PR #33012 完整报告

sgl-project/sglang

runtime_context: record the publishing process role

合并时间：2026-08-01 06:06

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33012

执行摘要

- 一句话：runtime_context 记录进程角色，全进程入口迁移至角色化 publish
- 推荐动作：值得精读。核心设计决策：(1) _ConfigBag 的 " 记账映射 + 真实属性 " 双写模式，解决了编译模型前向中配置读取的 dynamo 可追踪性问题，是一种可复制的模式；(2) _ServerArgsOverride 从 " 恢复发布对象 " 升级为 " 快照整个生命周期 "，避免嵌套作用域重投影导致的语义丢失；(3) legacy shim 刻意去别名，承认 tokenizer 与 scheduler 进程语义不同；(4) 角色先作 provenance、后做投影与 fail-closed 的分阶段落地思路务实。另外 TYPE_CHECKING 导入陷阱的修复过程（isort 挪导入 → 运行时 NameError → 无条件导入 + AST 扫描）也是对 Python 代码库的通用警示。

功能与动机

PR body 明确说明：role is provenance for now; per-role namespace projection and fail-closed enforcement come later。运行时存在多种角色进程（tokenizer / scheduler / encoder / expert_backup / weight_cache_daemon / launcher / test）共享同一套 runtime context 配置管道，但此前 set_global_server_args_for_scheduler 与 set_global_server_args_for_tokenizer 是同一函数别名，进程语义被抹平；同时 _ConfigBag 依赖 __getattr__ 动态查找，在 torch.compile 编译的模型 forward 中读取配置（如 embedding 层的 get_exec().comm.enable_symm_mem）会造成 graph-break。本 PR 要为后续 per-role 配置投影和 fail-closed 强制校验建立角色记录基础（RFC #30696）。

实现拆解

1. 角色发布 API 与进程入口迁移：在 python/sglang/srt/runtime_context.py 新增模块级 publish(server_args, *, role=...) 与 publish_role()，RuntimeContext 增加 _publish_role 槽位；re-publish 语义为 last-publish-wins（重新投影配置袋、重置 provenance、覆盖角色）。随后逐入口迁移：ray/scheduler_actor.py 的 SchedulerActor.__init__、managers/scheduler.py 的 run_scheduler_process 和 managers/data_parallel_controller.py 的 run_data_parallel_controller_process 以 role="scheduler" / role="dp_controller" 发布；disaggregation/encode_server.py 的 MMEncoder.__init__ 以 role="encoder" 发布（此前误用 scheduler setter）；elastic_ep/expert_backup_manager.py 与 weight_cache/daemon.py 分别以 expert_backup、weight_cache_daemon 发布。每个入口的注释都点明动机：Scheduler 构造会先于模型 worker 的 publish 读取配置命名空间。

2. _ConfigBag 可追踪化改造：去掉 __slots__，让 _set / _set_sub 同时写入记账映射（_fields / _subs）与真实实例属性（__dict__），override 作用域管理器统一改走 _set，_build_config_bags 改走 _set_sub；__getattr__ 退化为兜底。效果是 bag.leaf / bag.sub 变成 dynamo 可追踪的普通属性加载，同时记账映射仍承担 override 路由、成员判断与作用域恢复的权威职责。
3. 生命周期快照修复：_ServerArgsOverride.install 从只快照 server_args 与 capture 开关，扩展为快照整个生命周期（_server_args、_config_bags、_overrides_log、_publish_role、parallel._config、capture 开关），restore 原样复原而非重投影，保证嵌套作用域逐字恢复；preserve_config 同步快照 / 恢复 _publish_role；build_draft_tp_worker 在 preserve_config 作用域内发布 draft 配置副本，使 draft 层解析自己的配置且 target 的 post-publish override 可在恢复后存活；overrides_log() 改为返回逐条拷贝，防止外部调用方就地污染 provenance 记录。
4. legacy shim 去别名与 ratchet：python/sglang/srt/server_args.py 中 set_global_server_args_for_tokenizer 不再别名到 scheduler shim，而是独立函数以 role="tokenizer" 发布；scheduler shim 显式 role="scheduler"。配套 test/registered/unit/test_legacy_global_ratchet.py 把 legacy setter 调用计数 ratchet 从 5 下调到 4，体现进程入口迁移减少调用点。
5. 测试配套：test/registered/unit/test_runtime_context_override.py 新增 test_publish_records_role、test_legacy_shims_record_roles、test_reset_clears_role、test_direct_install_clears_role；test/registered/unit/test_runtime_context.py 把 test_tokenizer_alias_is_same_function（断言 alias）反转为 test_tokenizer_alias_is_distinct_role_shim（断言 assertIsNot）。

