PR #37029 完整报告

sgl-project/sglang

fix(frontend): bound stop strings and regex patterns

合并时间：2026-08-30 13:25

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37029

执行摘要

- 一句话：为 stop 与 stop_regex 加上限，超限请求返回 400
- 推荐动作：值得精读，虽然只有 60 行改动，但 review 交锋展示了 " 参数校验应放在数据归一化的必经漏斗（SamplingParams.normalize()）而不是请求入口（GenerateReqInput）" 的架构判断，以及 Python / Rust 双实现如何对齐（错误消息、常量名、限制数量）。作者第一版放错位置被驳回、第二版按要求迁移并补齐限制的过程，是 API 防护与校验分层的典型案例。

功能与动机

PR body 明确指出：请求目前可以携带无界数量的 stop 字符串或 stop 正则模式，而这些内容在解码期间会被逐一检查，超大输入会浪费 CPU 并拖慢其他请求（"oversized inputs can waste CPU and slow down other requests"）。reviewer 进一步补充：stop_regex 是更热的路径——每个解码步骤、每条模式都会执行 re.search（见 schedule_batch.py:1591-1599），因此正则的数量与单条长度都需要封顶，且限制阈值与错误文案必须与 Rust 侧 rust/sglang-server/src/message/sampling.rs 的 normalize_stops()（sampling.rs:334-348）保持一致。

实现拆解

1. 定义上限常量：在 python/sglang/srt/sampling/sampling_params.py 文件顶部新增 MAX_STOP_COUNT = 32、MAX_STOP_REGEX_COUNT = 32、MAX_STOP_REGEX_LEN = 256 三个模块级常量，与 Rust 侧 sampling.rs 的 normalize_stops() 镜像对齐，后续任一侧调整都需同步另一侧。
2. stop 字符串数量校验：在 normalize() 的 stop 分支中，先沿用原有逻辑把单个字符串包装成列表，再检查 len(self.stop_strs) > MAX_STOP_COUNT，超限抛 ValueError，文案严格复刻 Rust 的 "at most {MAX_STOP_COUNT} stop strings are allowed, got {n}"。
3. stop_regex 数量与字节长度校验：在 normalize() 的 stop_regex 分支中，先检查模式数量是否超过 MAX_STOP_REGEX_COUNT；随后在遍历每条模式时用 stop_regex.encode("utf-8") 计算字节长度，超过 256 字节即抛 ValueError。按字节而非字符计数，使中文、emoji 等多字节字符也能被正确计量。
4. 错误传播为 HTTP 400：normalize() 是所有采样参数的必经漏斗，包括 preferred_sampling_params 在 tokenizer_manager.py:1352-1356 合并后的参数；在这里抛出的 ValueError 沿请求处理链被 API 层映射为 HTTP 400。这正是第一版被驳回的关

键原因——在 io_struct.py 校验原始 dict 会漏掉合并进来的 preferred_sampling_params。

5. 测试配套：在 test/registered/unit/sampling/test_sampling_params.py 中新增 test_stop_count_limit、test_stop_regex_count_limit、test_stop_regex_byte_length_limit 三个边界测试，均验证 " 恰好等于上限可通过、上限 + 1 被拒 "；字节长度测试特意用多字节字符 "é" 构造恰好 256 字节的模式，验证字节计数语义。该测试文件已注册 CPU/XPU CI 标签（register_cpu_ci / register_xpu_ci）。

关键文件：

- python/sglang/srt/sampling/sampling_params.py（模块 采样参数；类别 source；类型 core-logic；符号 MAX_STOP_COUNT, MAX_STOP_REGEX_COUNT, MAX_STOP_REGEX_LEN, normalize）：核心变更文件：新增三个上限常量，并在 normalize() 中为 stop 字符串与 stop_regex 增加数量与字节长度校验，是所有请求路径的必经点。
- test/registered/unit/sampling/test_sampling_params.py（模块 采样参数；类别 test；类型 test-coverage；符号 test_stop_count_limit, test_stop_regex_count_limit, test_stop_regex_byte_length_limit）：配套边界测试：覆盖 stop 数量、stop_regex 数量与字节长度的 " 恰好上限通过、超限拒绝 " 边界，并用多字节字符验证字节计数语义。

