

PR #39015 完整报告

sgl-project/sglang

[Router] Add the shutdown-drain configuration surface

合并时间: 2026-09-16 15:26

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39015>

[Router] Add the shutdown-drain configuration surface (PR #39015)

执行摘要

`experimental/sgl-router` 新增优雅关闭所需的「配置面」: `--shutdown-drain-secs` (默认 30s) 与 `--termination-grace-secs` (可选、无默认), 以及 `K8S_DEFAULT_GRACE_SECS = 30`、`MAX_SHUTDOWN_DRAIN_SECS = 1800`、`ShutdownDrainAdvisory / shutdown_drain_advisory()` 和 `Config::validate` 中的上限硬闸门。本 PR 是 3 段提交栈的第 2 段, 没有任何代码读取 `shutdown_drain_secs`, 真正的 SIGTERM 暂停与在途 drain 在 #39016 才发生, 因此运行期行为只多了一次启动校验和一次启动告警; 23 个文件、+347/-12 的体量中, 语义核心集中在 `config/mod.rs`、`config/types.rs`、`config/cli.rs`, 其余 15+ 个文件都是给 `ServerConfig` 字面量补 `..Default::default()`。适合负责 router 部署形态与滚动升级语义的工程师精读, 对其它方向只需了解结论。

功能与动机

要解决的问题有三个层次:

1. 端点摘除与进程关闭的竞态: router 收到 SIGTERM 后若立即关闭 listener, k8s 的端点摘除还没到达 kube-proxy, 新请求仍会被转发到正在关闭的 pod 上。因此需要一个「已经 /readyz 503、但仍在服务」的窗口。
2. 探针驱动的负载均衡器: 这类 LB 要到 `failureThreshold * periodSeconds` 次 /readyz 失败之后才会摘除, 窗口必须按这个乘积来定尺寸。
3. 信息不对称: router 读不到自己的 pod spec。缺少真实的 `terminationGracePeriodSeconds` 时, 启动检查只能对着 k8s 默认值比较, 会把一个故意设长的正确 drain 误报成不安全。

作者对范围与边界的表述很明确:

「Adds the two flags a SIGTERM readiness drain needs, plus the checks that keep them honest. Nothing reads `shutdown_drain_secs` yet—the shutdown path lands in the next PR in the stack — so this changes no runtime behaviour beyond one startup validation and one startup warning.」

「The default equals the k8s default `terminationGracePeriodSeconds`, so a pod that has not raised its grace period has nothing left for the in-flight drain and

`shutdown_drain_advisory` warns at every startup. That is the intended reading rather than a misconfigured default]

栈的关系是：#39014（在途 HTTP 计数）→ #39015（本 PR，配置面）→ #39016（drain 本身 + k8s 证明，原 #39017 的测试已折入）。

实现拆解

1. 配置类型：ServerConfig 扩字段与默认值收敛（src/config/types.rs, +59）

新增 `shutdown_drain_secs: u64` 与 `termination_grace_secs: Option<u64>`；新增 `ServerConfig::shutdown_drain()` 把秒 → `Duration` 的转换固定进库内，理由写在注释里：放在 `main.rs` 的调用点，一个 `from_secs` / `from_millis` 手误会把每次 drain 静默缩短 1000 倍，而且没有任何东西会失败。同时抽出 `default_host()` / `default_port()` / `default_shutdown_drain_secs()`，并首次为 `ServerConfig` 实现 `Default`——注释说明这是因为 `tests/` 是独立 crate，`#[cfg(test)]` 构造器够不到集成测试 fixture。

2. CLI 入口：两个 flag 与穷举映射（src/config/cli.rs, +94/-8）

新增 `--shutdown-drain-secs` (`default_value_t = default_shutdown_drain_secs()`) 与 `--termination-grace-secs` (`Option<u64>`，缺省 `None` 表示「假设 k8s 默认」)；`host / port` 的默认值也改走 `default_*`()，消除默认值的第二处定义。`Cli::into_config` 填充两个字段并保持穷举，所以未来新增字段在生产路径上仍会编译报错、强制做出决定。help 文本承担了运维语义：`failureThreshold * periodSeconds`、`0` 表示关闭暂停、以及「调高 grace period 后要声明它」。

