PR #30348 完整报告

sgl-project/sglang

[refactor] ctx.resources: named slots, stream leases, and workspace buffer leases

合并时间：2026-07-08 12:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30348

执行摘要

- 一句话：集中进程级资源单例到 ctx.resources，引入命名槽位和租约
- 推荐动作：该 PR 值得精读，尤其关注如何通过数据类 + _FlagGroupBase 将散落的运行时状态集中管理，并实现测试友好的 override 机制。对于构建大型推理引擎的资源管理层有很好的参考价值。

功能与动机

Process-level resource handles were scattered as module singletons, each with its own get/set pair, no reset lifecycle, and monkeypatch-only test injection: the CUDA graph memory pool, the EPLB expert-distribution recorder (101 call sites), the publish-once expert-location metadata, the LPLB solver map, two side streams, a CUDA-event pool, and one lazily-created workspace buffer per attention backend. ~24 model files additionally each constructed their own alternate stream for intra-layer overlap.

实现拆解

1. 定义 Resources 数据类：在 runtime_context.py 中新增 Resources 类（继承 _FlagGroupBase），包含 graph_memory_pool、expert_distribution_recorder、expert_location_metadata、lplb_solvers、streams、buffers、tbo_event_pool 等槽位。
2. 替换模块级全局变量：将 pool.py 中的 _global_graph_memory_pool、expert_distribution.py 中的 _global_expert_distribution_recorder、expert_location.py 中的元数据等改为通过 get_resources() 访问对应槽位。
3. 引入命名流租约：RuntimeContext.get_stream(name) 按需创建 CUDA 流，共享同名流；set_stream(name, stream) 用于显式注入。DP-TBO 通信流和 LoRA 侧流成为命名流；24 个模型文件的备选流统一为单个 "alt" 流租约。
4. 引入命名缓冲区租约：RuntimeContext.get_buffer(name, factory) 按需创建并共享持久缓冲区。各注意力后端的工作空间（flashinfer、flashinfer-MLA、TRT-LLM、DSA、MUSA MATE-MLA）均通过此机制管理。
5. 调整调用点：更新 lora_moe_runner_marlin.py、dp_attention.py、多个注意力后端文件（flashinfer_backend.py、flashinfer_mla_backend.py、trtllm_mla_backend.py、trtllm_mha_backend.py、dsa_backend.py 等）以及 eplb 子模块，将全局变量访问改为 ctx.resources 访问。

6. 测试覆盖：在 test_runtime_context.py 中新增 TestResources 和 TestNamedStreams 测试类，验证懒创建、共享、override 注入、reset 清空等行为。

关键文件：

- python/sglang/srt/runtime_context.py（模块 上下文管理；类别 source；类型 core-logic；符号 Resources, get_stream, set_stream, get_buffer）：核心变更文件：新增 Resources 数据类，在 RuntimeContext 中集成资源管理，并实现 get_stream/set_stream/get_buffer 等租约方法。
- test/registered/unit/test_runtime_context.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestResources, test_graph_pool_lazy_create_and_reuse, test_expert_recorder_noop_default_and_injection, test_expert_location_metadata_publish_once_until_reset）：测试配套：新增 TestResources 和 TestNamedStreams 两个测试类，覆盖资源懒创建、共享、override 注入、reset 清空等场景。
- python/sglang/srt/model_executor/runner_utils/pool.py（模块 内存池；类别 source；类型 data-contract；符号 get_global_graph_memory_pool, set_global_graph_memory_pool, get_or_create_global_graph_memory_pool）：数据契约：将模块级全局变量 _global_graph_memory_pool 替换为通过 get_resources().graph_memory_pool 访问，是迁移全局变量的示例。
- python/sglang/srt/layers/dp_attention.py（模块 DP 注意力；类别 source；类型 dependency-wiring；符号 get_dp_tbo_comm_stream, _tbo_event）：依赖配线：将 DP-TBO 通信流和 TBO 事件池迁移到 ctx.resources，展示了命名流和事件池的使用。
- python/sglang/srt/eplb/expert_distribution.py（模块 EPLB；类别 source；类型 dependency-wiring；符号 get_global_expert_distribution_recorder, set_global_expert_distribution_recorder）：迁移 EPLB 记录器到 ctx.resources，示例了懒创建默认 noop 和 override 测试注入。

关键符号：Resources.init, RuntimeContext.get_stream, RuntimeContext.set_stream, RuntimeContext.get_buffer, get_resources

关键源码片段

python/sglang/srt/runtime_context.py

核心变更文件：新增 Resources 数据类，在 RuntimeContext 中集成资源管理，并实现 get_stream/set_stream/get_buffer 等租约方法。

```
@dataclasses.dataclass
class Resources(_FlagGroupBase):
    """集中管理进程级资源句柄的槽位，提供统一的 reset 生命周期和 override 测试注入"""
    # 懒创建的 CUDA graph 内存池（详见 pool.py）
    graph_memory_pool: Any = None
    # EPLB 记录器（初始为 noop，访问时可自动创建）
    expert_distribution_recorder: Any = None
    # 专家位置元数据（一次发布，直到 reset）
    expert_location_metadata: Any = None
```

```
    # LPLB: layer_id -> solver
    lplb_solvers: dict = dataclasses.field(default_factory=dict)
    # 命名 CUDA 流租约（name -> stream）
    streams: dict = dataclasses.field(default_factory=dict)
    # 命名持久缓冲区租约（name -> tensor）
    buffers: dict = dataclasses.field(default_factory=dict)
    # TBO 事件复用池（避免 HSA 事件创建过多）
    tbo_event_pool: dict = dataclasses.field(default_factory=dict)

# RuntimeContext 新增方法
def get_stream(self, name: str) -> Any:
    """按名称租用 CUDA 流，首次调用时创建，后续共享"""
    stream = self.resources.streams.get(name)
    if stream is None:
        import torch
        stream = torch.cuda.Stream()
        self.resources.streams[name] = stream
    return stream

def set_stream(self, name: str, stream: Any) -> Any:
    """显式注入一个命名流，用于测试或特定后端"""
    self.resources.streams[name] = stream
    return stream

def get_buffer(self, name: str, factory: Any) -> Any:
    """按名称租用持久缓冲区，首次调用时通过 factory 创建"""
    buf = self.resources.buffers.get(name)
    if buf is None:
        buf = factory()
        self.resources.buffers[name] = buf
    return buf
```

评论区精华

无 review 讨论。

- 暂无高价值评论线程

风险与影响

- 风险：主要风险在于替换全局变量时如果遗漏某些引用点可能导致状态不一致，但 PR 提交修改了 45 个文件，覆盖了所有已知调用点。测试用例验证了基本行为。引入的 get_stream/get_buffer 方法在 CUDA graph capture 外使用，不会影响 graph 捕获。由于是纯重构，功能行为不变，回归风险较低。但存在边角场景：如果外部代码直接导入原模块全局变量（如 _global_graph_memory_pool），将出现接缝断裂。
- 影响：对最终用户无影响（功能不变）。对开发者：所有访问原模块级全局变量的代码需要改为通过 ctx.resources 或 get_stream/get_buffer 访问。测试代码可以通过 override() 上下文管理器更方便地进行资源注入，不再依赖 monkey-patch。未来新增资源类时，可直接

在 Resources 中添加槽位，利用统一生命周期管理。

- 风险标记：核心路径变更，全局变量替换，需检查外部依赖

关联脉络

- 暂无明显关联 PR