus::NONE) {
            return convertErrorStatusToResultCode(prepareLaunchStatus);
        }

        // Immediately synchronize with callback object for now
        // TODO: change to asynchronous later
        //堵塞上层进程
        preparedModelCallback->wait();
        ErrorStatus prepareReturnStatus = preparedModelCallback-
>getStatus();
        mPreparedModel = preparedModelCallback->getPreparedModel();
        if (prepareReturnStatus != ErrorStatus::NONE) {
            return convertErrorStatusToResultCode(prepareReturnStatus);
        }
        if (mPreparedModel == nullptr) {
            return ANEURALNETWORKS_OP_FAILED;
        }
    }

    // We separate the input & output pools so that we reduce the copying
```

```
done if we
    // do an eventual remoting (hidl_memory->update()).  We could also use
it to set
    // protection on read only memory but that's not currently done.
    Memory inputPointerArguments;
    Memory outputPointerArguments;
    // 创建内存
    // Layout the input and output data
    int n = allocatePointerArgumentsToPool(&mInputs,
&inputPointerArguments);
    if (n != ANEURALNETWORKS_NO_ERROR) {
        return n;
    }
    n = allocatePointerArgumentsToPool(&mOutputs, &outputPointerArguments);
    if (n != ANEURALNETWORKS_NO_ERROR) {
        return n;
    }

    // Copy the input data that was specified via a pointer.
    // inputPointerArguments.update();
    for (auto& info : mInputs) {
        if (info.state == ModelArgumentInfo::POINTER) {
            DataLocation& loc = info.locationAndLength;
            uint8_t* data = nullptr;
            int n = inputPointerArguments.getPointer(&data);
            if (n != ANEURALNETWORKS_NO_ERROR) {
                return n;
            }
            memcpy(data + loc.offset, info.buffer, loc.length);
        }
    }
    // TODO: Add inputPointerArguments.commit() and .update() at all the
right places

    Request request;
    setRequestArgumentArray(mInputs, &request.inputs);
    setRequestArgumentArray(mOutputs, &request.outputs);
    uint32_t count = mMemories.size();
    request.pools.resize(count);
    for (uint32_t i = 0; i < count; i++) {
        request.pools[i] = mMemories[i]->getHidlMemory();
    }

    // Prepare the callback for asynchronous execution.
sp<ExecutionCallback>
    // object is returned when the execution has been successfully
launched,
    // otherwise a nullptr is returned. The executionCallback is abstracted
in
    // the NN API as an "event".
    //
```

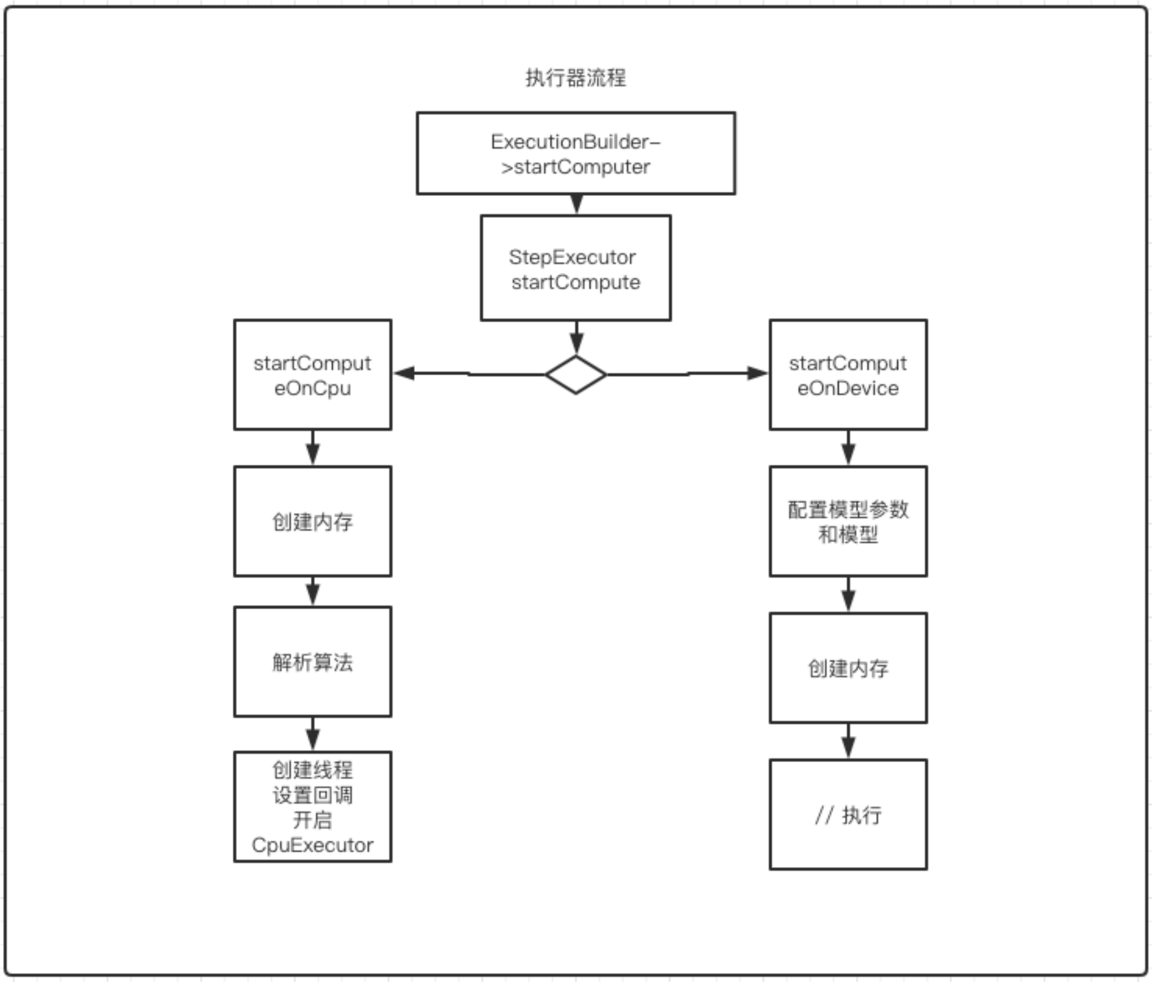
```
    // The sp is used for ref-counting purposes. Without it, the HIDL
service
    // could attempt to communicate with a dead callback object.
    //
    // TODO: Explain the "dead callback" problem further, either here or
    // in the design document.
    sp<ExecutionCallback> executionCallback = new ExecutionCallback();

    VLOG(EXECUTION) << "Before mPreparedModel->execute() " <<
SHOW_IF_DEBUG(toString(request));
    // Execute.
    // TODO: What happens to the Callback if the service dies abnormally
    // -- won't that keep the Callback live forever, because the service
    // never has the opportunity to bump the reference count down? Or
    // maybe the HIDL infrastructure handles this magically? At worst,
    // it seems like this is a small memory leak, if the Callback stays
    // alive forever.
    // 执行
    Return<ErrorStatus> executeStatus = mPreparedModel->execute(request,
executionCallback);
    if (!executeStatus.isOk() || executeStatus != ErrorStatus::NONE) {
        VLOG(EXECUTION) << "**Execute failed**";
        return executeStatus.isOk()
                ? convertErrorStatusToResultCode(executeStatus)
                : ANEURALNETWORKS_OP_FAILED;
    }

