Документация libpgprobackup

Документация libpgprobackup

libpgprobackup

libpgprobackup

libpgprobackup — библиотека с API для резервного копирования данных, восстановления из резервных копий, а также проверки и объединения резервных копий

Описание

Библиотека libpgprobackup содержит функции для резервного копирования, восстановления из резервных копий, а также проверки и объединения резервных копий. Предоставляемый API позволяет создать собственное приложение для резервного копирования и восстановления данных вместо использования утилиты командной строки pg_probackup.

Библиотека хранит резервные копии в файлах собственного формата.

Библиотека берёт на себя взаимодействие с сервером баз данных, обработку данных, кодирование и декодирование, а также создание и хранение метаданных для резервных копий.

Приложение, использующее библиотеку, должно предоставить ей доступ к хранилищу, такому как файловая система, хранилище S3 или лента. Однако за операции в файловой системе, такие как чтение и хранение резервных копий, а также хранение метаданных, отвечает приложение. Приложение взаимодействует с libpgprobackup следующим образом:

1. Приложение подготавливает экземпляр базы данных к резервному копированию и вызывает функцию libpgprobackup backup.
2. libpgprobackup преобразует файлы данных и файлы WAL в формат pg_probackup3 и передаёт их приложению.
3. Клиентское приложение отправляет данные в файл в целевом хранилище (файловая система, хранилище S3 или лента) и сохраняет метаданные резервной копии. Промежуточное хранилище не используется.
4. Для восстановления из резервной копии с помощью функции libpgprobackup restore, выполнения проверки целостности резервной копии с помощью функции validate или объединения резервных копий с помощью функции merge, приложение предоставляет библиотеке функции для получения данных и метаданных резервной копии.

libpgprobackup реализована на C++, но может использоваться в приложениях, написанных на различных языках программирования. Интеграция с приложениями на C/C++ должна вызывать наименьшие сложности. Библиотека использует соглашение о вызовах `extern "C"`.

Структуры и функции libprobackup объявлены в файле `probackup_lib.h`.

Функции

Библиотека libpgprobackup содержит функции, описанные ниже. При вызове функций библиотеки они принимают структуры с параметрами команд и структуру с указателями на функции, которые работают с файлами и метаданными.

`bool backup ( ConnectOptionsLib *conn_opt, BackupOptionsLib *backup_opt, MetadataProviderLib *metadata );`

Подключается к локальному или удалённому серверу и выполняет резервное копирование. Возвращает `true` в случае успеха и `false` в случае ошибки.

Аргументы:

- *conn_opt* — указатель на структуру ConnectOptionsLib, которая содержит параметры подключения к северу Postgres Pro, такие как `pgdatabase`, `pghost`, `pgport`, `pguser`, `password`.

- *backup_opt* — указатель на структуру BackupOptionsLib, которая содержит параметры резервного копирования, такие как режим резервного копирования или количество потоков.
- *metadata* — указатель на структуру MetadataProviderLib, которая предоставляет функции-обработчики для метаданных резервных копий и файловых операций.

bool restore (RestoreOptionsLib *restore_opt, MetadataProviderLib *metadata);

Восстанавливает данные из резервной копии с использованием параметров, переданных в соответствующих структурах, в локальную файловую систему. Возвращает true в случае успеха и false в случае ошибки.

Аргументы:

- *restore_opt* — указатель на структуру RestoreOptionsLib, содержащую параметры для восстановления из резервной копии.
- *metadata* — указатель на структуру MetadataProviderLib, которая предоставляет функции-обработчики для метаданных резервных копий и файловых операций.

bool validate (RestoreOptionsLib *restore_opt, MetadataProviderLib *metadata, bool withParents);

Проверяет, что все файлы, необходимые для восстановления кластера из резервной копии, присутствуют и не повреждены. Если параметр *withParents* установлен в значение true, также проверяются все резервные копии в цепочке родительских. Возвращает true в случае успеха и false в случае ошибки.