关键文件：

- python/sglang/srt/runtime_context.py（模块 配置上下文；类别 source；类型 core-logic；符号 publish, publish_role, _ConfigBag._set, _ConfigBag._set_sub）：核心变更文件：新增 publish/publish_role 角色发布与查询；_ConfigBag 从 __slots__迁移到真实实例属性以支持 torch.compile 追踪；_ServerArgsOverride 快照整个生命周期；overrides_log 返回副本。
- python/sglang/srt/server_args.py（模块 启动参数；类别 source；类型 core-logic；符号 set_global_server_args_for_scheduler, set_global_server_args_for_tokenizer）：legacy setter 从别名关系拆分为两个带角色的独立 shim，是本 PR 角色语义的关键载体，同时受 ratchet 测试约束。
- python/sglang/srt/ray/scheduler_actor.py（模块 调度器；类别 source；类型 dependency-wiring）：P1 讨论点所在文件：publish 导入曾被 isort 移入 TYPE_CHECKING 块导致运行时 NameError，修复为无条件模块级导入并在 actor 构造前以 scheduler 角色发布。
- test/registered/unit/test_runtime_context_override.py（模块 配置测试；类别 test；类型 test-coverage；符号 test_publish_records_role, test_legacy_shims_record_roles, test_reset_clears_role, test_direct_install_clears_role）：新增 4 个角色相关回归测试（publish 记录角色、legacy shim 记录角色、reset 清除角色、直接 install 清除角色），直接验证本 PR 的核心语义。

- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 dependency-wiring；符号 run_scheduler_process）：run_scheduler_process 是新调度器进程入口，在 configure_scheduler_process 之后、Scheduler 构造之前以 scheduler 角色发布配置，确保 Scheduler.__init__读取命名空间时已就绪。
- python/sglang/srt/weight_cache/daemon.py（模块 权重缓存；类别 source；类型 dependency-wiring）：weight cache daemon 首次获得显式角色 weight_cache_daemon，是角色模型覆盖新进程形态的关键样例。
- python/sglang/srt/disaggregation/encode_server.py（模块 编码器；类别 source；类型 dependency-wiring；符号 MMEncoder.init）：MMEncoder 此前误用 scheduler 的 legacy setter，现改为角色 encoder，修复角色语义错配。
- python/sglang/srt/elastic_ep/expert_backup_manager.py（模块 弹性 EP；类别 source；类型 dependency-wiring；符号 run_expert_backup_manager_process）：expert backup 进程入口迁移到角色化 publish（role=expert_backup），是弹性 EP 路径的配套更改。
- python/sglang/srt/managers/data_parallel_controller.py（模块 DP 控制器；类别 source；类型 entrypoint；符号 run_data_parallel_controller_process）：DP controller 进程入口新增 publish（role=dp_controller），注释说明其在派生 scheduler 前读取配置命名空间。
- test/registered/unit/test_runtime_context.py（模块 配置测试；类别 test；类型 test-coverage；符号 test_tokenizer_alias_is_distinct_role_shim）：把 tokenizer alias 测试反转为 distinct role shim 测试，验证去别名的语义变更。
- python/sglang/srt/speculative/draft_worker_common.py（模块 投机解码；类别 source；类型 core-logic）：draft worker 构建路径配套：在 preserve_config 作用域内发布 draft 配置副本，避免 draft 层读取到 target 的 post-publish override。
- test/registered/unit/test_legacy_global_ratchet.py（模块 配置测试；类别 test；类型 test-coverage）：legacy setter 计数 ratchet 从 5 下调至 4，对应进程入口迁移带来的调用点减少，是防回退的机制化约束。