3. 启动校验与告警（src/config/mod.rs, +174/-4）

新增 `K8S_DEFAULT_GRACE_SECS = 30`、`MAX_SHUTDOWN_DRAIN_SECS = 1800`、`ShutdownDrainAdvisory { shutdown_drain_secs, termination_grace_secs, grace_declared }` 与纯函数 `shutdown_drain_advisory()`：边界取 `>=`（drain 正好等于 grace period 已把它全部吃掉），`None` 视为假设默认值。`Config::validate` 加入唯一一条硬闸门，超过 1800 s 直接拒启，错误文本点名 flag 并给出补救动作。`advisory` 返回字段而不是拼好的字符串，是为了让日志聚合能按 `grace_declared` 分组去重。

4. 测试配套（8 个新单测）

- `config/mod.rs` 5 个：低于默认 grace 静默；默认 drain 必然告警、且调高 grace 后必须能静默；`>=` 边界（30 / 120 / 1800）都告警且 `grace_declared == false`；声明 grace 的双向行为（60/120 静默、60/60 告警、10/10 告警、1800/3600 静默）；上限本身可启动、超一被拒。
- `config/cli.rs` 3 个：0 / 17 / 1800 多值透传并校验 `shutdown_drain()` 与秒数一致；18000 在 CLI 路径被拒且错误信息点名 `shutdown_drain_secs`；`--termination-grace-secs` 映射且缺省为 `None`。

5. fixture plumbing（15+ 个文件）

范围	文件示例	变更
src/** 的 #[cfg(test)] 构造点	policies/factory.rs、server/app_context.rs、tokenizer/mod.rs、workers/manager.rs	ServerConfig 字面量补 ..Default::default()
tests/ 集成 fixture	tests/component/discovery/static_urls.rs、tests/proxy/**（含 graceful_shutdown.rs、pd_protocol_binding.rs）	同上

这与「保持 `into_config` 穷举」构成一对有意为之的取舍：生产路径强制决定，测试路径允许省略。

6. 明确未做的部分

没有任何代码读取 `shutdown_drain_secs`；`pause`、`/readyz 503`、在途 `drain` 的执行与 `k8s` 端到端证明都留给 #39016。换言之，本 PR 交付的是 契约与不变量，不是行为。

