弹性内存存储

Python SDK 接口参考

文档版本 01

发布日期 2026-05-25

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 使用前须知

本文介绍EMS SDK的版本变更，并提供版本兼容性说明，以及其他使用前须知。

变更及兼容性说明

如表1-1所示，本节将为您展示EMS SDK的版本变更情况，以及当前版本是否兼容某个历史版本的说明。

表 1-1 Python SDK 版本变更及兼容性说明

版本	变更类型	说明	是否兼容
26.4.0	新功能	添加分组存储功能	是
25.7.0	初始版本	支持访问Context Caching特性（大模型推理过程中计算产生的KV Cache缓存）相关读写接口。	-

其他使用前须知

请确认您已阅读弹性内存存储（Elastic Memory Service，EMS）产品文档。

2 SDK 接口概览

表2-1总结了EMS Python SDK支持的接口及功能描述，每个接口的详细介绍和示例代码请前往接口详情页查看。

SDK API 概览

表 2-1 Python SDK API 概览

接口名	方法	功能描述
EMS存储初始化	Ems.init	初始化EMS客户端。
获取Context Caching对象	Ems.get_cc	获取ContextCaching对象，用户后续可以访问对象中的读取/保存KVCache等接口。
健康检查	Ems.check_health	EMS健康检查，可以通过接口探测EMS内存池是否正常工作。
同步保存KVCache	ContextCaching.save	同步保存推理请求产生的KVCache到EMS内存池。
异步保存KVCache	ContextCaching.async_save	异步保存推理请求产生的KVCache到EMS内存池。
同步加载KVCache	ContextCaching.load	从EMS内存池中同步加载KVCache到加速卡显存。
异步加载KVCache	ContextCaching.async_load	从EMS内存池中异步加载KVCache到加速卡显存。
检查异步IO状态	ContextCaching.is_ready	检测返回的异步Future对象是否已完成计算。
获取异步IO结果	ContextCaching.get_result	获取返回的异步Future对象的结果。

3 使用前准备

在使用Python SDK访问华为云弹性内存服务EMS之前，您需要先完成推理/训练环境的准备。环境准备需要提前在本地完成环境搭建，比如下载安装依赖软件，安装开发工具等，以便您能顺利完成推理/训练环境准备，以及SDK的安装、基于SDK的代码开发与运行。

准备环境

- 在推理/训练容器内执行命令**npu-smio info**查看NPU卡信息，确保成功挂载NPU加速卡，同时安装CANN软件包。安装CANN软件包请参考：**昇腾CANN软件安装**。
- 确保将宿主机EMS服务端容器共享的unix domain socket目录"/mnt/paas/kubernetes/kubelet/ems"，通过增加负载配置文件**hostPath**项，将目录映射到推理/训练容器目录："**/dev/shm/ems**"；同时推理/训练容器内，运行服务的用户能够读写该文件夹及其文件。
- 从**Python官网**下载并安装合适的Python版本。

  推荐使用的Python 3.x版本：3.9、3.10版本。
- 从**PyCharm官网**下载并安装最新社区版本。
- 运行命令 **pip install pycryptodome==3.10.1**，安装加密库。
- 从**pip官网**下载并安装最新社区版本

4 下载与安装 SDK

本节提供Python SDK的下载链接，并介绍SDK的安装方式。

SDK 下载

EMS Python SDK安装包：请联系技术人员获取。

使用 pip 安装

步骤1 运行**pip -V**命令确保pip已安装。

步骤2 运行**pip install ems-.*linux_aarch64.whl**命令执行安装。

----结束

5 快速入门

初始化 EMS 客户端

本示例用于初始化EMS客户端配置并启动EMS服务。

```
# 引入模块
import os, torch, torch_npu
from ems import Ems, EmsConfig, EmsException, CcConfig, CcKvOption
# 初始化Ems
config = EmsConfig(cc_config=cc_config)
try:
    Ems.init(config)
except EmsException as e:
    print(f"exception: {e}.")
    exit(1)
```

更多关于EMS客户端初始化的内容请参考**初始化**章节。

读写 Context Caching

本示例通过初始化并获取Context Caching配置，保存和加载显存数据。

1. 初始化Ems,并向context cache注册一个KVCache结构。

```
import os, torch, torch_npu
from ems import Ems, EmsConfig, EmsException, CcConfig_v1, CcKvOption
# 初始化cc配置
cc_config = CcConfig_v1(rank_id=8, device_id=0, model_id="llama2-13b")
# 初始化Ems
config = EmsConfig(cc_config=cc_config)
try:
    Ems.init(config)
except EmsException as e:
    print(f"exception: {e}.")
    exit(1)
# 获取context caching对象
cc = Ems.get_cc()
if cc is None:
    print("cc is None.")
    exit(1)
# 注册到Context Caching
try:
     context_caching.register_kvcache(kvcache)
     # 期望kvcache形状: [layers, k_v_index, GPU_blocks, Block_size, heads, head_size]
     print("register_kvcache: success")
except EmsException as e:
     print(f"register_kvcache failed: {e}")
```

2. slot_mapping语义下的保存和加载显存数据，参数：slot_mapping + hashes + offsets。

```
# 设置save请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
# 保存
slot_mapping = [0,1,2,3,4,5,6,7]
hashes = [0xABCD, 0x1234]
offsets = [4, 4]
try:
    cc_result = cc.save(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets, option =
option)
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(1)
# 读取保存的数据
try:
    cc_result = cc.load(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets, option =
option)
except EmsException as e:
    print(f"failed to load, {e}.")
    exit(1)
```

3. chunk_descs 语义下的保存和加载显存数据，参数：hashes + chunk_descs。

```
# 设置save请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
# 保存
hashes = [0xABCD, 0x1234]
chunk_descs = [[(0, 4),(4, 4)]]
try:
    cc_result = cc.save(hashes = hashes, chunk_descs = chunk_descs, option = option)
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(1)
# 读取保存的数据
try:
    cc_result = cc.load(hashes = hashes, chunk_descs = chunk_descs, option = option)
except EmsException as e:
    print(f"failed to load, {e}.")
    exit(1)
```


6 初始化

6.1 EMS 存储初始化

功能说明

Ems是访问EMS服务的Python客户端，它为调用者提供一系列与EMS服务进行交互的接口，提供EMS内存池健康检查等能力，以及ContextCaching等存储功能。

方法定义

```
init(config)
```

构造函数参数描述

表 6-1 EmsConfig

参数名称	参数类型	是否必选	描述
cc_config	表6-2	否	**参数解释：** EMS存储SDK初始化配置。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

参数名称	参数类型	是否必选	描述
cc_config_v1	表6-3	否	**参数解释：** EMS存储V1版本SDK初始化配置。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。
access_id	string	否	**参数解释：** 业务访问内存池身份凭证，由用户指定并保证唯一性，用于多租隔离场景。 **约束限制：** 长度 1～512 个字符；支持数字、小写字母 "."、"-"、"_"；需全局唯一。 **取值范围：** 符合上述规则的字符串 **默认取值：** None
access_key	string	否	**参数解释：** SDK 访问内存池资源的密钥，与 access_id 配合使用。 **约束限制：** 长度 32～256 个字符；支持数字、大小写字母、"+"、"/"、"="。 **取值范围：** 符合上述规则的字符串 **默认取值：** None

表 6-2 CcConfig

参数名称	参数类型	是否必选	描述
rank_id	int	是	**参数解释：** 当前计算进程，使用加速卡的全局rank ID。 **约束限制：** ● rank ID实例内唯一。 **取值范围：** [0,当前实例rank个数)。 **默认取值**： 无。
device_id	int	是	**参数解释：** 当前计算进程，使用加速卡的本地节点rank ID。 **约束限制：** ● 节点内唯一。 **取值范围：** [0,当前实例本节点内rank个数) **默认取值**： 无。
model_id	string	是	**参数解释：** 唯一标识当前实例使用的推理模型ID。 **约束限制**： ● model_id创建规则：1～512个字符，支持数字、小写字母、“.”、“-”、“_”。 ● 需要保证全局唯一。 **取值范围：** 无。 **默认取值**： 无。
tp_world_size	int	否	**参数解释：** 实例TP并行度。 **约束限制**： 必须为数字。 **取值范围**： [1,当前实例总rank数]。 **默认取值：** 1。

