Calculet Runtime API C++ 开发文档

系统和应用

Exported on 07/10/2026

Table of Contents

1 安装

1.1 编译环境

- Linux 系统 (AlmaLinux7 或更高版本)
- C++17 编译器

1.2 安装步骤

1. 下载 Calculet Runtime API 源代码 (后续应该会改为提供镜像网站，不提供源码)
   a. 提供public header files
   b. cmake文件
   c. .so文件
2. 解压缩源代码包
   a. 后续改为镜像网站后，则只需要在运行自己的cmakelist时候加入
      `cmake -DCMAKE_PREFIX_PATH=https://mirror.company.com/calrt/v1.0.0 ..`
3. 使用Cmakelist工具生成lib。在hostrt目录下执行

   install .so file

   ```
   cmake -S . -B build -DBUILD_SHARED_LIBS=ON
   cmake --build build --config Release -j #开发版本不需要加入 “--config Release”
   cmake --install build (在PLD中，需sudo执行)
   ```

4. 在你的project的 CMakeLists.txt中加入如下命令。
   a. 如果安装在系统默认环境路径中，即没有-DCMAKE_INSTALL_PREFIX=PATH。
      `find_package(calrt REQUIRED)`
   b. 如果安装在自定义路径下，
      `find_package(calrt REQUIRED PATHS "path")`

1.3 快速上手

更多例子可访问 原粒半导体[1]

1 https://calculet.com.cn/

example

```
#include "calrt.hpp"

#include <memory>
#include <thread>
#include <vector>

void inference(calrt::VirtualDevice *vDev, calrt::CalbinModel *model,
calrt::CalrtInputBuf* inputBuffer, calrt::CalrtOutputBuf* outputBuffer)
{
    calrt::infer(vDev, model, inputBuffer, outputBuffer);
    outputBuffer->wait();
}

void threadFunc(calrt::VirtualDevice *vDev, calrt::CalbinModel &model)
{
    std::unique_ptr<calrt::CalrtInputBuf> pIbuf = calrt::createInputBuf(model);
    std::unique_ptr<calrt::CalrtOutputBuf> pObuf = calrt::createOutputBuf(model);
    uint32_t input[1024]; // Example input buffer size, adjust as needed

    for(auto &tensor : pIbuf->GetTensors())
    {
        // Fill input data here
        uint32_t *data = (uint32_t*)tensor.GetDataPtr(); // char * to uint32_t *

        for(size_t i = 0; i < 1024; i++)
        {
            data[i] = input[i]; // Assuming input is filled with appropriate data
        }
    }

    inference(vDev, &model, pIbuf.get(), pObuf.get());
}

int main()
{
    auto pCalbin = calrt::Calbin::CreateParser("...");

    auto vdevice = calrt::VirtualDevice::CreateVDevice();
    calrt::configure(vdevice.get(), pCalbin.get());

    std::vector<calrt::CalbinModel> models = pCalbin->GetAllModels();
    calrt::CalbinModel modelA = models[0];
    std::unique_ptr<std::thread> threads[2];
    for(int i = 0; i < 2; i++)
    {
        threads[i] = std::make_unique<std::thread>(&threadFunc, vdevice.get(),
std::ref(modelA));
    }
```

```
    for(int i = 0; i < 2; i++)
    {
        if(threads[i]->joinable())
        {
            threads[i]->join();
        }
    }

    return 0;
}
```

2 API 参考

这部分由3部分构成: 模块, 数据结构 (see page 22) and 数据类型 (see page 29)

2.1 模块

2.1.1 配置

2.1.1.1 *加载calbin文件*

```
CalrtError_e configure(VirtualDevice *vDev, Calbin *pParser)
```

```
CalrtError_e configure(std::unique_ptr<VirtualDevice> &vDev, Calbin *pParser)
```

配置calbin配置信息（模型参数等）到设备上

2.1.1.1.1 参数

`vDev` – calculet 设备指针。指针类型可为 `VirtualDevice` (see page 29) and `CalrtDevice` (see page 28)

2.1.1.1.2 返回值

CalrtSuccess (see page 35), CalrtErrorInvalidConfiguration (see page 35)

2.1.1.1.3 *描述*

返回值 `CalrtSuccess` 如果配置成功

2.1.1.2 *获取模型 – 按照名字*

CalbinModel GetModelByName(std::string_view modelName)*

根据模型名字获取模型相关信息。

2.1.1.2.1 参数

`modelName` – 模型名字

2.1.1.2.2 返回值

CalbinModel (see page 23)

2.1.2 设备管理

这一部分描述如何管理Calculet设备。通过 CalrtDevice (see page 28) 提供的接口即可完成相应的操作。

2.1.2.1 *创建设备*

```
static std::unique_ptr<CalrtDevice> CreateDevice()
```

```
static std::unique_ptr<CalrtDevice> CreatePCIeDevice()
```

创建CalrtDevice类，映射为一个实体设备.

