ケースバイケースで活用したい

PostgreSQLの搭載機能 自動エンコーディング変換の使い方

本パートでは、PostgreSQLに搭載されている「自動エンコーディング変換」の仕組みと、アプリケーション開発や運用管理における日本語の取り扱いについて解説する。この自動エンコーディング変換機能を利用すると、負荷が高くなると言われている。そこで、後半では代表的なエンコーディング変換が行なわれるケースと無変換のケースとを検証し、その負荷を比較する。

ユニアデックス株式会社 米田健治 YONEDA, Kenji ／中川 浩 NAKAGAWA, Hiroshi

日本語の文字コード

コンピュータと文字化け

コンピュータはデータを数値として取り扱い、文字も数値と対応付けて処理する。例えば、キーボードから入力された「A」という文字は「65」という数値で処理される。英語がアルファベットという、ごく少数の文字の組み合わせで多くの単語を表現するのとは異なり、漢字は文字の種類そのものが非常に多い。日本工業規格（JIS）は、コンピュータで扱う文字集合としてJIS X 0201、JIS X 0208、JIS X 0212、JIS X 0213などを定め（**表1**）、これらの文字に数値を割り振った。このような数値を割り振られた文字を「符号化文字集合」と呼ぶ。

一般的にこれらは、単に文字コードと呼ぶことが多い。文字コードにはさまざまな種類があり、コンピュータや個々のアプリケーションによっても使用している文字コードが異なる可能性がある。つまり、自分のコンピュータで、ある特定の文字に割り振られている数値が、ほかのコンピュータで同じ文字に割り振られている保証はない。そのため複数のコンピュータ間でデータを送受信すると、送信側と異なる文字が受信側で表示されたり、受信側に該当文字がないために所定の文字に勝手に置き換えられたりといった、いわゆる「文字化け」という現象が発生する。

マルチバイト文字の符号化

コンピュータで扱う文字の基本となるASCII文字集合（ISO-646）は英語で使用する文字や制御記号を規定したもので、7ビットですべてのASCII文字と制御記号を表現している。現在ではASCII文字集合は8ビットに拡張され、ヨーロッパ諸国で使用される文字を定義した文字集合規格（ISO-8859）などが定められている。日本ではJIS X 0201（通称ANK）によって、増えた領域に片仮名文字（半角カナ）を定義している。

漢字は日常的に使用するものだけでも数千種類にも及ぶため、8ビットに拡張しただけでは足らず、複数バイトを使って符号化されている。このようなマルチバイト文字を符号化する方法として、JISコードに代表される「制御コード」によって文字の種類の切り換えを行なう方法がある。

JISコードは、ASCIIコードと漢字の1バイト目の数値の重複箇所に、後続する文字の種類を示すための「ESCシーケンス」と呼ばれる制御コードを挿入することで文字の種類を切り換える。そのほか、PostgreSQLで扱うシフトJIS、EUC_JPおよびUTF-8などの文字コードは、1バイト目あるいは後続バイトの先行ビットの数値を見ることで文字の種類を判別している。

PostgreSQLがサポートする文字コード

PostgreSQLでは、一般的にデータベースに格納する文字コードを「データベースエンコーディング」、クライアント環境の文字コードを「クライアントエンコーディング」と呼ぶ。

日本語を扱う場合はDBエンコーディングとして、EUC_JPおよびUTF_8を選択できる。クライアントエンコーディングとしては、上記に加えてSJIS（シフトJIS）を指定できる。

PostgreSQLが扱うSJISは、SHIFT-JIS＋Windows拡張文字でWindows-31J（cp932）に相当し、EUC_JPは日本語EUCの亜種でeucJP_

表1：JISで扱う文字集合

JIS規格番号	説明
ASCII（ISO-646）	ASCII文字集合。英語で使用する文字と制御記号を規定。当初、米国の国内標準として定められた後、国際標準化機構によって国際標準となる
JIS X 0201（JIS C 6220※）	ANK（ASCII文字集合と半角カナ）
JIS X 0208(JIS C 6226※)	JIS基本漢字（第一水準／第二水準）。使用頻度の高い第一水準と、低い第二水準に分割した。当時の漢字処理用途の主流が名簿／帳簿の処理であったことから、人名漢字や地名（都道府県／市町村）漢字は無条件で第一水準に選出されている
JIS X 0212	JIS補助漢字
JIS X 0213	JIS拡張漢字（第三水準／第四水準）。2000年に制定（JIS2000）され、その後2004年に改訂（JIS2004）

