PR #27273 完整报告

sgl-project/sglang

[mem_cache][5/N] refactor: extract host KV cache base layer into pool_host package

合并时间：2026-06-20 20:44

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27273

执行摘要

- 一句话：提取主机 KV 缓存基类层到 pool_host 子包
- 推荐动作：推荐核心架构和缓存模块的开发者精读，了解模块化重构策略。PR 中的设计决策（如 HiSparse 单独文件避免依赖 MLA 族）和自动化还原脚本值得参考。

功能与动机

重构 memory_pool_host.py（约 2960 行），将其拆分为模块化子包，为后续将具体 Pool 类逐个分离到独立模块做准备。关联 issue #25371。

实现拆解

步骤 1：创建 pool_host 包，包含 base.py (HostKVCache 抽象基类)、common.py (HostTensorAllocator 与分配辅助函数)、hisparse.py (HiSparseHostPoolMixin)。步骤 2：从 memory_pool_host.py 中移除上述符号，改为从 pool_host 子包导入。步骤 3：更新 12 个外部文件的导入路径，包括 7 处 HostKVCache、4 处分配函数、1 处 mooncake_store.py 分离导入。步骤 4：保留 mooncake 分配器的惰性导入以维持向后兼容。步骤 5：运行单元测试验证 (21 passed) 并提供还原脚本确保可复现。

关键文件：

- python/sglang/srt/mem_cache/pool_host/base.py（模块 主机池；类别 source；类型 refactor；符号 synchronized, wrapper, HostKVCache, init）：新文件，包含 HostKVCache 抽象基类和 synchronized 装饰器，是整个 host KV cache 层的核心接口。
- python/sglang/srt/mem_cache/memory_pool_host.py（模块 主机池；类别 source；类型 refactor；符号 MHATokenToKVPoolHost, MLATokenToKVPoolHost）：原主机池实现文件，本次移除大量代码并改为从 pool_host 子包导入，仍保留所有具体 Pool 类。
- python/sglang/srt/mem_cache/pool_host/common.py（模块 主机池；类别 source；类型 refactor；符号 HostTensorAllocator, init, allocate, get_allocator_from_storage）：新文件，包含 HostTensorAllocator、获取分配器的工厂函数、cudaHostRegister 辅助函数以及 ALLOC_MEMORY_FUNCS 映射。
- python/sglang/srt/mem_cache/pool_host/hisparse.py（模块 主机池；类别 source；类型 refactor；符号 HiSparseHostPoolMixin, _round_up_to_page_size, alloc_page, alloc_paged_token_slots）：新文件，包含 HiSparseHostPoolMixin 混入类，为 MLA 和 DeepSeekV4 池提供分页分配方法。

- python/sglang/srt/mem_cache/pool_host/__init__.py（模块 主机池；类别 source；类型 refactor）：新包初始化文件，仅暴露 HostKVCache 和 HostTensorAllocator 为公共 API。
- python/sglang/srt/mem_cache/storage/mooncake_store/mooncake_store.py（模块 Mooncake 存储；类别 source；类型 dependency-wiring）：修改导入路径，从 memory_pool_host 切换到 pool_host 包，并继续保持 MLATokenToKVPoolHost 的导入。

关键符号：HostKVCache.init, HostKVCache.get_size_per_token, HostKVCache.init_kv_buffer, HostTensorAllocator.allocate, get_allocator_from_storage, alloc_with_host_register, alloc_with_pin_memory, HiSparseHostPoolMixin.alloc_paged_token_slots, synchronized

