PR #34679 完整报告

sgl-project/sglang

fix(constrained): reject NUL bytes in grammar specs to stop an xgrammar segfault

合并时间：2026-08-20 06:31

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34679

执行摘要

- 一句话：拒绝 grammar 中的 NUL 字节，修复 xgrammar 段错误
- 推荐动作：此 PR 值得精读，尤其是 _grammar_key_contains_nul 的实现细节，展示了如何在底层第三方库存在潜在崩溃风险时，通过上层校验来规避。也展示了好的防御性编程实践。

功能与动机

PR 描述说明了触发条件：以 NUL 字节开头的 regex 会使 xgrammar 的 regex 转换器追加自身的 NUL 终止符并索引越界，引发 SIGSEGV，导致调度器以 exit code -11 崩溃，后续请求全部被拒。由于是 OS 信号，异常无法被捕获。此外，JSON schema 通过 pattern 也可能导致同样的崩溃，但请求体中可能没有 NUL 字节（如 \u0000 转义）。

实现拆解

1. 新增 _grammar_key_contains_nul 函数（位于 base_grammar_backend.py）：
   - 先检查原始字符串是否包含 NUL 字节，若命中直接返回 True。
   - 对 json 和 structural_tag 类型，进一步解析 JSON 并递归遍历所有字符串值，检查是否包含 \u0000 转义形式。
   - 对于格式错误的 JSON，则返回 False，让后端正常报错。
2. 在 _init_value_dispatch 张入口调用该检查，若命中则记录错误并返回 InvalidGrammarObject，替代原本可能触发崩溃的后端调用。
3. 测试文件 test_base_grammar_backend.py 新增 TestNulByteGrammarRejection 测试类，覆盖含 NUL 的 regex、ebnf、structural_tag、JSON pattern（原始和转义形式），以及非 NUL 控制字符、转义形式的非 NUL、格式错误 JSON 等正常请求。
4. 后续提交精简了注释，并指示待 xgrammar 上游修复版本合并后移除该 guard。

关键文件：

- python/sglang/srt/constrained/base_grammar_backend.py（模块 约束解码；类别 source；类型 dependency-wiring；符号 _grammar_key_contains_nul）：核心变更文件，新增 NUL 字节检查函数并在调度入口调用，防止崩溃。
- test/registered/unit/constrained/test_base_grammar_backend.py（模块 约束测试；类别 test；类型 test-coverage；符号 TestNulByteGrammarRejection, setUp, tearDown,

test_nul_payload_never_reaches_backend）：新增测试类覆盖 NUL 拒绝和正常分派，验证修复效果且无回归。

关键符号：_grammar_key_contains_nul

关键源码片段

python/sglang/srt/constrained/base_grammar_backend.py

核心变更文件，新增 NUL 字节检查函数并在调度入口调用，防止崩溃。

```
import json

def _grammar_key_contains_nul(key_type: str, key_string: str) -> bool:
    """A NUL in a spec segfaults xgrammar's regex converter, which a JSON schema
    also reaches through `pattern` (possibly escaped). Drop once the upstream fix
    https://github.com/mlc-ai/xgrammar/pull/850 is in our pinned version."""
    # 先检查原始字符串中是否有 NUL 字节
    if "\x00" in key_string:
        return True
    # 对 json 和 structural_tag 类型，需要递归检查转义后的 \u0000
    if key_type not in ("json", "structural_tag"):
        return False
    try:
        decoded = json.loads(key_string)
    except ValueError:
        # 格式错误的 JSON，交给后端正常报错
        return False
    # 深度优先遍历 JSON 结构，检查所有字符串值
    stack = [decoded]
    while stack:
        node = stack.pop()
        if isinstance(node, str):
            if "\x00" in node:
                return True
        elif isinstance(node, dict):
            stack.extend(node.keys())
            stack.extend(node.values())
        elif isinstance(node, list):
            stack.extend(node)
    return False

# 在 _init_value_dispatch 中调用
if _grammar_key_contains_nul(key_type, key_string):
    logger.error(f"Rejecting {key_type} grammar containing a NUL byte")
    return InvalidGrammarObject(
        f"Invalid {key_type}: NUL bytes (\\u0000) are not allowed"
    )
```