2.1.2.1.1 返回值

`std::unique_ptr<CalrtDevice>` (see page 28) –unique pointer 描述的CalrtDevice对象

2.1.2.2 *创建虚拟设备*

```
static std::unique_ptr<VirtualDevice> CreateVDevice() #创建当前可用设备，优先选择PCIe设备
static std::unique_ptr<VirtualDevice> CreateVDevice(CalrtDeviceType_e type) # 创建指定类型设备
```

创建VirtualDevice类，

2.1.2.2.1 返回值

std::unique_ptr<VirtualDevice>

2.1.2.3 *重置设备*

```
void Reset()
```

重置设备为出场模式.

2.1.2.4 *设置设备电源模式*

```
void SetPowerMode(PowerMode power)
```

设置设备电源模式。 暂不支持

2.1.2.4.1 参数

PowerMode (see page 39)

2.1.2.5 *清除设备配置数据*

```
void ClearAllDevMem()
```

清除设备所有配置数据，即清除设备上的calbin

2.1.2.6 *获得设备信息*

```
std::string GetDevInfo()
```

获得设备信息，如内存使用情况，核心频率等

2.1.2.7 *获得设备运行状态*

```
CalrtError_e Status()
```

获得设备运行状态

2.1.2.7.1 返回值

CalrtSuccess (see page 35), CalrtErrorDeviceBusy (see page 35)

2.1.3 Buffer

这一部分描述跟缓冲相关的runtime内容。其主要用于推理和携带数据信息

2.1.3.1 *创建缓冲*

```
std::unique_ptr<CalrtInputBuf> createInputBuf(const CalbinModel &model)
std::unique_ptr<CalrtOutputBuf> createOutputBuf(const CalbinModel &model)
```

创建输入缓冲或输出缓冲

2.1.3.1.1 参数

`model` -- CalbinModel (see page 23) 从Calbin (see page 15)对象中获得

2.1.3.1.2 返回值

`std::unique_ptr<CalrtInputBuf>` (see page 28)
`std::unique_ptr<CalrtOutputBuf>` (see page 28)

2.1.3.2 *获得缓冲信息*

> `const CalrtBufferInfo_s& GetBufferInfo() const`

获取 i/o 缓冲的信息

2.1.3.2.1 返回值

CalrtBufferInfo_s (see page 26)

2.1.3.3 *获得并修改tensor张量信息*

> `std::vector<CalrtTensor>& GetTensors()`

获取i/o 缓冲中张量信息，并可以拷贝相应 形状的数据到张量中。

2.1.3.3.1 返回值

CalrtTensor (see page 29)

2.1.3.3.2 描述

With `CalrtTensor.GetDataPtr()` , it return a `char *` for input data or get output data.

2.1.3.4 *获得并修改当前超参数信息*

> `ModelHyperParameters_s &GetCurHyperParam()`

获得并修改当前超参数信息。

2.1.3.4.1 返回值

ModelHyperParameters_s (see page 28)

2.1.3.4.2 注意

在CNN模型下该结构体为无效结构体。此接口只在 `CalrtInputBuf` 中提供。

2.1.3.5 *等待推理完成并读取结果*

> `CalrtError_e Wait()`

阻塞接口，用于提交推理任务后，等待推理完成并读回推理结果。

2.1.3.5.1 注意

此接口只在 `CalrtOutputBuf` 中提供。

2.1.3.5.2 返还值

CalrtSuccess (see page 35)

2.1.3.6 *相关buffer指定Tensor切片*

> *CalrtError_e SliceTensorByName(void *src, const std::string &tensorName, uint64_t offset, uint64_t size)*

可截短tensor传输数据大小和数据所在tensor的位置。以减少数据传输的时间开销。

2.1.3.6.1 注意

参数offset永远会以tensor中buffer起始位置做偏移。不会被之前的设置的offset影响

2.1.3.6.2 返还值

CalrtSuccess (see page 35)

2.1.3.7 *重置Tensor设置*

> *void ResetTensorByName(const std::string &tensorName)*

重置tensor的传输size跟offset值为默认值。

2.1.3.8 *获取传输输入数据的时间开销*

> *float GetInputTransferTime()*

获取input传输数据时间开销。返还单位为 ms

2.1.3.8.1 注意

只有CalrtInputBuf类提供该接口

2.1.3.9 *获取传输输出数据的时间开销*

> *float GetOutputTransferTime()*

获取output传输数据时间开销。返还单位为 ms

2.1.3.9.1 注意

只有CalrtOutputBuf类提供该接口

2.1.3.10 获取推理时间

> *float GetWaitTime()*

获取纯推理时间。（考虑改名为 GetInferTime更为贴切）。返还单位为 ms

2.1.3.10.1 注意

只有CalrtOutputBuf类提供该接口

2.1.3.11 *动态设置Tensor地址*

> *CalrtError_e RelocateTensorAddress(const std::string &tensorName, uint64_t altAddr)*