Аргументы:

- *restore_opt* — указатель на структуру RestoreOptionsLib, содержащую параметры для проверки целостности резервной копии.
- *metadata* — указатель на структуру MetadataProviderLib, которая предоставляет функции-обработчики для метаданных резервных копий и файловых операций.
- *withParents* — логическое значение, определяющее, необходимо ли также проверять родительские резервные копии.

bool merge (MergeOptionsLib *merge_opt, MetadataProviderLib *metadata, bool with_file_map);

Объединяет цепочку инкрементных резервных копий в одну полную. Во время объединения создаётся новая полная резервная копия, включающая все родительские. Если у родительских копий нет дополнительных зависимостей, они удаляются после успешного объединения. Возвращает true в случае успеха и false в случае ошибки.

Аргументы:

- *merge_opt* — указатель на структуру MergeOptionsLib, содержащую параметры для объединения резервных копий.
- *metadata* — указатель на структуру MetadataProviderLib, которая предоставляет функции-обработчики для метаданных резервных копий и файловых операций.
- *with_file_map* — включает создание файлов сопоставления в метаданных.

void set_probackup_logger (Logger *info*, Logger *warning*, Logger *error*, Logger *debug*);

Определяет три функции-обработчика записи в журнал, которые libpgprobackup будет использовать для вывода отладочных, информационных сообщений, предупреждений или сообщений об ошибках. Каждая из этих функций должна принимать строковый параметр с сообщением, передаваемым библиотекой.

Аргументы:

- *info* — указатель на функцию, которая будет обрабатывать информационные сообщения.
- *warning* — указатель на функцию, которая будет обрабатывать предупреждения.
- *error* — указатель на функцию, которая будет обрабатывать сообщения об ошибках.
- *debug* — указатель на функцию, которая будет обрабатывать отладочные сообщения.

`void get_api_info`

Возвращает структуру `ProBackupApiInfo`, содержащую информацию о версии API, компиляторе и операционной системе.

`void set_cancel`

Сигнализирует текущей команде начать плавное завершение. Команда завершит текущую операцию, выполнит необходимую очистку, переведёт резервную копию в статус `ERROR` и прекратит работу.

Структуры для работы с файлами

Библиотека получает обратную связь от приложения через структуру типа `MetadataProviderLib`, которая должна содержать указатели на функции, работающие с файлами и метаданными.

Используются обработчики из библиотеки для функций, переданных приложением. Указатели на функции передаются в библиотеку через структуру `MetadataProviderLib`. Операции чтения или записи файлов определяются в структурах `CSource` и `CSink`, соответственно.

Для обратной связи от приложения структуры библиотеки содержат указатель `void *thisPointer`, в котором приложение может хранить указатель на свою собственную функцию или экземпляр класса. Этот указатель передаётся в функции-обработчики при их вызове из библиотеки.

MetadataProviderLib

Указатель на эту структуру передаётся всем основным функциям libpgprobackup, таким как backup, restore, validate, и merge. Эта структура определяет методы для работы с данными резервных копий, для записи и чтения метаданных резервных копий, а также для получения списка резервных копий.

```
typedef struct
{
    CSinkArray *(*get_sinks_for_backup)(const char *backup_id, size_t nsinks, void
 *thisPointer);
    CSourceArray *(*get_sources_for_backup)(const char *backup_id, void *thisPointer);

    CSink *(*get_sink_for_backup_map)(const char *backup_id, void *thisPointer);
    CSource *(*get_source_for_backup_map)(const char *backup_id,
                                          void       *thisPointer);

    void (*register_backup)(PgproBackup *backup, void *ptr);
    PgproBackup *(*get_backup_by_id)(const char *backup_id, void *thisPointer);
    void (*free_backup)(PgproBackup *backup, void *thisPointer);
    char **(*list_backup_ids)(void *thisPointer);

    bool (*write_backup_status)(const char *backup_id, BackupStatus status,
                                void *thisPointer);

    void *thisPointer;
} MetadataProviderLib;
```