test/registered/unit/constrained/test_base_grammar_backend.py

新增测试类覆盖 NUL 拒绝和正常分派，验证修复效果且无回归。

```
import json

class TestNulByteGrammarRejection(unittest.TestCase):
    def setUp(self):
        self.backend = BaseGrammarBackend()

    def tearDown(self):
        self.backend.executor.shutdown(wait=True)

    def test_nul_payload_never_reaches_backend(self):
        # 各种含 NUL 的 grammar 都应该被拒绝，且不会调用底层后端
        cases = [
            ("regex", "\x00\x01\x02\x1f"),
            ("regex", "\x00"),
            ("regex", "a\x00b"), # 非开头的 NUL 也会崩溃，固定该行为
            ("ebnf", "root ::= \x00"),
            ("structural_tag", '{"triggers": ["\x00"]}'),
            ("json", '{"type":"string","pattern":"\\u0000"}'), # 转义形式
            ("json", '{"type":"object","properties":{"f":{"type":"string","pattern":"\\u0000x"}}}'),
            ("structural_tag", '{"format":{"pattern":"\\u0000"}}'),
        ]
        for key_type, key_string in cases:
            with self.subTest(key_type=key_type, key_string=repr(key_string)):
                dispatch = MagicMock()
                setattr(self.backend, f"dispatch_{key_type}", dispatch)
                result = self.backend._init_value_dispatch((key_type, key_string), False)
                self.assertIsInstance(result, InvalidGrammarObject)
                dispatch.assert_not_called()

    def test_nul_free_payload_still_dispatches(self):
        # 正常的 grammar 仍然会正常分派
        cases = [
            ("regex", "[0-9]+"),
            ("regex", r"\x00"), # 字面量转义，不是 NUL 字节
            ("regex", "\x01\x02\x1f"), # 其他控制字符不受影响
            ("json", json.dumps({"type": "string", "pattern": "[0-9]+"})),
            ("json", "{not valid json"), # 格式错误 JSON 交给后端报错
        ]
        for key_type, key_string in cases:
            with self.subTest(key_type=key_type, key_string=repr(key_string)):
                dispatch = MagicMock(return_value=BaseGrammarObject())
                setattr(self.backend, f"dispatch_{key_type}", dispatch)
                self.backend._init_value_dispatch((key_type, key_string), False)
                dispatch.assert_called_once_with(key_string)
```

评论区精华

- kpham-sgl: 要求移除 AI 生成的注释，并添加备注提醒在 xgrammar 修复后移除该补丁。作者已在后续提交中完成。
- 维护者建议并鼓励作者到 xgrammar 仓库提交 issue 和 PR（作者已提交 https://github.com/mlc-ai/xgrammar/pull/850）。
- CI 运行通过，/rerun-test 成功。
- xgrammar 上游修复跟进 (design): 作者已提交 xgrammar PR #850，并在注释中保留指针，计划在升级 xgrammar 版本后移除 guard。

风险与影响

- 风险：
  - 该补丁只是防御性检查，未改变正常 grammar 的处理路径，对合法请求无影响。
  - 若 xgrammar 上游修复版本发布，若未及时移除 guard，可能对含 NUL 的合法请求（极少见）造成误拒，建议跟踪上游版本并按要求移除。
  - 对非常规的 encoding 或大 JSON 可能带来轻微性能开销，但仅在初次编译 grammar 时执行。
  - 影响：影响所有使用 xgrammar 后端的 constrained decoding 用户，防止恶意或意外输入的 grammar 导致服务器崩溃，提升了服务的稳定性和安全性。整体影响为正，变更极小。
  - 风险标记：依赖第三方库未修复，防御性校验可能误拒极少见合法请求

关联脉络

- PR #35679 [diffusion] Refresh eager optimization skills and benchmark safeguards: 同为代码质量与稳定性改进，展示了仓库近期对稳定性的关注。