Postgres 连接池器

李睿远

Jul 07, 2026

在高并发 Web 应用中，PostgreSQL 的每个后端进程通常会占用约 10 MB 内存，这意味着当并发连接数快速上升时，数据库服务器的内存占用和上下文切换成本会急剧增加。短连接风暴在 Serverless、Lambda 函数或 PHP-FPM 场景中尤为明显，因为每次请求都需要建立新的 TCP 连接并完成 TLS 握手，导致延迟显著上升。连接池的核心价值在于将有限的数据库连接进行复用，从而降低建立连接的开销、提升整体吞吐量，同时减少数据库服务器的资源消耗。本文聚焦 PgBouncer、Odyssey、ProxySQL 和 PgCat 等主流连接池器，探讨它们在生产环境中的选型与运维策略。

1　连接池核心原理

连接池按照作用范围可划分为三种模式：会话池、事务池和语句池。会话池在客户端整个会话期间保持连接，适合需要维护会话级状态的场景；事务池则在每个事务结束后立即归还连接，能最大程度提高复用率，但会丢弃会话级设置；语句池进一步将每个 SQL 语句独立管理，适合极简查询场景。关键指标包括 `max_client_conn` 和 `default_pool_size`，前者限制客户端最大并发连接数，后者控制连接池实际持有的后端连接数量。连接池还需要维护 Server Liveness 检测、DNS 缓存以及 TLS 握手状态，以保证后端连接的可用性。

内存与 CPU 模型因实现语言和并发机制而异。PgBouncer 采用 C 语言配合 epoll 事件循环，单进程即可处理数万连接，内存占用极低；Odyssey 使用协程模型，在保持 C 语言性能的同时提供了更灵活的调度能力。连接池必须妥善处理一致性问题，例如 Prepared Statements 的生命周期、GUC 参数的覆盖范围，以及 Listen/Notify 通知机制。事务级特性与会话级特性存在明显差异，前者无法保留临时表和会话级配置，而后者则需要更长的连接持有时间。

2　主流连接池器横向评测

PgBouncer 是最成熟的连接池实现，使用 C 语言和 epoll 模型，对 PostgreSQL 协议兼容度极高，支持完整的连接多路复用，运维复杂度低且获得广泛的云厂商支持。Odyssey 同样基于 C 语言，但引入协程机制，在保持高性能的同时提供了更低的延迟，适合对延迟敏感的遗留应用。ProxySQL 最初面向 MySQL 设计，对 PostgreSQL 的协议兼容度相对有限，但其读写分离和分片能力非常强大，适合需要复杂路由逻辑的场景。PgCat 采用 Rust 语言和 Tokio 异步运行时，具备现代语言的安全性优势，在读写分离和分片方面表现出色，但社区仍处于快速发展阶段。Supabase Pooler 基于 PgBouncer 二次开发，针对 Serverless 场景进行了优化，部署和运维成本都很低。

在协议兼容度方面，PgBouncer、Odyssey 和 PgCat 都能完整支持 PostgreSQL 协议，而 ProxySQL 由于历史原因对 PostgreSQL 的支持相对薄弱。连接多路复用是所有主流连接池器的共同特性，但 ProxySQL 在这

方面的实现较为保守。读写分离和分片能力是 ProxySQL 和 PgCat 的强项，而 PgBouncer 通常需要配合外部组件实现类似功能。运维复杂度上，PgBouncer 和 PgCat 配置相对简单，Odyssey 次之，ProxySQL 由于功能丰富而配置项较多。

3 场景化选型决策树

当应用完全基于 PostgreSQL 且追求极致 QPS 时，PgBouncer 的事务池模式是首选，它能在高并发下保持极低的资源占用。对于需要读写分离和查询缓存的场景，ProxySQL 能提供最直接的支持，其内置的路由规则引擎可以灵活处理复杂的分库分表需求。在容器化和云原生环境中，PgCat 的 Rust 实现提供了更好的内存安全性和并发性能，与 Kubernetes 等编排平台集成也更加自然。Serverless 或 FaaS 架构下，云厂商提供的托管连接池如 Supabase、Neon 或 Crunchy 的方案能消除运维负担。遗留的 PHP 或 Java 单体应用如果对延迟要求较高，可以考虑 Odyssey 的会话池模式，它能在保持会话状态的同时降低连接建立开销。

4 生产部署 checklist

容量规划需要遵循明确的公式。`max_client_conn` 应设置为 Web 服务器线程数的两倍再加一定缓冲，而 `default_pool_size` 则取 CPU 核心数的两倍与 `max_connections` 减去预留连接数中的较小值。PgBouncer 的典型配置包括将 `pool_mode` 设为 `transaction`，`server_lifetime` 设置为 3600 秒以定期刷新后端连接，并通过 `ignore_startup_parameters` 忽略 `extra_float_digits` 等不必要的启动参数。PostgreSQL 的 `max_connections` 需要设置为连接池大小乘以节点数再加 20，以留出管理连接的空间。
高可用部署通常采用双 PgBouncer 实例配合 VIP 或 HAProxy 实现负载均衡，当主实例故障时可快速切换。滚动重启策略要求先逐步降低 `default_pool_size`，待活跃连接数下降后再执行 reload 操作，避免连接被强制断开。可观测性方面，PgBouncer 可通过暴露 Prometheus `/metrics` 端点来采集连接数、等待队列长度和查询耗时等指标，Grafana 看板可以直观展示连接命中率、排队延迟和 TLS 握手时间的变化趋势。安全层面必须强制使用 TLS 1.2 或更高版本，并启用证书双向校验，同时将连接池与数据库置于不同 VPC 安全组，仅开放 6432 端口。

5 真实案例与压测数据

某电商平台在大促期间通过部署 8 组 PgBouncer 事务池配合 3 节点主从架构，将 QPS 从 8000 提升至 35000，p99 延迟从 120 毫秒降至 45 毫秒，同时将数据库内存占用从 64 GB 压缩至 16 GB。压测数据显示，在 500 个直连连接与 50 个池化连接的对比中，连接池方案的吞吐量提升了近 4 倍，上下文切换次数减少约 70%。实际部署中遇到 Prepared Statements 跨事务失效的问题，通过将 `server_prep_stmts` 设置为 `false` 得以解决；LISTEN/NOTIFY 通知被吞噬的情况则通过为特定功能创建独立的会话池子池来规避。

6 未来演进与云原生趋势

基于 QUIC 协议的无状态连接池正在成为新方向，Crunchy 和 Neon 等厂商已在探索这一技术以进一步降低连接建立延迟。eBPF 技术为连接池提供了更细粒度的可观测性能力，Cilium Service Mesh 的 sidecar 模式可以将连接池逻辑与应用解耦。连接池即 PaaS 的理念正在兴起，Knative 等 Serverless 框架结合

Scale-to-Zero 能力，可以在无流量时完全释放连接资源。

7 结论与行动清单

立即可以采取三项行动。首先通过 pg_stat_activity 视图统计当前数据库的峰值连接数，为后续容量规划提供数据支撑。其次部署 PgBouncer 事务池进行 A/B 测试，对比直连和池化方案的性能差异。最后在 Grafana 中新建连接池健康看板，持续监控关键指标以便及时发现异常。