Здесь:

get_sinks_for_backup

Указатель на функцию, которая возвращает `CSinkArray` — массив указателей на структуру `CSink` (см. `CSource` **и** `CSink` для получения более подробной информации). Требуется для записи информации в резервную копию. *backup_id* содержит строку с идентификатором резервной копии. В *thisPointer* приложение получает указатель, который ранее был передан в библиотеку через структуру.

get_sources_for_backup

Указатель на функцию, которая возвращает `CSourceArray` — массив указателей на структуру `CSource` (см. `CSource` **и** `CSink` для получения более подробной информации). Требуется для чтения информации из резервной копии. *backup_id* содержит строку с идентификатором резервной копии. В *thisPointer* приложение получает указатель, который ранее был передан в библиотеку через структуру.

get_sink_for_backup_map

Указатель на функцию, которая возвращает указатель на структуру `CSink` (см. `CSource` **и** `CSink` для получения более подробной информации). Требуется для записи информации в резервную копию. *backup_id* содержит строку с идентификатором резервной копии. В *thisPointer* приложение получает указатель, который ранее был передан в библиотеку через структуру.

get_source_for_backup_map

Указатель на функцию, которая возвращает указатель на структуру `CSource` (см. `CSource` **и** `CSink` для получения более подробной информации). Требуется для чтения информации о карте резервной копии из её метаданных. *backup_id* содержит строку с идентификатором резервной копии. В *thisPointer* приложение получает указатель, который ранее был передан в библиотеку через структуру.

register_backup

Указатель на функцию, которая сохраняет метаданные резервной копии. Принимает структуру `PgproBackup`.

get_backup_by_id

Указатель на функцию, которая получает метаданные резервной копии, определяемой параметром *backup_id*.

free_backup

Освобождает память для структуры `PgproBackup`, возвращаемой функцией `get_backup_by_id`.

list_backup_ids

Возвращает список доступных идентификаторов резервных копий. Список содержит указатели на строки, заканчивающиеся на ноль. Последний указатель в списке также должен быть нулевым. Память для списка должна быть выделена с помощью функции языка C, такой как `malloc` или `strdup`, поскольку библиотека освобождает память с использованием функции `free`.

write_backup_status

Указатель на функцию, которая сохраняет статус резервной копии. Статус резервной копии обновляется отдельно от сохранения остальных метаданных резервной копии.

*void *thisPointer*

Указатель, который передаётся из библиотеки во все функции-обработчики.

`CSource` и `CSink`

Эти структуры необходимы для чтения и записи блоков данных в файл резервной копии. Массивы указателей на эти структуры должны возвращаться функциями get_sources_for_backup и

get_sinks_for_backup, соответственно. Каждая из этих структур фактически представляет собой файл резервной копии, который открыт соответственно для чтения или записи.

```
/* Support structure */
typedef struct
{
 unsigned char *ptr;
 size_t         len;
} c_buffer_t;

typedef struct
{
 c_buffer_t (*read_one)(void *thisPointer);
 ssize_t (*read)(char *buf, size_t size, void *thisPointer);
 void (*seek)(off_t offset, void *thisPointer);
 off_t (*get_offset)(void *thisPointer);
 void (*close)(void *thisPointer);
 void *thisPointer;
} CSource;

typedef struct
{
 /* Write one Pb structure. See @PbStructs. */
 size_t (*write_one)(uint8_t *buf, size_t size, void *thisPointer);
 ssize_t (*write)(char *buf, size_t size, void *thisPointer);
 /* Close this Sink. No more calls will be done. */
 void (*close)(void *thisPointer);
 void *thisPointer;
} CSink;

typedef struct
{
 size_t nsinks;
 CSink **array;
} CSinkArray;

typedef struct
{
 size_t nsources;
 CSource **array;
} CSourceArray;
```

В поле `thisPointer` приложение может передать в структуру указатель на структуру или класс приложения, который, например, содержит дескриптор открытого файла.

