**PG内核分析**

hap

# 1 PG内核分析概述

* 1. **PG是一种先进的对象——关系型数据库**

PG不仅支持关系数据库的各种功能，而且还具备类、继承等对象数据库的特征。它是一种以关系数据库模型和SQL为基础，扩展了抽象数据类型，从而具备了面向对象特征的数据库。

PG不是完全的对象数据库，而是综合了关系型数据库的基础上，吸收了对象数据库的优点而发展起来的。

面向对象数据库强调与高级程序设计语言的无缝衔接，即用高级语言编写了程序，可以不需要或基本不需要改动，就将它作用于数据库，所以也有人把面向对象数据库理解为语言的持久化。由于实现了无缝衔接，是的面向对象数据库能支持非常复杂的数据模型。面向对象数据库同样也支持类、方法、继承等面向对象的概念。

面向对象数据库和关系型数据库优势互补，形成了对象关系型数据库，成为目前数据库发展的主流。

* 1. **PG的特性**

1. 完善的SQL标准支持，支持SQL92等标准。
2. 支持多版本并发控制（MVCC），时间点恢复。
3. 支持表空间机制、异步复制、嵌套事务、在线/热备份。
4. 有一个复杂的查询优化器。
5. 预写日志容错技术
6. 国际字符集，多字节字符编码，Unicode，并且对格式化、排序、大小写敏感提供本地化支持。
7. PG可以管理海量数据库，单机PG目前已出现具有管理超过4万亿字节数据能力的实用版本产品。
   1. **PostgreSQL的系统架构基础**

在数据库术语里，PostgreSQL使用一种客户端/服务器的模型（C/S模型）。一次PostgreSQL会话由下列相关的进程（程序）组成：

1. 一个服务器进程，它管理数据库文件、接受来自客户端应用与数据库的联接并且代表客户端在数据库上执行操作。 该数据库服务器程序叫做postgres。
2. 那些需要执行数据库操作的用户的客户端（前端）应用。 客户端应用可能本身就是多种多样的：可以是一个面向文本的工具， 也可以是一个图形界面的应用，或者是一个通过访问数据库来显示网页的网页服务器，或者是一个特制的数据库管理工具。 一些客户端应用是和 PostgreSQL发布一起提供的，但绝大部分是用户开发的。

和典型的客户端/服务器应用（C/S应用）一样，这些客户端和服务器可以在不同的主机上。 这时它们通过 TCP/IP 网络联接通讯。 你应该记住的是，在客户机上可以访问的文件未必能够在数据库服务器机器上访问（或者只能用不同的文件名进行访问）。

PostgreSQL服务器可以处理来自客户端的多个并发请求。 因此，它为每个连接启动（“forks”）一个新的进程。 从这个时候开始，客户端和新服务器进程就不再经过最初的 postgres进程的干涉进行通讯。 因此，主服务器进程总是在运行并等待着客户端联接， 而客户端和相关联的服务器进程则是起起停停

* 1. **PG代码逻辑结构**

PG源代码（14.1版本）有约2700个文件、190w行代码。

主要模块及用途如下：

Main：主程序，任何 Postgres 服务器进程都从此处开始执行。用于启动各种进程。

Bootstrap：--boot启动PG时用，创建初始的模板数据库。

函数调用：

main() -> AuxiliaryProcessMain()

Postmaster：守护进程。监听用户请求，控制postgres进程的启动和停止。

函数调用：

main() -> PostmasterMain()

Postgres: 服务器进程，管理数据库文件、接受来自客户端应用与数据库的联接并且代表客户端在数据库上执行操作。

函数调用：

PostmasterMain()

-> ServerLoop()

-> BackendStartup()

-> BackendRun()

-> PostgresMain()

-> InitPostgres()

Libpq: libpq是PostgreSQL的 C 应用程序接口。 libpq 是一套允许客户程序向PostgreSQL 服务器服务进程发送查询并且获得查询返回的库函数。 libpq同时也是其他几个PostgreSQL 应用接口下面的引擎，包括C++，Perl，Python，Tcl 和ECPG。

‘函数调用：

(pg\_rewind.c)main() -> init\_libpq\_source

parser:编译器

PostgresMain() -> exec\_parse\_message () -> pg\_parse\_query () -> raw\_parser()

optimizer:优化器

executor:执行器

Planer(查询规划模块):查询规划部分的在整个查询处理模块应该是在一个非常重要的地位上，这一步直接决定了查询的方式与路径，很大程度上影响了数据库查询的查询性能。因此这一块代码量也很大。

