Хост Postgres

Назначение: Обеспечивает работу с PostgreSQL через Oracle-совместимые интерфейсы. Предоставляет загрузку метаданных функций, пулы соединений, короткие и длинные транзакции, нотификации и утилиты генерации обёрток. Включает адаптеры, скрывающие различия между Oracle и PostgreSQL.

Пакет предназначен для работы в составе MorphCluster-приложения (например, Supervisor) и использует общие сервисы: `DbServices`, `RegistryHelper`, `GlobalServices` и т.д.

Точка входа (`src/index.mjs`)

```
export default {
  name: "Postgres",
  config,
  clients: {},
  services: {
    DbServices, PgFunctions, OraFunctions, PgPool, OraQueries,
    PgTransactions, PgTools, PgNotify, OraLongTransactions,
    OraQueriesPool, OLTTest, OraAdpTest
  }
}
```

Все сервисы регистрируются в `ServiceHost`. Они общаются через механизм зависимостей (`requirements`) и прямые вызовы методов друг друга.

Основные сервисы

1. PgFunctions

Файл: `src/pg-queries/pg-functions/index.mjs`

Загружает метаданные о всех функциях/процедурах из системных таблиц PostgreSQL и кэширует их. Используется другими сервисами для поиска функций по имени и пространству имён.

Зависимости: нет
Ключевые методы:

- `reload()` – перечитывает функции из БД (вызывается при старте).
- `find({namespace, name})` – поиск функции по схеме и имени.

- `findByName({name})` – поиск только по имени (менее точно).

2. OraFunctions

Файл: `src/ora-adapter/ora-functions/index.mjs`

Представляет функции PostgreSQL в «оракул-подобном» виде: пакеты и процедуры. Использует `PgFunctions` для получения сырых метаданных и конвертирует их в формат, привычный для Oracle-клиентов.

Зависимости: `PgFunctions`
Ключевые методы:

- `list()` – возвращает список всех схем (пакетов).
- `getPackage({packageName})` – список функций в схеме.
- `getFunc({packageName, funcName})` – детальное описание функции (параметры, типы, направления).
- `reload()` – обновляет данные, перезагружая `PgFunctions`.

3. PgPool

Файл: `src/pg-queries/pg-pool/index.mjs`

Пул «коротких» соединений с PostgreSQL. Каждый запрос выполняется в отдельной транзакции (BEGIN → COMMIT/ROLLBACK). Используется сервисами `OraQueries` для быстрых запросов.

Зависимости: `PgFunctions`, `RegistryHelper`
Управление через CommonRegistry:

- `pgQueryPool.connectionCount` – размер пула (по умолчанию 0 = отключен).

Методы:

- `execSql({sql, values, session})` – выполнить SQL.
- `execFunc({namespace, funcName, params, session})` – выполнить функцию.
- `reconnectAll()` – пересоздать все соединения.
- `state()` – текущий статус соединений (для `OraQueriesPool`).
- `reload()` – инициализация ссылок на служебные функции (`set_user_id`, `write_error_log`).

Особенности:

- Устанавливает контекст пользователя через `accounts.set_user_id`.
- Логирует ошибки в таблицу `logs.write_error_log`.
- Автоматически переподключается при обрывах (таймер).

4. PgTransactions

Файл: `src/pg-queries/pg-transactions/index.mjs`

Управляет длинными транзакциями. В отличие от `PgPool`, соединение удерживается открытым между вызовами, поддерживается ручное управление (BEGIN, выполнение нескольких запросов, COMMIT/ROLLBACK).

Зависимости: `RegistryHelper`, `DbServices`, `PgFunctions`
Управление через CommonRegistry:

- `PgTransactions.idleTimeout` – таймаут бездействия транзакции (мс).
- `PgTransactions.cleanTimeout` – через сколько очищать закрытые транзакции (мс).

Внутренний класс `PGTPool` содержит массив экземпляров `PGTransaction`. Каждая транзакция имеет жизненный цикл: `created → connecting → ready → executing → executed → fetching → closing → closed`.

Методы (внешние через `OraLongTransactions`):

- `create({session})` – создать новую транзакцию, возвращает `trxId`.
- `execFunc({trxId, namespace, funcName, params, session})` – выполнить функцию в транзакции.
- `execSql({trxId, sql, values, session})` – выполнить SQL.
- `getExecResult({trxId})` – получить результат после выполнения (блокирующий).
- `fetch({trxId, count})` – дочитать строки курсора.
- `commit({trxId})` / `rollback({trxId})` – завершить транзакцию.
- `poolStatus()` – состояние всех транзакций.

5. PgTools

Файл: `src/pg-queries/pg-tools/index.mjs`

Вспомогательный сервис, создающий функцию, возвращающую `TABLE(...)`, на основе существующей функции, возвращающей `REFCURSOR`. Упрощает миграцию с Oracle.

Методы:

- `funcTableFromRefcursor({funcFrom, funcTo})` – генерирует SQL-обёртку и создаёт её в БД.

6. PgNotify

Файл: `src/pg-queries/pg-notify/index.mjs`

Реализует механизм асинхронных уведомлений PostgreSQL (`LISTEN` / `NOTIFY`). Поддерживает подписку на каналы, переподключение и единое событие `onNotify`.

События:

- `onNotify` – генерируется при получении уведомления (полезная нагрузка парсится из JSON).

Методы:

- `subscribe({channelName})` / `unsubscribe({channelName})`.
- `isConnected()` – проверка соединения.
- `getSubscriptions()` – список активных подписок.

