Базы данных

Лекция 16 – Интерфейс Postgres

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2023

2023.12.08

Оглавление

libpq — библиотека для языка C

libpq — это интерфейс Postgres для программирования приложений на языке C.

Библиотека libpq содержит набор функций, используя которые клиентские программы могут передавать запросы серверу Postgres Pro и принимать результаты этих запросов.

libpq также является базовым механизмом для других прикладных интерфейсов Postgres, в частности, для C++, Perl, Python и Tcl.

Примеры использования `libpq`, в том числе несколько завершённых приложений, можно найти в каталоге `src/test/examples` дистрибутива в исходных текстах.

Клиентские программы, которые используют `libpq`, должны включать заголовочный файл `libpq-fe.h` и должны компоноваться с библиотекой `libpq`.

Интерфейс для C++ обычно находится в `libpqxx`.

Функции управления подключением к базе данных

Прикладная программа может иметь несколько подключений к серверу, открытых одновременно. Это нужно для доступа к более чем одной базе данных.

Каждое соединение представляется объектом типа PGconn, который можно получить от функций:

```
Pqconnectdb;
PqconnectdbParams;
PQsetdbLogin.
```

В случае успеха эти функции возвращают ненулевой указатель на объект.

Прежде чем передавать запросы через объект подключения, следует вызвать функцию PQstatus для проверки возвращаемого значения в случае успешного подключения.

PqconnectdbParams – создаёт новое подключение к серверу баз данных

```
PGconn *PQconnectdbParams(const char *const *keywords,
                          const char *const *values,
                          int expand_dbname);
```

Функция открывает новое соединение с базой данных, используя параметры, содержащиеся в двух массивах, завершающихся символом NULL.

keywords – массив строк, каждая из которых представляет собой ключевое слово.

values – значение для каждого ключевого слова.

Набор параметров может быть расширен без изменения сигнатуры функции.

Ключевые слова

`host` – имя компьютера для подключения.

Если это имя начинается с косой черты, оно выбирает подключение через Unix-сокет, а не через TCP/IP, и задаёт имя каталога, содержащего файл сокета.

По умолчанию, если параметр `host` отсутствует или пуст, выполняется подключение к Unix-сокету в `/tmp`.

В системах, где Unix-сокеты не поддерживаются, по умолчанию выполняется подключение к `localhost`.

Может принимаеть разделённый запятыми список имён узлов. В данном случае имена будут перебираться по порядку.

Для пустых элементов списка применяется поведение по умолчанию.

`hostaddr` – числовой IP-адрес компьютера для подключения.

Он должен быть представлен в стандартном формате адресов IPv4, например, 172.28.40.9.

Есть поддержка IPv6, можно использовать и эти адреса.

Если в качестве этого параметра передана непустая строка, всегда используется связь по протоколу TCP/IP.

Использование `hostaddr` вместо `host` позволяет приложению избежать поиска на сервере имён, что может быть важно для приложений, имеющих ограничения по времени.

Тем не менее, имя компьютера требуется для некоторых методов аутентификации, а также для проверки полномочий на основе SSL-сертификатов.

Применяются следующие правила:

1) Если адрес `host` задаётся без `hostaddr`, осуществляется разрешение имени[1].

2) Если указан `hostaddr`, а `host` нет, значение `hostaddr` задает сетевой адрес сервера. Однако, если метод аутентификации требует наличия имени компьютера, попытка подключения завершится неудачей.

3) Если указаны как `host`, так и `hostaddr`, значение `hostaddr` задает сетевой адрес сервера, а значение `host` игнорируется. Если метод аутентификации его требует, оно будет использоваться в качестве имени компьютера.

Аутентификация может завершится неудачей, если `host` не является именем сервера с сетевым адресом `hostaddr`. Когда указывается и `host`, и `hostaddr`, соединение в файле паролей идентифицируется по значению `host`.

4) Если `hostaddr` является списком значений, разделенных запятыми, узлы будут перебираться по порядку.

