PR #35910 完整报告

sgl-project/sglang

config: publish before the launcher reads effective configuration

合并时间：2026-08-23 16:20

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35910

执行摘要

- 一句话：将 publish 提前至校验后，launcher 从 config bag 读生效配置
- 推荐动作：值得精读，尤其是 test_resolution_reads_no_bag.py 的静态可达性设计与 snapshot_context 的按叶子快照方案。建议结合系列栈自底向上阅读（#35904 → #35910），并重点关注两点设计决策： 1) 用 inspect 从 live registries 推导注册条目而非扫描装饰器（后者只找到 39/65）； 2) 回滚采用"恢复而非清空"的语义以保留调用方已有上下文。对配置架构演进感兴趣的读者可以把它作为 ServerArgs 只读化的范例。

功能与动机

PR body 明确说明：The launcher published last — after validating, allocating ports, and deciding whether to start the engine-info bootstrap server — so all of those read effective configuration off the record, which is exactly what the record should stop carrying。系列目标是让 ServerArgs 从"构造时原地修改"演进为"resolve once, publish into config bags"，本 PR 消除 launcher 阶段对记录的读取，使记录在被发布前真正只读。

实现拆解

1. 变更入口 python/sglang/srt/entrypoints/engine.py::_launch_subprocesses：启动顺序由原来的 resolve → env → plugins → check_server_args → 分配端口 → 起 bootstrap server → resolve_auto_parsers → publish → spawn，调整为 resolve → env → plugins → check_server_args → resolve_auto_parsers → snapshot_context() → publish(server_args, role="tokenizer") → try 块内做端口分配、bootstrap server、daemon 启动 → except 时 restore_context()。原因：校验中的 LoRA 路径规范化和 auto-parser 检测属于晚解析，必须写记录且已发布记录拒绝写入，因此只能放在 publish 前。
2. 快照 / 回滚机制 python/sglang/srt/runtime_context.py：新增 snapshot_context() / restore_context() 及辅助函数 _group_leaves() / _restore_leaves()。状态从 RuntimeContext.__slots__ 枚举而非手写字段列表，避免新增 slot 时静默漏掉；flag 组按叶子快照，规避 publish 原位写入同一 Flags 对象导致的快照污染。同时 ROLE_NAMESPACE_SETS["dp_controller"] 从 {"exec"} 扩展为 {"exec", "parallel", "device", "disagg"}，匹配 DP 控制器新增的 bag 读取面。
3. DP 控制器与 rank 布局改用 bag python/sglang/srt/managers/data_parallel_controller.py：load_balance_method、soft_watchdog_timeout、线程标签里的

disaggregation_mode、rank 布局中的 pp_size / attn_cp_size / moe_dp_size 一律改从 get_parallel() / get_device() / get_disagg() / configured_*() 读取；engine.py::_compute_parallelism_ranks 同样用 configured_attn_cp_size() / configured_moe_dp_size()，并把反复读取的尺寸提升为局部变量。

4. 静态守卫测试（新增 test/registered/unit/server_args/test_resolution_reads_no_bag.py）：用 ast + inspect 从 runtime_context.py 推导完整 bag 访问器集合（避免手写列表与模块导出不同步）；入口集合来自 resolution pipeline 的 import 符号、live registries（POST_PROCESS_PASSES、_MODEL_OVERRIDE_FNS、_PREDICATE_OVERRIDE_FNS）以及 run_post_process_pass 按值传参的调用点，再沿模块内函数调用做可达性分析，断言解析期可达代码不读 bag。test_resolution_declarations.py 新增 _late_resolvers() 推导和两个 source-order pin（校验在 publish 前、launcher 完成全部晚解析后才 publish）。
5. 配套测试与 ratchet 清单更新：test_gpu_feature_transport.py 的启动失败用例从 SimpleNamespace 改为真实 ServerArgs（launcher 中途 publish 后读取的是 bag，只有 dataclass 能投影）；test_global_config_read_ratchet.py 为 launcher 与 DP 控制器的 configured_* 读取补充白名单理由；test_supplied_instance_exposure_ratchet.py 删除已不再从记录读取的字段条目（如 engine 的 attn_cp_size、moe_dp_size，DP controller 的 disaggregation_mode、load_balance_method 等）。

