PR #49114 完整报告

vllm-project/vllm

Add CachePolicyFactory for pluggable/external eviction policies

合并时间：2026-07-29 12:34

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49114

执行摘要

- 一句话：添加 CachePolicyFactory 支持可插拔缓存策略
- 推荐动作：值得精读，展示了遵循已有工厂模式（OffloadingSpecFactory）进行一致扩展的实践。review 中对 out-of-tree 加载的设计决策有参考意义。建议同时关注后续对 OffloadingSpec 警告位置的统一调整。

功能与动机

CPUOffloadingManager 通过私有字典解析 eviction_policy（'lru'/'arc'），外部包无法注册自己的 CachePolicy 实现。仿照 OffloadingSpecFactory 和 KVConnectorFactory 的模式，添加工厂以支持可插拔策略，避免用户需要手动替换 self._policy。

实现拆解

1. 创建工厂类：在 vllm/v1/kv_offload/cpu/policies/factory.py 中新增 CachePolicyFactory，包含类级字典 _registry、register_cache_policy 和 get_cache_policy_cls 方法。支持懒加载和 module_path 回退。
2. 改造管理器：修改 vllm/v1/kv_offload/cpu/manager.py，将 CPUOffloadingManager.__init__ 中的私有字典替换为 CachePolicyFactory.get_cache_policy_cls()，新增 cache_policy_module_path 参数。类型标注从 Literal['lru','arc'] 改为 str。
3. 预注册内置策略：在 factory.py 末尾预注册 lru 和 arc 策略，确保零行为变化。移除 manager.py 中直接导入 LRUCachePolicy 和 ARCCachePolicy 的语句。
4. 调整策略基类：修改 base.py，将 CachePolicy.__init__ 从抽象方法改为非抽象，并直接设置 self.cache_capacity，避免子类重复实现。
5. 更新调用链：修改 vllm/v1/kv_offload/tiering/spec.py 和 cpu/spec.py，传入 cache_policy_module_path 参数，移除已无效的 # type: ignore[arg-type] 注释。同步更新 tiering/manager.py 和 factory.py 的兼容性。
6. 完善测试：新增 tests/v1/kv_offload/cpu/policies/test_factory.py，覆盖预注册策略可达性、注册 / 解析自定义策略、未注册策略异常、重复注册异常、out-of-tree 动态加载等场景。
7. 更新文档：在 docs/features/kv_offloading_usage.md 中说明 eviction_policy 和 cache_policy_module_path 用法。

关键文件：

- vllm/v1/kv_offload/cpu/policies/factory.py（模块 策略工厂；类别 source；类型 dependency-wiring；符号 CachePolicyFactory, register_cache_policy, loader, get_cache_policy_cls）：新增文件，核心工厂类 CachePolicyFactory 实现策略注册和懒加载解析，是 PR 的中心变更。
- vllm/v1/kv_offload/cpu/manager.py（模块 管理器；类别 source；类型 dependency-wiring；符号 CPUOffloadingManager, CPUOffloadingManager.init）：修改管理器使用工厂解析策略，是依赖注入的关键变化。
- tests/v1/kv_offload/cpu/policies/test_factory.py（模块 测试；类别 test；类型 test-coverage；符号 _DummyCachePolicy, TestCachePolicyFactory, restore_cache_policy_registry）：新测试文件，全面覆盖工厂功能。
- vllm/v1/kv_offload/cpu/policies/base.py（模块 策略基类；类别 source；类型 core-logic；符号 CachePolicy.init）：修改 CachePolicy 基类，去除 __init__抽象性并设置 cache_capacity。
- docs/features/kv_offloading_usage.md（模块 文档；类别 docs；类型 documentation）：文档更新说明新参数和 out-of-tree 策略用法。

关键符号：CachePolicyFactory.register_cache_policy, CachePolicyFactory.get_cache_policy_cls, CPUOffloadingManager.init, CachePolicy.init

关键源码片段

vllm/v1/kv_offload/cpu/policies/factory.py

新增文件，核心工厂类 CachePolicyFactory 实现策略注册和懒加载解析，是 PR 的中心变更。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
import importlib
from collections.abc import Callable

from vllm.logger import init_logger
from vllm.v1.kv_offload.cpu.policies.base import CachePolicy

logger = init_logger(__name__)


class CachePolicyFactory:
    """Registry for CachePolicy implementations, resolved by name.

    Mirrors OffloadingSpecFactory (vllm/v1/kv_offload/factory.py): built-in
    policies are pre-registered below. External policies can either
    register_cache_policy() a friendly short name up front, or skip
    registration entirely and pass a module path at lookup time (out-of-tree,
    no vLLM fork/patch required) -- see get_cache_policy_cls.
    """

    _registry: dict[str, Callable[[], type[CachePolicy]]] = {}
```

```
@classmethod
def register_cache_policy(
    cls, name: str, module_path: str, class_name: str
) -> None:
    """Register a cache policy with a lazy-loading module and class name."""
    if name in cls._registry:
        raise ValueError(f"Cache policy '{name}' is already registered.")

    def loader() -> type[CachePolicy]:
        module = importlib.import_module(module_path)
        return getattr(module, class_name)

    cls._registry[name] = loader

@classmethod
def get_cache_policy_cls(
    cls, name: str, module_path: str | None = None
) -> type[CachePolicy]:
    """Get a cache policy class by name.