    // TODO: Remove this synchronization point when the block of code below
is
    // removed.
    executionCallback->wait();
    Return<ErrorStatus> callbackStatus = executionCallback->getStatus();
    if (!callbackStatus.isOk() || callbackStatus != ErrorStatus::NONE) {
        VLOG(EXECUTION) << "**Execute async failed**";
        return callbackStatus.isOk()
                ? convertErrorStatusToResultCode(callbackStatus)
                : ANEURALNETWORKS_OP_FAILED;
    }

    // Copy the output data from shared memory to the output buffers.
    // TODO: Move this block of code somewhere else. It should not be in
the
    // startCompute function.
    // TODO: outputMemory->update(); outputMemory->commit()
    for (auto& info : mOutputs) {
        if (info.state == ModelArgumentInfo::POINTER) {
            DataLocation& loc = info.locationAndLength;
            uint8_t* data = nullptr;
            int n = outputPointerArguments.getPointer(&data);
            if (n != ANEURALNETWORKS_NO_ERROR) {
                return n;
```

```
            }
            memcpy(info.buffer, data + loc.offset, loc.length);
        }
    }
    VLOG(EXECUTION) << "StepExecutor::startComputeOnDevice completed";

    *synchronizationCallback = executionCallback;
    return ANEURALNETWORKS_NO_ERROR;
}
```

也就是说，如果系统新增一个运算逻辑单元，需要实现 startComputeOnDevice 方法函数的部分功能即可。完整的函数结构和运行流程如下图所示：

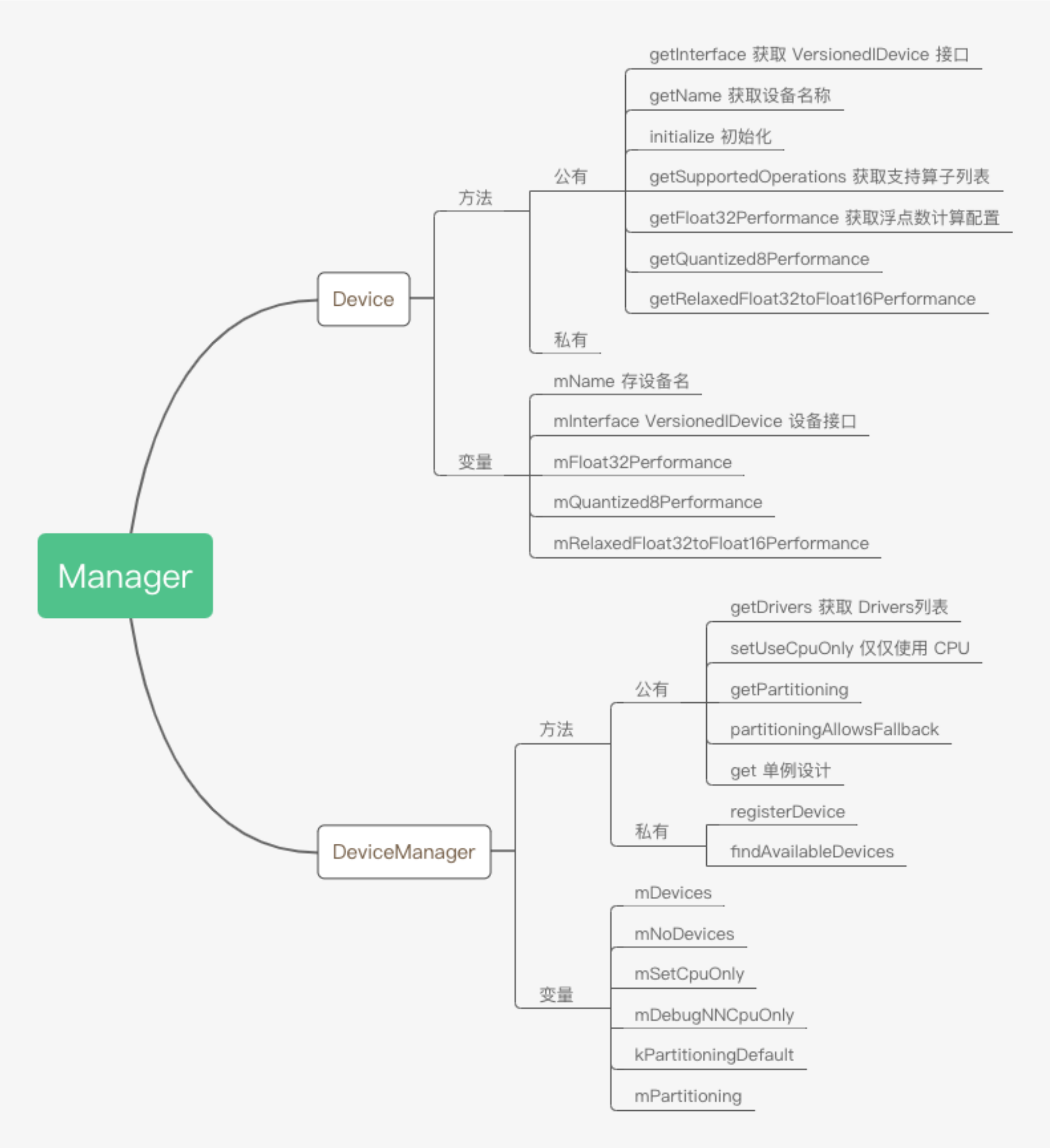


执行步骤：



- **Runtime--Manager.**

Runtime 中的 Manager 是对 Device的管理类。每每接入一个设备，都会增加一个 device 对象。首先看整体的成员方法和成员变量。



重点是看 find 方法。

```
void DeviceManager::findAvailableDevices() {
    using ::android::hidl::manager::V1_0::IServiceManager;
    VLOG(MANAGER) << "findAvailableDevices";

    sp<IServiceManager> manager = hardware::defaultServiceManager();
    if (manager == nullptr) {
        LOG(ERROR) << "Unable to open defaultServiceManager";
        return;
    }