关键符号：publish, publish_role, _ConfigBag._set, _ConfigBag._set_sub, _ServerArgsOverride.install, _ServerArgsOverride.restore, RuntimeContext.preserve_config, RuntimeContext.overrides_log, set_global_server_args_for_scheduler, set_global_server_args_for_tokenizer, run_scheduler_process, SchedulerActor.init, MMEncoder.init, run_expert_backup_manager_process, run_data_parallel_controller_process

关键源码片段

python/sglang/srt/runtime_context.py

核心变更文件：新增 publish/publish_role 角色发布与查询；_ConfigBag 从 __slots__迁移到真实实例属性以支持 torch.compile 追踪；_ServerArgsOverride 快照整个生命周期；overrides_log 返回副本。

```
class _ConfigBag:
    """已解析配置的命名空间袋。
```

数值在 ``publish`` 时从 ``server_args`` 快照而来，之后本袋是这些字段的
唯一事实来源。读取是普通属性访问；裸赋值只读。

Leaf 与 sub-bag 存放为 **真实实例属性**（在 ``__dict__`` 中），因此
``bag.leaf`` / ``bag.sub`` 是普通属性加载，``torch.compile`` / dynamo
可以追踪——编译后的模型 forward 中的配置读取（例如 embedding 层的
``get_exec().comm.enable_symm_mem``）不能 graph-break。
``_fields`` / ``_subs`` 仍是权威的 name→value 映射，负责 override 路由、
成员判断与作用域恢复；``__getattr__`` 只兜底真正缺失的名字。
刻意不用 ``__slots__``：leaf 是动态的，``__dict__`` 正是可追踪读取的来源。
"""

```
def __init__(self, path: str):
    object.__setattr__(self, "_path", path)
    object.__setattr__(self, "_fields", {}) # {leaf: value} 记账映射
    object.__setattr__(self, "_subs", {}) # {subname: _ConfigBag} 记账映射

def __getattr__(self, name: str) -> Any:
    # 仅兜底：真实 leaf / sub-bag 会先经 __dict__ 命中。这里用
    # object.__getattribute__（而非 self._fields）保证 __init__ 之前
    # 被调用也安全。
    fields = object.__getattribute__(self, "_fields")
    if name in fields:
        return fields[name]
    subs = object.__getattribute__(self, "_subs")
    if name in subs:
        return subs[name]
    path = object.__getattribute__(self, "_path")
    raise AttributeError(f"config namespace {path!r} has no leaf/subgroup {name!r}")

def __setattr__(self, name: str, value: Any) -> None:
    # 对外只读：所有写入必须走 publish / override 通道。
    raise AttributeError(
        f"config namespace {self._path!r} is read-only; write via "
        "get_context().override(source, ...) or the scoped .override(**kw)"
    )

def _set(self, name: str, value: Any) -> None:
    """内部写入（publish + override），绕过只读守卫。

    同时更新记账映射和真实属性（可追踪读取），保证两处永远一致。"""
    object.__getattribute__(self, "_fields")[name] = value
    object.__setattr__(self, name, value)

def _set_sub(self, name: str, sub: _ConfigBag) -> None:
    """注册嵌套 bag：同时进入记账映射与真实属性。

    这样 ``bag.sub`` 是普通、可追踪的属性加载。"""
    object.__getattribute__(self, "_subs")[name] = sub
```

```
        object.__setattr__(self, name, sub)

    def __contains__(self, name: str) -> bool:
        return name in object.__getattribute__(self, "_fields")

    @contextmanager
    def override(self, **kwargs):
        """作用域化、事务性的测试专用 override（写前校验 key，退出时恢复）。"""
        fields = object.__getattribute__(self, "_fields")
        unknown = set(kwargs) - set(fields)
        if unknown:
            path = object.__getattribute__(self, "_path")
            raise ValueError(f"unknown config leaf for {path!r}: {sorted(unknown)}")
        saved = {name: fields[name] for name in kwargs}
        for name, value in kwargs.items():
            self._set(name, value)
        try:
            yield self
        finally:
            for name, value in saved.items():
                self._set(name, value)
```

