PR #26676 完整报告

sgl-project/sglang

[mem_cache][2/N] refactor: move SWATokenToKVPoolAllocator to allocator/swa.py

合并时间：2026-06-04 15:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26676

执行摘要

- 一句话：移动 SWATokenToKVPoolAllocator 至 allocator/swa.py
- 推荐动作：该 PR 适合所有 mem_cache 模块的开发者精读。值得关注的设计决策包括： 1) 通过 git blame -C 保留历史的方法； 2) 不在 __init__.py 中重新导出的原因（避免循环导入）； 3) 机械重构时的零风险迁移实践。对于普通使用者，只需知道这是纯重构即可。

功能与动机

根据父 Issue #25371（分配器提升系列）的规划，将 SWATokenToKVPoolAllocator 从 swa_memory_pool.py 分离到独立的 allocator/swa.py 中。此举使分配器逻辑与内存池（SWAKVPool）解耦，让 allocator 子包成为所有分配器的统一存放位置，提升代码可维护性和模块化。PR body 明确说明这是系列的第 2 个 PR，构建于 #26675（分配器子包骨架）之上。

实现拆解

1. 创建 allocator/swa.py：将 SWATokenToKVPoolAllocator 类完整移入新文件，并调整导入关系：使用子包内的直接模块路径（base、paged、token）而非通过 __init__.py，避免字母顺序导致的循环导入。
2. 精简 swa_memory_pool.py：删除该类及关联的 NPU 导入（is_npu、_is_npu 等），因为这些仅用于分配器分支选择，现在随类移至新文件。清理了失效的 PagedTokenToKVPoolAllocator、TokenToKVPoolAllocator 等导入。
3. 更新 16 个调用站点：将 from sglang.srt.mem_cache.swa_memory_pool import SWATokenToKVPoolAllocator 改为 from sglang.srt.mem_cache.allocator.swa import SWATokenToKVPoolAllocator，其中部分站点拆分联合导入（如 model_runner_kv_cache_mixin.py 拆分为两行）。
4. 验证 git 历史：git blame -C -C -C 可追溯 99.7% 的原始提交历史（368/369 行），唯一新增行是改写后的子模块导入。
5. 测试配套：运行了 1162 个单元测试（含 SWA、radix cache、kv-canary 等）及 GSM8K 精度 / 速度基准，结果与基线一致，无回归。

关键文件：

- python/sglang/srt/mem_cache/allocator/swa.py（模块 分配器；类别 source；类型 core-logic；符号 SWATokenToKVPoolAllocator, init, available_size, full_available_size）：核心新增文件，包含移动后的 SWATokenToKVPoolAllocator 类实现，是重构的目标文件。

- python/sglang/srt/mem_cache/swa_memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 SWAKVPool）：源文件，移除了 SWATokenToKVPoolAllocator 类和相关 NPU 导入，仅保留 SWAKVPool 类。
- python/sglang/srt/model_executor/model_runner_kv_cache_mixin.py（模块 模型执行器；类别 source；类型 data-contract）：主调用点之一，导入路径从 swa_memory_pool 改为 allocator.swa。
- python/sglang/srt/kv_canary/api.py（模块 金丝雀；类别 source；类型 entrypoint）：另一个关键调用点，导入路径更新。
- python/sglang/srt/kv_canary/runner/canary_manager.py（模块 金丝雀；类别 source；类型 dependency-wiring）：类型检查块中的导入更新。
- python/sglang/srt/kv_canary/runner/swa_divergence.py（模块 金丝雀；类别 source；类型 dependency-wiring）：类型检查块中的导入更新。
- python/sglang/srt/layers/attention/flashinfer_backend.py（模块 注意力；类别 source；类型 dependency-wiring）：导入更新。
- python/sglang/srt/managers/schedule_policy.py（模块 调度器；类别 source；类型 dependency-wiring）：导入更新。
- python/sglang/srt/mem_cache/chunk_cache.py（模块 内存缓存；类别 source；类型 dependency-wiring）：导入更新。
- python/sglang/srt/mem_cache/common.py（模块 内存缓存；类别 source；类型 dependency-wiring）：导入更新。
- python/sglang/srt/mem_cache/swa_radix_cache.py（模块 内存缓存；类别 source；类型 dependency-wiring）：导入更新。
- python/sglang/srt/mem_cache/unified_cache_components/swa_component.py（模块 统一缓存；类别 source；类型 dependency-wiring）：导入更新。