关键文件：

- python/sglang/srt/runtime_context.py（模块 运行时上下文；类别 source；类型 core-logic；符号 _group_leaves, _restore_leaves, snapshot_context, restore_context）：新增 snapshot_context / restore_context 及叶子级快照辅助函数，是启动失败回滚的核心；同时把 dp_controller 角色命名空间从 exec 扩展为 exec / parallel / device / disagg。
- python/sglang/srt/entrypoints/engine.py（模块 启动入口；类别 source；类型 core-logic；符号 _launch_subprocesses, _compute_parallelism_ranks）：启动入口的时序重组：publish 提前到校验与 auto-parser 检测之后、端口分配之前，并用 snapshot / restore 包住后续可能失败的 pre-spawn 步骤；_compute_parallelism_ranks 改读 configured 尺寸。
- python/sglang/srt/managers/data_parallel_controller.py（模块 DP 控制器；类别 source；类型 entrypoint；符号 DataParallelController.init, launch_dp_schedulers, launch_tensor_parallel_group, run_data_parallel_controller_process）：DP 控制器进程发布后即刻布局 scheduler，因此其配置读取全部从记录切到 config bag：load_balance_method、watchdog 超时、disagg 线程标签、pp / attn_cp / moe_dp 尺寸。
- test/registered/unit/server_args/test_resolution_reads_no_bag.py（模块 静态守卫；类别 test；类型 test-coverage；符号 _accessor_names, _module_of, _imported_symbols, _registry_functions）：新增的静态守卫测试，是整个“解析期不读 bag”不变量最关键的保障：用 AST 可达性分析覆盖 import 符号、注册表条目和按值传参 pass，防止此类错误悄悄合入。
- test/registered/unit/server_args/test_resolution_declarations.py（模块 解析声明；类别 test；类型 test-coverage；符号 _late_resolvers, reaches, test_validation_can_still_resolve_before_the_record_is_published, test_the_launcher_finishes_resolving_before_it_publishes）：补充晚解析推导与两个 source-order pin：校验先于 publish、launcher 完成全部晚解析后才 publish，防止启动

时序回退。

- test/registered/unit/multimodal/test_gpu_feature_transport.py（模块 特性传输测试；类别 test；类型 test-coverage；符号 test_engine_startup_failure_releases_parent_pool）：启动失败用例从 SimpleNamespace 假记录改为真实 ServerArgs，因为 launcher 中途 publish 后读取的是 bag，只有 dataclass 能正确投影；同时补充 reset_context cleanup。
- test/registered/unit/test_global_config_read_ratchet.py（模块 全局读包白名单；类别 test；类型 test-coverage）：为 launcher 与 DP 控制器的 configured_* 读取补充白名单及理由，保证 pre-spawn 布局读取配置尺寸的合法性被审计。
- test/registered/unit/test_supplied_instance_exposure_ratchet.py（模块 实例暴露守卫；类别 test；类型 test-coverage）：删除已不再从 ServerArgs 记录读取的字段条目：engine 的 attn_cp_size / moe_dp_size / remote_instance_weight_loader_start_seed_via_transfer_engine，以及 DP controller 的若干字段，反映读取面已迁移到 bag。

关键符号：snapshot_context, restore_context, _group_leaves, _restore_leaves, _launch_subprocesses, _compute_parallelism_ranks, DataParallelController.init, run_data_parallel_controller_process