关键符号：normalize, test_stop_count_limit, test_stop_regex_count_limit, test_stop_regex_byte_length_limit

关键源码片段

python/sglang/srt/sampling/sampling_params.py

核心变更文件：新增三个上限常量，并在 normalize() 中为 stop 字符串与 stop_regex 增加数量与字节长度校验，是所有请求路径的必经点。

```
# 上限常量需与 Rust 侧 rust/sglang-server/src/message/sampling.rs 的
# normalize_stops() 保持同步，避免 Python / Rust 双实现行为分叉
MAX_STOP_COUNT = 32
MAX_STOP_REGEX_COUNT = 32
MAX_STOP_REGEX_LEN = 256 # 单条 stop_regex 的 UTF-8 字节上限


def normalize(self, tokenizer):
    # ---- stop 字符串：包装成列表后先做数量校验 ----
    if self.stop_strs is None:
        self.stop_strs = []
        self.stop_str_max_len = 0
    else:
        # API 允许传单个字符串，统一包装成列表后再校验数量
        if isinstance(self.stop_strs, str):
            self.stop_strs = [self.stop_strs]
        if len(self.stop_strs) > MAX_STOP_COUNT:
            # 文案与 Rust 侧 sampling.rs:335 保持一致，便于客户端统一处理
            raise ValueError(
                f"at most {MAX_STOP_COUNT} stop strings are allowed, "
```

```
                f"got {len(self.stop_strs)}"
            )

        stop_str_max_len = 0
        for stop_str in self.stop_strs:
            if tokenizer is not None:
                stop_str_ids = tokenizer.encode(stop_str, add_special_tokens=False)
                stop_str_max_len = max(stop_str_max_len, len(stop_str_ids))
            else:
                stop_str_max_len = max(stop_str_max_len, len(stop_str))
        self.stop_str_max_len = stop_str_max_len

    # ---- stop_regex：数量 + 单条字节长度双重校验 ----
    if self.stop_regex_strs is None:
        self.stop_regex_strs = []
        self.stop_regex_max_len = 0
    else:
        if isinstance(self.stop_regex_strs, str):
            self.stop_regex_strs = [self.stop_regex_strs]
        if len(self.stop_regex_strs) > MAX_STOP_REGEX_COUNT:
            raise ValueError(
                f"at most {MAX_STOP_REGEX_COUNT} stop_regex patterns are allowed, "
                f"got {len(self.stop_regex_strs)}"
            )

        stop_regex_max_len = 0
        for stop_regex in self.stop_regex_strs:
            # 按 UTF-8 字节计数：多字节字符（如 "é"）也会被正确计量，
            # 防止超长正则放大每个解码步骤的 re.search 开销
            stop_regex_len = len(stop_regex.encode("utf-8"))
            if stop_regex_len > MAX_STOP_REGEX_LEN:
                raise ValueError(
                    f"stop_regex is {stop_regex_len} bytes, over the "
                    f"{MAX_STOP_REGEX_LEN}-byte limit"
                )
            stop_regex_max_len = max(
                stop_regex_max_len, get_max_seq_length(stop_regex)
            )

        self.stop_regex_max_len = stop_regex_max_len
```