参数名称	参数类型	是否必选	描述
pp_world_size	int	否	**参数解释：** 实例PP并行度。 **约束限制**： 必须为数字。 **取值范围**： [1,当前实例总rank数]。 **默认取值：** 1。
rank_in_tp_group	int	否	**参数解释：** 实例TP索引。 **约束限制**： 必须为数字。 **取值范围**： [0,当前实例rank个数)。 **默认取值：** 0。
rank_in_pp_group	int	否	**参数解释：** 实例PP索引。 **约束限制**： 必须为数字。 **取值范围**： [0,当前实例rank个数)。 **默认取值：** 0。
kvcache_type	string	否	**参数解释：** 指定KVCache的实现类型。 **约束限制**： 必须为字符串枚举值。 **取值范围**： "GQA"、"MHA"、"MLA"、"MQA"。 **默认取值：** "MHA"。

表 6-3 CcConfig_v1

参数名称	参数类型	是否必选	描述
rank_id	int	是	**参数解释：** 当前计算进程，使用加速卡的全局rank ID。 **约束限制：** ● rank ID实例内唯一。 **取值范围：** [0,当前实例rank个数)。 **默认取值**： 无。
device_id	int	是	**参数解释：** 当前计算进程，使用加速卡的本地节点rank ID。 **约束限制：** ● 节点内唯一。 **取值范围：** [0,当前实例本节点内rank个数) **默认取值**： 无。
model_id	string	是	**参数解释：** 唯一标识当前实例使用的推理模型ID。 **约束限制**： ● model_id创建规则：1～512个字符，支持数字、小写字母、“.”、“-”、“_”。 ● 需要保证全局唯一。 **取值范围：** 无。 **默认取值**： 无。
tp_world_size	int	否	**参数解释：** 实例TP并行度。 **约束限制**： 必须为数字。 **取值范围**： [1,当前实例总rank数]。 **默认取值：** 1。

参数名称	参数类型	是否必选	描述
pp_world_size	int	否	**参数解释：** 实例PP并行度。 **约束限制**： 必须为数字。 **取值范围**： [1,当前实例总rank数]。 **默认取值：** 1。
rank_in_tp_group	int	否	**参数解释：** 实例TP索引。 **约束限制**： 必须为数字。 **取值范围**： [0,当前实例rank个数)。 **默认取值：** 0。
rank_in_pp_group	int	否	**参数解释：** 实例PP索引。 **约束限制**： 必须为数字。 **取值范围**： [0,当前实例rank个数)。 **默认取值：** 0。
llm_engine	string	否	**参数解释：** 当前实例的推理引擎。 **约束限制**： 必须为字符串。 **取值范围**： 无（通常为"vllm"，如需对接其他推理引擎时可修改）。 **默认取值：** "vllm"。

参数名称	参数类型	是否必选	描述
kvcache_type	string	否	**参数解释：** 指定KVCache的实现类型。 **约束限制：** 必须为字符串枚举值。 **取值范围：** "GQA"、"MHA"、"MLA"、"MQA"。 **默认取值：** "MHA"。

说明

- 如果初始化参数校验失败或者因与EMS内存池连接失败，可以参考**异常处理**捕获异常，避免阻塞推理服务启动；同时根据异常信息，初步分析原因并联系EMS工程师。

代码样例

- 初始化Ems，示例为支持ContextCaching对象初始化：

```
# 引入模块
import os
from ems import Ems, EmsConfig, EmsException, CcConfig

# 初始化cc配置
cc_config = CcConfig(rank_id=8, device_id=0, model_id="llama2-13b")
# 初始化Ems
config = EmsConfig(cc_config=cc_config)
try:
    Ems.init(config)
except EmsException as e:
    print(f"exception: {e}.")
```

说明

- 一个进程中只会初始化一次Ems，以及Context Caching对象。

6.2 环境变量

变量描述

EMS SDK提供了以下环境变量。

表 6-4 环境变量

变量名称	变量类型	描述
EMS_SDK_LOG	string	**参数解释：** EMS SDK的profiling等特殊用途日志的打印目录。 **约束限制**： 无。 **取值范围**： 无。 **默认取值：** "/var/log/ems"。

代码样例

```
# 设置SDK打印目录
export EMS_SDK_LOG="/tmp/ems_dir"
```

7 Ems 存储相关接口

7.1 获取 Context Caching 对象

功能介绍

获取Ems初始化的Context Caching对象，在访问Context Caching的相关接口前使用。

方法定义

```
Ems.get_cc()
```

请求参数说明

无。

返回结果说明

表 7-1 返回结果

类型	说明
Context Caching	**参数解释：** 获取ContextCaching对象，用于后续访问传输KVCache的接口。如果Ems未初始化成功则返回None。 **取值范围：** 无。

代码样例

以下为获取ContextCaching对象示例，用于后续执行该对象内部的方法。

```
from ems import Ems, EmsConfig, EmsException, CcConfig
# 初始化cc配置
cc_config = CcConfig(rank_id=8, device_id=0, model_id="llama2-13b")
# 初始化Ems
```



```
config = EmsConfig(cc_config=cc_config)
try:
    Ems.init(config)
except EmsException as e:
    print(f"exception: {e}.")
    exit(1)
cc = Ems.get_cc()
if cc is None:
    print(f"cc is not initialized.")
```

7.2 健康检查

功能介绍

健康检查接口主要用于探测EMS存储服务状态是否工作正常，可以用于服务熔断后恢复场景。

方法定义

```
Ems.check_health()
```

请求参数说明

无。

返回结果说明

表 7-2 返回结果

类型	说明
bool	**参数解释：** EMS存储服务是否正常。 **取值范围：** False：表示EMS服务异常。 True：表示EMS服务正常。

代码样例

当调用SDK接口访问EMS服务时返回失败，业务可能会熔断EMS服务，可以使用定时检查EMS服务状态来实现自动恢复。

```
from ems import Ems, EmsConfig, EmsException, CcConfig
# 初始化cc配置
cc_config = CcConfig(rank_id=8, device_id=0, model_id="llama2-13b")
# 初始化Ems
config = EmsConfig(cc_config=cc_config)
try:
    Ems.init(config)
except EmsException as e:
    print(f"exception: {e}.")
    exit(1)
status = Ems.check_health()
if status is not True:
    print(f"ems is not ok.")
```



```
else:
    print(f"ems is ok")
```


8 ContextCaching 相关接口

8.1 注册 KVCache

功能介绍

将推理过程中使用的KVCache内存布局一次性注册到EMS上下文缓存（Context Caching）中，用于后续save/load/async_save/async_load接口按块（block）进行定位与管理。注册仅建立元数据与地址映射，不执行实际读写。

接口约束

- 仅V1版本需要注册KVCache
- 接口为同步调用，完成初始化后返回。
- KVCache期望形状应为：[layers, k_v_index, GPU_blocks, Block_size, heads, head_size]。
- Block_size将作为后续slot_mapping解析的基准，注册成功后内部记录block_size。
- 仅支持华为昇腾NPU显存中的torch.Tensor；CPU张量或其他设备张量不支持。

方法定义

```
ContextCaching.register_kvcache(kvcache, store_layer_group_desc)
```

请求参数说明

表 8-1 请求参数列表

参数名称	参数类型	是否必选	描述
kvcache	List[List[torch.Tensor]]	是	**参数解释：** 每层的KVCache结构，二维列表； 外层维度为layers，内层维度为k_v_index（为2，对应 K/V）。 最内层张量形状必须为[GPU_blocks, Block_size, heads, head_size]。 **约束限制：** 不能为空；必须全部位于NPU设备。 **取值范围：** 无。 **默认取值：** 无。
store_layer_group_desc	List[**表2 StoreLayerGroupDesc**]	否	**参数解释：** 描述不同层组存储的block_size以及对应的具体层id。 **约束限制：** 无。 **取值范围**： 无。 **默认取值：** None。