    manager->listByInterface(V1_0::IDevice::descriptor, [this](const
```

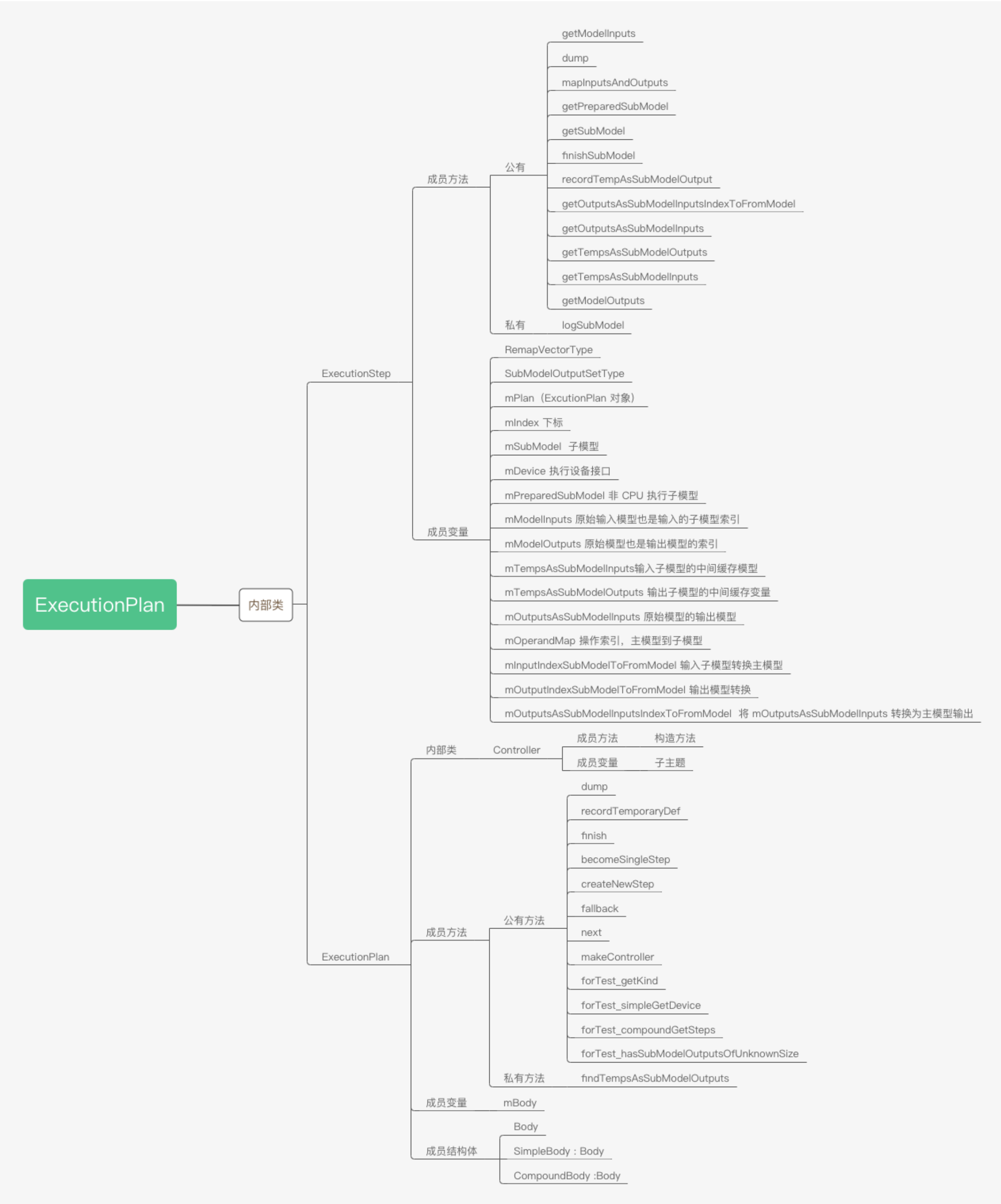
```
hidl_vec<hidl_string>& names) {
        for (const auto& name : names) {
            VLOG(MANAGER) << "Found interface " << name.c_str();
            sp<V1_0::IDevice> device = V1_0::IDevice::getService(name);
            if (device == nullptr) {
                LOG(ERROR) << "Got a null IDEVICE for " << name.c_str();
                continue;
            }
            registerDevice(name.c_str(), device);
        }
    });
}
```

这里看到是从 serviceManager 中获取 IDevice 的描述符，并将其添加到 device 列表中。

- **Runtime--策略类ExecutionPlan.**

ExecutionPlan 是执行策略类，里面包括了调度的方法。官方给予的说明是在多个 PU 中调度使用计算的策略类方法。

ExecutionPlan 定义了ExecutionStep和ExecutionPlan两个类，ExecutionPlan内部还定义了Controller和一些结构体数据类。



这个类是关于 模型运行策略的类，如何选择模型算子最优化的方案，我们可以看到 ModelBuilder 方法中的 findBestDeviceForEachOperation 方法。

```
private:
    // TODO: move partitionTheWork, findBestDeviceForEachOperation,
    // sortIntoRunOrder to CompilationBuilder?

    int findBestDeviceForEachOperation(uint32_t preference,
                                       const
```

```
std::vector<std::shared_ptr<Device>>& devices,
                                                     const size_t operationCount,
                                                     const size_t deviceCount,
                                                     std::vector<int>*
bestDeviceForOperation) const;
```

定义以及使用：

```
// 找出每个操作在哪里执行最好。
// 向量的值是设备向量中的索引，其中设备的个数
// 表示为CPU 的数量。
    // Figure out where each operation will best execute.
    // The value of the vector is the index in the devices vector, with
devices.size()
    // representing the CPU.
    std::vector<int> bestDeviceForOperation(operationCount);
    int status = findBestDeviceForEachOperation(preference, devices,
deviceCount,
                                                &bestDeviceForOperation);
    if (status != ANEURALNETWORKS_NO_ERROR) {
        return status;
    }
```

主力拆解一下代码：

```
int ModelBuilder::findBestDeviceForEachOperation(
        uint32_t preference,
        const std::vector<std::shared_ptr<Device>>& devices,
        const size_t deviceCount,
        std::vector<int>* bestDeviceForOperation) const {
    // 确定 CPU 的个数
    // Note that deviceCount includes CPU, which has no entry in devices[]
    const size_t nonCpuDeviceCount = deviceCount - 1;
    // 初始化 容器，Cando 定义与 ExecutionPlan 中。
    std::vector<CanDo> canDo(nonCpuDeviceCount);
    for (size_t deviceIndex = 0; deviceIndex < nonCpuDeviceCount;
deviceIndex++) {
        canDo[deviceIndex].initialize(this, devices[deviceIndex]);
    }

    // Figure out the best driver for each operation.
    // 找到执行端运算最佳的硬件解决方案
    const size_t operationCount = mOperations.size();
    for (size_t operationIndex = 0; operationIndex < operationCount;
operationIndex++) {
        // Find which non-CPU device gives the best performance for this
operation.
        int bestChoice = -1;
        float bestPerfVal = 0.0;  // Do not check bestPerfVal if bestChoice
< 0.
        for (size_t deviceIndex = 0; deviceIndex < nonCpuDeviceCount;
```

```
deviceIndex++) {
            const auto& device = devices[deviceIndex];
            // 判断 device 是否支持本运算符算子
            if (canDo[deviceIndex].check(operationIndex)) {
                const PerformanceInfo perf = getPerformanceInfo(device,
operationIndex);
                // 依旧执行时间来取得最优解
                const float perfVal =
                            (preference == ANEURALNETWORKS_PREFER_LOW_POWER
? perf.powerUsage