重新设置相关buffer的指定tensor在设备上的地址。需要通过 VirtrualDevice类的接口

`uint64_t AllocDevMem(uint64_t size, uint32_t alignLog2Byte=0)` 来动态开设备内存空间。或者 CalrtDevice类的接口

`uint64_t CreateBuf(uint64_t size, uint32_t alignLog2Byte=2)`来动态开设备空间内存

2.1.3.11.1 返还值

CalrtSuccess (see page 35)

2.1.4 推理

这一部分描述推理相关的接口

2.1.4.1 *推理接口*

```
void infer(std::unique_ptr<VirtualDevice> &vDev, CalbinModel *model,
CalrtInputBuf* inputBuffer, CalrtOutputBuf* outputBuffer);
```

2.1.4.1.1 参数

`vDev` -- VirtualDevice (see page 29) virtual device manages one or multiple physical device.
`model` -- CalbinModel (see page 23) try to infer specific model
`inputBuffer` -- CalrtInputBuf (see page 28) runtime input buffer pointer
`outputBuffer` -- CalrtOutputBuf (see page 28) runtime output buffer pointer

2.1.4.1.2 描述

非阻塞接口。用于通知单个或多个设备开始推理。通过 `CalrtOutputBuf.Wait()`来等待推理完成并获取推理结果。

2.1.5 Calbin

这部分描述calbin对象中提供的接口。主要用于解析calbin文件和提供相关calbin中的文件信息。

2.1.5.1 *创建calbin*

```
static std::unique_ptr<Calbin> CreateCalbin(const std::string &path)
```

创建calbin对象。

2.1.5.1.1 参数

`path` --calbin文件的路径信息。

2.1.5.1.2 返回值

`std::unique_ptr<Calbin>` 一个unique_ptr Calbin类型指针。

2.1.5.2 *获得calbin文件信息*

> `CalrtCalbin& GetCalbinBrief()`

获得calbin文件对应的结构体数据信息。

2.1.5.3 *获得模型信息*

> `std::optional<CalbinModel> GetModelInfo(const char *modelName)`
> `std::vector<CalbinModel> GetAllModels()`

两种calbin对象的成员接口均可提供calbin文件中，关于模型的相关结构体信息。不同在于，第一种可获取指定模型信息，第二种可获取全部模型信息。

2.1.5.4 *获得LLM 信息*

> `const CalbinLLM_s &GetLLMInfo()`

获取最大batch size和最大sequence length。

2.1.5.4.1 返回值

CalbinLLM_s

2.1.5.5 *打印kv-cache specs*

> `void Report()`

打印kv-cahce 配置信息。

2.1.6 内存管理

这一部分主要描述内存管理相关的接口。所有的接口均是 CalrtDevice[2] 成员函数。

---

2 https://file+.vscode-resource.vscode-cdn.net/c:/Users/YulinYang/Desktop/yulin_250731/subpage/class.md#calrtdevice

2.1.6.1 *注册设备内存*

> `CalrtError_e RegisterDram(uint64_t startAddr, uint64_t size)`

注册设备动态随机存取存储器(DRAM)空间。

> `CalrtError_e RegisterSyncUnit(uint64_t startAddr, uint64_t size)`

注册设备同步单元(sync unit)。

> `CalrtError_e RegisterSramBuf(uint64_t startAddr, uint64_t size)`

注册设备静态随机存取存储器(SRAM)空间。

2.1.6.1.1 参数

`startAddr` -- 注册内存起始地址
`size` -- 注册内存大小。以byte为单位

2.1.6.1.2 返回值

CalrtSuccess (see page 35), CalrtErrorMemoryAlreadyRegistered (see page 35)

2.1.6.1.3 用法

```
/*CalrtDevice pointer*/ device_ptr→RegisterDram(0x1000, 32);
```

2.1.6.2 *分配内存*

> `uint64_t CreateBuf(uint64_t size, uint32_t alignLog2Byte=0)`
>
> `uint64_t AllocDevMem(uint64_t size, uint32_t alignLog2Byte=0)`

动态分配设备动态随机存取存储器(DRAM)空间。

> `uint64_t CreateSramBuf(uint64_t size, uint32_t alignLog2Byte=0)`

动态分配设备静态随机存取存储器(SRAM)空间。

> `uint64_t ApplySyncUnit()`

动态分配设备同步单元(sync unit)。

2.1.6.2.1 参数

`size` -- 申请分配空间大小。byte单位。
`alignLog2Byte` -- 位对其。在2^n公式中，对应n。举例： 64 byte align. then n is 6。

