PR #24718 完整报告

sgl-project/sglang

feat(kv-events): publish SWA radix cache events

合并时间：2026-05-12 01:31

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24718

执行摘要

- 一句话：SWA radix cache 新增 KV 事件发布
- 推荐动作：建议精读。本 PR 是 SWA 缓存事件机制的首个实现，设计上复用已有 KVCacheEventMixin，耦合度低。值得关注的设计决策包括：事件记录仅覆盖主要生命周期（store/remove/evict），未覆盖所有细粒度操作（如 match 命中），这可能是与 RadixCache 已有实现对齐的权衡。后续可扩展至更多事件类型。

功能与动机

PR body 明确指出需要为 SWARadixCache 添加 KV 缓存事件发布能力，覆盖 store、tombstone、full eviction 和 split-hash 行为，以利用已有的事件机制统一缓存变更的观测与响应。

实现拆解

1. 导入 mixin 与辅助函数：在 python/sglang/srt/mem_cache/swa_radix_cache.py 中新增 from sglang.srt.mem_cache.events import KVCacheEventMixin，以及 from sglang.srt.mem_cache.utils import split_node_hash_value，为事件记录提供所需工具。
2. 修改类定义与初始化：将 SWARadixCache 的基类从单一 BasePrefixCache 改为多继承 KVCacheEventMixin, BasePrefixCache，并在 __init__ 中增加 self.enable_kv_cache_events 配置和 self.kv_event_queue = [] 队列。同时为 TreeNode 添加 hash_value 字段存储每个 page 的哈希值。
3. 在缓存操作中插入事件记录：在 reset、evict（两个分支：tombstone 分支和完整 eviction 分支）以及 _compact_single_child_chain 等关键方法中调用 self._record_all_cleared_event() 或 self._record_remove_event(x)，确保在适当生命周期节点发布事件。
4. 配套测试：在 test/registered/unit/mem_cache/test_swa_unittest.py 中新增 test_swa_radix_cache_kv_events 和 test_swa_radix_cache_kv_events_split_hash 两个测试，验证基本事件（store 数量、token 顺序、eviction 事件）和 split-hash 场景（插入后分裂节点、parent_block_hash 正确性）。

关键文件：

- python/sglang/srt/mem_cache/swa_radix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 SWARadixCache）：核心源码文件，实现 SWARadixCache 的事件发布

机制，包括继承 KVCacheEventMixin、添加事件队列、在 reset/evict/compact 等关键路径调用事件记录方法。

- test/registered/unit/mem_cache/test_swa_unittest.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_swa_radix_cache_kv_events, test_swa_radix_cache_kv_events_split_hash）：新增两个端到端测试，验证 KV 事件在 SWA radix cache 中的正确性（基本事件、token 顺序、eviction 事件类型、split-hash parent 关系）。

关键符号：SWARadixCache.init, SWARadixCache.reset, SWARadixCache.evict, SWARadixCache._compact_single_child_chain, test_swa_radix_cache_kv_events, test_swa_radix_cache_kv_events_split_hash

关键源码片段

python/sglang/srt/mem_cache/swa_radix_cache.py

核心源码文件，实现 SWARadixCache 的事件发布机制，包括继承 KVCacheEventMixin、添加事件队列、在 reset/evict/compact 等关键路径调用事件记录方法。

```
# python/sglang/srt/mem_cache/swa_radix_cache.py
# 导入事件 mixin 和辅助函数
from sglang.srt.mem_cache.events import KVCacheEventMixin
from sglang.srt.mem_cache.utils import convert_to_bigram_key, split_node_hash_value

# 类定义改为多继承：优先 KVCacheEventMixin，然后 BasePrefixCache
class SWARadixCache(KVCacheEventMixin, BasePrefixCache):
    def __init__(self, params: CacheInitParams):
        ...
        self.is_eagle = params.is_eagle
        # 新增：是否启用 KV 缓存事件；由外部配置控制，默认关闭
        self.enable_kv_cache_events = params.enable_kv_cache_events
        # 新增：事件队列，用于暂存缓存生命周期事件（如 BlockStored, BlockRemoved）
        self.kv_event_queue = []
        ...

    def reset(self) -> None:
        ...
        self.root_node.hash_value = [] # 树根的 hash_value 初始化为空列表
        ...
        # 新增：触发“全部清除”事件，通知下游缓存已被重置
        self._record_all_cleared_event()

    def evict(self, params: EvictParams) -> EvictResult:
        # 在节点 evict 时记录移除事件
        # 第一个分支：tombstone 后的节点完全释放
        if x.swa_tombstone:
            self._record_remove_event(x)
            self.token_to_kv_pool_allocator.free(x.value)
            ...
```

```
            # 第二个分支：直接释放完整叶节点
            else:
                self._record_remove_event(x)
                self.token_to_kv_pool_allocator.free(x.value)
                ...

    def _compact_single_child_chain(self, node: TreeNode) -> None:
        # 在节点合并时传播 hash_value，确保 split-hash 信息一致
        if node.hash_value:
            child.hash_value = node.hash_value
```