关键源码片段

python/sglang/srt/runtime_context.py

新增 snapshot_context / restore_context 及叶子级快照辅助函数，是启动失败回滚的核心；同时把 dp_controller 角色命名空间从 exec 扩展为 exec / parallel / device / disagg。

```
def _group_leaves(group: _FlagGroupBase) -> dict[str, Any]:
    """递归收集一个 flag 组的叶子值：嵌套组递归，dict / list 只拷贝容器。"""
    leaves: dict[str, Any] = {}
    for name in type(group).__dataclass_fields__:
        value = getattr(group, name)
        if isinstance(value, _FlagGroupBase):
            leaves[name] = _group_leaves(value)
        elif isinstance(value, (dict, list)):
            leaves[name] = type(value)(value) # 拷贝容器；元素共享，回滚时仍是原对象
        else:
            leaves[name] = value
    return leaves


def _restore_leaves(group: _FlagGroupBase, leaves: dict[str, Any]) -> None:
    """把叶子值逐层写回 flag 组；dict / list 原地更新，避免替换对象。"""
    for name, value in leaves.items():
        current = getattr(group, name)
        if isinstance(current, _FlagGroupBase):
            _restore_leaves(current, value)
        elif isinstance(current, dict):
            current.clear()
            current.update(value)
        elif isinstance(current, list):
```

```
            current[:] = value
        else:
            setattr(group, name, value)


def snapshot_context() -> dict[str, Any]:
    """捕获 publish 会替换的全部上下文状态，供启动失败时回滚。

    从 ``__slots__`` 枚举而不是手工列字段：手写列表总会落后于新增 slot，
    而静默漏掉一个字段的回滚比没有更糟。flag 组按叶子快照而非按引用保存，
    因为 publish 会*原位写入*同一个 ``Flags`` 对象（比如
    ``capture.enable_torch_compile``），按引用保存会在回滚时读到失败启动
    遗留的值。
    """
    state: dict[str, Any] = {}
    for name in RuntimeContext.__slots__:
        if name == "parallel":
            continue
        value = getattr(_CONTEXT, name)
        if isinstance(value, _FlagGroupBase):
            state[name] = (value, _group_leaves(value))
        elif isinstance(value, list):
            state[name] = list(value)
        else:
            state[name] = value
    state["__parallel__"] = {
        name: getattr(_CONTEXT.parallel, name)
        for name in type(_CONTEXT.parallel).__slots__
    }
    state["__dwdp__"] = get_global_dwdp_manager()
    return state


def restore_context(state: dict[str, Any]) -> None:
    """把 ``snapshot_context`` 捕获的状态原样放回。"""
    for name in RuntimeContext.__slots__:
        if name == "parallel":
            continue
        value = state[name]
        if isinstance(value, tuple) and isinstance(value[0], _FlagGroupBase):
            # 先恢复对象本身，再按叶子恢复：publish 复用了对象，两者缺一不可
            group, leaves = value
            setattr(_CONTEXT, name, group)
            _restore_leaves(group, leaves)
        else:
            setattr(_CONTEXT, name, value)
    for name, value in state["__parallel__"].items():
        setattr(_CONTEXT.parallel, name, value)
    _adaptive_draft_token_bound.cache_clear() # 自适应草稿 token 阈值缓存依赖配置
```

```
    set_global_dwdp_manager(state["__dwdp__"])
```

python/sglang/srt/entrypoints/engine.py

启动入口的时序重组：publish 提前到校验与 auto-parser 检测之后、端口分配之前，并用 snapshot / restore 包住后续可能失败的 pre-spawn 步骤；_compute_parallelism_ranks 改读 configured 尺寸。

```
# 校验看似只读，实则会在 late resolution 中规范化 LoRA adapter 路径，
# 而已发布的 config 拒绝写入，因此它必须位于 publish 之前；同时它也要
# 在 auto-parser 检测之前，保证被拒绝的记录仍可重试。
server_args.check_server_args()

# auto-parser 检测需要 tokenizer 和 chat template，无法放进 resolution
# pipeline；放在插件加载之后，因为插件可能注册检测到的 parser。
if (
    server_args.reasoning_parser == "auto"
    or server_args.tool_call_parser == "auto"
):
    resolve_auto_parsers(server_args)

# publish 会替换此前发布的内容，回滚要恢复“发布前”的上下文而不是清空
# 进程：调用方捕获启动错误后仍保有它原来的上下文。
context_before_publish = snapshot_context()
publish(server_args, role="tokenizer")

try:
    # 从这里开始的所有读取都来自 config bag；端口分配、bootstrap server、
    # daemon 启动都还没有 spawn 任何子进程。
    if port_args is None:
        port_args = PortArgs.init_new(server_args)
    logger.info(f"{server_args=}")
    # ... engine-info bootstrap server 与 weight_cache daemon 启动 ...
except BaseException:
    # 回滚发布前的上下文，使同一个 ServerArgs 实例仍可重试
    restore_context(context_before_publish)
    raise
```