test/registered/unit/sampling/test_sampling_params.py

配套边界测试：覆盖 stop 数量、stop_regex 数量与字节长度的 " 恰好上限通过、超限拒绝 " 边界，并用多字节字符验证字节计数语义。

```
  def test_stop_count_limit(self):
      # 恰好 MAX_STOP_COUNT 条时应通过，超限 1 条即抛 ValueError
      tokenizer = self._mock_tokenizer()
      SamplingParams(stop=["x"] * MAX_STOP_COUNT).normalize(tokenizer)
```

```
        with self.assertRaises(ValueError) as cm:
            SamplingParams(stop=["x"] * (MAX_STOP_COUNT + 1)).normalize(tokenizer)
        self.assertEqual(
            str(cm.exception),
            f"at most {MAX_STOP_COUNT} stop strings are allowed, got {MAX_STOP_COUNT + 1}",
        )


    def test_stop_regex_byte_length_limit(self):
        # "é" 在 UTF-8 中占 2 字节：用 MAX_STOP_REGEX_LEN // 2 个 "é" 构造恰好
        # 256 字节的边界模式，验证限制按字节而非字符计数
        tokenizer = self._mock_tokenizer()
        pattern = "é" * (MAX_STOP_REGEX_LEN // 2)
        SamplingParams(stop_regex=pattern).normalize(tokenizer)

        with self.assertRaises(ValueError) as cm:
            # 追加 1 个 ASCII 字符后达 257 字节，应被拒绝
            SamplingParams(stop_regex=pattern + "a").normalize(tokenizer)
        self.assertEqual(
            str(cm.exception),
            f"stop_regex is {MAX_STOP_REGEX_LEN + 1} bytes, over the "
            f"{MAX_STOP_REGEX_LEN}-byte limit",
        )
```

评论区精华

hnyls2002 的核心意见是：方向正确但层级错误且不完整。校验应放在 SamplingParams.normalize() 中，它是 Rust normalize_stops() 的镜像，也是真正的必经漏斗；在 io_struct.py 校验会漏掉 preferred_sampling_params（tokenizer_manager.py:1352-1356 才合并进采样参数）。此外 Rust 侧有三个限制，Python 只实现了 stop 数量一个——stop_regex 更贵（每个解码步骤每条模式都 re.search），必须一并补齐；continue 静默接受非 dict 参数会让限制变成 best-effort；错误文案与常量名需与 Rust 对齐。作者在第二轮提交 ab9b2a17 中全部落实，评审无进一步异议。

- 校验位置应在 SamplingParams.normalize() 而非 io_struct.py (design): 作者将校验从 GenerateReqInput 迁移到 SamplingParams.normalize()，覆盖所有请求路径。
- 缺失 stop_regex 的两个限制，需对齐 Rust 三项上限 (correctness): 作者补齐 stop_regex 数量（32 条）与单条 UTF-8 字节长度（256 字节）校验。
- continue 静默接受非 dict 参数使限制变成 best-effort (correctness): 随校验迁移到 SamplingParams.normalize()，该问题不复存在。
- 错误消息与常量名需与 Rust 一致 (style): 作者统一常量命名为 MAX_STOP_COUNT，错误文案逐字对齐 Rust。
- 测试随校验迁移并补充边界场景 (testing): 测试迁移至 test_sampling_params.py，新增三个边界测试，其中 test_stop_regex_byte_length_limit 覆盖裸字符串与多字节边界。

风险与影响

- 风险：
  1. 行为变更风险：此前可正常请求的存量调用方，若携带超过 32 条 stop / stop_regex，或单条 regex 超过 256 字节，将改为收到 HTTP 400；PR 未评估存量超限调用方分布，存在兼容性影响。
  2. 字节计数语义：MAX_STOP_REGEX_LEN 按 UTF-8 字节计算，非 ASCII 字符（中文、emoji）会更快触顶，错误消息暴露的是字节数而非字符数，客户端需理解该语义。
  3. 双端同步维护风险：Python sampling_params.py 与 Rust sampling.rs 的常量与错误文案需手工保持一致，任一侧后续改动都可能造成行为分叉。
  4. 校验覆盖正面效应：因校验落在 normalize() 必经点，覆盖 preferred_sampling_params 合并后的路径，比第一版在 io_struct.py 的入口校验覆盖面更完整，漏检风险更低。 - 影响：用户可见变化是超限请求从 " 被接受但拖慢服务 " 变为 " 快速拒绝并返回明确 400 错误 "；系统层面消除了解码阶段无界 re.search 匹配导致的 CPU 浪费与请求间相互拖慢的放大效应；团队层面确立了 Python / Rust 采样参数校验的镜像关系，为后续新增参数上限提供了明确落点范例。整体影响面覆盖所有走 SamplingParams.normalize() 的请求路径，但改动行数少、逻辑简单，风险可控。 - 风险标记：超限请求行为变更（400），Python/Rust 双端同步，存量调用方兼容性未评估

关联脉络

- 暂无明显关联 PR