    Args:
        name: Name of the cache policy. Checked against the registry
            first; if it's not registered and `module_path` is given,
            `name` is imported from there instead -- an out-of-tree
            policy needs no register_cache_policy() call at all, just
            this module path passed through config (mirrors
            OffloadingSpecFactory.get_spec_cls's spec_module_path
            fallback).
        module_path: Python import path to load `name` from when it is
            not a registered policy.

    Returns:
        The cache policy class.

    Raises ValueError if the cache policy is neither registered nor
    resolvable via `module_path`.
    """
    if name in cls._registry:
        return cls._registry[name]()
    if module_path is None:
        raise ValueError(
            f"Unknown cache policy: {name!r}. Supported: {list(cls._registry)}. "
            "For an out-of-tree policy, also set cache_policy_module_path."
        )
    logger.warning(
        "Loading out-of-tree cache policy '%s' from '%s'. This API is "
        "experimental and subject to change in the future as we "
        "iterate the design.",
        name,
```

```
            module_path,
        )
        module = importlib.import_module(module_path)
        policy_cls = getattr(module, name)
        assert issubclass(policy_cls, CachePolicy)
        return policy_cls


# Register built-in policies here.
CachePolicyFactory.register_cache_policy(
    "lru", "vllm.v1.kv_offload.cpu.policies.lru", "LRUCachePolicy"
)
CachePolicyFactory.register_cache_policy(
    "arc", "vllm.v1.kv_offload.cpu.policies.arc", "ARCCachePolicy"
)
```

vllm/v1/kv_offload/cpu/manager.py

修改管理器使用工厂解析策略，是依赖注入的关键变化。

```
# (manager.py 核心变更片段 )
def __init__(
    self,
    num_blocks: int,
    cache_policy: str = "lru", # 之前为 Literal["lru", "arc"]
    cache_policy_module_path: str | None = None, # 新增参数
    enable_events: bool = False,
    store_threshold: int = 1,
    max_tracker_size: int = 64_000,
):
    self.medium: Medium = Medium.CPU
    self._num_blocks: int = num_blocks
    self._num_allocated_blocks: int = 0
    self._free_list: list[int] = []
    self.events: list[OffloadingEvent] | None = [] if enable_events else None
    # 直接使用工厂解析，替代原有私有字典 _CACHE_POLICIES
    policy_cls = CachePolicyFactory.get_cache_policy_cls(
        cache_policy, cache_policy_module_path
    )
    self._policy: CachePolicy = policy_cls(cache_capacity=num_blocks)
    # ... 其余初始化逻辑不变
```

评论区精华

核心讨论围绕三方面：

- out-of-tree 策略支持：orozery 指出初始版本只支持注册不直接支持无注册的 out-of-tree 加载，要求复制 OffloadingSpecFactory 的 module_path 回退模式。最终 get_cache_policy_cls 增加了 module_path 参数，允许直接导入类。

- 测试文件位置：orozery 要求将测试从 test_manager.py 移到独立的 tests/v1/kv_offload/cpu/policies/test_factory.py，遵循其他工厂测试的惯例，作者在第二组 commit 中调整。
- 实验性警告位置：orozery 建议警告不要放在 base.CachePolicy.__init__ 中，而应放在 out-of-tree 加载处，并提议同时对齐 OffloadingSpec。作者将警告移至 get_cache_policy_cls 回退路径。
  - 支持 out-of-tree 策略加载 (design): 作者在 get_cache_policy_cls 中增加了 module_path 回退机制，实现类似 OffloadingSpecFactory 的动态加载。
  - 测试文件位置 (testing): 作者在第二 commit 中移动测试，并调整了测试结构与 fixture。
  - 实验性 API 警告位置 (design): 作者将 logger.warning 移至 get_cache_policy_cls 的回退路径，base.py 中不再包含该警告。

风险与影响

- 风险：
  1. 参数类型放宽：CPUOffloadingManager.cache_policy 从 Literal['lru','arc'] 改为 str，外部代码如果先前依赖类型检查（如传递枚举值）可能产生 mypy 错误，但运行时兼容。
  2. 工厂单例共享：全局 _registry 字典在进程间共享，多线程 / 进程场景下注册竞赛可能导致意外覆盖。当前注册仅在模块导入时进行，风险较低但需注意。
  3. 模块导入开销：get_cache_policy_cls 在第一次调用时执行 importlib.import_module，可能增加延迟。内置策略预注册已导入，out-of-tree 加载的首次热路径有额外开销，但通常可接受。
  4. cache_policy_module_path 路径错误：用户提供错误模块路径时，异常信息需足够清晰，当前 raised ValueError。 - 影响：用户影响：内置策略行为完全一致，零迁移成本。外部开发者现在可以通过注册或者直接指定 module_path 使用自定义 CachePolicy，无需 fork vLLM。系统影响：仅影响 CPU offloading 子模块，不改变核心推理路径。团队影响：解耦策略实现与管理器，降低添加新策略的维护成本，推动 offloading 基础设施的模块化。
- 风险标记：参数类型放宽，全局注册表竞赛，模块导入热路径开销，配置文件路径易错

关联脉络

- 暂无明显关联 PR