表 8-2 StoreLayerGroupDesc

参数名称	参数类型	是否必选	描述
store_block_size	int	是	**参数解释：** 该层组存储的block_size大小。 **约束限制：** 不能为None。 **取值范围**： 无。 **默认取值：** 无。

参数名称	参数类型	是否必选	描述
layer_ids	List[int]	是	**参数解释：** 该层组对应的具体层id集合。 **约束限制：** 不能为None。 **取值范围**： 无。 **默认取值：** 无。

返回结果说明

表 8-3 返回结果

类型	说明
None	**参数解释：** 无返回值。注册成功即表示KVCache结构就绪，可进行后续save/load/async_save/async_load。

代码样例

下面为向context cache注册一个KVCache结构，同时对异常进行容错处理。

```
import os, torch, torch_npu
from ems import Ems, EmsConfig, EmsException, CcConfig, CcKvOption
# 初始化cc配置
cc_config = CcConfig(rank_id=8, device_id=0, model_id="llama2-13b")
# 初始化Ems
config = EmsConfig(cc_config=cc_config)
try:
    Ems.init(config)
except EmsException as e:
    print(f"exception: {e}.")
    exit(1)
# 获取context caching对象
cc = Ems.get_cc()
if cc is None:
    print("cc is None.")
    exit(1)
# 设定形状
# 期望整体形状: [layers, k_v_index, GPU_blocks, Block_size, heads, head_size]
layers = 2
k_v_index = 2
GPU_blocks = 2
Block_size = 4
head = 2
head_size = 4
# 设置device和dtype
device = "npu:1"
dtype = torch.float16
# 构造最内层张量 (每个都是 [GPU_blocks, Block_size, heads, head_size])
```

```
k_t0 = torch.zeros((GPU_blocks, Block_size, heads, head_size), device=device, dtype=dtype)
v_t0 = torch.zeros((GPU_blocks, Block_size, heads, head_size), device=device, dtype=dtype)
k_t1 = torch.zeros((GPU_blocks, Block_size, heads, head_size), device=device, dtype=dtype)
v_t2 = torch.zeros((GPU_blocks, Block_size, heads, head_size), device=device, dtype=dtype)
# 组织成二维列表 List[List[Tensor]]：外层按 layer，内层按 K/V
# kvcache[l][0]->第l层的K，kvcache[l][1]-> 第l层的V
kvcache = [
    [k_t0, v_t0],  # 第 0 层
    [k_t1, v_t1],  # 第 1 层
]
# 注册到Context Caching
try:
    context_caching.register_kvcache(kvcache)
    print("register_kvcache: success")
except EmsException as e:
    print(f"register_kvcache failed: {e}")
```

8.2 同步保存 KVCache

功能介绍

将推理计算过程中加速卡产生的多个 **KVCache 数据块**，批量**同步保存到 EMS 缓存池**，并在保存完成后返回结果**表8-11**。

接口约束

- **同步接口**：接口会休眠阻塞。
- **加速卡限制**：仅支持华为昇腾加速卡显存拷贝。
- **异常处理**：若部分键值保存失败，接口不会直接返回失败，后续读取时将无法命中对应键值。
- **V1语义约束**：需遵循以下两种语义之一，二选一：
  - **slot_mapping 语义（使用 hashes + offsets + slot_mapping）**：通过 hashes , offsets来唯一标识 KVCache 数据块；
  - **chunk_descs 语义（使用 hashes+ chunk_descs）**：通过 **kwargs 传入 chunk_descs，标识每个层组中每个chunk对应的 start_slot, length。

方法定义

V0版本：

```
ContextCaching.save(option, key_list, value_list)
```

V1版本：

```
ContextCaching.save(slot_mapping, hashes, offsets, option, **kwargs)
```


请求参数说明

表 8-4 V0 请求参数列表

参数名称	参数类型	是否必选	描述
option	表8-9	是	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。
key_list	List[PrimaryKey]	是	**参数解释：** 访问内存池的键名列表。 **约束限制：** 所有键名必须唯一。 **取值范围：** 单个key的长度小于128，且保证全局唯一。 **默认取值：** None。
value_list	List[List[KvBufferWrapper]]	是	**参数解释：** ContextCaching访问内存池的值列表。 **约束限制：** 值列表的数目必须跟键列表中的数目相同，形成一一对应的键值对。 **取值范围：** 无。 **默认取值：** None。

表 8-5 PrimaryKey

参数名称	参数类型	是否必选	描述
distribute_key	string	必选	**参数解释：** 分布式key，DHT基于该key进行分布式路由。 **约束限制：** 只支持NPU显存地址。 **取值范围：** 大于0。 **默认取值：** 无
attribute_key	string	必选	**参数解释：** 属性key，相同的distribute_key下的所有attribute_key都会存储在同一个节点。 **约束限制：** 无 **取值范围：** 大于0。 **默认取值：** 无

表 8-6 KvBufferWrapper

参数名称	参数类型	是否必选	描述
data_ptr	int	必选	**参数解释：** 加速卡计算产生的KVCache连续显存起始地址。 **约束限制：** 只支持NPU显存地址。 **取值范围：** 大于0。 **默认取值：** 无

参数名称	参数类型	是否必选	描述
length	int	必选	**参数解释：** 加速卡计算产生的KVCache连续显存长度。 **约束限制：** 无 **取值范围：** 大于0。 **默认取值：** 无

表 8-7 V1 请求参数列表（slot_mapping 语义）

参数名称	参数类型	是否必选	描述
slot_mapping	List[int]	是	**参数解释：** 每个token索引映射到block标识的列表。 **约束限制：** 不能为空，必须为整数列表，sum(offsets) == len(slot_mapping)。 **取值范围：** 无。 **默认取值：** 无。
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** 需与offsets长度一致。 **取值范围：** 无。 **默认取值：** None。

参数名称	参数类型	是否必选	描述
offsets	List[int]	是	**参数解释：** 与hashes对应的每块token数，通常等于block_size。 **约束限制：** sum(offsets) == len(slot_mapping)。 **取值范围：** 无。 **默认取值：** None。
option	表8-9	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-8 V1 请求参数列表（chunk_descs 语义）

参数名称	参数类型	是否必选	描述
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** len(chunk_descs[i])=len(hashes)。 **取值范围：** 无。 **默认取值：** None。

参数名称	参数类型	是否必选	描述
chunk_descs	List[List[Tuple[int, int]]]	是	**参数解释：** 每个层组中每个chunk对应的start_slot, length。 **约束限制：** len(chunk_descs)=len(store_layer_group_desc)且len(chunk_descs[i])=len(hashes)。 **取值范围：** 无。 **默认取值：** None。
option	表8-9	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-9 CcKvOption

参数名称	参数类型	是否必选	描述
write_rcache	bool	可选	**参数解释：** 是否将本次写入保存为本地读缓存，默认值为True。 **约束限制：** 无。 **取值范围：** True：将本次写入保存为本地读缓存 False：不将本次写入保存为本地读缓存 **默认取值：** True

参数名称	参数类型	是否必选	描述
read_local_only	bool	可选	**参数解释：** 是否只读本地缓存，如果置为True，则不会从其他节点读取数据，只有读流程生效。 **约束限制：** 无。 **取值范围：** True：只读本地缓存 False：优先读本地缓存，如果本地未命中，则从其他节点读取数据 **默认取值：** False
timeout	int	可选	**参数解释：** 请求超时时间，单位为毫秒。 **约束限制：** 无。 **取值范围：** 大于等于0。 **默认取值：** 5000

返回结果说明

表 8-10 返回结果

类型	说明
表8-11	**参数解释：** Context Caching访问内存池的执行结果。 **取值范围：** 无。

表 8-11 CcResult

参数名称	参数类型	描述
success	int	**参数解释：** 请求的批量key读写连续成功的个数。 **约束限制：** 无。 **取值范围：** 0~请求key的个数。 **默认取值：** 无。
total	int	**参数解释：** 请求的批量key总个数。 **约束限制：** 无。 **取值范围：** 请求列表中批量key的个数。 **默认取值：** 无。