※情報処理分野「X」の電気分野「C」からの独立（1987年）にともない、JIS Xに改称

msに相当する。なお、バージョン8.3以降ではJIS X 0213:2004をサポートしており、UTF_8がこれに対応しているほか、EUC_JIS_2004およびSHIFT_JIS_2004といった新しいエンコーディングを追加し、相互変換を可能にしている。このため、より多くの環境に対応可能となった。

PostgreSQLの文字コードサポート

DBクラスタ作成時のエンコーディングの指定

DBクラスタ作成時（initdb）で指定したエンコーディング（**LIST1**）は、このとき同時に作成されるtemplate0およびtemplate1のDBエンコーディングとなる。template1はデフォルトのテンプレートDBとして使用されれるため、ここで指定したエンコーディングはDB作成時に上書きしないかぎり、その後に作成するDBのエンコーディングとなる（LIST1の②）。

DBエンコーディングの指定

DBエンコーディングは、LIST1の①のようにCREATE DATABASE指令のENCODING句で指定する。ENCODING句による指定がない場合はテンプレートDB（TEMPLATE句）のエンコーディングを引き継ぎ、さらにTEMPLATE句の指定もない場合はtemplate1データベースのエンコーディング、つまりinitdbで指定したエンコーディングが適用される（LIST1の②）。

なお、執筆時での最新メジャーバージョンであるPostgreSQL 8.4では、テンプレートDBと異なるエンコーディングのDBの作成は基本的にはエラーとなるように変更された。例外的に、template0をテンプレートDBとして指定する場合にのみ、異なるエンコーディングの指定が許される。したがって、LIST1の③のようにTEMPLATE template0を指定することで、initdbの指定とは異なるエンコーディングのDBを作成できる。

LIST1：initdbで指定したエンコーディング

```
$ initdb --encoding=EUC_JP
:
CREATE DATABASE utf_8_db ENCODING 'UTF_8';―――①
CREATE DATABASE euc_jp_db;―――②
CREATE DATABASE utf_8_db1 ENCODING 'UTF_8' TEMPLATE template0;―――③
CREATE DATABASE euc_jis_2004_db ENCODING 'euc_jis_2004' TEMPLATE template0;
```

LIST2：DBのエンコーディングを調べるには

```
postgres=# ¥l―――①
                                  List of databases
      Name       | Owner |   Encoding   | Collation | Ctype | Access privileges
-----------------+-------+--------------+-----------+-------+-------------------
 euc_jis_2004_db | gaw84 | EUC_JIS_2004 | C         | C     |
 euc_jp_db       | gaw84 | EUC_JP       | C         | C     |
 postgres        | gaw84 | EUC_JP       | C         | C     |
 template0       | gaw84 | EUC_JP       | C         | C     | =c/gaw84
                                                            : gaw84=CTc/gaw84
 template1       | gaw84 | EUC_JP       | C         | C     | =c/gaw84
                                                            : gaw84=CTc/gaw84
 utf_8_db        | gaw84 | UTF8         | C         | C     |
(6 rows)
postgres=# select datname, pg_encoding_to_char(encoding) from pg_database;―――②
    datname     | pg_encoding_to_char
----------------+---------------------
template1       | EUC_JP
template0       | EUC_JP
postgres        | EUC_JP
euc_jis_2004_db | EUC_JIS_2004
utf_8_db        | UTF8
euc_jp_db       | EUC_JP
(6 rows)
```

DBエンコーディングの確認方法

DBのエンコーディングを調べるには、システムカタログpg_databaseのencoding列を検索する。また、**LIST2**の①のように、psql -lまたは¥lコマンドなどでも調べることができる。pg_databaseのencoding列は、各文字コードに割り振られた内部的な番号であるため、LIST2の②のようにpg_encoding_to_char関数を使ってエンコーディングを示す文字列に変換する必要がある。そのほか、後述するpgAdminという管理ツールを使えば、より簡単にDBのエンコーディングを調べることができる。

なお、日本語文字データを格納するデータ型には、character型、character varying(n)型、text型の3種類があるが、日本語を扱ううえで特にデータ型の選択の基準になるような観点はない。

クライアントエンコーディングと自動エンコーディング変換

DBはさまざまなクライアント環境からアクセスされる。クライアント／サーバー双方が日本語をそのまま送受信すると文字化けが発生する可能性があるため、PostgreSQLのサーバープロセスは、DBとクライアント側のエンコーディングが異なる場合には自動的にエンコーディング変換を行なうことで文字化けを回避している。変換処理はサーバー側で実行され、サーバーからクライアントに送信するデータや、クライアントから送られてくるSQL指令文なども変換対象になる。