5) Если не указаны ни имя компьютера, ни его адрес, `libpq` будет производить подключение, используя локальный Unix-сокет. В системах, не поддерживающих Unix-сокеты, будет попытка подключиться к `localhost`.

`port` – номер порта, к которому нужно подключаться на сервере.

Если в параметрах `host` или `hostaddr` задано несколько серверов, в данном параметре может задаваться через запятую список портов такой же длины, либо может указываться один номер порта для всех узлов.

Пустая строка или пустой элемент в списке через запятую воспринимается как номер порта по умолчанию, установленный при сборке PostgreSQL.

---

1) Есть нюансы при использовании PqconnectPoll()

dbname – имя базы данных.
По умолчанию оно совпадает с именем пользователя.

user – имя пользователя Postgres, используемое для подключения.
По умолчанию используется то же имя, которое имеет в операционной системе пользователь, от лица которого выполняется приложение.

password – пароль, используемый в случае, когда сервер требует аутентификации по паролю.

passfile – имя файла, в котором будут храниться пароли.
По умолчанию это `~/.pgpass` или `%APPDATA%\postgresql\pgpass.conf` в Microsoft Windows. Файл может отсутствовать.

connect_timeout – максимальное время ожидания подключения.
Задаётся десятичным целым числом.
При нуле, отрицательном или если не указано, ожидание будет бесконечным.
Минимальный допустимый тайм-аут равен 2 секундам, таким образом, значение 1 воспринимается как 2 секунды.
Этот тайм-аут применяется для каждого отдельного IP-адреса или имени сервера.
Если заданы адреса двух серверов и значение `connect_timeout` равно 5, тайм-аут при неудачной попытке подключения к каждому серверу произойдёт через 5 секунд, а общее время ожидания подключения может достигать 10 секунд.

client_encoding – параметр устанавливает конфигурационный параметр client_encoding для данного подключения.

В дополнение к значениям, которые принимает соответствующий параметр сервера, можно использовать значение auto. В этом случае правильная кодировка определяется на основе текущей локали на стороне клиента.

options – задаёт параметры командной строки, которые будут отправлены серверу при установлении соединения.

application_name – устанавливает значение для конфигурационного параметра application_name.

fallback_application_name – устанавливает альтернативное значение для конфигурационного параметра application_name.

Это значение будет использоваться, если для параметра application_name не было передано никакого значения с помощью параметров подключения или переменной системного окружения PGAPPNAME.

Задание альтернативного имени полезно для универсальных программ-утилит, которые желают установить имя приложения по умолчанию, но позволяют пользователю изменить его.

keepalives – управляет использованием сообщений keepalive протокола TCP на стороне клиента. Значение по умолчанию равно 1, что означает использование сообщений. Можно изменить его на 0, если эти сообщения не нужны.

keepalives_idle — управляет длительностью периода отсутствия активности, выраженного числом секунд, по истечении которого TCP должен отправить сообщение keepalive серверу.

При значении 0 действует системная величина.

Этот параметр игнорируется для соединений, установленных через Unix-сокеты, или если сообщения keepalive отключены.

Он поддерживается только в системах, воспринимающих параметр сокета TCP_KEEPIDLE или равнозначный. В других системах он не оказывает влияния.

keepalives_count — задаёт количество сообщений keepalive протокола TCP, которые могут быть потеряны, прежде чем соединение клиента с сервером будет признано неработающим.

Нулевое значение этого параметра указывает, что будет использоваться системное значение по умолчанию.

Этот параметр игнорируется для соединений, установленных через Unix-сокеты, или если сообщения keepalive отключены.

Он поддерживается только в системах, воспринимающих параметр сокета TCP_KEEPCNT или равнозначный; в других системах он не оказывает влияния.

sslmode – определяет, будет ли согласовываться с сервером защищённое SSL-соединение по протоколу TCP/IP, и если да, то в какой очередности.