代码样例

下面为同步保存KVCache到EMS中，同时对异常进行容错处理。

1. V0版本同步保存

```
import torch, torch_npu
from ems import CcKvOption, PrimaryKey, KvBufferWrapper
option = CcKvOption(timeout=5000)
# 组成键值列表
key_list = [PrimaryKey("1", "123"), PrimaryKey("1", "66")]
tensor1 = torch.ones(2, device="npu:1")
tensor2 = torch.ones(6, device="npu:1")
len1 = tensor1.numel() * tensor1.element_size()
len2 = tensor2.numel() * tensor2.element_size()
val_list = [[KvBufferWrapper(tensor1.data_ptr, len1)], [KvBufferWrapper(tensor2.data_ptr, len2)]]
# 可以根据不同异常，采取不同处理方式，例如超时错误可以重试。
try:
    cc_result = cc.save(option, key_list, val_list)
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(2)
print(cc_result)
```

2. V1版本 slot_mapping语义下的同步保存，参数：slot_mapping + hashes + offsets

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
# 组成参数列表
slot_mapping = [0,1,2,3,4,5,6,7]
hashes = [0xABCD, 0x1234]
offsets = [4, 4]
# 可以根据不同异常，采取不同处理方式，例如超时错误可以重试。
```



```
try:
    cc_result = cc.save(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets, option =
option)
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(2)
print(cc_result)
```

3. V1版本 chunk_descs 语义下的同步保存，参数：hashes + chunk_descs

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
# 组成参数列表
hashes = [0xABCD, 0x1234]
chunk_descs = [[(0, 4),(4, 4)]]
# 可以根据不同异常，采取不同处理方式，例如超时错误可以重试。
try:
    cc_result = cc.save(hashes = hashes, chunk_descs = chunk_descs, option = option)
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(2)
print(cc_result)
```

8.3 异步保存 KVCache

功能介绍

将推理计算过程中产生的多个 **KVCache 数据块**，批量**异步保存到 EMS 缓存池**。方法调用在提交后会立即返回一个**表8-19句柄**，用户可通过该句柄在后续获取保存结果。

接口约束

- **加速卡限制**：仅支持华为昇腾加速卡显存拷贝。
- **异常处理**：若部分键值保存失败，接口不会直接返回失败，后续读取时将无法命中对应键值。
- **V1语义约束**：需遵循以下两种语义之一，二选一：
  - **slot_mapping 语义（使用 hashes + offsets + slot_mapping）**：通过 hashes , offsets来唯一标识 KVCache 数据块；
  - **chunk_descs 语义（使用 hashes+ chunk_descs）**：通过 **kwargs 传入 chunk_descs，标识每个层组中每个chunk对应的 start_slot, length。

方法定义

V0版本：

```
ContextCaching.async_save(option, key_list, value_list)
```

V1版本：

```
ContextCaching.async_save(slot_mapping, hashes, offsets, option, **kwargs)
```


请求参数说明

表 8-12 V0 请求参数列表

参数名称	参数类型	是否必选	描述
option	表8-17	是	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。
key_list	List[PrimaryKey]	是	**参数解释：** 访问内存池的键名列表。 **约束限制：** 所有键名必须唯一。 **取值范围：** 单个key的长度小于128，且保证全局唯一。 **默认取值：** None。
value_list	List[List[KvBufferWrapper]]	是	**参数解释：** ContextCaching访问内存池的值列表。 **约束限制：** 值列表的数目必须跟键列表中的数目相同，形成一一对应的键值对。 **取值范围：** 无。 **默认取值：** None。

表 8-13 PrimaryKey

参数名称	参数类型	是否必选	描述
distribute_key	string	必选	**参数解释：** 分布式key，DHT基于该key进行分布式路由。 **约束限制：** 只支持NPU显存地址。 **取值范围：** 大于0。 **默认取值：** 无
attribute_key	string	必选	**参数解释：** 属性key，相同的distribute_key下的所有attribute_key都会存储在同一个节点。 **约束限制：** 无 **取值范围：** 大于0。 **默认取值：** 无

表 8-14 KvBufferWrapper

参数名称	参数类型	是否必选	描述
data_ptr	int	必选	**参数解释：** 加速卡计算产生的KVCache连续显存起始地址。 **约束限制：** 只支持NPU显存地址。 **取值范围：** 大于0。 **默认取值：** 无

参数名称	参数类型	是否必选	描述
length	int	必选	**参数解释：** 加速卡计算产生的KVCache连续显存长度。 **约束限制：** 无 **取值范围：** 大于0。 **默认取值：** 无

表 8-15 请求参数列表（slot_mapping 语义）

参数名称	参数类型	是否必选	描述
slot_mapping	List[int]	是	**参数解释：** 每个token索引映射到block标识的列表。 **约束限制：** 不能为空，必须为整数列表，sum(offsets) == len(slot_mapping)。 **取值范围：** 无。 **默认取值：** 无。
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** 需与offsets长度一致。 **取值范围：** 无。 **默认取值：** None。

参数名称	参数类型	是否必选	描述
offsets	List[int]	是	**参数解释：** 与hashes对应的每块token数，通常等于block_size。 **约束限制：** sum(offsets) == len(slot_mapping)。 **取值范围：** 无。 **默认取值：** None。
option	表8-17	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-16 请求参数列表（chunk_descs 语义）

参数名称	参数类型	是否必选	描述
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** len(chunk_descs[i])=len(hashes)。 **取值范围：** 无。 **默认取值：** None。

参数名称	参数类型	是否必选	描述
chunk_descs	List[List[Tuple[int, int]]]	是	**参数解释：** 每个层组中每个chunk对应的start_slot, length。 **约束限制：** len(chunk_descs)=len(store_layer_group_desc)且len(chunk_descs[i])=len(hashes)。 **取值范围：** 无。 **默认取值：** None。
option	表8-17	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-17 CcKvOption

参数名称	参数类型	是否必选	描述
write_rcache	bool	可选	**参数解释：** 是否将本次写入保存为本地读缓存，默认值为True。 **约束限制：** 无。 **取值范围：** True：将本次写入保存为本地读缓存 False：不将本次写入保存为本地读缓存 **默认取值：** True

参数名称	参数类型	是否必选	描述
read_local_only	bool	可选	**参数解释：** 是否只读本地缓存，如果置为True，则不会从其他节点读取数据，只有读流程生效。 **约束限制：** 无。 **取值范围：** True：只读本地缓存 False：优先读本地缓存，如果本地未命中，则从其他节点读取数据 **默认取值：** False
timeout	int	可选	**参数解释：** 请求超时时间，单位为毫秒。 **约束限制：** 无。 **取值范围：** 大于等于0。 **默认取值：** 5000

返回结果说明

表 8-18 返回结果

类型	说明
表8-19	**参数解释：** 返回异步执行Future句柄。 **取值范围：** 无。

表 8-19 CcFuture

方法名称	参数	返回结果	描述
result	无	表8-11	**参数解释：** 获取异步执行的结果。 **取值范围：** 无。

表 8-20 CcResult

参数名称	参数类型	描述
success	int	**参数解释：** 请求的批量key读写连续成功的个数。 **约束限制：** 无。 **取值范围：** 0~请求key的个数。 **默认取值：** 无。
total	int	**参数解释：** 请求的批量key总个数。 **约束限制：** 无。 **取值范围：** 请求列表中批量key的个数。 **默认取值：** 无。