* 1. **PG代码文件夹结构**

src/backend/access ☆ 各种存储访问的方法，如hash、nbtree等

src/backend/bootstrap 数据库初始化时调用

src/backend/catalog 系统目录

src/backend/commands ☆ SQL命令

src/backend/common.mk

src/backend/executor 执行器相关代码。

src/backend/foreign （扩展）FDW[1] 相关的代码

src/backend/jit Just-In-Time Compilation，即时编译相关，提高查询性能

src/backend/lib 通用函数

src/backend/libpq c/cpp库函数，处理和客户端通信

src/backend/main 主程序

src/backend/Makefile

src/backend/nls.mk

src/backend/nodes 链表、结点等数据结构，和相关方法

src/backend/optimizer ☆ 优化器相关代码

src/backend/parser ☆ 编译器相关代码

src/backend/partitioning 分片相关代码

src/backend/po 语言文件配置

src/backend/port 平台兼容性处理

src/backend/postmaster postmaster进程和相关辅助进程

src/backend/regex 正则处理

src/backend/replication 有关流复制的相关代码（）

src/backend/rewrite 规则与视图相关的重写处理

src/backend/snowball 全文检索，语干处理

src/backend/statistics 收集统计信息相关，估算

src/backend/storage ☆ 管理各种类型存储系统的相关代码

src/backend/tcop ☆ 交通警察，pg服务进程的主要处理部分，查询流程调用的相关代码

src/backend/tsearch （扩展）全文检索

src/backend/utils 各种支持函数、初始化操作、内存管理、错误报告

[1] FDW:foreign-data wrapper,外部封装数据,PostgreSQL的数据库逻辑上是相互独立的，和Oracle类似，如果要访问其他数据库，需要做跨库操作。FDW使用户可以通过SQL访问没有存在数据库中的数据。

* 1. **PG查询流程梳理**

pg查询请求处理的流程为：

→ 解析SQL语句：经过解析器。 pg\_parse\_query()

→ 分析处理：pg\_analyze\_and\_rewrite()->parse\_analyze()

→ 语句重写：pg\_analyze\_and\_rewrite()->pg\_rewrite\_query()

→ 查询优化：经过优化器。查询优化器的入口点：planner()->standard\_planner()

→ 执行计划：经过执行器。

Postgres 接受用户提供的SQL，传入 exec\_simple\_query 函数。

exec\_simple\_query(query\_string)

Note：

→ : 分别的代码流程，在处理查询消息时，依次通过

-> : 纵深，代码一层层往下走

→ parsetree\_list = pg\_parse\_query(query\_string);

// 基本解析，生成语法树

// 对查询或查询进行基本解析（即使我们处于中止事务状态，这也应该是安全的！）

// 不涉及语义检查。只是做语法扫描

→querytree\_list = pg\_analyze\_and\_rewrite(parsetree, query\_string,

NULL, 0, NULL);

// 分析和重写

→pg\_plan\_queries()->pg\_plan\_query()

// 对于工具类语句[2]，只是包装一下，不走优化

// 可优化语句，使用 pg\_plan\_query(query,

// query\_string, cursorOptions, boundParams)

// （pg\_plan\_query 后面没有s的一个新函数）进行优化

->planner()

->standard\_planner() //标准查询优化处理

->subquery\_planner() // 输入一个查询树，返回一个计划树

// 写一个巨——大的结构体 PlannerInfo （计划树）

->grouping\_planner()

// 对集合进行处理

// 对非SPJ函数进行优化

// SPJ (SELECT， 投影（PROJECT）， 连接（JOIN）)

// 非SPJ（在SPJ的基础上存在GROUP BY

// 操作的查询，这是一种复杂的查询）

// 对SQL查询语句进行物理优化

[2] 工具类语句，语句类型：

CMD\_UNKNOWN， CMD\_SELECT， CMD\_UPDATE， CMD\_INSERT， CMD\_DELETE

CMD\_UTILITY， CMD\_NOTHING

枚举类型： CmdType

CMD\_UNKNOWN,

CMD\_SELECT, /\* select stmt \*/

CMD\_UPDATE, /\* update stmt \*/

CMD\_INSERT, /\* insert stmt \*/

CMD\_DELETE,

CMD\_UTILITY, /\* cmds like create, destroy, copy, vacuum,

\* etc. \*/

CMD\_NOTHING /\* dummy command for instead nothing rules

\* with qual \*/