2.1.6.2.2 返回值

`uint64_t` 内存空间起始地址。

2.1.6.2.3 注意

如何分配内存失败，则会终止程序。

2.1.6.2.4 用法

```
/*CalrtDevice pointer*/ device_ptr->CreateBuf(32, 20) // allocate 32 byte
memory aligned to 1MB on DRAM
```

2.1.6.3 *拷贝用户内存到设备内存*

> `void WriteToDevice(void *srcAddr, const uint64_t devAddr, const uint64_t size) #CalrtDevice 接口`

拷贝用户端内存数据到设备端

2.1.6.3.1 参数

`srcAddr` -- 客户端内存地址。
`devAddr` -- 设备端内存地址。
`size` -- 数据大小。byte单位。

2.1.6.4 *拷贝用户数据到设备指定chip*

> *CalrtError_e WriteRemoteChipMem(void *srcAddr, const uint64_t devAddr, const uint64_t size, uint32_t chipId) #CalrtDevice 接口*
>
> *CalrtError_e WriteMem(void *srcAddr, const uint64_t dst, const uint64_t size, uint32_t chipId) #VirtualDevice 接口*

2.1.6.4.1 参数

`srcAddr` – 客户端内存地址
`devAddr` – 设备端内存地址
`size` – 数据大小。byte单位
`chipId` – 指定chip id

2.1.6.5 拷贝指定chip内存到用户

> *CalrtError_e ReadRemoteChipMem(void *srcAddr, const uint64_t devAddr, const uint64_t size, uint32_t chipId) #CalrtDevice 接口*
>
> *CalrtError_e ReadMem(void *srcAddr, const uint64_t dst, const uint64_t size, uint32_t chipId) #VirtualDevice 接口*

2.1.6.5.1 参数

`srcAddr` – 客户端内存地址
`devAddr` – 设备端内存地址
`size` – 数据大小。byte单位
`chipId` – 指定chip id

2.1.6.6 *拷贝设备内存到用户内存*

> `void ReadFromDevice(void *srcAddr, const uint64_t devAddr, const uint64_t size)`

2.1.6.6.1 参数

`srcAddr` – 客户端内存地址。
`devAddr` – 设备端内存地址。
`size` – 数据大小。byte单位。

2.1.6.7 *写设备寄存器*

> `void WriteReg(const uint64_t regAddr, uint32_t data) #CalrtDevice 接口`

> `void WriteRegister(uint64_t regAddr, uint32_t regValue, uint32_t chipId)`
> `#VirtualDevice接口`

写设备寄存器。

2.1.6.7.1 参数

`regAddr` – 寄存器地址。
`data` – 写入的数据。
`chipId` – 指定chipId

2.1.6.8 *读设备寄存器*

> `void ReadReg(const uint64_t regAddr, uint32_t &data) #CalrtDevice 接口`
>
> `void ReadRegister(uint64_t regAddr, uint32_t regValue, uint32_t chipId)`
> `#VirtualDevice接口`

读取设备寄存器。

2.1.6.8.1 参数

`regAddr` – 寄存器地址。
`data` – 读回的数据。
`chipId` – 指定chipId

2.1.6.9 *释放设备内存*

> `void FreeBuf(uint64_t addr)`

释放设备动态随机存取存储器(DRAM)地址空间。

> `void FreeSramBuf(uint64_t addr)`

释放静态随机存取存储器空间(SRAM)地址空间。

> `void FreeSyncUnitByAddress(uint64_t addr)`

释放同步单元依据地址信息。

> `void FreeSyncUnitByIndex(uint32_t idx)`

释放同步单元依据角标信息。

2.1.7 错误处理

这一部分描述错误处理的接口。

2.1.7.1 *CALRT_CHECK*

> `CALRT_CHECK(CalrtError_e, msg)`

检查错误代码是否是CalrtSuccess. 否则就会抛出错误信息

2.1.7.1.1 用法

```
CalrtError_e status = calrt::configure(device_ptr, calbin_ptr);
CALRT_CHECK(status, "hello world. your input: %s", "he11o w0r1d");
```

2.1.7.2 *REPORT_CALRT_ERROR*

> `REPORT_CALRT_ERROR(CalrtError_e, msg)`

反馈错误信息和错误类型

2.1.7.2.1 用法

```
REPORT_CALRT_ERROR(CalrtErrorInvalidValue, "hello world. your input: %s",
"he11o w0r1d");
```