test/registered/unit/mem_cache/test_swa_unittest.py

新增两个端到端测试，验证 KV 事件在 SWA radix cache 中的正确性（基本事件、token 顺序、eviction 事件类型、split-hash parent 关系）。

```
# test/registered/unit/mem_cache/test_swa_unittest.py
# 新增导入事件类型
from sglang.srt.disaggregation.kv_events import BlockRemoved, BlockStored

# 辅助函数无需修改，但 _build_swa_tree 新增 enable_kv_cache_events 参数
def _build_swa_tree(..., enable_kv_cache_events: bool = False):
    ...
    tree = SWARadixCache(
        params=CacheInitParams(
            ...,
            enable_kv_cache_events=enable_kv_cache_events, # 传入配置
        ),
    )
    return tree, allocator, req_to_token_pool

class TestSWA(unittest.TestCase):

    def test_swa_radix_cache_kv_events(self):
        # 构建树并启用事件
        tree, allocator, _ = _build_swa_tree(is_eagle=False, enable_kv_cache_events=True)
        tree.take_events() # 清除 reset 事件

        # 插入 [1,2,3,4]，应生成 4 个 BlockStored 事件
        _insert(tree, allocator, [1, 2, 3, 4])
        first_insert_events = [e for e in tree.take_events() if isinstance(e, BlockStored)]
        self.assertEqual(len(first_insert_events), 4)
        self.assertEqual([e.token_ids[0] for e in first_insert_events], [1, 2, 3, 4])

        # 插入 [1,2,3,4,5,6]（前缀共享），应仅对新 token 5,6 生成事件
        _insert(tree, allocator, [1, 2, 3, 4, 5, 6])
        second_insert_events = [e for e in tree.take_events() if isinstance(e, BlockStored)]
        self.assertEqual(len(second_insert_events), 2)
        self.assertEqual([e.token_ids[0] for e in second_insert_events], [5, 6])
        stored_hashes = [e.block_hashes[0] for e in first_insert_events + second_insert_events]
```

```
        # 仅 evict SWA tokens: 触发 tombstone，不应产生 BlockRemoved
        result = tree.evict(EvictParams(num_tokens=0, swa_num_tokens=1))
        self.assertEqual([e for e in tree.take_events() if isinstance(e, BlockRemoved)], [])

        # evict 完整 token: 应产生与之前 store 数量相同的 BlockRemoved
        result = tree.evict(EvictParams(num_tokens=1, swa_num_tokens=0))
        removed_hashes = [e.block_hashes[0] for e in tree.take_events() if isinstance(e,
        BlockRemoved)]
        self.assertCountEqual(removed_hashes, stored_hashes)

    def test_swa_radix_cache_kv_events_split_hash(self):
        tree, allocator, _ = _build_swa_tree(is_eagle=False, enable_kv_cache_events=True)
        tree.take_events() # Clear reset event

        # 插入 [1,2,3,4]
        _insert(tree, allocator, [1, 2, 3, 4])
        first_events = [e for e in tree.take_events() if isinstance(e, BlockStored)]
        self.assertEqual(len(first_events), 4)
        split_parent_hash = first_events[1].block_hashes[0] # 记录第二次插入的 hash

        # 插入 [1,2,5,6]，应在 1-2 节点处分裂，新分支的 parent_block_hash 应与 split_parent_
        hash 一致
        _insert(tree, allocator, [1, 2, 5, 6])
        second_events = [e for e in tree.take_events() if isinstance(e, BlockStored)]
        self.assertEqual(len(second_events), 2)
        self.assertEqual(second_events[0].token_ids, [5])
        self.assertEqual(second_events[0].parent_block_hash, split_parent_hash)
```

评论区精华

该 PR 只有一个提交，无 review 评论（0 个 review_comments），因此无公开讨论记录。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 回归风险：SWARadixCache 是混合 KV 缓存的核心模块，新增事件队列和 hash 字段可能引入内存开销或时序问题。但改动集中于额外的事件记录路径，不影响原有 evict/insert/match 主逻辑，且测试已有覆盖。
  2. 性能风险：每个事件记录操作会增加少量开销（对象分配、队列追加），但对吞吐敏感的场景（如高并发推理）可能有多毫秒级影响。不过事件队列仅在显式启用时才工作（由 enable_kv_cache_events 控制），默认不会有额外开销。
  3. 兼容性风险：CacheInitParams 新增 enable_kv_cache_events 字段，但该配置项由外层赋值，不会引入破坏性变更。 - 影响：影响范围：限于 SWA（Sliding Window Attention） radix cache 模块。启用后，下游消费者（如调度器、monitoring 系统）可订阅 BlockStored、BlockRemoved 等事件，实现缓存状态的可观测性。影响程度：中等

——这是一个可选功能，默认关闭，对现有用户无影响；但为未来 disaggregatedserving 高级场景提供基础。 - 风险标记：核心路径变更，新增可选功能

关联脉络

- PR #24950 fix: SGLANG_RADIX_FORCE_MISS chunk-cache passthrough: 同样修改了 mem_cache 模块（base_prefix_cache），但本 PR 专注于 SWA 的事件发布，两者共同构成 KV 缓存层的可观测性增强