Функции из структуры `CSource` используются для чтения файла резервной копии, а функции из структуры `CSink` — для записи. Для этого каждая из этих структур в настоящее время имеет по две функции. Функции `read_one` и `write_one` выполняют операции низкого уровня с блоком данных. `write_one` сохраняет размер блока, в то время как `read_one` сначала читает размер блока, а затем сам блок данных этого размера.

Функции `read` и `write` используются для упрощённой реализации чтения/записи. Их аргументы аналогичны аргументам общих функций для работы с файлами. Обработка размера блока выполняется внутри библиотеки. Эти функции должны возвращать размер прочитанных/записанных данных или 1 в случае ошибки.

Функция `seek` необязательна и на данный момент работает только вместе с файлами сопоставления. Если реализация этой функции слишком затратна для вашей системы хранения, её можно опустить.

Функция `get_offset` возвращает текущую позицию чтения в выходном потоке данных (*приёмнике*).

Функция `close` закрывает файл.

PgproBackup

В этой структуре передаются метаданные резервной копии.

```
typedef struct
{
    BackupStatus     backup_status;
    BackupMode       backup_mode;      /* Mode - one of BACKUP_MODE_xxx */
    char            *backup_id;        /* Identifier of the backup. */
    char            *parent_backup_id; /* Identifier of the parent backup. */
    BackupTimeLineID tli;       /* timeline of start and stop backup lsns */
    BackupXLogRecPtr start_lsn; /* backup's starting transaction log location */
    BackupXLogRecPtr stop_lsn; /* backup's finishing transaction log location */
    BackupTimestampTz   start_time; /* UTC time of backup creation */
    BackupTransactionId minxid;     /* min Xid for the moment of backup start */
    BackupMultiXactId
            minmulti; /* min multixact for the moment of backup start */

    bool stream; /* Was this backup taken in stream mode? I.e. does it include
                  all needed WAL files? */
    bool  from_replica;     /* Was this backup taken from replica */
    char *primary_conninfo; /* Connection parameters of the backup in the format
                               suitable for recovery.conf */
    char *note;

    /* For compatibility check */
    uint32_t block_size;
    uint32_t wal_block_size;
    char    *program_version;
    int      server_version;

    size_t uncompressed_bytes; ///< Size of data and non-data files before
                               ///< compression is applied
    size_t data_bytes; ///< Size of data and non-data files after compression is
                       ///< applied
    CompressAlg compress_alg;
    int         compress_level;

    BackupTimestampTz
            end_time; /* the moment when backup was finished, or the moment
                       * when we realized that backup is broken */
    BackupTimestampTz
            end_validation_time; /* UTC time when validation finished */

    BackupSource backup_source;/* direct, base or pro backup method*/

    size_t wal_bytes;// not used in pb3
    BackupTimestampTz   recovery_time;
} PgproBackup;
```

После выполнения функции backup библиотека вызывает функцию-обработчик register_backup, в которую передаётся заполненная структура `PgproBackup`. Переданную информацию можно сохранять по своему усмотрению.

Для получения информации о существующей резервной копии библиотека использует функцию-обработчик get_backup_by_id. Из этой функции приложение должно вернуть указатель на

структуру `PgproBackup` с метаданными резервной копии, имеющей указанный идентификатор, или нулевой указатель в случае ошибки.

Чтобы освободить память для структуры `PgproBackup` приложение должно вызвать функцию-обработчик free_backup.

`ProBackupApiInfo`

Эта структура содержит следующую информацию о libpgprobackup API:

- Версия libpgprobackup:
  - Компоненты версии (основная, дополнительная, патч)
  - Версия в виде строки, выводимой пользователю
- Поддерживаемые алгоритмы сжатия
- Версия компилятора
- Версия операционной системы
- Тип сборки

```
typedef struct tag_ProBackupApiInfo {
 const int pbk_ver_major;
 const int pbk_ver_minor;
 const int pbk_ver_patch;

 /* User-visible probackup lib version. */
 const char *version;
 /* Used for dynamic linking */
 const int dynamic_api_version;
 /* List of supported compressions. */
 const char **supported_compressions;
 /* Compiler info */
 const char *compiler_info;
 /* OS info */
 const char *os_info;
 /* Release/Debug */
 const char *build_type;
} ProBackupApiInfo;
```

`dynamic_api_version` помогает предотвратить использование клиентами несовместимых версий библиотеки. `supported_compressions` — завершающийся нулём список поддерживаемых алгоритмов сжатия.