python/sglang/srt/server_args.py

legacy setter 从别名关系拆分为两个带角色的独立 shim，是本 PR 角色语义的关键载体，同时受 ratchet 测试约束。

```
# NOTE: 进程级 ServerArgs 由 runtime context（sglang.srt.runtime_context）持有。
# 下面两个函数是 LEGACY shim，保留给既有调用方；它们按引用发布 / 读取同一个
# 活对象。不要新增调用方——调用次数由
# test/registered/unit/test_legacy_global_ratchet.py 以 ratchet（只减不增）约束。
# 函数内 import 是为了让两个模块在 import 期保持无环。
def set_global_server_args_for_scheduler(server_args: ServerArgs):
    """Legacy publish shim（role=scheduler）——新代码请改用
    ``runtime_context.publish(server_args, role=...)``。"""
    from sglang.srt.runtime_context import publish

    publish(server_args, role="scheduler")


def set_global_server_args_for_tokenizer(server_args: ServerArgs):
    """Legacy publish shim（role=tokenizer）。刻意不与 scheduler shim 做别名：
    两个进程的角色不同，去别名后角色记录才能反映真实进程语义。"""
    from sglang.srt.runtime_context import publish

    publish(server_args, role="tokenizer")


def get_global_server_args() -> ServerArgs:
    """Legacy accessor shim——新代码请改用
```

```
    ``get_server_args()`` (来自 ``sglang.srt.runtime_context``) 。"""
    from sglang.srt.runtime_context import get_context

    return get_context().server_args
```

python/sglang/srt/ray/scheduler_actor.py

P1 讨论点所在文件：publish 导入曾被 isort 移入 TYPE_CHECKING 块导致运行时 NameError，修复为无条件模块级导入并在 actor 构造前以 scheduler 角色发布。

```
# 顶部无条件导入（不能放在 TYPE_CHECKING 下）：Ray actor 运行时需要调用
# publish，若被 import 期跳过，会在 __init__ 中触发 NameError，且 py_compile
# 无法捕获这类运行时名字错误（isort 曾把该导入挪进 TYPE_CHECKING 块）。
from sglang.srt.runtime_context import publish
from sglang.srt.server_args import PortArgs, ServerArgs


@ray.remote
class SchedulerActor:
    """Ray actor wrapper for SGLang Scheduler.

    Each actor manages one GPU and runs the Scheduler + TpModelWorker stack.
    Ray is used for process lifecycle; ZMQ handles request/response communication.
    """

    def __init__(
        self,
        server_args: ServerArgs,
        port_args: PortArgs,
        gpu_id: int,
        tp_rank: int,
        attn_cp_rank: int,
        moe_dp_rank: int,
        moe_ep_rank: int,
        pp_rank: int,
        dp_rank: Optional[int],
        dist_init_addr: Optional[str] = None,
    ):
        ...
        # 该 actor 直接构造 Scheduler（不走 run_scheduler_process），而
        # Scheduler.__init__ 会在 model worker 自己 publish 之前读取配置
        # 命名空间——因此必须先以 scheduler 角色发布解析后的配置。
        publish(server_args, role="scheduler")

        self.scheduler = Scheduler(
            server_args=server_args,
            port_args=port_args,
            gpu_id=actual_gpu_id,
            tp_rank=tp_rank,
            moe_ep_rank=moe_ep_rank,
```

```
        pp_rank=pp_rank,
        attn_cp_rank=attn_cp_rank,
        moe_dp_rank=moe_dp_rank,
        dp_rank=dp_rank,
    )
    ...
```