Всего предусмотрено шесть режимов:

`disable` – следует пытаться установить только соединение без использования SSL;

`allow` – сначала следует попытаться установить соединение без использования SSL, но если попытка будет неудачной, нужно попытаться установить SSL-соединение;

`prefer` (по умолчанию) – сначала следует попытаться установить SSL-соединение, но если попытка будет неудачной, нужно попытаться установить соединение без использования SSL;

`verify-ca` – следует попытаться установить только SSL-соединение, при этом проконтролировать, чтобы сертификат сервера был выпущен доверенным центром сертификации (CA);

`require` – следует попытаться установить только SSL-соединение. Если присутствует файл корневого центра сертификации, то нужно верифицировать сертификат таким же способом, как будто был указан параметр verify-ca;

`verify-full` – следует попытаться установить только SSL-соединение, при этом проконтролировать, чтобы сертификат сервера был выпущен доверенным центром сертификации (CA) и чтобы имя запрошенного сервера соответствовало имени в сертификате;

!!! Замечание !!!

`sslmode` игнорируется при использовании Unix-сокетов. В случаях, когда Postgres скомпилирован без поддержки SSL, использование параметров `require`, `verify-ca` или `verify-full` приведёт к ошибке. Параметры `allow` и `prefer` будут приняты, но `libpq` не будет пытаться установить SSL-соединение.

`sslcompression` – если установлено значение 1 (по умолчанию), данные, пересылаемые через SSL-соединения, будут сжиматься. Если установлено значение 0, сжатие будет отключено.

Параметр игнорируется, если выполнено подключение без SSL, или если используемая версия OpenSSL не поддерживает его.

Сжатие требует процессорного времени, но может улучшить пропускную способность, если узким местом является сеть.

Отключение сжатия может улучшить время отклика и пропускную способность, если ограничивающим фактором является производительность CPU.

`sslcert` – указывает имя файла для SSL-сертификата клиента, заменяющего файл по умолчанию `~/.postgresql/postgresql.crt`. Этот параметр игнорируется, если SSL-подключение не выполнено.

`sslkey` – указывает местоположение секретного ключа, используемого для сертификата клиента.

Он может либо указывать имя файла, которое будет использоваться вместо имени по умолчанию `~/.postgresql/postgresql.key`, либо он может указывать ключ, полученный от внешнего криптомодуля (загружаемого модуля OpenSSL).

`sslrootcert` – указывает имя файла, содержащего SSL-сертификаты, выданные Центром сертификации (CA). Если файл существует, сертификат сервера будет проверен на предмет его подписания одним из этих центров. Имя по умолчанию – `~/.postgresql/root.crt`.

`sslcrl` – указывает имя файла, содержащего список отозванных серверных сертификатов (CRL) для SSL. Сертификаты, перечисленные в этом файле, если он существует, будут отвергаться при попытке установить подлинность сертификата сервера. Имя по умолчанию такое `~/.postgresql/root.crl`.

`requirepeer` – указывает имя пользователя операционной системы, предназначенное для сервера, например, `requirepeer=postgres`.

При создании подключения через Unix-сокет, если этот параметр установлен, клиент проверяет в самом начале процедуры подключения, что серверный процесс запущен от имени указанного пользователя и, если это не так, соединение аварийно прерывается с ошибкой.

Этот параметр можно использовать, чтобы обеспечить аутентификацию сервера, подобную той, которая доступна с помощью SSL-сертификатов при соединениях по протоколу TCP/IP.

`krbsrvname` – имя сервиса Kerberos, предназначенное для использования при аутентификации на основе GSSAPI. Оно должно соответствовать имени сервиса, указанному в конфигурации сервера, чтобы аутентификация на основе Kerberos прошла успешно.

`gsslib` – библиотека GSS, предназначенная для использования при аутентификации на основе GSSAPI.

В настоящее время это действует только в сборках для Windows, поддерживающих одновременно и GSSAPI, и SSPI. Значение gssapi в таких сборках позволяет указать, что libpq

должна использовать для аутентификации библиотеку GSSAPI, а не подразумеваемую по умолчанию SSPI.

`service` – имя сервиса, используемое для задания дополнительных параметров. Оно указывает имя сервиса в файле `pg_service.conf`, который содержит дополнительные параметры подключения. Это позволяет приложениям указывать только имя сервиса, поскольку параметры подключения могут поддерживаться централизованно.