2.1.7.3 *REPORT_CALRT_ERROR_IF*

> `REPORT_CALRT_ERROR_IF(CalrtError_e, condition, msg)`

反馈错误信息和错误类型，如果条件为false。

2.1.7.3.1 描述

条件判断使用跟 `assert` 逻辑相同。

2.1.7.4 *CALRT_LOG*

> `CALRT_LOG(verbosity, msg)`

打印runtime信息。

2.1.7.4.1 参数

`verbosity` (see page 39)– log层级

2.1.8 Bug Replay

提供bug复现功能。生成运行时所需要的input 数据和output数据。

2.1.8.1 *设置重现或debug方式*

> *void SetFullDebug(DebugOps_e debOp, const char* path = nullptr)*

2.1.8.1.1 参数

DebugOps_e (see page 39): debug 操作选项

path: 默认路径为当前calbin路径。可传入自定义路径，但DebugOps_e需要有对应的设置。详情查看选项描述部分。

2.2 数据结构

这一部分由两部分组成: struct (see page 22)and class (see page 28)。

2.2.1 结构体

2.2.1.1 *CalrtCalbin*

描述 `CalrtCalbin` 结构体.

2.2.1.1.1 变量

> `uint64_t name`

Calbin文件hash名字。

> `std::vector<CalbinModel> models`

文件描述的模型结构体。

2.2.1.2 *CalbinModel*

描述模型相关信息。

2.2.1.2.1 变量

> `bool isConfigured`

描述当前模型是否配置到设备上。

> `CalDevAccType_e devAccType`

CalDevAccType_e (see page 29)

> `std::string modelType`

CalbinModelType_e

> `std::string modelArch`

CalbinModelArch_e

> `ModelChipMode_e chipMode`

ModelChipMode_e (see page 30)

> `std::string modelName`

描述当前模型名字

> `std::unordered_multimap<CalbinSectionType_e, CalbinSection_s> sections`

描述当前模型组成的相关组件。 CalbinSectionType_e (see page 31), CalbinSection_s (see page 23)。

> `std::vector<int32_t> tarChipN`

描述当前模型部署的目标芯粒。

2.2.1.3 *CalbinSection_s*

描述模型的组件。

2.2.1.3.1 变量

> `CalbinSectionPlace_e place`

描述当前组件应放置于设备区域。 CalbinSectionPlace_e (see page 30)

> `CalbinSectionType_e type`

CalbinSectionType_e (see page 31)

> `std::string name`

描述当前组件名字。

> `std::string filePath`

描述当前组件相应二进制文件路径。

> `int64_t offset`

描述当前组件.bss段 entry address。

> `int64_t devAddr`

描述当前组件储存在设备的内存地址信息。

> `int64_t size`

描述当前组件大小。byte单位

> `int64_t progEntry`

描述当前组件相关的程序 header entry address

> `std::vector<uint32_t> chipMask`

描述当前组件应放置于的目标芯粒。

> `std::vector<CalbinTensorInfo_s> tensorInfos`

描述当前组件相关的张量信息。 CalbinTensorInfo_s (see page 25)

2.2.1.4 *CalbinLLM_s*

描述当前非CNN模型的最大batch size，最大sequence length和相关的kv cache。

2.2.1.4.1 变量

> `uint32_t max_batch_size`

描述当前模型支持的最大batch。

> `uint32_t max_seq_len`

描述当前模型支持的最大sequence length。

> `KV_Cache_s` *kv_cache`*

KV_Cache_s (see page 25)

2.2.1.5 *KV_Cache_s*

描述当前非CNN模型 kv cache信息.

2.2.1.5.1 变量

> `PrimitiveType dtype`

描述当前数据类型。默认是 `INVALID` . PrimitiveType (see page 32)。

> `uint32_t layer`

描述当前层数。

> `int64_t kv_base[2]`

描述K cache的基地址(`kv_base[0]`)和V cache的基地址(`kv_base[1]`)。

> `int64_t size`

描述当前kv cache 大小

> `std::vector<int32_t> shape`

描述当前kv cache 形状。*{head, batch, head_dim, d0, d2}*

2.2.1.6 *CalbinTensorInfo_s*

描述张量相关信息。

2.2.1.6.1 变量

> `std::string name`

描述当前张量名字。

> `std::vector<int32_t> shape`

描述当前张量形状。

> `PrimitiveType dataType`

描述当前张量的数据类型. PrimitiveType (see page 32)

> `int64_t ping; int64_t pong`

描述当前张量的ping-pong地址。

> `int64_t size`

描述当前张量的ping-pong地址大小。

2.2.1.7 *CalrtDevBuf_s*

Descript Buffer[3] info. Reserved.