experimental/sgl-router/src/config/mod.rs

语义核心：新增 `K8S_DEFAULT_GRACE_SECS`、`MAX_SHUTDOWN_DRAIN_SECS`、`ShutdownDrainAdvisory`、`shutdown_drain_advisory()`，并在 `Config::validate` 中加入唯一一条硬校验（超过 1800 s 拒启）；同时承载 5 个把边界语义钉死的单测。

```
/// k8s 默认的 `terminationGracePeriodSeconds`。运维没有声明真实值时按它假设：
/// `shutdown_drain_secs` 大于等于 grace period 就没给随后的在途 drain 留时间，
/// pod 会在 drain 跑完前被 SIGKILL —— 这正是 drain 想避免的结果。
pub const K8S_DEFAULT_GRACE_SECS: u64 = 30;
```

```
/// `shutdown_drain_secs` 的上限，由 [`Config::validate`] 强制。它按工作负载定尺寸，
/// 而不是按 k8s 默认的 grace period：一条流式 completion 可以让 router 忙上数分钟，
/// 因此不想被 termination 打断的部署会把 `terminationGracePeriodSeconds` 设到
/// 数十分钟，并配上等长的 drain。判断某个 drain 是否适配某个 grace period 是
/// [`shutdown_drain_advisory`] 的职责 —— 那只是建议，因为运维可以调高预算；
/// 这个常量是另一道更硬的闸门，用来拒绝「根本不是一个 drain」的值（多打一位数字，
/// 或者把秒和毫秒搞混），这些值任何 grace period 都救不回来。
pub const MAX_SHUTDOWN_DRAIN_SECS: u64 = 1800;
```

```
/// 一次 `shutdown_drain_secs` 在 grace period 下没给在途 drain 留空间时的告警载荷。
/// 被比较的两个值作为字段携带，调用方因此可以打静态消息 + 结构化字段，而不是把数字
/// 插值进消息文本里（那样日志聚合无法分组、去重）。
#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub struct ShutdownDrainAdvisory {
    pub shutdown_drain_secs: u64,
    /// drain 被拿来比较的预算。
    pub termination_grace_secs: u64,
    /// 该预算来自运维声明，还是来自 [`K8S_DEFAULT_GRACE_SECS`]：
    /// 假设出来的预算让告警只是猜测，声明过的预算才让它成为事实。
```

```

    pub grace_declared: bool,
}

/// 对「drain 没给在途 drain 留空间」给出建议。注意这不是硬错误：drain 可能本来就是对的，
/// 该调高的是 grace period。边界取 `>=` 而不是 `>`，因为 drain 正好等于 grace period 时，
/// grace period 已经被它全部吃掉。
pub fn shutdown_drain_advisory(
    shutdown_drain_secs: u64,
    termination_grace_secs: Option<u64>,
) -> Option<ShutdownDrainAdvisory> {
    // router 读不到自己的 pod spec，所以 `None` 只能退回 k8s 默认值；
    // 这也解释了为什么「声明真实 grace period」才是让告警安静的正确方式，
    // 而不是把一个本来正确的 drain 调小。
    let grace = termination_grace_secs.unwrap_or(K8S_DEFAULT_GRACE_SECS);
    (shutdown_drain_secs >= grace).then_some(ShutdownDrainAdvisory {
        shutdown_drain_secs,
        termination_grace_secs: grace,
        grace_declared: termination_grace_secs.is_some(),
    })
}

impl Config {
    /// 校验类型系统和 `clap` 兜不住的不变量；由 [`cli::Cli::into_config`]
    /// 在组装完 `Config` 之后调用。
    pub(crate) fn validate(&self) -> Result<()> {
        // ..... model id、bucket config、sampling overrides 的既有校验略 .....
        // 本 PR 新增的唯一硬闸门：超过上限直接拒绝启动，因为那样的值会让之后
        // 每一次 termination 都变成 SIGKILL；错误文本同时点名 flag 与补救动作。
        if self.server.shutdown_drain_secs > MAX_SHUTDOWN_DRAIN_SECS {
            return Err(anyhow!(
                "shutdown_drain_secs must be at most {MAX_SHUTDOWN_DRAIN_SECS} (got {}); \
                past the ceiling a value is a typo rather than a drain, and the pod \
                would be SIGKILLED long before the pause elapsed. A long but deliberate \
                drain is fine — declare --termination-grace-secs so startup can check \
                it against the pod's real budget",
                self.server.shutdown_drain_secs,
            ));
        }
        // ..... discovery、proxy、active_load 的既有校验略 .....
        Ok(())
    }
}

```

experimental/sgl-router/src/config/types.rs

定义配置契约：ServerConfig 新增 shutdown_drain_secs 与 termination_grace_secs，新增 shutdown_drain() 把秒 → Duration 的转换固定进库内，并抽出 default_host() / default_port() / default_shutdown_drain_secs() 与 Default impl，供独立 crate 的测试 fixture 使用。

```

#[derive(Debug, Clone)]
pub struct ServerConfig {
    pub host: String,
    pub port: u16,
    /// SIGTERM 之后继续服务（同时 `/readyz` 翻成 503）多少秒，然后 HTTP server
    /// 停止 accept。默认值同时覆盖两条摘除路径：pod 打上 `deletionTimestamp` 之后
    /// 端点摘除到达 kube-proxy，以及由探针驱动的负载均衡器（它必须等到
    /// `failureThreshold * periodSeconds` 次 `/readyz` 失败才会动作）。
    ///
    /// 注意它等于 k8s 默认的 `terminationGracePeriodSeconds`，所以没有调高 grace
    /// period 的 pod 在暂停之后什么都没留给在途 drain，启动时会一直告警。这是有意
    /// 为之而不是默认值配错：completion 会流式跑数分钟的 router 本来就无法在 30 s
    /// 内干净退出，该调高的是 grace period。0 表示关闭暂停。
    pub shutdown_drain_secs: u64,
    /// pod 真实的 `terminationGracePeriodSeconds`，由运维声明。router 读不到自己的
    /// pod spec，所以没有这个值时启动检查只能对着 k8s 默认值比较——对一个故意调高
    /// grace period 的部署会误报。`None` 表示「按默认值假设」。
    pub termination_grace_secs: Option<u64>,
}

impl ServerConfig {
    /// 把 [`Self::shutdown_drain_secs`] 转成 `Duration`。转换放在库里（有测试钉住），
    /// 而不是放在 `main.rs` 的调用点：那里一个 `from_secs` / `from_millis` 的手误
    /// 会把每次 drain 静默缩短 1000 倍，而且没有任何东西会失败。
    pub fn shutdown_drain(&self) -> std::time::Duration {
        std::time::Duration::from_secs(self.shutdown_drain_secs)
    }
}

pub fn default_host() -> String {
    "127.0.0.1".into()
}

pub fn default_port() -> u16 {
    30000
}

pub fn default_shutdown_drain_secs() -> u64 {
    30
}

/// 存在的理由：让测试 fixture 只写出自己关心的字段（`tests/` 是独立 crate，
/// `#[cfg(test)]` 构造器够不到集成测试 fixture）。`Cli::into_config` 保持穷举，
/// 这样将来新增字段时，生产路径上仍然被迫做出决定。
impl Default for ServerConfig {
    fn default() -> Self {
        Self {
            host: default_host(),
            port: default_port(),
        }
    }
}