`target_session_attrs` – если этот параметр равен `read-write`, по умолчанию будут приемлемы только подключения, допускающие транзакции на чтение/запись.

При успешном подключении будет отправлен запрос `SHOW transaction_read_only`; если он вернёт on, соединение будет закрыто.

Если в строке подключения указано несколько серверов, будут перебираться остальные серверы, как и при неудачной попытке подключения. Со значением по умолчанию (any) приемлемыми будут все подключения.

Pqconnectdb – создаёт новое подключение к серверу баз данных

```
PGconn *PQconnectdb(const char *conninfo);
```

Эта функция открывает новое соединение с базой данных, используя параметры, полученные из строки `conninfo`.

Передаваемая строка может быть пустой. В этом случае используются все параметры по умолчанию. Она также может содержать одно или более значений параметров, разделённых пробелами, или URI.

Пример:

```
host=localhost port=5432 dbname=mydb connect_timeout=10
```

PqsetdbLogin – создаёт новое подключение к серверу баз данных

```
PGconn *PQsetdbLogin(const char *pghost,
                     const char *pgport,
                     const char *pgoptions,
                     const char *pgtty,      // не используется в н.в.
                     const char *dbName,
                     const char *login,
                     const char *pwd);
```

Предшественница функции PQconnectdb( ) с фиксированным набором параметров.

Она имеет такую же функциональность, за исключением того, что непереданные параметры (NULL или пустая строка) всегда принимают значения по умолчанию.

Pqsetdb – создаёт новое подключение к серверу баз данных.

```
PGconn *PQsetdb(char *pghost,
                char *pgport,
                char *pgoptions,
                char *pgtty,
                char *dbName);
```

Это макрос. Он вызывает `PQsetdbLogin( )` с нулевыми указателями в качестве значений параметров `login` и `pwd`.

Обеспечивает обратную совместимость с очень старыми программами.

Подключение к серверу баз данных неблокирующим способом

```
PGconn *PQconnectStartParams(const char *const *keywords,
                             const char *const *values,
                             int expand_dbname);

PGconn *PQconnectStart(const char *conninfo);

PostgresPollingStatusType PQconnectPoll(PGconn *conn);
```

Эти функции используются для того, чтобы открыть подключение к серверу баз данных таким образом, чтобы вызывающий их поток исполнения не был заблокирован при выполнении удаленной операции ввода/вывода во время подключения.

Суть этого подхода в том, чтобы ожидание завершения операций ввода/вывода могло происходить в главном цикле приложения, а не в внутри функций `PQconnectdbParams( )` или `PQconnectdb( )`, с тем, чтобы приложение могло управлять этой операцией параллельно с другой работой.

С помощью функции `PQconnectStartParams( )` подключение к базе данных выполняется, используя параметры, взятые из массивов `keywords` и `values`.

С помощью функции PQconnectStart подключение к базе данных выполняется, используя параметры, взятые из строки `conninfo`.

Эти функции не блокируются до тех пор, пока выполняется ряд ограничений:

1) Параметр `hostaddr` должен использоваться так, чтобы для разрешения заданного имени не требовалось выполнять запросы DNS.

2) Если вызывается PQtrace, необходимо обеспечить, чтобы поток, в который выводится трассировочная информация, не заблокировался.

Детали в документации.

Pqconndefaults – возвращает значения по умолчанию для параметров подключения

```
PQconninfoOption *PQconndefaults(void);

typedef struct {
    char   *keyword;   // Ключевое слово для данного параметра
    char   *envvar;    // Имя альтернативной переменной окружения
    char   *compiled;  // Альтернативное значение по умолчанию,
                       // назначенное при компиляции
    char   *val;       // Текущее значение параметра или NULL
    char   *label;     // Обозначение этого поля в диалоге подключения
    char   *dispchar;  // Показывает, как отображать это поле
                       // в диалоге подключения. Значения следующие:
                       // ""  Отображать введённое значение "как есть"
                       // "*" Поле пароля – скрывать значение
                       // "D" Параметр отладки
    int     dispsize;  // Размер поля в символах для диалога
} PQconninfoOption;
```

Возвращает массив параметров подключения. Он может использоваться для определения всех возможных параметров PQconnectdb и их текущих значений по умолчанию.