 : perf.execTime);
                // 执行策略记录
                if (bestChoice < 0 || perfVal < bestPerfVal) {
                    bestChoice = deviceIndex;
                    bestPerfVal = perfVal;
                }
            } else {
                // Somewhat noisy logging, but only place where the user of
                // NNAPI can get feedback on why an operation was not run
on a
                // specific device.
                // Logs O(operationCount * nonCpuDeviceCount) times, but
                // typically nonCpuDeviceCount is very small.
                VLOG(COMPILATION) << "Device " << device->getName()
                                  << " can't do operation "
                                  <<
toString(getOperation(operationIndex).type);
            }
        }
        // If it's the OEM op, we'd better have a device able to do it.
        if (mOperations[operationIndex].type ==
OperationType::OEM_OPERATION) {
            if (bestChoice < 0) {
                LOG(ERROR) << "No driver can do the OEM op";
                return ANEURALNETWORKS_BAD_DATA;
            }
        } else {
            // 默认 CPU 的权重是 1
            // If no driver has been found, or if the best driver is not
better than the CPU,
            // prefer the CPU. Since the performance is a ratio compared to
the CPU performance,
            // by definition the performance of the CPU is 1.0.
            if (bestChoice < 0 || bestPerfVal >= 1.0) {
                bestChoice = nonCpuDeviceCount;  // The ID of the CPU.
                // 记录最优解决方案
            }
        }
        // 记录下标
        (*bestDeviceForOperation)[operationIndex] = bestChoice;
```

```
        VLOG(COMPILATION) <<
"ModelBuilder::findBestDeviceForEachOperation("
                          << toString(getOperation(operationIndex).type)
                          << ") = "
                          << (*bestDeviceForOperation)[operationIndex];
    }
    return ANEURALNETWORKS_NO_ERROR;
}
```

这个方法的执行是在 ModelBuilder 方法定义中，是导入模型阶段。依据各个 PU 执行速度不同所记录不同的 PU ID。从而达到加速运行的原则。在调用这个方法的地方，看 ModelBuilder 的另一个方法`partitionTheWork`，这个方法做的主要是拆分工作：

```
int ModelBuilder::partitionTheWork(const
std::vector<std::shared_ptr<Device>>& devices,
                                   uint32_t preference, ExecutionPlan*
plan) const {
    // This function uses a heuristic approach to partitioning the graph.
    // It should be good enough for the first release.

    const size_t nonCpuDeviceCount = devices.size();
    // The device count is the number of HAL devices + 1. The +1 is for the
CPU.
    // Note that deviceCount includes CPU, which has no entry in devices[].
    const size_t deviceCount = nonCpuDeviceCount + 1;
    const size_t operationCount = mOperations.size();

    VLOG(COMPILATION) << "ModelBuilder::partitionTheWork: deviceCount = "
<< deviceCount
                      << ", operationCount = " << operationCount;

    // If we only have the CPU, or if the graph has no operations, no need
to try to partition.
    if (nonCpuDeviceCount == 0 || operationCount == 0) {
        // Make sure no op is an OEM operation.
        if (mHasOEMOperation) {
            LOG(ERROR) << "No driver can do the OEM op";
            return ANEURALNETWORKS_BAD_DATA;
        }
        // 结束 如果找不到可用的就结束
        plan->becomeSingleStep(nullptr /* CPU */, this);
        return plan->finish(this, preference);
    }

    // Figure out where each operation will best execute.
    // The value of the vector is the index in the devices vector, with
devices.size()
    // representing the CPU.
    // 开始优选计划
    std::vector<int> bestDeviceForOperation(operationCount);
    int status = findBestDeviceForEachOperation(preference, devices,
```

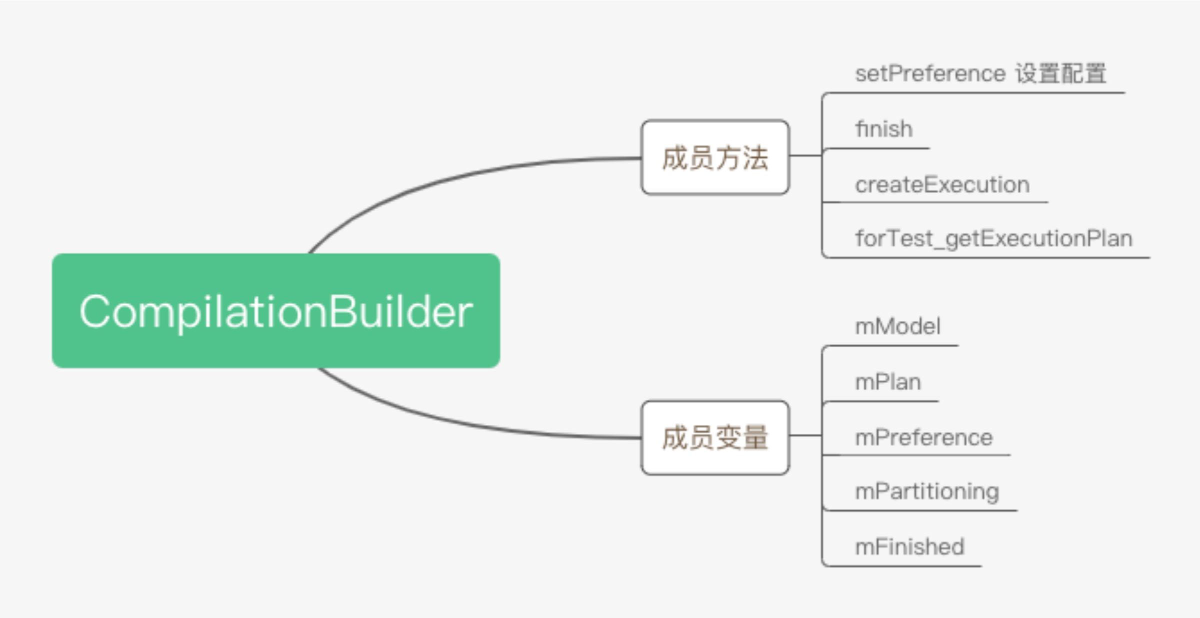
```
deviceCount,
                                                &bestDeviceForOperation);
    if (status != ANEURALNETWORKS_NO_ERROR) {
        return status;
    }
    // 如果一个 device 可以运行所有的
    // If one device will run all the operations, we don't need to split
the work.
    if (std::adjacent_find(bestDeviceForOperation.begin(),
bestDeviceForOperation.end(),
                           std::not_equal_to<int>()) ==
bestDeviceForOperation.end()) {
        const int bestDeviceIndex = bestDeviceForOperation[0];
        const bool cpu = (size_t(bestDeviceIndex) == deviceCount - 1);
        VLOG(COMPILATION) << "ModelBuilder::partitionTheWork: only one best
device: "
                          << bestDeviceIndex << " = "
                          << (cpu ? "CPU" : devices[bestDeviceIndex]-
>getName());
        plan->becomeSingleStep(cpu ? nullptr : devices[bestDeviceIndex],
this);
        return plan->finish(this, preference);
    }

    // No easy solution, we need to split the work.