关键符号：SWATokenToKVPoolAllocator.init, SWATokenToKVPoolAllocator.available_size, SWATokenToKVPoolAllocator.full_available_size, SWATokenToKVPoolAllocator.swa_available_size, SWATokenToKVPoolAllocator.alloc, SWATokenToKVPoolAllocator.alloc_extend

关键源码片段

python/sglang/srt/mem_cache/allocator/swa.py

核心新增文件，包含移动后的 SWATokenToKVPoolAllocator 类实现，是重构的目标文件。

```
# allocator/swa.py (new file)
# 从 swa_memory_pool.py 移动而来，导入改为子包内直接路径以避免循环导入

import torch

from sglang.srt.mem_cache.allocator.base import BaseTokenToKVPoolAllocator
from sglang.srt.mem_cache.allocator.paged import PagedTokenToKVPoolAllocator
from sglang.srt.mem_cache.allocator.token import TokenToKVPoolAllocator
from sglang.srt.mem_cache.base_swa_memory_pool import BaseSWAKVPool
```

```
from sglang.srt.utils import is_npu
from sglang.srt.utils.common import get_num_new_pages

_is_npu = is_npu()

if _is_npu:
    from sglang.srt.hardware_backend.npu.allocator_npu import (
        NPUPagedTokenToKVPoolAllocator,
    )


class SWATokenToKVPoolAllocator(BaseTokenToKVPoolAllocator):
    """Allocator for SWA hybrid KV cache."""

    def __init__(
        self,
        size: int,
        size_swa: int,
        page_size: int,
        dtype: torch.dtype,
        device: str,
        kvcache: BaseSWAKVPool,
        need_sort: bool,
    ):
        # 初始化全注意力和 SWA 分配器
        full_kv_pool = getattr(kvcache, "full_kv_pool", None)
        swa_kv_pool = getattr(kvcache, "swa_kv_pool", None)

        if page_size == 1:
            self.full_attn_allocator = TokenToKVPoolAllocator(
                size, dtype, device, full_kv_pool, need_sort,
            )
            self.swa_attn_allocator = TokenToKVPoolAllocator(
                size_swa, dtype, device, swa_kv_pool, need_sort,
            )
        else:
            if _is_npu:
                PagedTokenToKVPoolAllocatorClass = NPUPagedTokenToKVPoolAllocator
            else:
                PagedTokenToKVPoolAllocatorClass = PagedTokenToKVPoolAllocator
            self.full_attn_allocator = PagedTokenToKVPoolAllocatorClass(
                size, page_size, dtype, device, full_kv_pool, need_sort,
            )
            self.swa_attn_allocator = PagedTokenToKVPoolAllocatorClass(
                size_swa, page_size, dtype, device, swa_kv_pool, need_sort,
            )
        # 映射全注意力位置到 SWA 位置，末尾附加 -1 用于边界处理
        self.full_to_swa_index_mapping = torch.cat([
            torch.zeros(size + self.page_size, dtype=torch.int64, device=device),
```

```
            torch.tensor([-1], dtype=torch.int64, device=device),
        ])
        self._kvcache = kvcache
        self.clear()
        self._kvcache.register_mapping(self.full_to_swa_index_mapping)

    def available_size(self):
        return min(
            self.full_attn_allocator.available_size(),
            self.swa_attn_allocator.available_size(),
        )
```