关键源码片段

python/sglang/srt/mem_cache/pool_host/base.py

新文件，包含 HostKVCache 抽象基类和 synchronized 装饰器，是整个 host KV cache 层的核心接口。

```
from __future__ import annotations

import abc
import logging
import threading
from functools import wraps
from typing import Optional

import psutil
import torch

from sglang.srt.mem_cache.memory_pool import KVCache
from sglang.srt.mem_cache.pool_host.common import get_allocator_from_storage

logger = logging.getLogger(__name__)

# 保留 10 GB 主机内存用于操作系统和其他进程
HICACHE_HOST_MEMORY_RESERVE_BYTES: int = 10 * (1024**3)


def synchronized(func):
    """装饰器，用 self.lock 保护方法执行。"""
    @wraps(func)
    def wrapper(self, *args, **kwargs):
        with self.lock:
            return func(self, *args, **kwargs)
    return wrapper


class HostKVCache(abc.ABC):
    """主机端 KV 缓存抽象基类，所有具体 Host Pool 的公共接口。"""
```

```
def __init__(
    self,
    device_pool: KVCache,
    host_to_device_ratio: float,
    host_size: int,
    page_size: int,
    layout: str,
    pin_memory: bool,
    device: str,
    allocator_type: str = "default",
):
    self.device_pool = device_pool
    self.page_size = page_size
    self.layout = layout
    self.pin_memory = pin_memory
    self.device = device
    self.allocator = get_allocator_from_storage(allocator_type)
    self.can_use_write_back_jit = False

    self.dtype = device_pool.store_dtype
    self.size_per_token = self.get_size_per_token()
    if host_size > 0:
        self.size = int(host_size * 1e9 // self.size_per_token)
    else:
        self.size = int(device_pool.size * host_to_device_ratio)
    # 页面数向上对齐（注意：+1 会多分配一页，后续可优化）
    self.page_num = self.size // self.page_size + 1
    self.size = self.page_num * self.page_size
    self.start_layer = device_pool.start_layer
    self.end_layer = device_pool.end_layer

    assert self.size > device_pool.size, "主机内存必须大于设备内存"

    # 验证可用主机内存
    host_mem = psutil.virtual_memory()
    requested_bytes = self.size * self.size_per_token
    available_bytes = host_mem.available - HICACHE_HOST_MEMORY_RESERVE_BYTES
    if requested_bytes > available_bytes:
        raise ValueError(
            f"Not enough host memory available. Requesting "
            f"{requested_bytes / 1e9:.2f} GB but only have "
            f"{available_bytes / 1e9:.2f} GB free."
        )

    self.kv_buffer = self.init_kv_buffer()
    self.lock = threading.RLock()
    self.clear()

@abc.abstractmethod
```

```
    def get_size_per_token(self):
        ...

    @abc.abstractmethod
    def init_kv_buffer(self):
        ...

    @abc.abstractmethod
    def load_to_device_per_layer(self, device_pool, host_indices, device_indices, layer_id, io_
    backend) -> None:
        ...

    @abc.abstractmethod
    def backup_from_device_all_layer(self, device_pool, host_indices, device_indices, io_backend)
    -> None:
        ...
```

评论区精华

review 中 gemini-code-assist[bot] 指出 base.py 中 page_num 计算使用了多余的 +1（应使用 (size + page_size - 1) // page_size 对齐），以及 common.py 中 alloc_with_pin_memory 的类型提示 allocator: None 过于严格，应改为 HostTensorAllocator | None。另外 xiezhq-hermann 询问 common.py 是否有必要单独存在，alphabetc1 回答基类与实用工具分离是合理的架构。

- 页面数向上对齐算法 (correctness): PR 未采纳此建议，仍使用原公式。后续可优化。
- 类型提示过于严格 (style): PR 未修改，类型提示仍为 None，可能引起类型检查器警告。
- common.py 是否必要 (design): 设计被坚持，文件保持独立。

风险与影响

- 风险：本 PR 为纯机械迁移，逻辑无变更，但若遗留的导入路径未完全更新或第三方代码直接引用 memory_pool_host 中的符号，可能导致 ImportError。PR 提供了全仓库扫描确认无外部调用者引用已移动符号。风险较低。
- 影响：对用户透明，无运行时行为变化。对开发者，host KV 缓存模块的组织更清晰，便于后续按 pool 类型逐步剥离。影响范围覆盖所有使用 host KV cache 的存储后端（Mooncake、EIC、HF3FS、NIXL 等）。
- 风险标记：核心路径变更（host KV cache），缺少新测试覆盖（依赖已有单元测试），多存储后端导入调整

关联脉络

- PR #26675 [mem_cache] allocator series PR 1（推测）: PR body 指出本 PR 是已完成分配器系列的 sibling，该系列包括 #26675 和 #26676。
- PR #26676 [mem_cache] allocator series PR 2（推测）: 同上，分配器系列的另一部分。