    // We keep track of the operations that are ready to run for each
device.
    std::vector<std::queue<uint32_t>> perDeviceQueue(deviceCount);

    // This helper function enqueues the operation on the appropriate
queue.
    auto enqueueOnAppropriateDevice = [&](uint32_t operationIndex) {
        int deviceIndex = bestDeviceForOperation[operationIndex];
        perDeviceQueue[deviceIndex].push(operationIndex);
        VLOG(COMPILATION) << "enqueueOnAppropriateDevice " <<
operationIndex << " onto "
                          << deviceIndex;
    };
    // 查找已准备并且空闲的设备，分布计算方式，并行处理，发挥某一时刻的最佳执行效果
    // This helper function finds a device that has operations ready to
process.
    // We start by looking at the CPU. We do this to try to maximize the
    // size of the graph we'll send to non-CPU devices. If the CPU runs
first,
    // it will have the chance to prepare more of the inputs required by
the
    // other devices. This function returns -1 if all queues are empty.
    auto findNextDeviceToProcess = [&]() -> int {
        for (int i = deviceCount - 1; i >= 0; i--) {
            if (!perDeviceQueue[i].empty()) {
```

```
                return i;
            }
        }
        return -1;
    };

    OperandTracker tracker(this, enqueueOnAppropriateDevice);
    // For each iteration of this loop, we'll create an execution step.
    while (true) {
        // Find the device we'll do this step for.
        int deviceIndex = findNextDeviceToProcess();
        VLOG(COMPILATION) << "findNextDeviceToProcess: " << deviceIndex;
        if (deviceIndex < 0) {
            break;
        }
        // nullptr represents the CPU.
        std::shared_ptr<Device> device =
                static_cast<size_t>(deviceIndex) < nonCpuDeviceCount
                        ? devices[deviceIndex] : nullptr;

        // Assign as much as possible to this device.
        std::shared_ptr<ExecutionStep> step = plan->createNewStep(device);
        auto& queue = perDeviceQueue[deviceIndex];
        while (!queue.empty()) {
            uint32_t operationIndex = queue.front();
            queue.pop();
            int n = step->addOperation(operationIndex, *this);
            if (n != ANEURALNETWORKS_NO_ERROR) {
                LOG(ERROR) << "failed to add operation " << operationIndex
<< " to step";
                return n;
            }
            tracker.markProcessed(operationIndex,
enqueueOnAppropriateDevice);
        }
    }
    // 结束工作流
    int n = plan->finish(this, preference);
    if (VLOG_IS_ON(COMPILATION)) {
        Model model;
        setHidlModel(&model);
        VLOG(COMPILATION) << "ModelBuilder::partitionTheWork: original
model: ";
        logModelToInfo(model);
        plan->dump();
    }
    return n;
}
```

使用这个方法的上一层逻辑在 CompilationBuilder 中，下面是整个 CompilationBuilder 的内部逻辑。

- **Runtime--运行类CompilationBuilder**

该类是运行执行类，接口定义也比较简单，主要是关联 ModelBuilder 和 ExectionBuilder 两个。提供的方法如CompilationBuilder，就是组合创建一个 ModelBuilder 的过程。



- **Runtime--运行类ModelBuilder**

模型执行者类，这个可以被定义为，也是其中相对而言比较复杂的一款。按照上面提到的 CompilationBuilder 当中的起始关联，看相关的代码和结构。

这里定义的是一个新建的模型类，存储了模型的操作符，内存，输入等等信息。也就是说，应用层所训练的模型，会在运行环境中创建一个新的 ModelBuilder，ModelBuilder 会把对应的模型操作符、数据，开辟完成，在导入模型文件中的操作符、特征向量、计算图等信息，由此建立一个属于应用模型的 modelbuilder。并且上文也提到了，在 working 的时候会选择最优的解决策略。剩下的只是执行器的事情。

- **Runtime--运行类NeuralNetworks 接口定义类**

这个是 NNAPI 对外的接口函数方法实现类，里面定义了基本的函数接口还有枚举定义。首先看操作符定义：

枚举	枚举值	说明
ANEURALNETWORKS_ADD	0	函数增
ANEURALNETWORKS_AVERAGE_POOL_2D	1	均值计算 计算公式： output[batch, row, col, channel] = sum_{i, j}(input[batch, row + i, col + j, char
ANEURALNETWORKS_CONCATENATION	2	* Inputs: * * 0 ~ n-1: The list of n input tensors, o * [D0, D1, ..., Daxis(i), ..., Dm]. Fo * {@link ANEURALNETWORKS_ }, all input tensors * must have the same scale and * * n: An {@link ANEURALNETWORKS_ * concatenation axis. * * Outputs: * * 0: The output, a tensor of the same { input * tensors. The output shape is [D0, D */
ANEURALNETWORKS_CONV_2D	3	2 维卷积 The output dimensions are functions of the d * padding. * * The values in the output tensor are co * * output[batch, row, col, channel] = * sum_{i, j} (* input[batch, row + i, col + j, k] * filter[channel, row + i, col + j, k * bias[channel] *) * * Supported tensor {@link OperandCod * * {@link ANEURALNETWORKS_TENS * * {@link ANEURALNETWORKS_TENS * * Supported tensor rank: 4, with "NHW

