PR #23991 完整报告

sgl-project/sglang

Add registry for custom speculative algorithms

合并时间：2026-05-08 07:11

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23991

执行摘要

- 一句话：为自定义推测算法添加注册机制
- 推荐动作：值得精读，它展示了如何在不破坏现有代码的前提下提供扩展点。对于需要自定义推测算法的用户尤其有价值，设计模式具有参考意义。

功能与动机

PR 描述明确指出目标是让下游用户可以注册自定义推测方法，而无需修改 SGLang 源码。此前推测算法以枚举值硬编码，第三方无法扩展；本 PR 通过注册表机制解耦。

实现拆解

1. 新增注册表模块(spec_registry.py)：定义 CustomSpecAlgo 鸭式类型（默认 is_*() 方法除 is_speculative 外均返回 False）和全局注册表 _REGISTRY，提供 register_algorithm 装饰器和 get_spec 查找函数。
2. 集成到 SpeculativeAlgorithm 枚举(spec_info.py)：from_string 方法在枚举查找失败后回退到注册表查询；新增 register 类方法作为对外 API。
3. 调整服务参数验证(server_args.py)：移除 --speculative-algorithm 的硬编码 choices，先统一转为大写，通过 from_string 解析；若返回 CustomSpecAlgo 实例且设有 validate_server_args，则调用进行自定义校验。
4. 增加单元测试(test_spec_registry.py)：覆盖注册成功、重复名称冲突、保留名称保护、大小写不敏感、鸭式接口行为等，确保注册表隔离不污染测试环境。

关键文件：

- python/sglang/srt/speculative/spec_registry.py（模块 注册表；类别 source；类型 core-logic；符号 CustomSpecAlgo, init, repr, is_none）：核心新增文件，定义 CustomSpecAlgo 鸭式类型和全局注册表，提供 register_algorithm 装饰器和 get_spec 查找函数
- python/sglang/srt/speculative/spec_info.py（模块 枚举集成；类别 source；类型 dependency-wiring；符号 from_string, register）：修改枚举类，集成注册表查找和 register 类方法
- python/sglang/srt/server_args.py（模块 服务参数；类别 source；类型 dependency-wiring；符号 _handle_speculative_decoding）：移除 --speculative-algorithm 硬编码 choices，集成自定义算法解析

- test/registered/unit/spec/test_spec_registry.py（模块 单元测试；类别 test；类型 test-coverage；符号 _RegistryIsolated, TestFromString, TestRegister, TestCustomSpecAlgoInterface）：单元测试，覆盖注册、冲突、保留名称、大小写不敏感、鸭式接口等场景

关键符号：CustomSpecAlgo.init, CustomSpecAlgo.is_speculative, CustomSpecAlgo.create_worker, register_algorithm, get_spec, SpeculativeAlgorithm.from_string, SpeculativeAlgorithm.register

关键源码片段

python/sglang/srt/speculative/spec_info.py

修改枚举类，集成注册表查找和 register 类方法

```
# python/sglang/srt/speculative/spec_info.py ( 节选 )

from __future__ import annotations
from typing import TYPE_CHECKING, Callable, Optional, Type, Union

from sglang.srt.speculative.spec_registry import (
    CustomSpecAlgo,
    ServerArgsValidator,
    WorkerFactory,
)
from sglang.srt.speculative.spec_registry import get_spec as _get_registered_spec
from sglang.srt.speculative.spec_registry import register_algorithm as _register_algorithm


class SpeculativeAlgorithm(Enum):
    """Builtin speculative algorithms. Plugin-registered ones are ``CustomSpecAlgo`` instances.
    """

    DFLASH = auto()
    EAGLE = auto()
    EAGLE3 = auto()
    FROZEN_KV_MTP = auto()
    STANDALONE = auto()
    NGRAM = auto()
    NONE = auto()

    @classmethod
    def from_string(
        cls, name: Optional[str]
    ) -> Union[SpeculativeAlgorithm, CustomSpecAlgo]:
        """解析算法名称：先尝试内置枚举，再查询注册表"""
        if name is None:
            return cls.NONE
        upper = name.upper()
        try:
```

```
            return cls[upper] # 内置枚举匹配
        except KeyError:
            pass
        spec = _get_registered_spec(upper) # 查询插件注册表
        if spec is not None:
            return spec
        raise ValueError(f"Unknown speculative algorithm name: {name}")

    @classmethod
    def register(
        cls,
        name: str,
        *,
        supports_overlap: bool = False,
        validate_server_args: Optional[ServerArgsValidator] = None,
        spec_class: Type[CustomSpecAlgo] = CustomSpecAlgo,
    ) -> Callable[[WorkerFactory], WorkerFactory]:
        """对外注册 API，委托给 _register_algorithm"""
        return _register_algorithm(
            name,
            supports_overlap=supports_overlap,
            validate_server_args=validate_server_args,
            spec_class=spec_class,
        )
```

评论区精华

- jasonjk-park在 spec_registry.py 中询问 supports_spec_v2 是否应返回 self.supports_overlap（代码已如此）。
- merrymercy质疑自定义算法为何需要 is_eagle 等方法；hnyls2002解释公共代码路径使用这些方法进行调度，若不实现可能导致自定义算法崩溃。
- supports_spec_v2 实现 (correctness): 代码已正确返回 self.supports_overlap，该评论确认正确性后关闭
- 自定义算法需要 is_eagle 等方法的原因 (design): hnyls2002 解释了必要性，merrymercy 未进一步质疑，设计决策被接受

风险与影响

- 风险：
  1. 全局注册表冲突：不同插件可能覆盖相同名称，但通过保留名称集和重复注册检查缓解。
  2. 自定义算法接口兼容：部分调度代码仍通过 is_xxx() 硬编码分支，自定义算法默认返回 False 可能跳过优化路径，导致性能降级或功能异常。
  3. 硬编码字符串检查：server_args.py 中与特定算法相关的检查（如 DFLASH）仍使用字符串比较，自定义算法需额外处理。
  4. 迁移工作：并非所有 is_xxx 使用点都被识别为 capability 检查，后续需逐步重构为 supports_xxx。 - 影响：用户：可自由集成自定义推测算法，降低二次开发成本。系统：

注册表查找增加极小开销，命名冲突有保护机制。团队：明确了扩展点，未来可将内置算法逐步迁移至同一框架，减少 special case。 - 风险标记：全局注册表冲突，自定义算法接口兼容，硬编码路径未完全适配

关联脉络

- PR #24436 [Gemma 4] Adding MTP support: 同属 speculative decoding 功能演进，展示本 PR 注册机制可服务的场景