代码样例

本示例用于批量多次下发异步保存KVCache的请求，并获取每个请求的最终执行结果。

1. V0版本异步保存

```
import torch, torch_npu
from ems import CcKvOption, PrimaryKey, KvBufferWrapper
option = CcKvOption(timeout=5000)
# 组成键值列表
key_list = [PrimaryKey("1", "123"), PrimaryKey("1", "66")]
key_list2 = [PrimaryKey("2", "hello_world")]
tensor1 = torch.ones(2, device="npu:1")
tensor2 = torch.ones(6, device="npu:1")
tensor3 = torch.ones(1, 4, 2, device="npu:1")
len1 = tensor1.numel() * tensor1.element_size()
len2 = tensor2.numel() * tensor2.element_size()
val_list = [[KvBufferWrapper(tensor1.data_ptr, len1)], [KvBufferWrapper(tensor2.data_ptr, len2)]]
```

```
len3 = tensor2.numel() * tensor3.element_size()
val_list2 = [[KvBufferWrapper(tensor1.data_ptr, len3)]]
future_list = []
# 多次异步下发异步save请求，增加save并发能力。
try:
    for idx in range(2):
        future_list.append(cc.async_save(option, key_list, val_list))
        future_list.append(cc.async_save(option, key_list2, val_list2))
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(2)
try:
    for future in future_list:
        result = future.result()
        print(f"result:{result}")
except EmsException as e:
    print(f"failed to get result, {e}.")
```

2. V1版本 slot_mapping语义下的异步保存，参数：slot_mapping + hashes + offsets

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
future_list = []
# 多次异步下发异步save请求，增加save并发能力。
try:
    slot_mapping = [0, 1, 2, 3, 4, 5]
    hashes = [0x1111, 0x2222]
    offsets = [4, 4]
    future_list.append(cc.async_save(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets,
option = option))
    slot_mapping = [10, 11, 12, 13, 14, 15, 16, 17]
    hashes = [0xAAAA, 0xBBBB]
    offsets = [4, 4]
    future_list.append(cc.async_save(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets,
option = option))
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(2)
try:
    for future in future_list:
        result = future.result()
        print(f"rsult:{result}")
except EmsException as e:
    print(f"failed to get result, {e}.")
```

3. V1版本 chunk_descs 语义下的异步保存，参数：hashes + chunk_descs

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
future_list = []
# 多次异步下发异步save请求，增加save并发能力。
try:
    hashes = [0x1111, 0x2222]
    chunk_descs = [[(0, 4),(4, 4)]]
    future_list.append(cc.async_save(hashes = hashes, chunk_descs = chunk_descs, option = option))
    hashes = [0xAAAA, 0xBBBB]
    chunk_descs = [[(10, 4),(14, 4)]]
    future_list.append(cc.async_save(hashes = hashes, chunk_descs = chunk_descs, option = option))
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(2)
try:
    for future in future_list:
        result = future.result()
        print(f"result:{result}")
except EmsException as e:
    print(f"failed to get result, {e}.")
```


8.4 同步加载 KVCache

功能介绍

将推理请求需要的多个**KVCache数据块**，批量**同步加载到EMS缓存**中，并在保存完成后返回结果**表8-28**。

接口约束

- **同步接口**：接口会休眠阻塞。
- **加速卡限制**：仅支持华为昇腾加速卡显存拷贝。
- **异常处理**：若部分键值保存失败，接口不会直接返回失败，后续读取时将无法命中对应键值。
- **V1语义约束**：需遵循以下两种语义之一，二选一：
  - **slot_mapping 语义（使用 hashes + offsets + slot_mapping）**：通过 hashes , offsets来唯一标识 KVCache 数据块；
  - **chunk_descs 语义（使用 hashes+ chunk_descs）**：通过 **kwargs 传入 chunk_descs，标识每个层组中每个chunk对应的 start_slot, length。

方法定义

V0版本：

```
ContextCaching.load(option, key_list, value_list)
```

V1版本：

```
ContextCaching.load(slot_mapping, hashes, offsets, option, **kwargs)
```

请求参数说明

表 8-21 V0 请求参数列表

参数名称	参数类型	是否必选	描述
option	**表8-26**	是	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

参数名称	参数类型	是否必选	描述
key_list	List[**PrimaryKey**]	是	**参数解释：** 访问内存池的键名列表。 **约束限制：** 所有键名必须唯一。 **取值范围：** 单个key的长度小于128，且保证全局唯一。 **默认取值：** None。
value_list	List[List[**KvBufferWrapper**]]	是	**参数解释：** ContextCaching访问内存池的值列表。 **约束限制：** 值列表的数目必须跟键列表中的数目相同，形成一一对应的键值对。 **取值范围：** 无。 **默认取值：** None。

表 8-22 PrimaryKey

参数名称	参数类型	是否必选	描述
distribute_key	string	必选	**参数解释：** 分布式key，DHT基于该key进行分布式路由。 **约束限制：** 只支持NPU显存地址。 **取值范围：** 大于0。 **默认取值：** 无

参数名称	参数类型	是否必选	描述
attribute_key	string	必选	**参数解释：** 属性key，相同的distribute_key下的所有attribute_key都会存储在同一个节点。 **约束限制：** 无 **取值范围：** 大于0。 **默认取值：** 无

表 8-23 KvBufferWrapper

参数名称	参数类型	是否必选	描述
data_ptr	int	必选	**参数解释：** 加速卡计算产生的KVCache连续显存起始地址。 **约束限制：** 只支持NPU显存地址。 **取值范围：** 大于0。 **默认取值：** 无
length	int	必选	**参数解释：** 加速卡计算产生的KVCache连续显存长度。 **约束限制：** 无 **取值范围：** 大于0。 **默认取值：** 无

表 8-24 请求参数列表（slot_mapping 语义）

参数名称	参数类型	是否必选	描述
slot_mapping	List[int]	是	**参数解释：** 每个token索引映射到block标识的列表。 **约束限制：** 不能为空，必须为整数列表，sum(offsets) == len(slot_mapping)。 **取值范围：** 无。 **默认取值：** 无。
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** 需与offsets长度一致。 **取值范围：** 无。 **默认取值：** None。
offsets	List[int]	是	**参数解释：** 与hashes对应的每块token数，通常等于block_size。 **约束限制：** sum(offsets) == len(slot_mapping)。 **取值范围：** 无。 **默认取值：** None。
option	表8-26	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-25 请求参数列表（chunk_descs 语义）

参数名称	参数类型	是否必选	描述
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** len(chunk_descs[i])=len(hashes)。 **取值范围：** 无。 **默认取值：** None。
chunk_descs	List[List[Tuple[int, int]]]	是	**参数解释：** 每个层组中每个chunk对应的start_slot, length。 **约束限制：** len(chunk_descs)=len(store_layer_group_desc)且len(chunk_descs[i])=len(hashes)。 **取值范围：** 无。 **默认取值：** None。
option	表8-26	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-26 CcKvOption

参数名称	参数类型	是否必选	描述
write_rcache	bool	可选	**参数解释：** 是否将本次写入保存为本地读缓存，默认值为True。 **约束限制：** 无。 **取值范围：** True：将本次写入保存为本地读缓存 False：不将本次写入保存为本地读缓存 **默认取值：** True。
read_local_only	bool	可选	**参数解释：** 是否只读本地缓存，如果置为True，则不会从其他节点读取数据，只有读流程生效。 **约束限制：** 无。 **取值范围：** True：只读本地缓存 False：优先读本地缓存，如果本地未命中，则从其他节点读取数据 **默认取值：** False。
timeout	int	可选	**参数解释：** 请求超时时间，单位为毫秒。 **约束限制：** 无。 **取值范围：** 大于等于0。 **默认取值：** 5000。

返回结果说明

表 8-27 返回结果

类型	说明
表8-28	**参数解释：** Context Caching访问内存池的执行结果。 **取值范围：** 无。

表 8-28 CcResult

参数名称	参数类型	描述
success	int	**参数解释：** 请求的批量key读写连续成功的个数。 **约束限制：** 无。 **取值范围：** 0~请求key的个数。 **默认取值：** 无。
total	int	**参数解释：** 请求的批量key总个数。 **约束限制：** 无。 **取值范围：** 请求列表中批量key的个数。 **默认取值：** 无。

代码样例

下面示例描述了同步加载KVCache流程，同时对返回异常进行处理。

1. V0版本同步加载

```
import torch, torch_npu
from ems import CcKvOption, PrimaryKey, KvBufferWrapper
option = CcKvOption(timeout=5000)
# 组成键值列表
key_list = [PrimaryKey("1", "123"), PrimaryKey("1", "66")]
tensor1 = torch.ones(2, device="npu:1")
tensor2 = torch.ones(6, device="npu:1")
len1 = tensor1.numel() * tensor1.element_size()
len2 = tensor2.numel() * tensor2.element_size()
```

```
val_list = [[KvBufferWrapper(tensor1.data_ptr, len1)], [KvBufferWrapper(tensor2.data_ptr, len2)]]
# 可以根据不同异常，采取不同处理方式，例如超时错误可以重试。
try:
    cc_result = cc.load(option, key_list, val_list)
except EmsException as e:
    print(f"failed to load, {e}.")
    exit(2)
print(cc_result)
```

2. V1版本 slot_mapping语义下的同步加载，参数：slot_mapping + hashes + offsets

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
# 组成参数列表
slot_mapping = [0, 1, 2, 3, 4, 5]
hashes = [0x1111, 0x2222]
offsets = [4, 4]
# 可以根据不同异常，采取不同处理方式，例如超时错误可以重试。
try:
    cc_result = cc.load(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets, option =
option)
except EmsException as e:
    print(f"failed to load, {e}.")
    exit(2)
print(f"succeed to load key num {cc_result.success}.")
print(cc_result)
```

3. V1版本 chunk_descs 语义下的同步加载，参数：hashes + chunk_descs

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
# 组成参数列表
hashes = [0xABCD, 0x1234]
chunk_descs = [[(0, 4),(4, 4)]]
# 可以根据不同异常，采取不同处理方式，例如超时错误可以重试。
try:
    cc_result = cc.load(hashes = hashes, chunk_descs = chunk_descs, option = option)
except EmsException as e:
    print(f"failed to load, {e}.")
    exit(2)
print(f"succeed to load key num {cc_result.success}.")
print(cc_result)
```

8.5 异步加载 KVCache

功能介绍

将推理请求需要的多个**KVCache数据块**，批量异**步加载到EMS缓存**中。方法调用在提交后会立即返回一个**表8-36句柄**，用户可通过该句柄在后续获取保存结果。

接口约束

- **加速卡限制**：仅支持华为昇腾加速卡显存拷贝。
- **异常处理**：若部分键值保存失败，接口不会直接返回失败，后续读取时将无法命中对应键值。
- **V1语义约束**：需遵循以下两种语义之一，二选一：
  - **slot_mapping 语义（使用 hashes + offsets + slot_mapping）**：通过 hashes , offsets来唯一标识 KVCache 数据块；
  - **chunk_descs 语义（使用 hashes+ chunk_descs）**：通过 **kwargs 传入 chunk_descs，标识每个层组中每个chunk对应的 start_slot, length。

方法定义

V0版本：

```
ContextCaching.async_load(option, key_list, value_list)
```

V1版本：

```
ContextCaching.async_load(slot_mapping, hashes, offsets, option, **kwargs)
```

请求参数说明

表 8-29 V0 请求参数列表

参数名称	参数类型	是否必选	描述
option	表8-34	是	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。
key_list	List[PrimaryKey]	是	**参数解释：** 访问内存池的键名列表。 **约束限制：** 所有键名必须唯一。 **取值范围：** 单个key的长度小于128，且保证全局唯一。 **默认取值：** None。
value_list	List[List[KvBufferWrapper]]	是	**参数解释：** ContextCaching访问内存池的值列表。 **约束限制：** 值列表的数目必须跟键列表中的数目相同，形成一一对应的键值对。 **取值范围：** 无。 **默认取值：** None。

表 8-30 PrimaryKey

参数名称	参数类型	是否必选	描述
distribute_key	string	必选	**参数解释：** 分布式key，DHT基于该key进行分布式路由。 **约束限制：** 只支持NPU显存地址。 **取值范围：** 大于0。 **默认取值：** 无
attribute_key	string	必选	**参数解释：** 属性key，相同的distribute_key下的所有attribute_key都会存储在同一个节点。 **约束限制：** 无 **取值范围：** 大于0。 **默认取值：** 无

表 8-31 KvBufferWrapper

参数名称	参数类型	是否必选	描述
data_ptr	int	必选	**参数解释：** 加速卡计算产生的KVCache连续显存起始地址。 **约束限制：** 只支持NPU显存地址。 **取值范围：** 大于0。 **默认取值：** 无

参数名称	参数类型	是否必选	描述
length	int	必选	**参数解释：** 加速卡计算产生的KVCache连续显存长度。 **约束限制：** 无 **取值范围：** 大于0。 **默认取值：** 无

表 8-32 请求参数列表（slot_mapping 语义）

参数名称	参数类型	是否必选	描述
slot_mapping	List[int]	是	**参数解释：** 每个token索引映射到block标识的列表。 **约束限制：** 不能为空，必须为整数列表，sum(offsets) == len(slot_mapping)。 **取值范围：** 无。 **默认取值：** 无。
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** 需与offsets长度一致。 **取值范围：** 无。 **默认取值：** None。

参数名称	参数类型	是否必选	描述
offsets	List[int]	是	**参数解释：** 与hashes对应的每块token数，通常等于block_size。 **约束限制：** sum(offsets) == len(slot_mapping)。 **取值范围：** 无。 **默认取值：** None。
option	表8-34	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-33 请求参数列表（chunk_descs 语义）

参数名称	参数类型	是否必选	描述
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** len(chunk_descs[i])=len(hashes)。 **取值范围：** 无。 **默认取值：** None。

参数名称	参数类型	是否必选	描述
chunk_descs	List[List[Tuple[int, int]]]	是	**参数解释：** 每个层组中每个chunk对应的start_slot, length。 **约束限制：** len(chunk_descs)=len(store_layer_group_desc)且len(chunk_descs[i])=len(hashes)。 **取值范围：** 无。 **默认取值：** None。
option	表8-34	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-34 CcKvOption

参数名称	参数类型	是否必选	描述
write_rcache	bool	可选	**参数解释：** 是否将本次写入保存为本地读缓存，默认值为True。 **约束限制：** 无。 **取值范围：** True：将本次写入保存为本地读缓存。 False：不将本次写入保存为本地读缓存。 **默认取值：** True。

参数名称	参数类型	是否必选	描述
read_local_only	bool	可选	**参数解释：** 是否只读本地缓存，如果置为True，则不会从其他节点读取数据，只有读流程生效。 **约束限制：** 无。 **取值范围：** True：只读本地缓存。 False：优先读本地缓存，如果本地未命中，则从其他节点读取数据。 **默认取值：** False。
timeout	int	可选	**参数解释：** 请求超时时间，单位为毫秒。 **约束限制：** 无。 **取值范围：** 大于等于0。 **默认取值：** 5000。

返回结果

表 8-35 返回结果

类型	说明
表8-23	**参数解释：** 返回异步执行Future句柄。 **取值范围：** 无。

表 8-36 CcFuture

方法名称	参数	返回结果	描述
result	无	表8-24	**参数解释：** 获取异步执行的结果。 **取值范围：** 无。

表 8-37 CcResult

参数名称	参数类型	描述
success	int	**参数解释：** 请求的批量key读写连续成功的个数。 **约束限制：** 无。 **取值范围：** 0~请求key的个数。 **默认取值：** 无。
total	int	**参数解释：** 请求的批量key总个数。 **约束限制：** 无。 **取值范围：** 请求列表中批量key的个数。 **默认取值：** 无。

代码样例

本示例用于多次下发异步加载KVCache的请求，并获取每个请求的最终执行结果 。

1. V0版本异步加载

```
import torch, torch_npu
from ems import CcKvOption, PrimaryKey, KvBufferWrapper
option = CcKvOption(timeout=5000)
# 组成键值列表
key_list = [PrimaryKey("1", "123"), PrimaryKey("1", "66")]
key_list2 = [PrimaryKey("2", "hello_world")]
tensor1 = torch.ones(2, device="npu:1")
tensor2 = torch.ones(6, device="npu:1")
tensor3 = torch.ones(1, 4, 2, device="npu:1")
len1 = tensor1.numel() * tensor1.element_size()
len2 = tensor2.numel() * tensor2.element_size()
val_list = [[KvBufferWrapper(tensor1.data_ptr, len1)], [KvBufferWrapper(tensor2.data_ptr, len2)]]
len3 = tensor2.numel() * tensor3.element_size()
```



```
val_list2 = [[KvBufferWrapper(tensor1.data_ptr, len3)]]
future_list = []
# 多次异步下发异步load请求，增加load并发能力。
try:
    for idx in range(2):
        future_list.append(cc.async_load(option, key_list, val_list))
        future_list.append(cc.async_load(option, key_list2, val_list2))
except EmsException as e:
    print(f"failed to load, {e}.")
    exit(2)
try:
    for future in future_list:
        result = future.result()
        print(f"result:{result}")
except EmsException as e:
    print(f"failed to get result, {e}.")
```

2. V1版本 slot_mapping语义下的异步加载，参数：slot_mapping + hashes + offsets

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
future_list = []
# 多次异步下发异步load请求，增加load并发能力。
try:
    slot_mapping = [0, 1, 2, 3, 4, 5]
    hashes = [0x1111, 0x2222]
    offsets = [4, 4]
    future_list.append(cc.async_load(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets,
option = option))
    slot_mapping = [10, 11, 12, 13, 14, 15, 16, 17]
    hashes = [0xAAAA, 0xBBBB]
    offsets = [4, 4]
    future_list.append(cc.async_ load (slot_mapping = slot_mapping, hashes = hashes, offsets = offsets,
option = option))
except EmsException as e:
    print(f"failed to save, {e}.")
    exit(2)
try:
    for future in future_list:
        result = future.result()
        print(f"rsult:{result}")
except EmsException as e:
    print(f"failed to get result, {e}.")
```

3. V1版本 chunk_descs 语义下的异步加载，参数：hashes + chunk_descs

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
future_list = []
# 多次异步下发异步load请求，增加load并发能力。
try:
    for idx in range(2):
        hashes = [0x1111, 0x2222]
        chunk_descs = [[(0, 4),(4, 4)]]
        future_list.append(cc.async_load(hashes = hashes, chunk_descs = chunk_descs, option = option))
        hashes = [0xAAAA, 0xBBBB]
        chunk_descs = [[(10, 4),(14, 4)]]
        future_list.append(cc.async_load(hashes = hashes, chunk_descs = chunk_descs, option = option))
except EmsException as e:
    print(f"failed to load, {e}.")
    exit(2)
try:
    for future in future_list:
        result = future.result()
        print(f"result:{result}")
except EmsException as e:
    print(f"failed to get result, {e}.")
```


8.6 检查异步 IO 状态

功能介绍

用于检测async_save、async_load等接口返回的异步Future对象是否已完成计算。调用is_ready不会阻塞，适合在轮询或超时控制场景下使用。

接口约束

- 接口为非阻塞调用。
- 输入必须是CcFuture类型对象，由async_save/async_load返回。
- 仅判断完成状态，不返回计算结果。

方法定义

```
ContextCaching.is_ready(ccfuture)
```

请求参数说明

表 8-38 请求参数列表

参数名称	参数类型	是否必选	描述
ccfuture	表8-39	是	**参数解释：** 异步计算返回的 Future 对象。 **约束限制：** 必须为async_save或async_load等接口返回的 CcFuture 实例，不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-39 CcFuture

方法名称	参数	返回结果	描述
result	无	表8-44	**参数解释：** 获取异步执行的结果。 **取值范围：** 无。

返回结果

表 8-40 返回结果

类型	说明
bool	**参数解释：** 异步计算是否完成。返回True表示已完成；返回False表示尚未完成。 **取值范围：** 无。

代码样例

本示例用于多次下发异步加载KVCache的请求，并获取每个请求的最终执行结果。

1. slot_mapping语义下的验证异步计算是否完成，参数：slot_mapping + hashes + offsets

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
slot_mapping = [0, 1, 2, 3, 4, 5, 6, 7]
hashes = [0xABCD, 0x1234]
offsets = [4, 4]
# 发起异步保存
future = cc.async_save(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets, option = option)
# 使用 is_ready 非阻塞检查
if cc.is_ready(future):
    print("async_save(hash) 已完成")
else:
    print("async_save(hash) 仍在进行中")
```

2. chunk_descs 语义下的验证异步计算是否完成，参数：hashes + chunk_descs

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
block_size = 4
hashes = [0xABCD, 0x1234]
chunk_descs = [[(0, 4),(4, 4)]]
# 发起异步保存
future = cc.async_save(hashes = hashes, chunk_descs = chunk_descs, option = option)
# 使用 is_ready 非阻塞检查
if cc.is_ready(future):
    print("async_save(token) 已完成")
else:
    print("async_save(token) 仍在进行中")
```

8.7 获取异步 IO 结果

功能介绍

用于获取async_save、async_load等接口返回的异步Future对象的结果。调用get_result将阻塞，直到异步计算完成，并返回以token为单位的结果对象。

接口约束

- 接口会休眠阻塞，直到异步结果可用。
- 输入必须是CcFuture类型对象，由 async_save/async_load返回。

方法定义

```
ContextCaching.get_result(ccfuture)
```

请求参数说明

表 8-41 请求参数列表

参数名称	参数类型	是否必选	描述
ccfuture	表8-42	是	**参数解释：** 异步计算返回的Future对象。 **约束限制：** 必须为async_save或async_load等接口返回的CcFuture实例，不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-42 CcFuture

方法名称	参数	返回结果	描述
result	无	表8-44	**参数解释：** 获取异步执行的结果。 **取值范围：** 无。

返回结果

表 8-43 返回结果

类型	说明
表8-44	**参数解释：** Context Caching访问内存池的执行结果。 **取值范围：** 无。

表 8-44 CcResult

参数名称	参数类型	描述
success	int	**参数解释：** 请求的批量key读写连续成功的个数。 **约束限制：** 无。 **取值范围：** 0~请求key的个数。 **默认取值：** 无。
total	int	**参数解释：** 请求的批量key总个数。 **约束限制：** 无。 **取值范围：** 请求列表中批量key的个数。 **默认取值：** 无。

代码样例

本示例用于异步保存KVCache的请求，并获取每个请求的最终执行结果。

1. slot_mapping语义下的保存和加载显存数据，参数：slot_mapping + hashes + offsets

```
# 设置save请求的超时时间
option = CcKvOption(timeout=5000)
# 异步保存，获取结果
block_size = 4
slot_mapping = [0, 1, 2, 3, 4, 5, 6, 7]
hashes = [0xABCD, 0x1234]
offsets = [4, 4]
try:
    fut = cc.async_save(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets, option = option)
    res = cc.get_result(fut)
    print("async_save(hash) result:", res)
except EmsException as e:
    print("async_save(hash) result:", res)
# 异步读取，获取结果
try:
    fut = cc.async_load(slot_mapping = slot_mapping, hashes = hashes, offsets = offsets, option = option)
    res = cc.get_result(fut)
    print("async_load(hash) result:", res)
except EmsException as e:
    print(f"async_load(hash) failed: {e}")
```

2. chunk_descs 语义下的保存和加载显存数据，参数：hashes + chunk_descs

```
# 设置save请求的超时时间
option = CcKvOption(timeout=5000)
# 异步保存，获取结果
```



```
block_size = 4
hashes = [0xABCD, 0x1234]
chunk_descs = [[(0, 4),(4, 4)]]
try:
    fut = cc.async_save(hashes = hashes, chunk_descs = chunk_descs, option = option)
    res = cc.get_result(fut)
    print("async_save(token) result:", res)
except EmsException as e:
    print(f"async_save(token) failed: {e}")
# 异步读取，获取结果
try:
    fut = cc.async_load(hashes = hashes, chunk_descs = chunk_descs, option = option)
    res = cc.get_result(fut)
    print("async_load(token) result:", res)
except EmsException as e:
    print(f"async_load(token) failed: {e}")
```