Возвращаемое значение указывает на массив структур PQconninfoOption, который завершается элементом, имеющим нулевой указатель keyword.

Если выделить память не удалось, то возвращается нулевой указатель.

Pqconninfo – Возвращает параметры подключения действующего соединения

```
PQconninfoOption *PQconninfo(PGconn *conn);
```

Возвращает массив параметров подключения.

Используется для определения всех возможных параметров PQconnectdb и значений, которые были использованы для подключения к серверу.

Возвращаемое значение указывает на массив структур PQconninfoOption, который завершается элементом, имеющим нулевой указатель keyword.

Pqfinish – Закрывает соединение с сервером

```
void PQfinish(PGconn *conn);
```

Освобождает память, используемую объектом PGconn.

Приложение в любом случае должно вызвать PQfinish( ), чтобы освободить память, используемую объектом PGconn, даже если попытка подключения к серверу потерпела неудачу (как показывает PQstatus).

После того, как была вызвана функция PGfinish, указатель PGconn не должен использоваться повторно.

Pqreset – переустанавливает канал связи с сервером

```
void PQreset(PGconn *conn);
```

Эта функция закроет подключение к серверу, а потом попытается установить новое подключение, используя все те же параметры, которые использовались прежде. Это может быть полезным для восстановления после ошибки, если работающее соединение было разорвано.

Переустановка канала связи с сервером неблокирующим способом

`PQresetStart`
`PQresetPoll`

```
int PQresetStart(PGconn *conn);

PostgresPollingStatusType PQresetPoll(PGconn *conn);
```

Эти функции закроют подключение к серверу, после чего попытаются установить новое подключение, используя все те же параметры, которые использовались прежде.

Это может быть полезным для восстановления после ошибки, если работающее соединение оказалось потерянным.

Они отличаются от `PQreset` тем, что действуют неблокирующим способом.

На эти функции налагаются те же ограничения, что и на `PQconnectStartParams()`, `PQconnectStart()` и `PQconnectPoll()`.

Переустановка подключения `PQresetStart()` — если она возвратит 0, переустановка завершилась неудачно.

Если она возвратит 1, следует опросить результат переустановки, используя `PqresetPoll()`.

Детали в документации

Далее нужно организовать цикл ожидания, пока PQconnectPoll(conn) будет возвращать PGRES_POLLING_READING.

при последнем вызове возвращает, ожидайте, пока сокет не окажется готовым для чтения (это покажет функция select(), poll() или подобная системная функция). Затем снова вызовите PQconnectPoll(conn). Если же PQconnectPoll(conn) при последнем вызове возвратила PGRES_POLLING_WRITING, дождитесь готовности сокета к записи, а затем снова вызовите PQconnectPoll(conn). На первой итерации, то есть когда вы ещё не вызывали PQconnectPoll, реализуйте то же поведение, что и после получения PGRES_POLLING_WRITING. Продолжайте этот цикл, пока PQconnectPoll(conn) не выдаст значение PGRES_POLLING_FAILED, сигнализирующее об ошибке при установлении соединения, или PGRES_POLLING_OK, показывающее, что соединение установлено успешно.

В любое время в процессе подключения его состояние можно проверить, вызвав PQstatus. Если этот вызов возвратит CONNECTION_BAD, значит, процедура подключения завершилась сбоем; если вызов возвратит CONNECTION_OK, значит, соединение готово. Оба эти состояния можно определить на основе возвращаемого значения функции PQconnectPoll, описанной выше. Другие состояния могут также иметь место в течение (и только в течение) асинхронной процедуры подключения. Они показывают текущую стадию процедуры подключения и могут быть полезны, например, для предоставления обратной связи пользователю. Вот эти состояния: