PR #27411 完整报告

sgl-project/sglang

Add scripted-runtime harness core and wire scheduler/IPC hooks

合并时间：2026-06-06 09:07

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27411

执行摘要

- 一句话：新增 scripted-runtime 测试框架核心与调度器 IPC 钩子
- 推荐动作：值得对 scripted-runtime 感兴趣或有复杂调度测试需求的工程师阅读，尤其 ScriptedSchedulerHook 的 IPC 分发和 ScriptedHttpServer 的生命周期管理设计。

功能与动机

为支持可脚本化的运行时测试框架（scripted-runtime），需要构建核心调度器钩子和 IPC 通信层，以允许外部脚本控制调度器的行为（如启停请求、检查状态），从而实现自动化、确定性测试。（来自 PR 标题及系列先决 PR 的上下文）

实现拆解

1. 新增 Scheduler Hook：在 python/sglang/test/scripted_runtime/scheduler_hook.py 中定义 ScriptedSchedulerHook，通过 _run_dispatch_loop 监听 ZMQ 消息，收到 RunScript 命令后重置引擎状态并执行用户脚本的生成器。
2. 暴露 Context API：在 context/api.py 中提供 ScriptedContext，封装 start_req、pause_generation、abort 等操作，内部委托给 lifecycle、queries、engine 子模块。
3. 构建 HTTP Server 包装：ScriptedHttpServer 类封装 SGLang HTTP 服务器的启动、心跳等待、脚本执行和优雅关闭，使用 ZMQ PAIR 与 hook 通信。
4. 连接调度器：修改 scheduler.py 新增 maybe_init_scripted_scheduler_hook 函数，在调度器初始化时按 SGLANG_TEST_SCRIPTED_RUNTIME_IPC_ADDR 环境变量判断是否创建 hook 并注入。
5. 编写支持模块：包括 BackgroundHttpPoster（异步 POST）、ScriptedTokenizerRecvProxy（代理 token receipt）、io_struct（IPC 消息定义）、req_handle、context/lifecycle.py、context/queries.py、context/radix.py、context/req_starter.py 等。

关键文件：

- python/sglang/test/scripted_runtime/scheduler_hook.py（模块 调度器钩子；类别 test；类型 test-coverage；符号 ScriptedBatchRecord, _reset_engine_state, ScriptedSchedulerHook, init）：核心组件：实现 ZMQ 消息循环分发，将外部脚本命令映射到调度器控制（重置、运行脚本、记录批次）。是测试框架与调度器的桥梁。

- python/sglang/test/scripted_runtime/context/api.py（模块 API 层；类别 test；类型 test-coverage；符号 ScriptedContext, init, start_req, pause_generation）：用户可见 API 层：ScriptedContext 提供 start_req、pause_generation、abort、flush_cache 等高层操作，并整合查询和生命周期子模块。
- python/sglang/test/scripted_runtime/http_server.py（模块 HTTP 服务器；类别 test；类型 test-coverage；符号 ScriptedHttpServer, init, start, execute_script）：HTTP 服务器包装：负责启动 SGLang 服务进程、等待就绪、执行脚本、优雅关闭，是测试入口。
- python/sglang/srt/managers/scheduler.py（模块 调度器入口；类别 source；类型 core-logic；符号 maybe_init_scripted_scheduler_hook）：生产代码唯一改动：新增 maybe_init_scripted_scheduler_hook 函数，条件初始化钩子并注入 scheduler，是测试框架与调度器的桥梁。
- python/sglang/test/scripted_runtime/context/queries.py（模块 查询模块；类别 test；类型 test-coverage；符号 _get_all_reqs, list_active_reqs, batch_composition, is_idle）：提供对调度器状态的查询函数（list_active_reqs, batch_composition, is_idle 等），供测试脚本断言。
- python/sglang/test/scripted_runtime/context/lifecycle.py（模块 生命周期；类别 test；类型 test-coverage；符号 _await_control, pause_generation, continue_generation, abort_all）：实现暂停、继续、中止、flush 等生命周期控制，通过 HTTP POST 发送控制请求并等待回应。
- python/sglang/test/scripted_runtime/background_http_poster.py（模块 后台 HTTP；类别 test；类型 test-coverage；符号 BackgroundHttpPoster, init, _run_loop, submit_coro）：后台异步 HTTP 客户端（基于 aiohttp），用于将控制请求异步 POST 到调度器，避免阻塞脚本生成器。

关键符号：ScriptedSchedulerHook, ScriptedContext, ScriptedHttpServer, maybe_init_scripted_scheduler_hook, ScriptedContext.start_req, ScriptedSchedulerHook.on_run_batch

关键源码片段

python/sglang/test/scripted_runtime/context/api.py

用户可见 API 层：ScriptedContext 提供 start_req、pause_generation、abort、flush_cache 等高层操作，并整合查询和生命周期子模块。

```
# python/sglang/test/scripted_runtime/context/api.py

from __future__ import annotations
import logging
from typing import TYPE_CHECKING, Dict, List, Literal, Optional

from sglang.test.scripted_runtime.context import (
    engine,
    lifecycle,
    queries,
    radix,
```

```
)
from sglang.test.scripted_runtime.context.req_starter import ScriptedContextReqStarter

if TYPE_CHECKING:
    from sglang.srt.managers.schedule_batch import Req
    from sglang.test.scripted_runtime.background_http_poster import BackgroundHttpPoster
    from sglang.test.scripted_runtime.req_handle import ScriptedReqHandle
    from sglang.test.scripted_runtime.scheduler_hook import ScriptedSchedulerHook
    from sglang.test.scripted_runtime.tokenizer_recv_proxy import ScriptedTokenizerRecvProxy

logger = logging.getLogger(__name__)

class ScriptedContext:
    """测试脚本上下文：提供对调度器的控制与查询接口。"""

    def __init__(
        self,
        *,
        scheduler_hook: "ScriptedSchedulerHook",
        tokenizer_recv_proxy: Optional["ScriptedTokenizerRecvProxy"],
        http_poster: "BackgroundHttpPoster",
    ) -> None:
        # 只能在 driver rank 上构造
        assert scheduler_hook._is_driver, "ScriptedContext only exists on the driver rank"
        self.scheduler = scheduler_hook.scheduler
        self._scheduler_hook = scheduler_hook
        self._tokenizer_recv_proxy = tokenizer_recv_proxy
        self._http_poster = http_poster

        self._seen_rids: set[str] = set() # 记录所有见过的 rid，用于判断已结束的请求
        self._req_starter = ScriptedContextReqStarter(self) # 请求启动器（负责实际 HTTP 调用）

    def start_req(
        self,
        *,
        prompt_len: int,
        max_new_tokens: int = 8,
        rid: Optional[str] = None,
        ignore_eos: bool = False,
        priority: Optional[int] = None,
        dp_rank: Optional[int] = None,
        prompt_token: int = 1,
        return_logprob: bool = False,
        logprob_start_len: Optional[int] = None,
        top_logprobs_num: Optional[int] = None,
        lora_path: Optional[str] = None,
    ) -> "ScriptedReqHandle":
        """启动一个新的请求，返回句柄用于后续控制。"""
        return self._req_starter.start_req(
```

```
            prompt_len=prompt_len,
            max_new_tokens=max_new_tokens,
            rid=rid,
            ignore_eos=ignore_eos,
            priority=priority,
            dp_rank=dp_rank,
            prompt_token=prompt_token,
            return_logprob=return_logprob,
            logprob_start_len=logprob_start_len,
            top_logprobs_num=top_logprobs_num,
            lora_path=lora_path,
        )

    def pause_generation(self, *, mode: Literal["retract", "in_place"]) -> None:
        return lifecycle.pause_generation(self, mode=mode)

    def continue_generation(self, *, torch_empty_cache: bool = False) -> None:
        return lifecycle.continue_generation(self, torch_empty_cache=torch_empty_cache)

    def abort_all(self) -> None:
        return lifecycle.abort_all(self)

    def abort(self, handle: "ScriptedReqHandle") -> None:
        return lifecycle.abort(self, rid=handle.rid)

    def flush_cache(self) -> None:
        return lifecycle.flush_cache(self)

    # ... 其他方法（evict_radix, get_all_node_hit_counts, 属性等）
```

评论区精华

无 review 评论，PR 由作者自行合并。

- 暂无高价值评论线程

风险与影响

- 风险：scheduler.py 中新增的条件钩子初始化路径可能干扰原有调度流程，但钩子仅在测试环境通过 SGLANG_TEST_SCRIPTED_RUNTIME_IPC_ADDR 环境变量激活；新增的 IPC 通信依赖 ZMQ，若端口冲突或连接超时可能导致测试孤悬。
- 影响：影响范围限于测试框架使用者：提供了一套完整的脚本化测试 API 和 HTTP 服务器包装。对生产调度器无影响（条件初始化）。团队可以获得更可靠的自动化回归测试能力。
- 风险标记：新增调度器条件钩子，依赖 ZMQ IPC 通信

关联脉络

- PR #27413 Add scripted-runtime unit, core integration, and chunked-prefill tests: 添加针对 scripted-runtime 的单元测试、集成测试和 chunked-prefill 测试，依赖本 PR 的核心组件。
- PR #27412 Add scripted-runtime KV-pool and lock-ref exhauster primitives: 添加 KV 池和锁引用耗尽原语，扩展 scripted-runtime 的测试能力。
- PR #27405 Don't write crash dump on graceful exit: 同一系列中修复优雅退出时误写 crash dump 的问题。