评论区精华

Codex 机器人提出两条有效问题，作者均确认为真实缺陷并修复：

1. P1（correctness）：ray/scheduler_actor.py 中 publish 的导入被 isort 挪进 TYPE_CHECKING 块，Ray actor 运行时直接 NameError: name 'publish' is not defined，所有 Ray scheduler actor 初始化必挂；py_compile 无法捕获这种运行时名字错误。作者回应：修复为无条件模块级导入（用 stub ray 实际导入模块验证 publish 已绑定），并做全树 AST 扫描确认没有其他运行时使用的名字被 TYPE_CHECKING 守卫。
2. P2（correctness）：直接调用 get_context().set_server_args(...)（测试 override、draft worker 构建等合法路径）后，publish_role() 仍返回旧角色，provenance 与实际安装的生命周期不一致。作者回应：set_server_args 现在清除角色（直接安装即无角色），publish 仅在安装成功返回后赋值角色，并补回归测试 test_direct_install_clears_role。

- publish 导入被 isort 移入 TYPE_CHECKING 导致 Ray actor 运行时 NameError (correctness): publish 改为无条件模块级导入；补充 AST 全树扫描确认同类问题不存在。
- 直接 set_server_args 后 publish_role 返回陈旧角色（P2） (correctness): set_server_args 清除角色，publish 在安装成功后赋值；补回归测试。

风险与影响

- 风险：
  1. 配置容器内部表示变更：runtime_context.py 的 _ConfigBag 从 __slots__ 改为 __dict__ 动态属性，_set / _set_sub 的双写路径是把双刃剑——一旦记账映射与真实属性失同步，__getattr__ 兜底与 __dict__ 读取结果会分歧；当前 override/restore 全部收敛到 _set 单点写入，一致性依赖这条路径不被后续改动绕过。
  2. 进程入口大范围迁移：scheduler、DP controller、encoder、expert backup、weight cache daemon 六个进程入口同时改发布方式，任何一个入口漏发布或角色填错，在后续 fail-closed 阶段会被放大为启动失败；weight_cache/daemon.py 是首次引入角色（此前无 setter 调用语义），需要额外验证。
  3. 导入路径陷阱：TYPE_CHECKING 守卫导入的教训说明静态检查无法拦截运行时 NameError，未来新增 runtime 使用的名字仍有被 isort 等工具挪进守卫块的风险；本次只做了单次全树 AST 扫描，没有形成持续校验机制。
  4. draft worker 构建路径：build_draft_tp_worker 在 preserve_config 作用域内发布 draft 副本，若恢复逻辑遗漏某个生命周期字段（如 parallel._config），target 的 post-publish override 会丢失，影响投机解码配置的正确性。
  5. ratchet 行为约束：server_args.py 的 legacy setter 计数 ratchet 从 5 降到 4，新代码若再调用 legacy setter 会 CI 失败——这是设计意图，但对不熟悉该约束的贡献者构

成隐性行为限制。 - 影响：对用户与外部 API 无可见变化，publish 角色是内部 provenance，不影响请求 / 响应语义。对系统而言，运行时配置从 " 无角色全局对象 "走向 " 带角色语义的过程级配置 "，为 RFC #30696 的 per-role 命名空间投影与 fail-closed 校验奠定数据基础。对性能有正面影响：_ConfigBag 读取从 __getattr__ 动态查找变为 __dict__ 普通属性加载，配合 torch.compile 可减少编译前向中的 graph-break。对团队开发流程：新代码应优先使用 publish(server_args, role=...)，legacy setter 受 ratchet 约束只能减少不能新增，overrides_log() 返回副本也杜绝了外部篡改 provenance 的隐患。 - 风险标记：进程入口大范围迁移，配置容器内部表示变更，导入路径陷阱，role 语义约束

关联脉络

- PR #33011 base of 3-PR stack (runtime context config): 本 PR body 明确说明 base: #33011，是 3-PR stack 的第一部分，本 PR 在其基础上引入角色记录。
- PR #33013 config: read resolved config via namespace accessors: 同属 RFC #30696 配置体系重构线，覆盖 scheduler.py 与 test_runtime_context.py，与本 PR 的 _ConfigBag 与 runtime_context 改造直接衔接。
- PR #30696 RFC: runtime context config: PR body 引用的 RFC，定义了 per-role namespace projection 与 fail-closed enforcement 的后续方向，本 PR 是其中间落地步骤。