Структура выделяется статически — приложения не должны пытаться освободить её память.

Используйте get_api_info, чтобы получить указатель на эту структуру.

Общие параметры

В этом разделе перечисляются структуры, используемые для передачи параметров командам. Для получения более подробной информации обратитесь к заголовочному файлу библиотеки `probackup_lib.h`.

`connectOptionsLib`

Следующая структура определяет параметры для подключения к серверу Postgres Pro:

```
typedef struct connectOptionsLib
{
 /* The database name. */
```

```
 const char *pgdatabase;
 /* Name of host to connect to. */
 const char *pghost;
 /* Port number to connect to at the server host, or socket file name
  * extension for Unix-domain connections.*/
 const char *pgport;
 /* PostgreSQL user name to connect as. */
 const char *pguser;
 /* Password to be used if the server demands password authentication. */
 const char *password;
} ConnectOptionsLib;
```

backupOptionsLib

Ниже представлена структура, которая определяет параметры для создания резервной копии. На данный момент поддерживаются только режимы резервного копирования `FULL` и `DELTA`.

```
typedef struct backupOptionsLib
{
 /* Number of threads, if backup mode supports multithreading */
 int num_threads;
 /* Backup mode PAGE, PTRACK, DELTA, AUTO, FULL*/
 BackupMode backup_mode;
 /* Backup source DIRECT, BASE or PRO */
 BackupSource backup_source;
 /* For DIRECT source is required to set up PGDATA. It is not required for
  * other sources */
 const char *pgdata;
 /* Backup Id if you wants to use custom id, otherwise it will be generated a
  * unique id using datetime of creation */
 const char *backup_id;
 /* Id of parent backup. It is not required for FULL backup mode */
 const char *parent_backup_id;
 /* Name of replication slot, if you use custom slot created by
  * pg_create_physical_replication_slot(). Otherwise will be used auto
  * generated slot */
 const char *replication_slot;
 bool        create_slot;
 const char *backup_note;
 /* If true, then WAL will be sent in stream mode, otherwise in archive mode
  */
 bool stream_wal;
 /* Progress flag. If true progress will be logged */
 bool        progress;
 const char *external_dir_str;
 /* Verify check sums. It is available only if checksums turn on on
  * PostgresPro server */
 bool verify_checksums;
 /* Compression algorithm, that is used for sending data between PostgresPro
  * server and libpgprobackup */
 CompressAlg compress_alg;
 /* Compression level, that is used for sending data between PostgresPro
  * server and libpgprobackup */
 int compress_level;
 /* Wait timeout for WAL segment archiving */
 uint32_t archive_timeout;
 /* Number of backup segment files. */
 uint32_t num_segments;
 /* Should we send raw 8k pages, or package them into larger groups. */
```

```
 TransferMode transfer_mode;
 /* Number of threads which write into segments. */
 uint32_t num_write_threads;
 /* Number of threads used for backup validation. */
 uint32_t num_validate_threads;
} BackupOptionsLib;
```