例えば、**図1**が示すようにクライアントが発行したSQL指令はそのままサーバーに送信される。サーバーは受信したSQL指令をDBエンコーディングに変換する。ここではWHERE句などに記述した定数やテーブルなどのオブジェクト名など、SQL指令文全体がコード変換の対象となる（図1の①）。一方、検索結果をクライアントに送信するときにも、データをクライアントエンコーディングに変換して文字化けを防いでいる（図1の②）。

クライアントエンコーディングの宣言

サーバープロセスが自動エンコーディング変換を実施する、あるいは実施が不要であることを判断するためには、クライアント側のエンコーディン

グをサーバーに対して宣言する必要がある。以下にその宣言方法を示す。

① postgresql.confのパラメータclient_encodingの指定

DBクラスタ全体に有効となる広域な宣言であり、この宣言は次の②または③の宣言によって上書きすることができる。

② 環境変数PGCLIENTENCODINGの指定

DBへの接続確立時に、クライアント環境で有効となる（上記①の指定を上書きする）宣言。

③ SET CLIENT_ENCODING指令の指定

現在のDB接続において有効となる（上記①、②の宣言を上書きする）宣言。

クライアント／DBエンコーディングの組み合わせ

クライアントとサーバーのエンコーディングの組み合わせは、**表2**のように定められており、標準ではこれ以外の組み合わせでDBにアクセスすることはできない。表2の赤字は、バージョン8.3以降でサポートされるものだ。なお、CREATE CONVERSION指令を実行して、標準で提供されていない変換関数を定義することもできる。

クライアントからDBへの接続要求があると、前述のルールに従ってクライアントエンコーディングが設定される。

例えば、postgresql.confで次のように設定すると、クライアントエンコーディングはsjisになる。

```
client_encoding = 'sjis'
```

しかし、sjisとeuc_jis_2004の組み合わせはサポートされていない（**図2**のケースA）。図2のケースBのように環境変数PGCLIENTENCODINGにeuc_jis_2004と変換可能なエンコーディングを指定すれば、DBに接続することができる。パラメータclient_encodingまたは環境変数PGCLIENTENCODINGによる指定がない場合は、DBエンコーディングと同じクライアントエンコーディングが仮定され、DB接続が可能となる。

クライアントエンコーディングは、接続後にset client_encoding指令を使用して変更することもできるが、もちろんこの場合も同様に表2の組み合わせに限定される（図2のケースC）。なお、reset client_encoding指令を実行することによって、DBへの接続確立時のクライアントエンコーディングに戻すことが可能である。

