PR #24719 完整报告

sgl-project/sglang

[sgl-model-gateway] Close PyO3 binding gaps and add regression tests

合并时间：2026-05-14 08:43

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24719

执行摘要

- 一句话：补齐 PyO3 绑定缺口并添加回归测试
- 推荐动作：值得精读，特别是 PyO3 绑定模式和测试策略。该 PR 展示了如何安全扩展跨语言绑定，并通过直接调用底层 Rust 类的测试防止接口漂移。对于即将参与绑定开发的工程师，是很好的 reference。

功能与动机

The Python sglang_router wrapper did not surface several RouterConfig fields the Rust binary exposes, plus a few CLI knobs were inert. This PR closes the actionable gaps without changing user-visible defaults.

实现拆解

1. Rust 绑定层（lib.rs）：扩展 PyJwtConfig 结构体，新增 role_claim: String 字段，并转发到 to_auth_jwt_config；在 Router 结构中添加 5 个 #[pyo3(get)] 字段（pool_idle_timeout_secs、connect_timeout_secs、pool_max_idle_per_host、tcp_keepalive_secs、enable_wasm），通过 builder 链式调用填入 RouterConfig。
2. Python 数据类（router_args.py）：在 RouterArgs 中添加对应字段及默认值；集中定义 _POLICY_CHOICES 元组，统一 --policy、--prefill-policy、--decode-policy 的 choices 为全部 8 种策略；暴露 --bootstrap-port-annotation、--jwt-role-claim、--pool-idle-timeout-secs 等 CLI 参数；移除 from_cli_args 中对 bootstrap_port_annotation 的静默覆盖。
3. Python 包装逻辑（router.py）：更新 build_control_plane_auth_config，读取 jwt_role_claim 并传入 PyJwtConfig(role_claim=...)；添加 JWT 部分配置检测，当设置相关字段但缺少 jwt_issuer/jwt_audience 时输出警告；在 Router.from_args 中弹出 jwt_role_claim 避免传递给 Rust 构造函数。
4. 全面回归测试（test_pyo3_binding.py）：新增 726 行测试文件，直接使用 _Router 而非 mock。覆盖所有枚举转换函数（policy_from_str、backend_from_str、history_backend_from_str、role_from_str）的每种变体；验证所有配置类（PyOracleConfig、PyPostgresConfig、PyRedisConfig、PyJwtConfig、PyControlPlaneAuthConfig）的默认值和验证逻辑；通过 Router.from_args(RouterArgs(**every_field)) 构造并断言新字段值到达 Rust 端；端到端测试 --jwt-role-claim 从 CLI 解析到底层 _Router 的完整链路。

关键文件：

- sgl-model-gateway/bindings/python/tests/test_pyo3_binding.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 TestEnumConversions, test_policy_from_str_covers_all_variants, test_policy_from_str_none, test_backend_from_str）：新增 726 行 PyO3 边界测试，覆盖所有枚举转换、配置类验证和端到端 CLI 链路，是确保绑定一致性的关键。
- sgl-model-gateway/bindings/python/src/sglang_router/router_args.py（模块 配置层；类别 source；类型 data-contract；符号 RouterArgs, _POLICY_CHOICES, add_cli_args）：集中化策略选择列表，新增 5 个 RouterArgs 字段和对应 CLI 参数，修复静默覆盖，是配置接口的主要变更点。
- sgl-model-gateway/bindings/python/src/sglang_router/router.py（模块 路由逻辑；类别 source；类型 data-contract；符号 build_control_plane_auth_config, Router.from_args）：更新 build_control_plane_auth_config 以支持 jwt_role_claim，添加 JWT 配置警告，是包装逻辑的核心。
- sgl-model-gateway/bindings/python/src/lib.rs（模块 绑定层；类别 source；类型 data-contract；符号 PyJwtConfig, to_auth_jwt_config, Router::new, Router::to_router_config）：Rust 端绑定，添加 PyJwtConfig.role_claim 字段和 Router 中 5 个新字段，是数据契约的根源。

关键符号：policy_from_str, build_control_plane_auth_config, Router.from_args, add_cli_args, PyJwtConfig::new, PyJwtConfig::to_auth_jwt_config