```
     *
     * Both explicit padding and implicit pad
     *
     * Inputs (explicit padding):
     * * 0: A 4-D tensor, of shape [batches, h
     *      specifying the input.
     * * 1: A 4-D tensor, of shape
     *      [depth_out, filter_height, filter_widtl
     *      filter.
     * * 2: A 1-D tensor, of shape [depth_out
     *      For input tensor of {@link ANEURA
OAT32}, the bias
     *      should also be of {@link ANEURAL
AT32}. For input
     *      tensor of {@link ANEURALNETWOI
YMM}, the bias
     *      should be of {@link ANEURALNET\
with zeroPoint of
     *      0 and bias_scale == input_scale * f
     * * 3: An {@link ANEURALNETWORKS_
padding on
     *      the left, in the ‘width’ dimension.
     * * 4: An {@link ANEURALNETWORKS_
padding on
     *      the right, in the ‘width’ dimension.
     * * 5: An {@link ANEURALNETWORKS_
padding on
     *      the top, in the ‘height’ dimension.
     * * 6: An {@link ANEURALNETWORKS_
padding on
     *      the bottom, in the ‘height’ dimensi
     * * 7: An {@link ANEURALNETWORKS_
stride when
     *      walking through input in the ‘width’
     * * 8: An {@link ANEURALNETWORKS_
stride when
     *      walking through input in the ‘height
     * * 9: An {@link ANEURALNETWORKS_
one of the
     *      {@link FuseCode} values. Specifies
     *      invoke on the result.
     *
     * Inputs (implicit padding):
     * * 0: A 4-D tensor, of shape [batches, h
     *      specifying the input.
     * * 1: A 4-D tensor, of shape
     *      [depth_out, filter_height, filter_widtl
     *      filter.
     * * 2: A 1-D tensor, of shape [depth_out
put
     *      tensor of {@link ANEURALNETWOI
```

		he bias should
		    * also be of {@link ANEURALNETWC
		For input tensor
		    * of {@link ANEURALNETWORKS_TI
		the bias should be
		    * of {@link ANEURALNETWORKS_TI
		int of 0 and
		    * bias_scale == input_scale * filter_sc
		    * * 3: An {@link ANEURALNETWORKS_
		implicit
		    * padding scheme, has to be one of
		    * {@link PaddingCode} values.
		    * * 4: An {@link ANEURALNETWORKS_
		stride when
		    * walking through input in the ‘width’
		    * * 5: An {@link ANEURALNETWORKS_
		stride when
		    * walking through input in the ‘height
		    * * 6: An {@link ANEURALNETWORKS_
		one of the
		    * {@link FuseCode} values. Specifies
		    * invoke on the result.
		    *
		    * Outputs:
		    * * 0: The output 4-D tensor, of shape
		    * [batches, out_height, out_width, de
		of
		    * {@link ANEURALNETWORKS_TEN
		e following condition
		    * must be satisfied: output_scale > in
		    */

ANEURALNETWORKS_DEPTHWISE_CONV_2D	4	深度2维卷积

ANEURALNETWORKS_DEPTH_TO_SPACE	5	深度转换宽度，转置
		    * Inputs:
		    * * 0: A 4-D tensor, of shape [batches, h
		    * specifying the input.
		    * * 1: An {@link ANEURALNETWORKS_
		block_size.
		    * block_size must be >=1 and block_
		divisor
		    * of the input depth.
		    *
		    * Outputs:
		    * * 0: The output 4-D tensor, of shape [b
		    * width*block_size, depth/(block_size
		    */

ANEURALNETWORKS_DEQUAN	6	* Inputs:

TIZE		* * 0: A tensor of {@link ANEURALNETW ASYMM}. * * Outputs: * * 0: The output tensor of same shape a * {@link ANEURALNETWORKS_TEN
ANEURALNETWORKS_EMBEDDING_LOOKUP	7	* Inputs: * * 0: Lookups. A 1-D tensor of {@link A OR_INT32}. * The values are indices into the first * 1: Values. An n-D tensor, where n >= 2 e extracted. * Output: * * 0: A n-D tensor with the same rank a * tensor, except for the first dimensio * as Lookups' only dimension.
ANEURALNETWORKS_FLOOR	8	计算梯度 Supported tensor rank: up to 4 Inputs:0: A tensor. Outputs:0: The output tensor, of the same {@link ons as the input tensor.
ANEURALNETWORKS_FULLY_CONNECTED	9	Inputs: * * 0: A tensor of at least rank 2, specify * greater than 2, then it gets flattened * (flattened) 2-D Tensor is reshaped (* [batch_size, input_size], where "inp * number of inputs to the layer, matc of * weights, and "batch_size" is calcul of * elements by "input_size". * * 1: A 2-D tensor, specifying the weigh * [num_units, input_size], where "nur number * of output nodes. * * 2: A 1-D tensor, of shape [num_units put * tensor of {@link ANEURALNETWOI he bias should * also be of {@link ANEURALNETWC For input tensor * of {@link ANEURALNETWORKS_TI the bias should be * of {@link ANEURALNETWORKS_TI int of 0 and * bias_scale == input_scale * filter_s

		* * 3: An {@link ANEURALNETWORKS_ one of the * {@link FuseCode} values. Specifies * invoke on the result. * * Outputs: * * 0: The output tensor, of shape [batch ut * tensor of {@link ANEURALNETWOI YMM}, the following * condition must be satisfied: * output_scale > input_scale * filter_s */
ANEURALNETWORKS_HASHTABLE_LOOKUP	10	* Inputs: * * 0: Lookups. A 1-D {@link ANEURALN 2} tensor with * shape [k]. * * 1: Keys. A 1-D {@link ANEURALNET nsor with shape * [n]; Keys and Values pair represer * in Keys (Keys[i]) is the key to select * (Values[i]), where 0 <= i <= n-1. Key n * ascending order. * * 2: Values. A tensor with shape of [n, * must be n. * * Outputs: * * 0: Output. A tensor with shape [k … * * 1: Hits. A boolean tensor with shape okup * hits (True) or not (False). * Stored as {@link ANEURALNETWC SYMM} with offset 0 * and scale 1.0f. * A non-zero byte represents True, a se. */
ANEURALNETWORKS_L2_NORMALIZATION	11	* Inputs: * * 0: A 4-D tensor, of shape [batches, h * * Outputs: * * 0: The output 4-D tensor, of shape * [batches, out_height, out_width, de
ANEURALNETWORKS_L2_POOL_2D	12	* Inputs (implicit padding): * * 0: A 4-D tensor, of shape [batches, h ng * the input.

