PR #34994 完整报告

sgl-project/sglang

Build Rust extensions on demand in source checkouts

合并时间：2026-08-17 05:58

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34994

执行摘要

- 一句话：源码检出按需构建 Rust 扩展，统一三 crate 模块命名
- 推荐动作：值得精读，尤其 loader.py 的指纹缓存、文件锁与原子暂存设计，对构建系统开发者有直接借鉴价值。建议关注两个后续优化点：一是缓存失效以整个 rust 工作区为粒度，可优化为按 crate 依赖子图计算 digest；二是 auto 模式下无 Rust 工具链时直接抛错，可考虑先探测 cargo 可用性并给出更友好的降级提示。

功能与动机

PR body 明确指出：Source checkouts (editable installs) currently require the PyO3 extensions (sglang.srt.grpc._core, sglang.srt.server._core, sglang.srt.multimodal._core) to be prebuilt into the wheel: importing them from a plain checkout fails with ModuleNotFoundError (or, for --grpc-port, a hard RuntimeError)。目标是让带有 Rust 工具链的源码检出开箱即用，消除纯 checkout 开发时必须先构建 wheel 的额外步骤，降低 SGLang 开发者调试 gRPC、RustServer 与 inkling 多模态路径的门槛。

实现拆解

按以下 5 步推进实现：

1. 新增按需加载器：新建 python/sglang/srt/rust_extensions/loader.py（412 行），核心函数 load_rust_extension 实现三级加载；crate 通过扫描 rust/ 工作区所有 Cargo.toml 的 [package.metadata.sglang] python-module 元数据定位（与 setup.py 同源），新增 crate 无需任何注册代码；Python 3.10 兼容通过 tomllib/tomli 回退实现。
2. 统一扩展模块命名与 wheel 布局：三个 Rust crate 的 #[pymodule] 入口从 _core 分别改为 _multimodal（sglang-mm）、_grpc（sglang-grpc）、_server（sglang-server），Python 侧模块路径统一收敛到 sglang.srt.rust_extensions.*；新建 rust_extensions/init.py 导出入口；python/setup.py 同步 wheel 内模块名；python/sglang/srt/server/init.py 调整使扩展包在源码检出可导入；同时删除 python/sglang/srt/grpc/init.py 中的旧模块残留。
3. 三处调用点接线：http_server.py 的 _start_native_grpc_server_for_runtime、rust_server.py 的 RustServer.launch、image_processing_rust.py 的 inkling 导入全部改走 load_rust_extension；其中 gRPC 启动路径删除了原先“扩展不在 wheel 中则硬报错”的 try/except 分支。

4. 缓存、并发与三态控制：_build_context 以源码内容 hash（_source_digest）、工具链版本、Python ABI（EXT_SUFFIX/SOABI 等）、构建环境变量（RUSTFLAGS 等）生成 fingerprint 与 target_fingerprint；_filesystem_lock 用 fcntl 文件锁串行化多进程并发构建；构建完成后复核源码 digest 防中途变化；_stage_atomically 原子暂存产物；environ.py 新增 SGLANG_RUST_BUILD_MODE=auto|never|force 环境变量契约。
5. 配套依赖与测试：rust/Cargo.lock 通过 cargo update --precise 定向升级 dynamo 家族（dynamo-protocols 5.1.0→5.3.1 等），并在 rust/sglang-server/src/api_server/openai/chat.rs 的 chat_logprobs 中填充新 token_id 字段；scripts/ci/utils/stage_rust_ext_modules.sh 更新 CI 暂存路径；新增 test/registered/rust/test_rust_extension.py（350 行，纯 mock 不触发真实 cargo），另有多处测试/导入路径跟随调整（test_utils.py、_mm_rust_utils.py、test_server_args.py、bench_parity.py）。

关键文件：

- python/sglang/srt/rust_extensions/loader.py（模块 扩展加载；类别 source；类型 dependency-wiring；符号 _CrateSpec, _BuildContext, load_rust_extension, _import_bundled_extension）：本 PR 核心新增文件，412 行实现三级按需加载、crate 发现、指纹缓存、并发锁与原子暂存，是所有 Rust 扩展导入的统一入口。
- test/registered/rust/test_rust_extension.py（模块 单元测试；类别 test；类型 test-coverage；符号 _hold_filesystem_lock, _FailingExtensionLoader, TestRustExtension, test_bundled_wheel_extension_never_touches_source_or_cargo）：350 行 mock 测试，覆盖加载器全部决策路径，是理解指纹与缓存设计意图的最佳入口，也是后续维护的回归护栏。
- python/sglang/srt/entrypoints/http_server.py（模块 启动入口；类别 source；类型 dependency-wiring）：gRPC 启动路径从硬报错改为按需加载，是用户可见行为变化最明显的接线点。
- python/sglang/srt/managers/rust_server.py（模块 Rust 服务；类别 source；类型 dependency-wiring）：RustServer.launch 是 Rust 原生 server 的入口，Type 引用与运行导入同步迁移到新命名空间。
- python/sglang/srt/multimodal/inkling/image_processing_rust.py（模块 多模态预处理；类别 source；类型 dependency-wiring）：inkling 多模态预处理是第三个接线点，验证 sglang-mm 扩展在 Python 侧的消费方式。
- rust/sglang-server/src/lib.rs（模块 服务内核；类别 source；类型 core-logic；符号 _core, _server）：Rust server crate 的 PyO3 入口更名，是命名空间统一的关键一端。
- rust/sglang-grpc/src/lib.rs（模块 gRPC 内核；类别 source；类型 core-logic；符号 _core, _grpc）：gRPC crate 的 PyO3 入口更名，与 http_server 接线对应。
- rust/sglang-mm/src/lib.rs（模块 多模态内核；类别 source；类型 core-logic；符号 _core, _multimodal）：多模态 crate 的 PyO3 入口更名，并同步更新 crate 文档注释。
- python/setup.py（模块 构建配置；类别 source；类型 core-logic）：wheel 构建配置必须与加载器发现逻辑保持一致，否则打包出的模块路径与运行期期望不符。
- scripts/ci/utils/stage_rust_ext_modules.sh（模块 CI 脚本；类别 infra；类型 infrastructure）：CI 暂存脚本路径同步更新，否则打包产物在 CI 环境中的布局与本地不

一致。

- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 core-logic）：新环境变量 SGLANG_RUST_BUILD_MODE 的运行契约在这里登记。
- rust/sglang-server/src/api_server/openai/chat.rs（模块 对话接口；类别 source；类型 entrypoint）：伴随 Cargo.lock 升级，chat_logprobs 填充新增 token_id 字段，是依赖升级在业务逻辑上的落地。
- rust/Cargo.lock（模块 依赖锁；类别 other；类型 core-logic）：定向升级 dynamo 家族，是 chat_logprobs token_id 填充与 --locked 可复现构建的基础。

关键符号：load_rust_extension, _discover_crate, _build_context, _source_digest, _source_files, _import_bundled_extension, _filesystem_lock, chat_logprobs

关键源码片段

test/registered/rust/test_rust_extension.py

350 行 mock 测试，覆盖加载器全部决策路径，是理解指纹与缓存设计意图的最佳入口，也是后续维护的回归护栏。

```
def test_fingerprint_is_content_based_and_covers_build_inputs(self):
    with TemporaryDirectory() as directory:
        workspace = self._workspace(Path(directory))
        crate = rust_extension._discover_crate(workspace, "demo._core")
        with mock.patch.object(
            rust_extension,
            "_command_version",
            side_effect=lambda command, *args, **kwargs: f"{command} 1.0",
        ):
            first = rust_extension._build_context(crate)
            source = workspace / "demo" / "lib.rs"
            # 只改动 mtime 不得改变指纹，证明指纹基于内容 hash 而非时间戳，
            # 避免 git checkout 等操作触发无谓的全量重建。
            os.utime(source, (1, 1))
            self.assertEqual(first, rust_extension._build_context(crate))

            # 内容变化必须让 fingerprint 变化，从而命中新的缓存路径。
            source.write_text("fn changed() {}\n", encoding="utf-8")
            changed_source = rust_extension._build_context(crate)
            self.assertNotEqual(first.fingerprint, changed_source.fingerprint)

            # RUSTFLAGS 属于构建环境变量，改变它应同时影响
            # fingerprint 与 target_fingerprint（后者用于隔离编译产物）。
            with mock.patch.dict(os.environ, {"RUSTFLAGS": "-Ctarget-cpu=native"}):
                changed_flags = rust_extension._build_context(crate)
            self.assertNotEqual(changed_source.fingerprint, changed_flags.fingerprint)
            self.assertNotEqual(
                changed_source.target_fingerprint,
                changed_flags.target_fingerprint,
            )
```

```
def test_auto_builds_once_then_uses_cache(self):
    # auto 模式首次触发 cargo build，之后同一 fingerprint 直接命中缓存，
    # 验证 cargo_build 只被调用一次，避免重复编译。
    with TemporaryDirectory() as directory:
        root = Path(directory)
        workspace = self._workspace(root)
        artifact = root / "libdemo_extension.so"
        artifact.write_bytes(b"extension")
        context = rust_extension._BuildContext("source", "fingerprint", "target")
        loaded = ModuleType("demo._core")
        with (
            mock.patch.object(
                rust_extension, "_import_bundled_extension", return_value=None
            ),
            mock.patch.object(rust_extension, "_build_context", return_value=context),
            mock.patch.object(rust_extension, "_source_digest", return_value="source"),
            mock.patch.object(
                rust_extension, "_cargo_build", return_value=artifact
            ) as cargo_build,
            mock.patch.object(
                rust_extension, "_load_extension_from_path", return_value=loaded
            ),
        ):
            self.assertIs(
                rust_extension.load_rust_extension(
                    "demo._core", workspace=workspace, cache_dir=root / "cache"
                ),
                loaded,
            )
            self.assertIs(
                rust_extension.load_rust_extension(
                    "demo._core", workspace=workspace, cache_dir=root / "cache"
                ),
                loaded,
            )
        cargo_build.assert_called_once()
```