关键源码片段

sgl-model-gateway/bindings/python/tests/test_pyo3_binding.py

新增 726 行 PyO3 边界测试，覆盖所有枚举转换、配置类验证和端到端 CLI 链路，是确保绑定一致性的关键。

```
from sglang_router.router import (
    policy_from_str,
    backend_from_str,
    history_backend_from_str,
    role_from_str,
)
from sglang_router.sglang_router_rs import (
    PolicyType,
    BackendType,
    HistoryBackendType,
    PyRole,
)


class TestEnumConversions:
    """确保 Python ↔ Rust 枚举转换覆盖所有变体。"""

    def test_policy_from_str_covers_all_variants(self):
        # 此映射必须与 Rust 端 PolicyType 枚举保持同步。
```

```
        # 如果 Rust 端增加新变体而 Python 端未更新，本测试将立即失败。
        cases = {
            "random": PolicyType.Random,
            "round_robin": PolicyType.RoundRobin,
            "cache_aware": PolicyType.CacheAware,
            "power_of_two": PolicyType.PowerOfTwo,
            "bucket": PolicyType.Bucket,
            "manual": PolicyType.Manual,
            "consistent_hashing": PolicyType.ConsistentHashing,
            "prefix_hash": PolicyType.PrefixHash,
        }
        for s, expected in cases.items():
            assert policy_from_str(s) == expected

    def test_policy_from_str_none(self):
        assert policy_from_str(None) is None

    def test_backend_from_str(self):
        assert backend_from_str("sglang") == BackendType.Sglang
        assert backend_from_str("openai") == BackendType.Openai
        assert backend_from_str("SGLANG") == BackendType.Sglang
        assert backend_from_str(None) == BackendType.Sglang
        assert backend_from_str(BackendType.Openai) == BackendType.Openai
        with pytest.raises(ValueError, match="Unknown backend"):
            backend_from_str("vllm")

    def test_history_backend_from_str(self):
        assert history_backend_from_str("memory") == HistoryBackendType.Memory
        assert history_backend_from_str("none") == getattr(HistoryBackendType, "None")
        # ... 其他后端 ...
        with pytest.raises(ValueError, match="Unknown history backend"):
            history_backend_from_str("dynamodb")

    def test_role_from_str(self):
        assert role_from_str("admin") == PyRole.Admin
        assert role_from_str("ADMIN") == PyRole.Admin
        assert role_from_str("user") == PyRole.User
        # 未知角色回退为 User
        assert role_from_str("unknown") == PyRole.User
```

sgl-model-gateway/bindings/python/src/lib.rs

Rust 端绑定，添加 PyJwtConfig.role_claim 字段和 Router 中 5 个新字段，是数据契约的根源。

```
use std::collections::HashMap;

#[pyclass]
#[derive(Clone, Debug, PartialEq)]
pub struct PyJwtConfig {
```

```
    #[pyo3(get, set)]
    pub issuer: String,
    #[pyo3(get, set)]
    pub audience: String,
    #[pyo3(get, set)]
    pub jwks_uri: Option<String>,
    #[pyo3(get, set)]
    pub role_mapping: HashMap<String, String>,
    // 新增字段，默认 "roles"，兼容旧有 4 参数位置调用
    #[pyo3(get, set)]
    pub role_claim: String,
}

#[pymethods]
impl PyJwtConfig {
    #[new]
    // role_claim 放在最后并加上默认值，使得现有 4 参数调用继续正常
    #[pyo3(signature = (
        issuer,
        audience,
        jwks_uri = None,
        role_mapping = HashMap::new(),
        role_claim = String::from("roles"),
    ))]
    fn new(
        issuer: String,
        audience: String,
        jwks_uri: Option<String>,
        role_mapping: HashMap<String, String>,
        role_claim: String,
    ) -> Self {
        PyJwtConfig {
            issuer,
            audience,
            jwks_uri,
            role_mapping,
            role_claim,
        }
    }
}

impl PyJwtConfig {
    pub fn to_auth_jwt_config(&self) -> auth::JwtConfig {
        let mut config = auth::JwtConfig::new(&self.issuer, &self.audience);
        // 将 role_claim 传递给底层 auth 配置
        config.role_claim = self.role_claim.clone();

        if let Some(ref uri) = self.jwks_uri {
            config = config.with_jwks_uri(uri);
```

```
        }

        for (idp_role, gateway_role) in &self.role_mapping {
            let role = match gateway_role.to_lowercase().as_str() {
                "admin" => auth::Role::Admin,
                _ => auth::Role::User,
            };
            config = config.with_role_mapping(idp_role, role);
        }

        config
    }
}
```

评论区精华

无用户评论，但 PR 作者在 body 中说明了关键设计决策：

- role_claim 放在 PyJwtConfig 签名末尾以避免破坏现有 4 参数位置调用者。
- 测试直接使用 _Router 而非 mock，确保任何绑定偏移都会被测试捕获。
- 新增字段默认值与 Rust 二进制现有默认值一致，不改变行为。
- 暂无高价值评论线程

风险与影响

- 风险：
  1. 绑定默认值同步风险：新增字段默认值在 Python 端和 Rust 端硬编码，若 Rust 端 RouterConfig 未来调整默认值，Python 端需手动同步。测试覆盖了新建时传入值验证，但未覆盖默认值同步。
  2. CLI 覆盖移除行为变更：原代码中 from_cli_args 静默覆盖用户提供的 bootstrap_port_annotation，移除后依赖该覆盖的用户可能行为变化，但该覆盖本身是错误，更可能符合预期。
  3. Rust 二进制策略子集未统一：src/main.rs 的 value_parser 仍接受较少策略，造成 CLI 表面不一致，但属故意推迟。 - 影响：影响范围：限于 sgl-model-gateway/bindings/python/ 模块。对用户：新增 5 个 CLI 选项（ --pool-idle-timeout-secs 等）和 --jwt-role-claim，默认行为不变；--policy 等现在接受 bucket 和 consistent_hashing。对开发者：新增全面测试套件，确保 PyO3 绑定与 Rust 端同步，减少后续开发中绑定偏移风险。对系统：无性能影响。
- 风险标记：绑定默认值同步依赖 , CLI 覆盖移除行为变更 , Rust 二进制策略子集未统一

关联脉络

- 暂无明显关联 PR