		* * 1: An {@link ANEURALNETWORKS_ implicit * padding scheme, has to be one of * {@link PaddingCode} values. * * 2: An {@link ANEURALNETWORKS_ stride when * walking through input in the ‘width’ * * 3: An {@link ANEURALNETWORKS_ stride when * walking through input in the ‘height * * 4: An {@link ANEURALNETWORKS_ filter * width. * * 5: An {@link ANEURALNETWORKS_ filter * height. * * 6: An {@link ANEURALNETWORKS_ one of the * {@link FuseCode} values. Specifies * invoke on the result. * * Outputs: * * 0: The output 4-D tensor, of shape * [batches, out_height, out_width, de
ANEURALNETWORKS_LOCAL_ RESPONSE_NORMALIZATION	13	* Inputs: * * 0: A 4-D tensor, of shape [batches, h ng * the input. * * 1: An {@link ANEURALNETWORKS_ radius of * the normalization window. * * 2: An {@link ANEURALNETWORKS_ the bias, must * not be zero. * * 3: An {@link ANEURALNETWORKS_ the scale * factor, alpha. * * 4: An {@link ANEURALNETWORKS_ the exponent, * beta. * * Outputs: * * 0: The output tensor of same shape a
ANEURALNETWORKS_LOGISTI C	14	Logistic 算法，回归算法一种 * Inputs: * * 0: A tensor, specifying the input. * * Outputs:

		* * 0: The output tensor of same shape a * For {@link ANEURALNETWORKS_T },
ANEURALNETWORKS_LSH_PROJECTION	15	LSH 算法 * 1: Input. Dim.size >= 1, no restriction on * * 2: Weight. Optional. Dim.size == 1, D * If not set, each input element is con eight * of 1.0. * Tensor[1].Dim[0] == Tensor[2].Dim[0 * * 3: Type: * Sparse: Value LSHProjectionType_ * Computed bit vector is consider * Each output element is an int32 * computed from hash functions. * * Dense: Value LSHProjectionType_ * Computed bit vector is consider * element represents a bit and car * 0 or 1. * * Outputs: * * 0: If the projection type is sparse: * Output.Dim == { Tensor[0].Dim[0] * A tensor of int32 that represents h * If the projection type is Dense: * Output.Dim == { Tensor[0].Dim[0] * A flattened tensor that represents
ANEURALNETWORKS_LSTM	16	LSTM 算法，long short term memory(LSTM 了解决长短期记忆网络而专门设计出来的算法
ANEURALNETWORKS_MAX_POOL_2D	17	* Inputs (implicit padding): * * 0: A 4-D tensor, of shape [batches, h ng * the input. * * 1: An {@link ANEURALNETWORKS_ implicit * padding scheme, has to be one of * {@link PaddingCode} values. * * 2: An {@link ANEURALNETWORKS_ stride when * walking through input in the 'width' * * 3: An {@link ANEURALNETWORKS_ stride when * walking through input in the 'height * * 4: An {@link ANEURALNETWORKS_

		filter * width. * * 5: An {@link ANEURALNETWORKS_ filter * height. * * 6: An {@link ANEURALNETWORKS_ one of the * {@link FuseCode} values. Specifies * invoke on the result. * * Outputs: * * 0: The output 4-D tensor, of shape * [batches, out_height, out_width, de
ANEURALNETWORKS_MUL	18	* Inputs: * * 0: A tensor. * * 1: A tensor of the same {@link Opera mensions * as input0. * * 2: An {@link ANEURALNETWORKS_ one of the * {@link FuseCode} values. Specifies * invoke on the result. * * Outputs: * * 0: The product, a tensor of the same put0. * For output tensor of {@link ANEUR UANT8_ASYMM}, * the following condition must be sat * output_scale > input1_scale * input
ANEURALNETWORKS_RELU	19	* Inputs: * * 0: A tensor, specifying the input. * * Outputs: * * 0: The output tensor of same shape
ANEURALNETWORKS_RELU1	20	* Inputs: * * 0: A tensor, specifying the input. * * Outputs: * * 0: The output tensor of same shape
ANEURALNETWORKS_RELU6	21	* Inputs: * * 0: A tensor, specifying the input. * * Outputs: * * 0: The output tensor of same shape
ANEURALNETWORKS_RESHAP	22	* Inputs:

E		* * 0: A tensor, specifying the tensor to l * * 1: A 1-D tensor of {@link ANEURALN }, defining the * shape of the output tensor. The nu shape * must be the same as the number o or. * * Outputs: * * 0: The output tensor, of shape specif
ANEURALNETWORKS_RESIZE_BILINEAR	23	* Inputs: * * 0: A 4-D tensor, of shape [batches, h ng * the input. * * 1: An {@link ANEURALNETWORKS_ output * height of the output tensor. * * 2: An {@link ANEURALNETWORKS_ output * width of the output tensor. * * Outputs: * * 0: The output 4-D tensor, of shape * [batches, new_height, new_width,
ANEURALNETWORKS_RNN	24	RNN 卷积算法
ANEURALNETWORKS_SOFTMAX	25	投票算法
ANEURALNETWORKS_SPACE_TO_DEPTH	26	空间转深度，矩阵逆转
ANEURALNETWORKS_SVDF	27	SVD 向量机算法
ANEURALNETWORKS_TANH	28 __ANDROID_API_P__	TANH 算法 * Inputs: * * 0: A tensor, specifying the input. * * Outputs: * * 0: The output tensor of same shape
ANEURALNETWORKS_BATCH_TO_SPACE_ND	29	* Inputs: * * 0: An n-D tensor, specifying the tens * * 1: A 1-D Tensor of {@link ANEURALN 2}, the block * sizes for each spatial dimension of * must be >= 1.

		* * Outputs: * * 0: A tensor of the same {@link Opera */
ANEURALNETWORKS_DIV	30	* Element-wise division of two tensors. Example: * input1.dimension = {4, 1, 2} * input2.dimension = {5, 4, 3, 1} * output.dimension = {5, 4, 3, 2} *
ANEURALNETWORKS_MEAN	31	* Inputs: * * 0: A tensor, specifying the input. * * 1: A 1-D Tensor of {@link ANEURALN 2}. The dimensions * to reduce. If None (the default), red e in * the range [-rank(input_tensor), rank * * 2: An {@link ANEURALNETWORKS_ positive, * retains reduced dimensions with le * * Outputs: * * 0: A tensor of the same {@link Opera
ANEURALNETWORKS_PAD	32	PDA 分析算法，用于文字 * Inputs: * * 0: An n-D tensor, specifying the tens * * 1: A 2-D Tensor of {@link ANEURALN 2}, the paddings * for each spatial dimension of the in * tensor must be {rank(input0), 2}. * padding[i, 0] specifies the number he * front of dimension i. * padding[i, 1] specifies the number er the * end of dimension i. *