评论区精华

本 PR 无外部 review 讨论（review 评论为 0），唯一评论是作者 merrymercy 触发的 CI 重跑指令 "/tag-and-rerun-ci"。设计决策主要沉淀在 PR body 与 commit 演进中：

> commit 2 "Fix Python 3.10 TOML compatibility"：确认 loader 需要兼容 3.10，引入 tomli 回退，避免新加载器自身成为版本门槛。
>
> commit 3 "Group Rust extensions under explicit package"：扩展从散落在 sglang.srt.grpc/server/multimodal 下收敛为独立 rust_extensions 包，说明作者在迭代中意识到统一命名空间对发现与缓存的重要性。

> commit 4/5/6 依次跟进 CI 暂存路径与各导入点验证，体现“命名迁移必须同步修改所有引用方”的工程约束。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 首次编译延迟：auto 模式在源码检出且无缓存时会触发 cargo build --release，Rust release 编译可能耗时数分钟，首次启动 gRPC/RustServer/inkling 相关路径会显著变慢，且编译失败信息依赖异常向上传播。
  2. 无 Rust 工具链的源码检出会直接失败：_command_version 在 crate 发现前就会抛 RuntimeError；用户只能显式设置 SGLANG_RUST_BUILD_MODE=never 来获得更友好的错误。
  3. 模块路径迁移破坏性：sglang.srt.grpc._core、sglang.srt.server._core、sglang.srt.multimodal._core 三个旧路径不再存在，任何直接 import 旧路径的第三方代码、文档示例会 ModuleNotFoundError；setup.py 与 CI 脚本已同步，但外部引用无法枚举。
  4. 缓存失效粒度过粗：_source_digest 遍历整个 rust/ 工作区，任一 crate 的源码变化都会让全部三个扩展的缓存集体失效，触发多 crate 重编译。
  5. fcntl 平台限制：_filesystem_lock 依赖 fcntl，仅支持 Unix 系平台，Windows 源码检出不可用；当前 SGLang 主要目标平台为 Linux，属已知边界。
  6. dynamo 依赖升级：Cargo.lock 升级 dynamo-protocols 5.1.0→5.3.1 并新增 token_id 字段，依赖旧版字段行为的其他路径可能存在细微差异；--locked 构建已保证可复现性。 - 影响：影响范围集中在构建期与开发环境，运行时推理路径不变。
- 开发者：源码检出（editable install）搭配 Rust 工具链即可直接使用 gRPC 端口、RustServer 与 inkling 多模态预处理，无需预构建 wheel，显著降低 SGLang 本地开发与调试门槛。
- 系统 / 发布：wheel 内扩展模块路径发生 breaking 变更（sglang.srt.rust_extensions.*），setup.py、CI 暂存脚本、测试导入全部联动更新；新增 SGLANG_RUST_BUILD_MODE 环境变量作为新的运行契约。
- 团队 / 工程：未来新增 Rust crate 只需在 Cargo.toml 声明 python-module 元数据即可自动接入加载器，无需在 Python 侧登记，降低 Rust 化组件的扩展成本，推动更多子模块向 Rust 迁移。
- 风险标记：首次使用触发全量 Rust 编译（分钟级延迟），无 Rust 工具链时 auto 模式直接失败，扩展模块路径迁移破坏旧导入，缓存粒度为整个 workspace，易集体失效，文件锁依赖 fcntl，仅支持 Unix

关联脉络

- PR #34309 标题未在材料中提供（PR body 提及移除 test_cargo_workspace.py）：本 PR 原计划将 test/registered/rust/test_cargo_workspace.py 的 cargo test --workspace 改为 --locked，但该文件已被 #34309 移除，故该部分变更被丢弃而非重新添加。