```

```

        shutdown_drain_secs: default_shutdown_drain_secs(),
        termination_grace_secs: None,
    }
}
}

```

experimental/sgl-router/src/config/cli.rs

用户可见的配置入口：新增两个 flag 并接入 `Cli::into_config`，`host / port` 的默认值同时改走 `default_*` (函数以消除默认值重复定义；help 文本承载了 `failureThresholdPeriodSeconds`、`0` 关闭暂停、需要声明真实 `grace period` 等运维语义。

```

pub struct Cli {
    // --- server ---
    /// HTTP server 绑定的地址。
    #[arg(long, default_value_t = default_host())]
    pub host: String,
    /// HTTP server 绑定的端口。
    #[arg(long, default_value_t = default_port())]
    pub port: u16,
    /// SIGTERM 之后、server 停止 accept 之前继续服务多少秒（其间 `readyz` 返回
    /// 503），好让端点摘除先到达 kube-proxy。要给随后的在途 drain 在
    /// terminationGracePeriodSeconds 之下留出空间。如果你依赖 readiness 探针
    /// （而不是 pod 删除）来摘除，就要把它设得大于 failureThreshold * periodSeconds。
    /// 默认值等于 k8s 默认的 terminationGracePeriodSeconds，因此在没有调高 grace
    /// period 的 pod 上启动会一直告警——用 --termination-grace-secs 声明真实预算，
    /// 让检查用上它。0 表示关闭暂停。
    #[arg(long, default_value_t = default_shutdown_drain_secs())]
    pub shutdown_drain_secs: u64,
    /// pod 的 terminationGracePeriodSeconds（如果你把它从 k8s 默认的 30 调高了）。
    /// 只用于启动时检查 --shutdown-drain-secs 是否为在途 drain 留出空间：router
    /// 读不到自己的 pod spec，没有它就只能对着默认值告警，而一个故意设长的 drain
    /// 没有办法说明自己是安全的。
    #[arg(long)]
    pub termination_grace_secs: Option<u64>,

    // --- model（恰好一个）---
    // …… 其余 flag 略 ……
}

impl Cli {
    /// 把 flag 解析成一份经过校验的 [`Config`]。
    pub fn into_config(&self) -> Result<Config> {
        // …… discovery、policy、bucket、sampling 的既有解析略 ……
        let config = Config {
            server: ServerConfig {
                host: self.host.clone(),
                port: self.port,
                // 两个新字段在这里落地；保持字段穷举，未来加字段仍会编译报错，
                // 从而强制在生产路径上做出决定。
            }
        }
    }
}

```

```

        shutdown_drain_secs: self.shutdown_drain_secs,
        termination_grace_secs: self.termination_grace_secs,
    },
    // ..... observability / model / discovery / proxy / active_load 略 .....
};
// 上限硬闸门在这里触发：超过 MAX_SHUTDOWN_DRAIN_SECS 直接拒绝启动。
config.validate()?;
Ok(config)
}
}

#[cfg(test)]
mod tests {
    /// 多个取值而不是只有默认值：任何 clamp 或缩放都能骗过单值断言。
    #[test]
    fn shutdown_drain_secs_maps_into_config() {
        for secs in ["0", "17", "1800"] {
            // ..... 解析 --shutdown-drain-secs 并断言原值透传 .....
            // 同时断言 Duration accessor 与配置的秒数一致。
        }
    }
}
}

```

评论区精华

本 PR 的 review 记录非常单薄：`comments_count = 0`、`review_comments_count = 0`，唯一信号是 sherlockwu 的一次无正文 APPROVED，没有任何行内评论与争议线程。因此讨论的「精华」只能来自作者在 PR body 与代码注释中的自我论证：