ANEURALNETWORKS_SPACE_TO_BATCH_ND	33	* Inputs: * * 0: An n-D tensor, specifying the inpu * * 1: A 1-D Tensor of {@link ANEURALN 2}, the block * sizes for each spatial dimension of * must be >= 1. * * 2: A 2-D Tensor of {@link ANEURALN 2}, the paddings

		* for each spatial dimension of the in be * >= 0. The shape of the tensor must * padding[i, 0] specifies the number he * front of dimension i. * padding[i, 1] specifies the number er the * end of dimension i. *
ANEURALNETWORKS_SQUEEZE	34	* Inputs: * * 0: An n-D tensor, the tensor to be sq * * 1: An optional 1-D tensor of {@link A OR_INT32}. The * dimensions to squeeze. If specified ons * listed. Otherwise, squeezes all dim x * starts at 0. An error must be report that * is not 1. * * Outputs: * * 0: A tensor of the same {@link Opera s the * same data as input, but has one or * removed.
ANEURALNETWORKS_STRIDED_SLICE	35	
ANEURALNETWORKS_SUB	36	* Example: * input1.dimension = {4, 1, 2} * input2.dimension = {5, 4, 3, 1} * output.dimension = {5, 4, 3, 2} * * Supported tensor {@link OperandCod * * {@link ANEURALNETWORKS_TENS * * Supported tensor rank: up to 4 * * Inputs: * * 0: An n-D tensor, specifying the first * * 1: A tensor of the same {@link Opera mensions * as input0. * * 2: An {@link ANEURALNETWORKS_ one of the * {@link FuseCode} values. Specifies * invoke on the result.

		* * Outputs: * * 0: A tensor of the same {@link Opera */
ANEURALNETWORKS_TRANSPOSE	37	* * Inputs: * * 0: An n-D tensor, specifying the tens * * 1: An optional 1-D Tensor of {@link A OR_INT32}, * the permutation of the dimensions * * Outputs: * * 0: A tensor of the same {@link Opera */

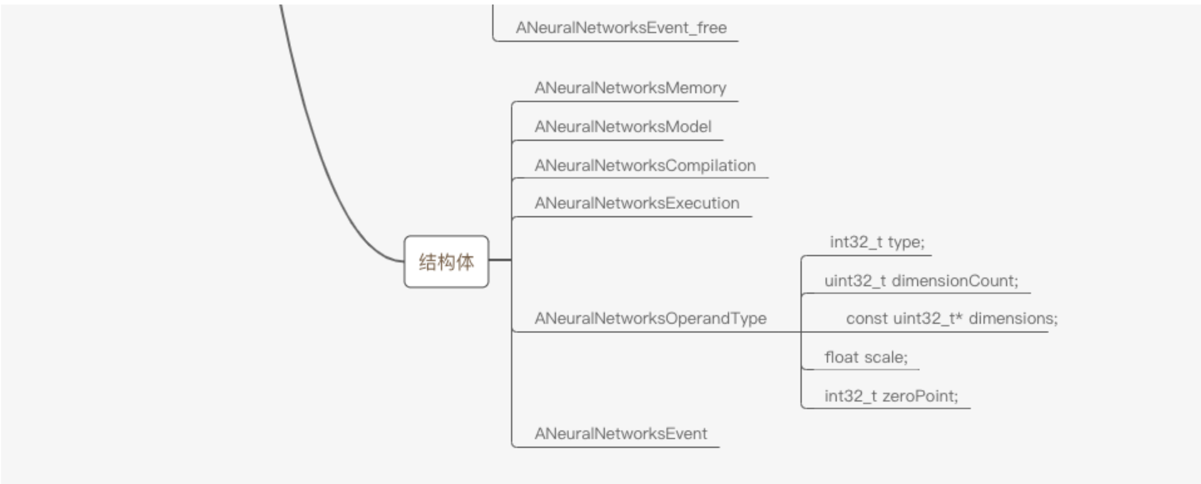
其次，定义了输入数据类型。

定义	键值	说明
ANEURALNETWORKS_FLOAT32	0	32 位 float
ANEURALNETWORKS_INT32	1	32 位 int
ANEURALNETWORKS_UINT32	2	32为 无符号 int
ANEURALNETWORKS_TENSOR_FLOAT32	3	Tensor张量 float32
ANEURALNETWORKS_TENSOR_INT32	4	张量 32 位 int
ANEURALNETWORKS_TENSOR_QUANT8_ASYMM	5	

NeuralNetworks算是一个对上层应用调用的接口封装，串联了 Memory、Execution、Compilation等等部件的功能，共同完成了 AI 算法的部署与模型的执行过程。详细的接口和定义如下：

- 枚举
  - OperandCode
    - ANEURALNETWORKS_TENSOR_QUANT8_ASYMM
    - ANEURALNETWORKS_FLOAT32
    - ANEURALNETWORKS_INT32
    - ANEURALNETWORKS_UINT32
    - ANEURALNETWORKS_TENSOR_FLOAT32
  - OperationCode
    - ANEURALNETWORKS_ADD
    - ANEURALNETWORKS_AVERAGE_POOL_2D
    - ANEURALNETWORKS_CONCATENATION
    - ANEURALNETWORKS_CONV_2D
    - ANEURALNETWORKS_DEPTHWISE_CONV_2D
    - ANEURALNETWORKS_DEPTH_TO_SPACE
    - ANEURALNETWORKS_DEQUANTIZE
    - ANEURALNETWORKS_EMBEDDING_LOOKUP
    - ANEURALNETWORKS_FLOOR
    - ANEURALNETWORKS_FULLY_CONNECTED
    - ANEURALNETWORKS_HASHTABLE_LOOKUP
    - ANEURALNETWORKS_L2_NORMALIZATION
    - ANEURALNETWORKS_L2_POOL_2D
    - ANEURALNETWORKS_LOCAL_RESPONSE_NORMALIZATION
    - ANEURALNETWORKS_LOGISTIC
    - ANEURALNETWORKS_LSH_PROJECTION
    - ANEURALNETWORKS_LSTM
    - ANEURALNETWORKS_MAX_POOL_2D
    - ANEURALNETWORKS_MUL
    - ANEURALNETWORKS_RELU
    - ANEURALNETWORKS_RELU1
    - ANEURALNETWORKS_RELU6
    - ANEURALNETWORKS_RESHAPE
    - ANEURALNETWORKS_RESIZE_BILINEAR
    - ANEURALNETWORKS_RNN
    - ANEURALNETWORKS_SOFTMAX
    - ANEURALNETWORKS_SPACE_TO_DEPTH
    - ANEURALNETWORKS_SVDF
    - ANEURALNETWORKS_TANH
    - ANEURALNETWORKS_BATCH_TO_SPACE_ND
      Android p
    - ANEURALNETWORKS_DIV
      Android p
    - ANEURALNETWORKS_MEAN
      Android p
    - ANEURALNETWORKS_PAD
      Android p
    - ANEURALNETWORKS_SPACE_TO_BATCH_ND
      Android p
    - ANEURALNETWORKS_SQUEEZE
      Android p
    - ANEURALNETWORKS_STRIDED_SLICE
      Android p

- NeuralNetworks
  - ANEURALNETWORKS_SUB
    - Android p
  - ANEURALNETWORKS_TRANSPOSE
    - Android p
  - FuseCode
    - ANEURALNETWORKS_FUSED_NONE
    - ANEURALNETWORKS_FUSED_RELU
    - ANEURALNETWORKS_FUSED_RELU1
    - ANEURALNETWORKS_FUSED_RELU6
  - PaddingCode
    - ANEURALNETWORKS_PADDING_VALID
    - ANEURALNETWORKS_PADDING_SAME
  - PreferenceCode
    - ANEURALNETWORKS_PREFER_LOW_POWER
    - ANEURALNETWORKS_PREFER_FAST_SINGLE_ANSWER
    - ANEURALNETWORKS_PREFER_SUSTAINED_SPEED
  - ResultCode
    - ANEURALNETWORKS_NO_ERROR
    - ANEURALNETWORKS_OUT_OF_MEMORY
    - ANEURALNETWORKS_INCOMPLETE
    - ANEURALNETWORKS_UNEXPECTED_NULL
    - ANEURALNETWORKS_BAD_DATA
    - ANEURALNETWORKS_OP_FAILED
    - ANEURALNETWORKS_BAD_STATE
    - ANEURALNETWORKS_UNMAPPABLE
  - ANEURALNETWORKS_MAX_SIZE_OF_IMMEDIATELY_COPIED_VALUES =128
  - 方法
    - ANeuralNetworksMemory_createFromFd
    - ANeuralNetworksMemory_free
    - ANeuralNetworksModel_create
    - ANeuralNetworksModel_free
    - ANeuralNetworksModel_finish
    - ANeuralNetworksModel_addOperand
    - ANeuralNetworksModel_setOperandValue
    - ANeuralNetworksModel_setOperandValueFromMemory
    - ANeuralNetworksModel_addOperation
    - ANeuralNetworksModel_identifyInputsAndOutputs
    - ANeuralNetworksModel_relaxComputationFloat32toFloat16
      - Android p
    - ANeuralNetworksCompilation_create
    - ANeuralNetworksCompilation_free
    - ANeuralNetworksCompilation_setPreference
    - ANeuralNetworksCompilation_finish
    - ANeuralNetworksExecution_create
    - ANeuralNetworksExecution_free
    - ANeuralNetworksExecution_setInput
    - ANeuralNetworksExecution_setInputFromMemory
    - ANeuralNetworksExecution_setOutput
    - ANeuralNetworksExecution_setOutputFromMemory
    - ANeuralNetworksExecution_startCompute
    - ANeuralNetworksEvent_wait


ANeuralNetworksEvent_free
结构体
ANeuralNetworksMemory
ANeuralNetworksModel
ANeuralNetworksCompilation
ANeuralNetworksExecution
ANeuralNetworksOperandType
int32_t type;
uint32_t dimensionCount;
const uint32_t* dimensions;
float scale;
int32_t zeroPoint;
ANeuralNetworksEvent