test/registered/unit/server_args/test_resolution_reads_no_bag.py

新增的静态守卫测试，是整个“解析期不读 bag”不变量最关键的保障：用 AST 可达性分析覆盖 import 符号、注册表条目和按值传参 pass，防止此类错误悄悄合入。

```
def _reaches_a_bag(path, entry):
    """entry 在 path 中是否可达某个 config bag 访问器，只沿模块内调用展开。"""
    tree = ast.parse(path.read_text(encoding="utf-8-sig"))
    functions = {
        node.name: node
        for node in ast.walk(tree)
        if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef))
    }
```

```
    seen = set()

    def walk(name):
        if name in seen or name not in functions:
            return None
        seen.add(name)
        for node in ast.walk(functions[name]):
            if not isinstance(node, ast.Call):
                continue
            if isinstance(node.func, ast.Attribute):
                # rc.get_exec()、self.get_schedule()：经模块别名或对象到达的访问器
                if node.func.attr in _ATTRIBUTE_SPELLED:
                    return node.lineno
                continue
            if not isinstance(node.func, ast.Name):
                continue
            if node.func.id in _BAG_ACCESSORS:
                return node.lineno
            found = walk(node.func.id) # 沿模块内函数调用继续追踪
            if found is not None:
                return found
        return None

    return walk(entry)


class TestResolutionReadsNoBag(CustomTestCase):
    def test_the_accessor_set_is_derived_and_whole(self):
        """访问器集合必须来自推导而非手写，否则集合缩水会让其它检查静默通过。"""
        self.assertGreaterEqual(
            len(_BAG_ACCESSORS),
            20,
            f"only {len(_BAG_ACCESSORS)} accessors were derived from "
            "runtime_context; the derivation broke",
        )
        # 显式列出，重命名删掉一个时这里会失败
        for name in ("get_exec", "get_flags", "get_parallel", "get_resources"):
            self.assertIn(name, _BAG_ACCESSORS)

    def test_nothing_the_pipeline_calls_reads_a_bag(self):
        # resolution 可达的入口除了 import 符号，还有注册表条目与按值传参的 pass
        imported = _imported_symbols(
            [_SRT / "server_args.py", _SRT / "arg_groups" / "overrides.py"]
        )
        reachable = {
            (path, symbol) for path, symbols in imported.items() for symbol in symbols
        } | _registered_entries()
        found = []
        for path, symbol in sorted(reachable):
```

```
            line = _reaches_a_bag(path, symbol)
            if line is not None:
                found.append(
                    f"{path.relative_to(_SRT)}:{line} reached from "
                    f"{symbol}(), which resolution calls"
                )
        # bag 读取只会在到达它的那条分支上抛错，测试无法覆盖所有配置，
        # 因此用静态可达性守住“解析期不读包”的约束。
        self.assertEqual(
            found,
            [],
            "resolution reaches a config-bag read, which raises on whichever "
            "branch gets there first; read the resolving state instead:\n  "
            + "\n  ".join(found),
        )
```

评论区精华

本 PR 无人工 review 评论，6 条评论均来自 chatgpt-codex-connector 自动化审查：

- P1（data_parallel_controller.py）：SGLANG_ROLE_NAMESPACES=enforce 下，publish(role="dp_controller") 只允许 *exec* 命名空间，而新增的 get_device() / get_disagg() 会直接抛错。结论：head 已将 ROLE_NAMESPACE_SETS["dp_controller"] 扩展为 {"exec", "parallel", "device", "disagg"}，并同步更新了审计注释，已解决。
- P2（engine.py）：publish 提前后，若端口分配等后续步骤失败，发布会残留，导致同一 ServerArgs 实例重试时 check_server_args 内的 late resolution 失败。结论：head 引入 snapshot_context / restore_context，启动失败时回滚发布前的上下文，已解决。
- P2（engine.py）：回滚不应清空进程已有上下文（reset_context 会丢掉此前配置），head 改为恢复快照而不是清空，已解决。
- P2（runtime_context.py）：snapshot_context 若按引用保存 Flags 对象，publish 原位写入后快照即被污染；head 采用 _group_leaves / _restore_leaves 按叶子保存与恢复，已解决。
- P2（engine.py 顺序）：auto-parser 检测在端口分配前替换 "auto" 哨兵，若分配端口失败，重试会跳过检测且可能静默缺少 parser。结论：head 保留 check_server_args 先于 resolve_auto_parsers 的顺序，解决“校验前消费 auto 哨兵”的问题；但检测仍在端口分配之前，snapshot_context 不覆盖外部 ServerArgs 实例字段，此点仅部分缓解，属于未完全关闭的疑虑。
  - DP 控制器角色命名空间需扩展以允许 device / disagg 读取 (correctness): head 将 ROLE_NAMESPACE_SETS["dp_controller"] 扩为 {"exec", "parallel", "device", "disagg"}，并同步更新审计注释，问题已解决。
  - 启动失败后 publish 残留导致同一实例不可重试 (correctness): head 引入 snapshot_context() / restore_context()，在失败时回滚发布前的上下文，保持实例可重试。
  - 回滚应恢复先前上下文而非清空进程 (correctness): head 用 restore_context 恢复快照而不是清空，保留调用方原有上下文。