8.8 同步查询 KvCache

功能介绍

用于查询 EMS 缓存池中是否存在已保存的 KVCache 数据块，查询结果将以**表CcResult**形式返回。

接口约束

- 同步接口：接口会休眠阻塞。
- 加速卡限制：仅支持华为昇腾加速卡显存查询。
- 异常处理：若 KVCache 数据块查询超时或失败，接口返回0命中，框架将重新进行计算。

方法定义

```
ContextCaching.exists(hashes, option)
```

请求参数说明

表 8-45 请求参数列表

参数名称	参数类型	是否必选	描述
hashes	List[int]	是	**参数解释：** 预计算的块前缀哈希列表。 **约束限制：** 无。 **取值范围：** 无。 **默认取值：** None。

参数名称	参数类型	是否必选	描述
option	表8-46	否	**参数解释：** ContextCaching访问内存池的KV操作选项。 **约束限制：** 不能为None。 **取值范围：** 无。 **默认取值：** 无。

表 8-46 CcKvOption

参数名称	参数类型	是否必选	描述
write_rcache	bool	可选	**参数解释：** 是否将本次写入保存为本地读缓存，默认值为True。 **约束限制：** 无。 **取值范围：** True：将本次写入保存为本地读缓存 False：不将本次写入保存为本地读缓存 **默认取值：** True
read_local_only	bool	可选	**参数解释：** 是否只读本地缓存，如果置为True，则不会从其他节点读取数据，只有读流程生效。 **约束限制：** 无。 **取值范围：** True：只读本地缓存 False：优先读本地缓存，如果本地未命中，则从其他节点读取数据 **默认取值：** False

参数名称	参数类型	是否必选	描述
timeout	int	可选	**参数解释：** 请求超时时间，单位为毫秒。 **约束限制：** 无。 **取值范围：** 大于等于0。 **默认取值：** 5000

返回结果说明

表 8-47 返回结果

类型	说明
表8-48	**参数解释：** Context Caching访问内存池的执行结果。 **取值范围：** 无。

表 8-48 CcResult

参数名称	参数类型	描述
success	int	**参数解释：** 请求的批量key读写连续成功的个数。 **约束限制：** 无。 **取值范围：** 0~请求key的个数。 **默认取值：** 无。

参数名称	参数类型	描述
total	int	**参数解释：** 请求的批量key总个数。 **约束限制：** 无。 **取值范围：** 请求列表中批量key的个数。 **默认取值：** 无。

代码样例

下面为从EMS中同步查询KVCache，同时对异常进行容错处理。

```
# 设置请求的超时时间
option = CcKvOption(timeout=5000)
# 组成参数列表
hashes = [0xABCD, 0x1234]
# 可以根据不同异常，采取不同处理方式，例如超时错误可以重试。
try:
    cc_result = cc.exists(hashes = hashes, option = option)
except EmsException as e:
    print(f"failed to exists, {e}.")
    exit(2)
print(cc_result)
```

9 异常处理

9.1 获取错误码

调用EMS接口时，需要对EmsExcepiton异常进行捕获，并根据异常中不同错误码进行不同策略处理。

功能介绍

EMS捕获到接口异常，可以通过接口获取异常状态码。

方法定义

```
EmsExcepiton.status_code()
```

常见的状态码及其含义：

EMS异常状态码	描述	常见原因	解决方法
EMS_INVALID_ARGUMENT	请求参数错误	• 请求参数不合法。	检查参数，修改后重试。
EMS_IO_ERROR	请求IO错误	• EMS内存池故障。 • SDK到EMS内存池连接断开。	将EMS服务隔离，待健康检查通过后恢复。
EMS_IO_TIMEOUT	请求IO超时	• EMS内存池IO压力较大，业务繁忙。 • SDK业务压力大，导致IO排队时间久。	增加超时时间后重试，或者将EMS服务隔离，待健康检查通过后恢复。

EMS异常状态码	描述	常见原因	解决方法
EMS_INTERNAL_ERROR	内部错误	• SDK内部故障。 • 访问加速卡内存故障。	将EMS服务隔离，待健康检查通过后恢复。

代码样例

本示例用于获取异常错误码为超时，进行重试 。

```
import os
import time
import torch, torch_npu
from ems import Ems, EmsConfig, EmsException, CcConfig_v1, CcKvOption
from ems.common.exception import EmsErrorCode
# 初始化cc配置
cc_config = CcConfig_v1(rank_id=8, device_id=0, model_id="llama2-13b")
# 初始化Ems
config = EmsConfig(cc_config_v1=cc_config)
try:
    Ems.init(config)
except EmsException as e:
    print(f"exception: {e}.")
    exit(1)
# 获取context caching对象
cc = Ems.get_cc()
if cc is None:
    print("cc is None.")
    exit(1)
# 设置save请求的超时时间
option = CcKvOption(timeout=100)
 # slot_mapping语义下的保存和加载显存数据，参数：slot_mapping + hashes + offsets
block_size = 4
# 组成参数列表
slot_mapping = [0, 1, 2, 3, 4, 5]
hashes = [0x1111, 0x2222]
offsets = [4, 4]
# 支持超时重试。
retries = 0
while True:
    try:
        result = cc.load(slot_mapping, hashes, offsets, option)
        break
    except EmsException as e:
        if e.status_code() != EmsErrorCode.EMS_IO_TIMEOUT or retries >= 3:
            print(f"failed to load cause unrecoverable error or retries is more than 3:  {e.status_code()}.")
            exit(2)
        else:
            # 增加超时时间，并睡眠10ms后重试
            option.timeout = 1000
            time.sleep(0.01)
            print(f"retry after sleep 10ms.")
print(f"load result: {result}")
```

9.2 获取详细信息

功能介绍

EMS捕获到接口执行异常，可以通过接口获取异常详细信息。

方法定义

```
EmsExcepiton.message()
```

返回结果

表 9-1 返回结果

类型	说明
string	**参数解释：** 返回异常详细说明。 **取值范围：** 无。

代码样例

本示例用于获取异常详细描述信息 。

```
import os
import torch, torch_npu
from ems import Ems, EmsConfig, EmsException, CcConfig_v1, CcKvOption
# 初始化cc配置
cc_config = CcConfig_v1(rank_id=8, device_id=0, model_id="llama2-13b")
# 初始化Ems
config = EmsConfig(cc_config_v1=cc_config)
try:
    Ems.init(config)
except EmsException as e:
    print(f"exception: {e}.")
    exit(1)
# 获取context caching对象
cc = Ems.get_cc()
if cc is None:
    print("cc is None.")
    exit(1)
# 设置save请求的超时时间
option = CcKvOption(timeout=5000)
 # slot_mapping语义下的保存和加载显存数据，参数：slot_mapping + hashes + offsets
block_size = 4
# 组成参数列表
slot_mapping = [0, 1, 2, 3, 4, 5]
hashes = [0x1111, 0x2222]
offsets = [4, 4]
# 多次异步下发异步load请求，增加load并发能力。
try:
    result = cc.load(slot_mapping, hashes, offsets, option)
except EmsException as e:
    print(f"failed to load, message: {e.message()}.")
```

10 常见问题

10.1 EMS 初始化失败如何定位?

在推理框架的日志中搜索**"EMS init failed"**关键字，如果对应的时间为出问题的时间点，则收集该日志，联系EMS工程师定位。

10.2 ContextCaching 接口超时时间如何设置?

Context Caching的读写相关接口执行时间，跟请求并发数、每个请求的键值对数量有关系，当前单个请求超时时间默认5秒，用户可以根据SLO（Service Level Objective，服务级别目标，例如吞吐、首token时延等）、请求batch数和KVCache数据量，合理设置超时时间。例如：长序列场景要求的首token时延是5秒，超时时间建议设置为3秒。