7. OraQueries

Файл: `src/ora-adapter/ora-queries/index.mjs`

Основной адаптер для выполнения «коротких» Oracle-подобных запросов. Принимает вызовы вида `package.func` с именованными параметрами, преобразует их через `DbServices` в реальные вызовы PostgreSQL и исполняет через `PgPool`.

Зависимости: `PgPool`, `DbServices`
Методы:

- `exec({queryName, params, count, offset, session})` – универсальный вызов.
- `execFunc(...)` – явное указание пакета и функции.
- `execSql({sql, params, session})` – выполнение SQL с заменой именованных параметров `:param` на позиционные `$1`.
- `resetUser()` / `resetAllUsers()` – сброс сессионного контекста (не реализованы).

8. OraLongTransactions

Файл: `src/ora-adapter/ora-long-transactions/index.mjs`

Аналог `OraQueries`, но для длинных транзакций. Использует `PgTransactions` для удержания соединения. Поддерживает многошаговые сценарии: создать транзакцию, выполнить несколько запросов, получить результаты, зафиксировать.

Зависимости: `PgTransactions`, `DbServices`
Методы:

- `create({session})` – создать транзакцию (возвращает `oltId`).
- `execFunc({oltId, package, func, params, session})` / `execSql(...)`.
- `getExecResult({oltId})` – получить результат выполненного запроса.
- `fetch({oltId, count})` – получить строки из открытого курсора.
- `commit({oltId})` / `rollback({oltId})`.
- `poolStatus()` – состояние пула длинных транзакций.

9. OraQueriesPool

Файл: `src/ora-adapter/ora-queries-pool/index.mjs`

Административный интерфейс для просмотра состояния пула `PgPool`.

Зависимости: `PgPool`, `DbServices`
Методы:

- `state()` – возвращает детализацию по соединениям (количество, статусы).

Вспомогательные классы и утилиты

ArgumentParser

Распарсивает строку аргументов функции PostgreSQL (из `pg_get_function_arguments`) в структурированный вид: `{ name, type, isOut, hasDefault }`.

PgFormats

Отвечает за:

- Конвертацию типов полей при чтении из БД (числа, даты).
- Формирование SQL-вызова функции с именованными параметрами (`"arg" => $1`).
- Проверку кодов ошибок PostgreSQL (отделение логических ошибок от проблем соединения).

PgConnection

Управляет одним подключением для «коротких» запросов (`PgPool`). Выполняет SQL в транзакции (BEGIN → запрос → COMMIT/ROLLBACK). Обрабатывает refcursor, загружая данные из курсора.

PGTransaction

Класс одной длинной транзакции. Хранит состояние, управляет жизненным циклом, поддерживает выполнение SQL и функций, фетч курсора. Используется внутри `PGTPool`.

PGTPool

Менеджер пула `PGTransaction`. Создаёт транзакции по требованию, отслеживает таймауты, автоматически подчищает закрытые соединения, пишет ошибки в лог БД.

PgNotifyConnection

Подключение для `LISTEN`/`NOTIFY`. Инкапсулирует логику переподключения и подписки.

Конфигурация

Файл `config.mjs` содержит параметры по умолчанию:

- `NatsConnection.queue` = `"pgqueries"`, `timeout` = 60000
- `NatsPublisher.interval` = 60000

Остальные настройки берутся из `config.common` и `CommonRegistry`:

Параметр	Описание
`common.postgresUri`	Строка подключения к PostgreSQL
`common.timeZone`	Часовой пояс для сессий БД
`pgQueryPool.connectionCount`	Размер пула коротких запросов (в `CommonRegistry`)
`PgTransactions.idleTimeout`	Таймаут простоя транзакции (мс)
`PgTransactions.cleanTimeout`	Интервал очистки закрытых транзакций (мс)

Все сервисы получают конфигурацию через стандартный механизм `Config` и могут динамически обновляться через подписку на `CommonRegistry`.

Схема взаимодействия

1. **Загрузка метаданных** `PgFunctions` при старте загружает список всех функций PostgreSQL. `OraFunctions` на его основе строит «оракул-подобное» представление.
2. **Регистрация в DbServices** Сервис `DbServices` (из внешнего пакета) хранит каталог всех доступных вызовов (жёсткие и мягкие). `OraQueries` и `OraLongTransactions` используют `DbServices.getOraSql()` для генерации PL/SQL-блока с подстановкой параметров.
3. **Выполнение запросов**
   - **Короткие запросы**: `OraQueries.exec` → `DbServices.getOraSql` → формирование вызова → `PgPool.execFunc` → `PgConnection`.
   - **Длинные транзакции**: `OraLongTransactions.create` → `PgTransactions.create` (создаётся `PGTransaction`) → далее `execFunc/execSql` → по окончании `getExecResult` + `commit/rollback`.
4. **Нотификации** `PgNotify` слушает каналы PostgreSQL и генерирует событие `onNotify`, на которое могут подписываться другие сервисы.
5. **Администрирование** `OraQueriesPool` и метод `poolStatus` у длинных транзакций дают мониторинг соединений.

Примечания

- Для корректной работы необходима настройка `common.postgresUri` и наличие в БД схемы `carabimeta` с процедурами `set_user_id`, `write_error_log`.

Revision #4
Created 21 June 2026 20:05:30 by Admin
Updated 26 June 2026 15:00:35 by Admin