python/sglang/srt/mem_cache/swa_memory_pool.py

源文件，移除了 SWATokenToKVPoolAllocator 类和相关 NPU 导入，仅保留 SWAKVPool 类。

```
# swa_memory_pool.py (modified)
# 移除了分配器类和相关导入，仅保留 SWAKVPool

import logging
from typing import Dict, List, Optional, Tuple

import torch

from sglang.srt.layers.radix_attention import RadixAttention
from sglang.srt.mem_cache.base_swa_memory_pool import BaseSWAKVPool
from sglang.srt.mem_cache.memory_pool import KVCache, MHATokenToKVPool
from sglang.srt.mem_cache.utils import maybe_init_custom_mem_pool

logger = logging.getLogger(__name__)
GB = 1024 * 1024 * 1024


class SWAKVPool(BaseSWAKVPool):
    """KV cache with separate pools for full and SWA attention layers."""
    # ... 类体保持不变
```

python/sglang/srt/model_executor/model_runner_kv_cache_mixin.py

主调用点之一，导入路径从 swa_memory_pool 改为 allocator.swa。

```
# model_runner_kv_cache_mixin.py ( 部分 )
# 导入更新：从旧路径拆分为两行

from sglang.srt.mem_cache.allocator import (
    PagedTokenToKVPoolAllocator,
    TokenToKVPoolAllocator,
)
from sglang.srt.mem_cache.allocator.swa import SWATokenToKVPoolAllocator # 新增行
from sglang.srt.mem_cache.deepseek_v4_memory_pool import DeepSeekV4TokenToKVPool
# ...
```

```
from sglang.srt.mem_cache.swa_memory_pool import SWAKVPool # 只保留 SWAKVPool
```

评论区精华

- 遗漏测试文件导入（P1，correctness）：Codex 指出 test_unified_radix_cache_bench.py 仍从旧路径导入，可能导致基准测试失败。作者未直接回复，但 PR 测试覆盖率显示该文件已通过，推测已修复。
- 包级 API 一致性建议（P2，design）：Codex 建议在 allocator/__init__.py 中重新导出 SWATokenToKVPoolAllocator。作者在 PR body 解释：由于内部模块必须直接互相导入（通过 __init__ 会因字母加载顺序导致循环导入），故选择不重新导出。

导入位置调整（style）：ispoboc要求将from future 语句放在注释下方，作者已调整。

- 遗漏测试文件中的导入路径 (correctness): 作者未直接回复，但 PR 测试全部通过（包括该基准测试文件），推测此问题已在后续修正或该测试文件实际未受影响。
- 是否在 init.py 中重新导出 (design): 作者在 PR body 解释：因为内部模块必须直接互相导入，通过 __init__ 会因字母加载顺序导致循环导入（swa 在 token 之前加载），故选择不重新导出。
- 导入语句位置调整 (style): 作者回应“Done”，调整了导入位置。

风险与影响

- 风险：该 PR 是纯代码移动，不改变运行时行为。主要风险：若有外部模块或未发现的测试文件仍从旧路径导入，会导致 ImportError。作者提供了可复现的转换脚本和全面的测试套件（1162 个测试通过），且通过 git blame -C 确保历史可追溯，大幅降低风险。另外，allocator 子包内部不使用 __init__.py 导出，可能造成包级 API 不一致，但属于有意设计，无功能影响。对生产环境无性能或安全影响。
- 影响：影响范围：仅涉及 SGLang 运行时中 SWA（滑动窗口注意力）KV 缓存分配器相关代码。对用户透明，无 API 变更。对团队：重构后分配器集中在 allocator/swa.py，便于未来引入新分配器类型。涉及 16 个文件导入更新，但通过脚本保证一致性。测试覆盖充分，回归风险低。影响程度：中等，因属于内部重构，外部依赖方无需改动。
- 风险标记：核心路径变更，导入重定向

关联脉络

- PR #26675 allocator subpackage skeleton: 本 PR 基于 #26675（分配器子包骨架）构建，是系列重构的第 2 步。
- PR #25371 allocator-promotion series: 父 Issue，跟踪整个分配器提升系列，本 PR 是其中的一部分。