2.2.1.8 *CalrtBufferInfo_s*

用于构建 `CalrtInputBuf` 和 `CalrtOutputBuf` 的结构体。描述基础buffer信息。

2.2.1.8.1 变量

> `CalrtBufferDirection_e direction`

描述当前buffer数据流向。 CalrtBufferDirection_e (see page 34)

> `std::string name`

描述当前buffer名字。

> `std::vector<CalbinTensorInfo_s> tensorInfos`

描述当前buffer的储存的相关张量信息。

> *`uint64_t devAddr0; uint64_t devAddr1; uint64_t size"*

描述当前buffer目标设备ping pong内存信息。重复出现在 `CalbinTensorInfo_s`。

---

3 https://file+.vscode-resource.vscode-cdn.net/c:/Users/YulinYang/Desktop/yulin_250731/subpage/class.md#calrtinputbuf

2.2.1.9 *TaskInfo_s*

内部 `JobEgnine` 使用。描述任务抽象类型。

2.2.1.10 *CalrtDeviceId_s*

当前设备基础id信息。

2.2.1.10.1 变量

> `CalrtDeviceType_e devType`

描述当前设备类型。 CalrtDeviceType_e (see page 38)

> `std::string devPath`

描述当前设备节点路径

> *int serialNum = 0*

设备系列号

2.2.1.11 *CalrtDeviceInfo_s*

当前设备的详细信息。通过 `CalrtDeviceId_s` 生成。

> `CalrtDeviceId_s devId`

描述当前设备 id。

> `std::string name`

描述当前设备名字

> `int64_t ddrSize`

描述当前设备动态随机存取存储器(DRAM)大小。

> `float sramFreqMhz`

静态随机存取存储器(SRAM)频率。

> `int64_t sramSizeMb`

描述当前设备静态随机存取存储器(SRAM)大小。

2.2.1.12 *ModelHyperParameters_s*

提供给上层用户 (llama, etc) 结构体。用于调整运行时的csr配置和sequence length。

> `uint32_t GetCsrValueByName(const std::string &csrName)`

根据csr 名字来获得对应data

> `void SetCsrByName(const std::string &csrName, uint32_t value)`

给对应csr配置data

2.2.2 类

相关类的简易说明。开发者需要更多的信息，可阅读相关头文件。Related class brief introduction. For more details, please read header files.

2.2.2.1 *Calbin*

Calculet runtime object used for parsing calbin file and related program data.

2.2.2.2 *CalrtInputBuf*

Calculet runtime object used for carraying tensors information and related data for inference. Multi-threads friendly.

2.2.2.3 *CalrtOutputBuf*

Calculet runtime object used for carray tensors information and inference result. Multi-threads friendly.

2.2.2.4 *CalrtDevice*

Calculet runtime abstract object used for controlling physical device for either memory management or device work. Actual functionality would be override by its derived class.

2.2.2.5 *CalrtEmuDevice*

Calculet runtime object, derived class of `CalrtDevice`. Used as an emulator without physical device.

2.2.2.6 *CalrtPcieDevice*

Calculet runtime object, derived class of `CalrtDevice`. Used to map with physical PCIE device.

2.2.2.7 *VirtualDevice*

Calculet runtime abstract object. Used for managing one or multiple physical device.

2.2.2.8 *CalrtJobEngine*

Calculet runtime object. `VirtualDevice` module used for handling ping-pong task.

2.2.2.9 *CalrtAllocator*

Calculet runtime object. `CalrtDevice` module used for memory management.

2.2.2.10 *CalrtDriver*

Calculet runtime object. `CalrtPcieDevice` module used for transfering data via PCI.

2.2.2.11 *CalrtJob*

Calculet runtime object. Used for descripting each inference.

2.2.2.12 *CalrtTensor*

Calculet runtime object, Used for recoding each tensors information inside i/o buffer object

---

2.3 数据类型

这一部分描述Runtime相关数据类型。

2.3.1 枚举

enum items.