- publish 原位写入 Flags 对象，按引用快照会被污染 (correctness): head 以 (value, _group_leaves(value)) 按叶子快照并在 restore_context 中逐层恢复叶子，已解决。
- 校验顺序与 auto-parser 哨兵的消费时机 (design): 校验先于 auto-parser 检测已通过 source-order pin 固定；端口分配失败时检测结果已写入外部 ServerArgs 且不被 snapshot 回滚，属于未完全关闭的已知限制。

风险与影响

- 风险：
  1. 启动时序敏感（核心路径变更）：_launch_subprocesses 的每一行都处于“发布前可写、发布后只读”的边界上；任何新代码如果在 publish 之后调用晚解析、或 publish 之前读取 bag（bag 未发布会抛 config namespace ... not published），都会在特定配置下启动失败。AST 守卫能覆盖模块级函数调用和注册表条目，但正如测试 docstring 自述“It is a ratchet, not a proof”，方法调用、动态分派或未跟踪的 import 仍是盲区。
  2. 快照范围有限：snapshot_context 只覆盖 RuntimeContext 状态，不覆盖外部持有的 ServerArgs 实例；resolve_auto_parsers 对记录字段的写入（reasoning_parser 从 "auto" 变为具体值）无法回滚，端口分配失败后重试可能跳过检测。
  3. fail-closed 命名空间：dp_controller 角色被限制在 4 个命名空间内，未来任何未登记的新 bag 读取都会在 enforce 模式下直接抛错，对依赖该开关用户是显性兼容性变化。
  4. 测试耦合：test_gpu_feature_transport.py 改用真实 ServerArgs(model_path="dummy") 构造，若 ServerArgs 必填字段或构造约束变化，该用例会变脆；同时要求 reset_context 作为 cleanup，跨用例状态泄漏风险需关注。 - 影响：影响面集中在启动路径：所有经过 Engine._launch_subprocesses 的部署形态（TP / PP / DP attention / PD disagg / expert backup / daemon 模式）都会先发布 config bag 再做端口分配与进程拉起；DP 控制器进程内部的配置读取从记录切到 bag。对使用者而言，启动失败后的重试语义有细微变化（auto-parser 字段可能已被写入），但整体行为经 A/B 验证与 base 一致。对团队而言，本 PR 为整个 ServerArgs → config bag 系列提供了可执行的静态守卫，后续解析器新增任何 bag 读取都会被 CPU CI（est_time=5）拦截；HTTP lifespan 的 gRPC 分支、DP controller 的线程标签等也同步切换。 - 风险标记：核心启动路径时序变更，静态守卫非证明（可达性有盲区），auto-parser 字段无法回滚，角色命名空间 fail-closed，快照不覆盖外部 ServerArgs 实例

关联脉络

- PR #35904 config-bag 系列 PR 0（base，据 PR body）：PR body 的 stack 表格说明 #35910 基于 #35904 系列，整体目标是 ServerArgs 从记录原地修改迁移到 resolve-once / publish-into-bags。
- PR #35909 config-bag 系列 PR 5（前序，据 PR body）：本 PR 是系列第 6 步，head 基于 #35909；#35909 很可能引入了 test_publish_precedes_bag_reads.py，本 PR 的 test_resolution_reads_no_bag.py docstring 明确提到它是同一担忧的另一侧。
- PR #35917 Whole-series CI vehicle（not for merge，据 PR body）：PR body 说明 #35917 是整个系列（#35904..#35910）的 CI 载体，用于全栈验证，本 PR 的 GPU 与

Ray round 验证依赖它。