「the default equals the k8s default `terminationGracePeriodSeconds` ... That is the intended reading rather than a misconfigured default」——默认值必然告警是刻意设计，不是默认值配错。

「Whether a given drain fits a given grace period is the advisory's job, because the operator can raise the budget; the ceiling is the separate hard gate for a value that is not a drain at all」——软建议与硬闸门的分工。

「a message carrying the configured seconds cannot be grouped or deduped by a log aggregator」——为什么 advisory 返回结构体字段而不是拼好的字符串。

「where a `from_secs/from_millis` slip would silently shorten every drain by a factor of 1000 with nothing to fail」——为什么把秒 → `Duration` 的转换收进库内。

需要点明的缺口：上述四条都属于作者单方面论证，没有第三方复核；尤其「默认 drain 等于默认 grace，导致每个未声明 grace 的部署每次启动都告警」这一会影响全部部署的取舍，以及 1800 s 上限、`>=` 边界这两个数值选择，都没有 reviewer 提出替代方案或反对意见。

风险与影响

风险	具体位置	严重程度	说明
flag 当前是 no-op	src/config/cli.rs → 无读取点	中	若 #39016 延期而镜像先发布，运维按 <code>--help</code> 会以为 drain 已生效，实际仍是立即关闭，滚动升级期间继续丢流式 completion
常态化启动告警	shutdown_drain_advisory + 默认 30 s	中	未声明 <code>--termination-grace-secs</code> 的部署每次启动都告警，可能稀释其它启动告警； <code>grace_declared</code> 只能帮助分类，不能消除噪声
上限不挡 drain > grace	MAX_SHUTDOWN_DRAIN_SECS vs advisory	中	<code>--shutdown-drain-secs 600</code> + grace 仍为 30 的配置可以正常启动，之后每次 termination 都会在 drain 中被 SIGKILL
校验入口唯一	Config::validate 只由 Cli::into_config 调用	低	未来引入配置文件或程序化构造 Config 的路径时可能绕过上限检查（当前仓库内没有其它构造点）
公开结构体加字段	ServerConfig 是 pub	低 - 中	下游以字面量构造会编译失败，属源码级 breaking；仓库内靠公开 Default + fixture 补 .Default::default() 化解
CI 证据不完整	PR Test (Extra) Run #35066732674 失败	低	与 PR body 的「fmt / clippy / test 全绿」声明不一致，且无解释；鉴于无运行期改动，疑为 flaky / 环境问题
commit 信息失真	commit f313a5bf	低	message 写「default 5」而实现是 30，历史记录与实际默认值不符

影响层面：对运维本 PR 只是把「调参对象」确定下来，行为仍由 #39016 交付；对系统不触碰 worker 选择、KV 路由、PD 转发等热路径，无吞吐 / 延迟影响；对团队，ServerConfig 是散布在测试中的公共构造点，本次一次性改 15+ 个 fixture 才让编译通过，说明新增字段的 plumbing 成本真实存在，而本次引入的公开 Default impl 会显著降低后续同类改动成本。

关联脉络

- 提交栈内部：#39014（在途 HTTP 计数）提供「何时可以退出」的判据，#39015（本 PR）提供配置契约，#39016 提供 drain 行为与 k8s e2e 证明（原 #39017 已折入并关闭）。三者必须一起评估：单独看本 PR，无法回答「默认 30 s 是否真的够用」。
- 同一作者的 sgl-router 配置栈：#39000 / #39001（fleet-wide sampling contract 1/3、2/3）与本次结构高度同构——同样在 config/{types,cli,mod}.rs 三个文件上「先落配置面与启动校验，再落请求路径行为」，同样把 Config::validate 当作值级不变量的集中闸门。这已经是该模块稳定的演进范式。

- Router 基础设施同期演进：#39004（按 worker 解析转发协议）与本 PR 都触碰 workers/manager.rs 的测试构造点；#39001 也修改了本 PR 动过的 tests/proxy/pd_protocol_binding.rs。这说明 router 的测试 fixture 与配置构造点是跨 PR 的共享热区，ServerConfig 这类公共结构体加字段会稳定地外溢成多文件改动。
- 方向判断：sgl-router 正从「能路由」转向「可运维」——采样契约、转发协议协商、KV 分层缓存路由、优雅关闭排空依次补齐，本次是优雅关闭这条线上把配置契约与校验先立住的一步。