2.3.1.1 *CalDevAccType_e*

设备加速类型。

2.3.1.1.1 变量

`CAL_DEV_ACC_NULL = 0`

undefined acceleration type

`CAL_DEV_ACC_CPU = 1`

仅CPU加速

`CAL_DEV_ACC_CCU = 2`

仅CCU加速

`CAL_DEV_ACC_CPU_CCU = 3`

co-op CPU and CCU acceleration。

2.3.1.2 *ModelChipMode_e*

描述当前模型的芯粒调度指示。

2.3.1.2.1 变量

`SCHIP = 0`

单芯粒模式。

`MCHIP = 1`

多芯粒模式

`UNDEFINED_CHIP_MODE

未知调度模式。

2.3.1.3 *CalbinSectionPlace_e*

描述calbin section在设备的位置。

2.3.1.3.1 变量

`DRAM = 0`

DRAM。

> `SRAM = 1`

SRAM。

CalbinSectionType_e

Calbin section 类型。

2.3.1.3.2 变量

> `CALBIN_SECTION_NULL = 0`

undefined section. 未定义部分。

> `CALBIN_SECTION_CPU_CMD = 1`

a section about CPU program and related commands. 有关CPU程序和相关命令的部分。

> `CALBIN_SECTION_CPU_SO = 2`

a section about shared libs used by CPU. 有关 CPU 使用的共享库的部分。

> `CALBIN_SECTION_CCU_ELF = 3`

a section about CCU ELF. 关于 CCU ELF 的部分。

> `CALBIN_SECTION_RODATA = 4`

a section about model parameter. 关于模型参数的部分。

> `CALBIN_SECTION_IBUF = 5`

a section about input buffer. 有关输入缓冲区的部分。

> `CALBIN_SECTION_OBUF = 6`

a section about output buffer. 关于输出缓冲区的部分。

> `CALBIN_SECTION_MEMRSVD = 7`

a section about reserved memory for your calbin applying to Calculet device. 有关应用于设备 calbin 保留内存的部分。

> `CALBIN_SECTION_KV = 8`

a section about kv cache and related information. 有关 kv cache和相关信息的部分。

> *CALBIN_SECTOIN_CSR = 9*

有关CSR配置相关信息的部分。

> `CALBIN_SECTION_NUM = 10`

the number of `CalbinSectionType_e` 类型数量。

2.3.1.4 *PrimitiveType*

Descript data type.

2.3.1.4.1 变量

> `U1=0`

1bit

> `PRED=1`

bool, int8

> `U8=2`

unsigned int

> `S8=3`

signed int

> `FP8_143=4`

exp_bias = 8; has subnormal, no inf/nan. when biased_exp=0 it is fixed explained as unbiased_exp=-7;

> `FP8_152=5`

exp_bias = 16; has subnormal, no inf/nan. when biased_exp=0, it is fixed explained as unbiased_exp=-15;

> `U16=6`

16b unsigned int

> `S16=7`

16b signed int

> `BF16=8`

16b

> `U32=9`

32b unsigned integer

> `S32=10`

32b signed integer

> `F32=11`

32b float

> `U4=12`

4b unsigned char

> `S4=13`

4b signed char

> `F35=14`

35b float

> `U64=15`

not support

> `S64=16`

u64和s64需要保留，仅做类型转换，因为pytorch要求tensor做index时必须为 long或byte或bool。

> `TF32=17`

19b. not support

> `F16=18`
> `F64=19`
> `C64`
> `C128`

reserved

> `TUPLE`

0b

`TOKEN`

32b

`OPAQUE_TYPE`

32b

`INVALID`

0b

`TYPE_SIZE`

the number of data types

2.3.1.5 *CalrtBufferDirection_e*

描述缓冲区的数据流方向。

2.3.1.5.1 变量

`HostToDevice = 0`

transfer data from host to device only. 仅从主机向设备传输数据。

`DeviceToHost = 1`

transfer data from device to host only. 仅从设备向主机传输数据。

`Both = 4`

2-ways transfer. 双向传输。

2.3.1.6 TaskType_e

Describe the task type. Used for ping-pong arrangement.

2.3.1.6.1 变量

`TASK_PING = 0`

ping property task. ping 属性任务。

`TASK_PONG = 1`

pong property task. pong 属性任务。

> `UNDEFINED_TASK = 2`

undefined property task. error report purpose. 未定义的属性任务

2.3.1.7 CalJobAlloStatus_e

`CalrtJobEngine` (see page 29)状态描述。

2.3.1.7.1 变量

> `JOB_PING = 0`

ping condition job. executed by ping thread. ping 条件作业。由 ping 线程执行。

> `JOB_PONG`

pong condition job. executed by pong thread. pong 条件作业。由 pong 线程执行。

> `JOB_PENDING`

pending condition job. 待定条件作业。

2.3.1.8 *CalrtError_e*

runtime 错误代码

2.3.1.8.1 变量

> `CalrtSuccess = 0`

no error. 成功。

> `CalrtErrorMemoryAllocation = 1`

failed to allocate device memory. 动态分配内存空间失败。

> `CalrtErrorMemoryAlreadyRegistered = 2`

failed to register(reserve) device memory due to target memory address is already taken. 由于目标内存地址已被占用，无法注册（保留）设备内存。

> `CalrtErrorAssert = 3`

runtime assert error.

> `CalrtErrorInvalidValue = 4`

invalid parameters paased to runtime API. 传递给运行时 API 的参数无效。

> `CalrtErrorInvalidConfiguration = 5`

indicate that the current model or Calbin file is invalid to be configured on device. 表示当前模型或 Calbin 文件无效，无法在设备上配置。

> `CalrtErrorNoDevice = 6`

indicate there is no valid calculet device detected. 表示没有检测到有效的 calculet 设备.