restoreOptionsLib

Следующая структура определяет параметры для восстановления из резервной копии. Эти параметры также используются для проверки резервной копии:

```
typedef struct restoreOptionsLib
{
 /* Number of threads */
 uint32_t    num_threads;
 /* Path to pgdata for restoration */
 const char *pgdata;
 /* Backup ID for restoration */
 char       *backup_id;
 /* Progress flag. If true progress will be logged */
 bool        progress;
 /* Skip external directories */
 bool        skip_external_dirs;
 const char *external_mapping;
 const char *tablespace_mapping;
 const char *db_str_oids;
 const char *db_str_names;
 PartialRestoreType restoreTyp
 /* Should we verify postgres page checksumse */
 bool        verify_checksums;

 /* Restore command to write in postgresql.auto.conf */
 /* https://postgrespro.ru/docs/enterprise/16/runtime-config-wal#GUC-RESTORE-COMMAND */
 const char *config_content;
 /* Need recovery signal */
 bool        need_recovery_signal;
 /* Need standby signal */
 bool need_standby_signal;
 /* No synchronization */
 bool no_sync;
 /* Number of threads to write restored data */
 uint32_t num_write_threads;
 /* Number of threads used for validation */
 uint32_t num_validate_threads;
} RestoreOptionsLib;
```

mergeOptionsLib

Следующая структура определяет параметры для объединения резервных копий:

```
typedef struct mergeOptionsLib
{
 /* Number of threads */
 int         num_threads;
 /* Path to pgdata to restore to */
 const char *pgdata;
 /* Incremental backup ID for the merge */
 const char *backup_id;
```

```
    /* Target backup ID for the merge */
    const char *target_backup_id;
    /* Progress flag. If true progress will be logged */
    bool        progress;
    /* ID of the last increment */
    const char *merge_from_id;
    /* Time interval within which to merge backups */
    int         interval;
    /* Number of threads to write merged data */
    uint32_t    num_write_threads;
    /* Number of threads used for validation */
    uint32_t    num_validate_threads;
   } MergeOptionsLib;
```

Константы

BackupMode

```
   /*
    Backup Mode is how backup is taken.
    All DIFF modes require parent backup id passed in the BackupOptions.
    Parent backup id is ignored in the FULL mode.

    DIFF_AUTO returns selected mode in the metadata. Tentatively it prefers
    DIFF_PAGE if WAL summarization is available, if not it tries to do
    DIFF_PTRACK if ptrack is enabled and finally it falls back to DIFF_DELTA.
    In case when even DIFF_DELTA is not possible (no parent full backup exists)
    FULL backup is taken.
   */
   typedef enum BackupMode
   {
       BACKUP_MODE_INVALID = 0,
       BACKUP_MODE_DIFF_PAGE,   /* incremental page backup */
       BACKUP_MODE_DIFF_PTRACK, /* incremental page backup with ptrack system */
       BACKUP_MODE_DIFF_DELTA,  /* incremental page backup with lsn comparison */
       BACKUP_MODE_DIFF_AUTO, /* library selects diff backup mode automatically */
       BACKUP_MODE_FULL       /* full backup */
   } BackupMode;
```

CompressAlg

```
   /*
   * Compression mode which is used to transfer data between PG server and the client lib.
   * Default is NONE_COMPRESS.
   */
   typedef enum CompressAlg
   {
       NONE_COMPRESS = 0,
       ZLIB_COMPRESS,
       LZ4_COMPRESS,
       ZSTD_COMPRESS,
   } CompressAlg;
```

BackupSource

```
   /**
   Backup Source is a method used by the client to access PGDATA.
   */
   typedef enum
```

```
{
    /*
     * Direct access. The client reads data files directly.
     * Opens normal connection to execute PG_BACKUP_START/STOP.
     * Local file access. Multithreaded. No special PostgresPro edition
     * required.
     */
    BACKUP_SOURCE_DIRECT,
    /*
     * Uses pg_basebackup protocol.
     * Opens replication connection.
     * Remote file access. No special PostgresPro edition required.
     */
    BACKUP_SOURCE_BASE_BACKUP,
    /*
     * Uses pg_probackup protocol.
     * Opens replication connection.
     * Remote file access. Multithreaded. Only works with PostgresPro builds.
     * Supported PostgresPro versions start with ent-15.
     */
    BACKUP_SOURCE_PRO_BACKUP
} BackupSource;
```

Предметный указатель

L