> `CalrtErrorAlreadyConfigured = 7`

indicate the current calbin file is already configured on device. 表示当前 calbin 文件已在设备上配置

> `CalrtErrorIncompatibleDriver = 8`

update device driver. too old to be supported by runtime. 更新设备驱动程序太旧，runtime不支持。

> `CalrtErrorRequireNewerRT = 9`

update runtime library. too old to be compatible with current device. 更新runtime库。太旧，与当前设备不兼容。

> `CalrtErrorFileNotFound = 10`

indicate the target file cannot be found. 表示找不到目标文件。

> `CalrtErrorBrokenFile = 11`

failed to read or write the target file. 无法读取或写入目标文件。

> `CalrtErrorInvalidELF = 12`

ELF version is invalid. ELF 版本无效。

> `CalrtErrorTimeout = 13`

indicate connection between host and device is timeout. 表示主机与设备之间的连接超时。

> `CalrtErrorOutOfRange = 14`

indicate the passed argument is out of range. 表示传递的参数超出范围。

> `CalrtErrorNotExpected = 15`

indicate the feedback or result is not expected. 表示反馈或结果不及预期。

> `CalrtErrorMissConfiguration = 16`

indicate the current model is not configured on the current device. 表示当前设备上未配置当前型号。

> `CalrtErrorDeviceBusy = 17`

indicate the current device is still connected and connection is not timeout. But it is too busy to execute the new request. 表示当前设备仍处于连接状态，且连接未超时，但太忙，无法执行新的请求。

> `CalrtErrorDeviceUnavailable = 18`

device disconnected, PCIe loose contact, power supply low. 设备断开连接、PCIe接触不良、电源电压低。

> `CalrtErrorInvalidCalbin = 19`

indicate that calbin file is invalid to parse any valid data for configure models on the current device. 表明 calbin 文件无效，无法解析当前设备上配置模型的任何有效数据。

> `CalrtErrorUnknown = 20`

indicate unknown error. 指示未知错误。

> *CalrtErrorInvalidDataType*

表明非法数据类型

> *CalrtErrorInvalidDataShape*

表明非法数据shape

> *CalrtErrorInvalidLibDep*

表明库依赖出错

> *CalrtErrorDeviceMapBroken*

表明设备列表失效

> *CalrtErrorFailedSoftReset*

表明设备软重置失败

> *CalrtErrorFailedConnectServer*

表明断开连接服务器

> *CalrtErrorDeviceUnsupport*

表明设备不支持当前操作

> *CalrtErrorCCUExecFailed*

表明设备CCU执行错误

CalrtErrorWrongDirection

表明数据传输方向错误

2.3.1.9 *CalrtDeviceType_e*

描述当前设备类型。.

2.3.1.9.1 变量

`NO_TYPE = 0`

当前设备为未知类型。

`PCIE = 1`

the current device is the PCIe device. 当前设备是PCIe设备。

`USB = 2`

the current device is an USB device. 当前设备是一个USB设备。

EMU

当前设备是一个模拟器

2.3.1.10 *CalrtOutputBufStatus_e*

描述当前输出缓冲区的工作状态。

2.3.1.10.1 变量

`CALRT_OBUF_STATUS_DEFAULT = 0`

闲置。

`CALRT_OBUF_STATUS_PENDING = 1`

在软件启动队列中，尚未启动。

`CALRT_OBUF_STATUS_RUNNING = 2`

正在推理中，输出数据还没准备好

> `CALRT_OBUF_STATUS_DONE = 3`

在软件完成队列中，准备将数据读回主机。

2.3.1.11 *CalrtLogVerbosity_e*

描述log信息层级

2.3.1.11.1 变量

> `INFO = 0`

打印runtime消息

> `WARNING = 1`

打印警告消息和runtime消息

> `ERROR = 2`

打印错误消息，警告消息和runtime消息

> `DEBUG = 3`

打印更详细的消息，包括文件名和出错位置

> `TRACE = 4`

输出log文件。

2.3.1.12 *PowerMode*

描述设备电源模式

2.3.1.12.1 变量

> *BALANCE = 1*
> *HIGH_PERFORMANCE = 2*

2.3.1.13 *DebugOps_e*

描述bug 复现的选项

2.3.1.13.1 变量

DEB_DISABLE = 0

关闭debug功能

DEB_DEFUALT = 1

按默认方式debug，dump运行时的数据。infer index递增，文件输出路径为calbin路径。

DEB_ALTPATH = 2

infer index递增，但文件输出路径为自定义路径。会自动创建多层目录。但要有读写权限。

DEB_RSTALT = 3

infer index重置，不再递增；同时文件输出路径为自定义路径。会自动创建多层目录。但要有读写权限。

3 修改历史 Changing Log