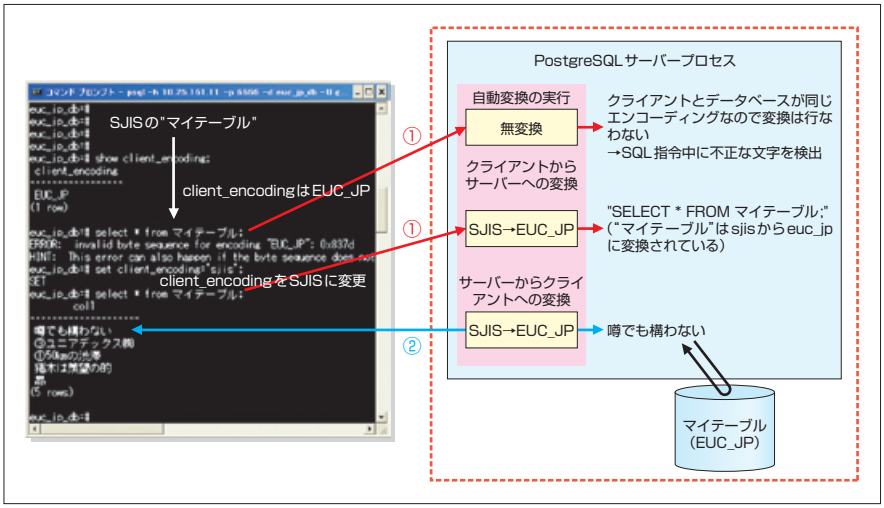


図1：自動エンコーディング変換

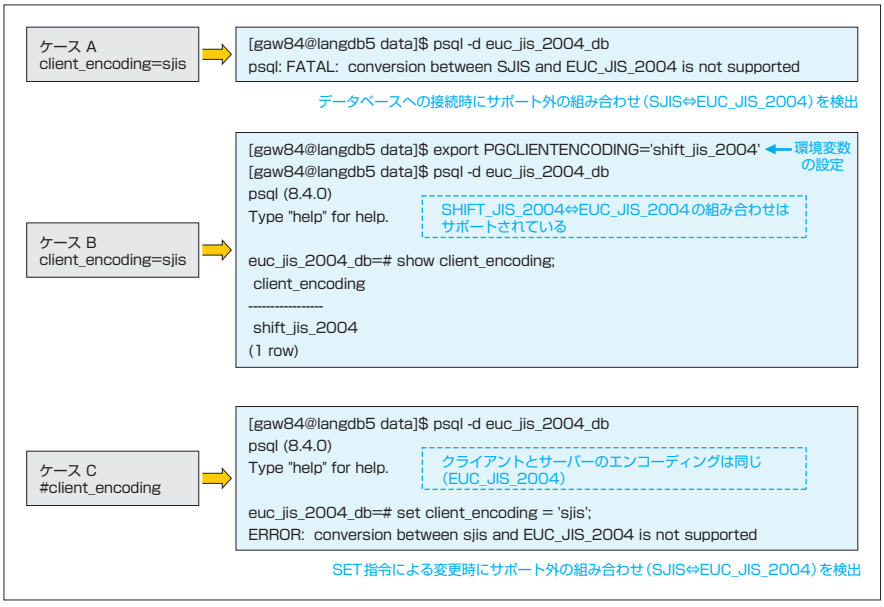


図2：postgresql.conf／環境変数／SET指令によるclient_encodingの設定

日本語の識別子を使用するときの注意点

PostgreSQLは、テーブル名やカラム名などにも日本語を使用できる。クライアント側のエンコーディングを適切に指定すれば、クライアントが実行

表2：C/Sのエンコーディングの組み合わせ

サーバーエンコーディング	クライアントエンコーディング
EUC_JP	EUC_JP、SJIS、UTF-8
EUC_JIS_2004	UTF-8、SHIFT_JIS_2004
UTF-8	EUC_JP、SJIS、SHIFT_JIS_2004、UTF-8

したテーブル定義文やテーブルにアクセスするためのSQL指令文はDBのエンコーディングに変換される。そのため、テーブルやカラムに正しくアクセスできる。逆に、テーブルへアクセスするときに定義時と異なるクライアントエンコーディングが宣言されている場合は、テーブル名やカラム名が正しく認識できなくなるなどの混乱が生じる。

また、識別子は最大長が決まっていて（レベル8では63バイト）、最大長を超えた分は後方の文字が切り取られる。日本語は必ずしも2バイトではなく、半角カタカナなど見た目の長さと異なる文字が多い。システムの要件やその他の事情により、万が一非常に長い日本語をテーブル名やカラム名として使用して論理設計しなければならなくなった場合は、DB物理設計の段階で、このことを気に留めておく必要がある。

各種インターフェイス

次に、PostgreSQLにアクセスするための各インターフェイスについて説明する。

libpq - C言語ライブラリ

libpqはC言語によるPostgreSQLインターフェイスであり、C++/Perl/Python/Tcl/ECPGなどのクライアントプログラムからの問い合わせを、バックエンドサーバーに渡し、その結果を受け取るためのライブラリ関数の集合である。

libpqにはライブラリ関数PQclientEncoding()、PQsetClientEncoding()が用意されており、クライアントエンコーディングの調査や設定が可能である。**図3**のlibpqの処理が示すように、これらの関数はそれぞれSHOW CLIENT_ENCODING指令やSET CLIENT_ENCODING指令に相当する。

ODBCドライバ

PostgreSQLの標準のODBCドライバとしてpsqlODBCが提供されている。psqlODBCはプリプロセッサによる条件文やプラットフォームの環境に応じてクライアントエンコーディングを決定し、

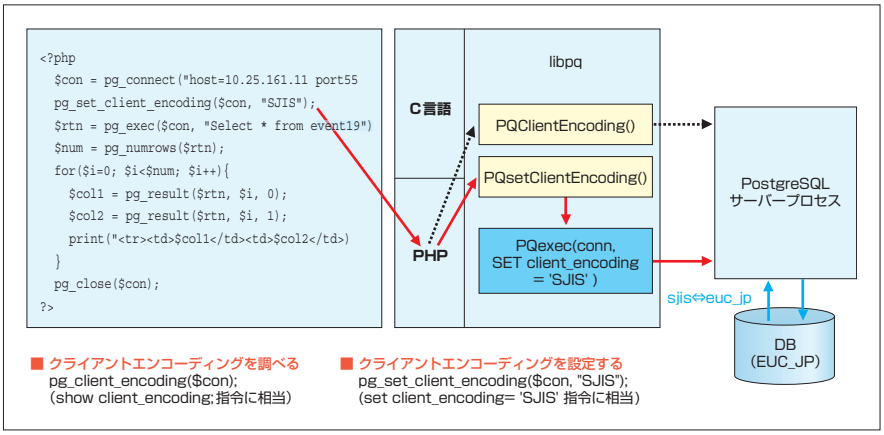


図3：PHP→libpq→PostgreSQLコマンド実行の流れ

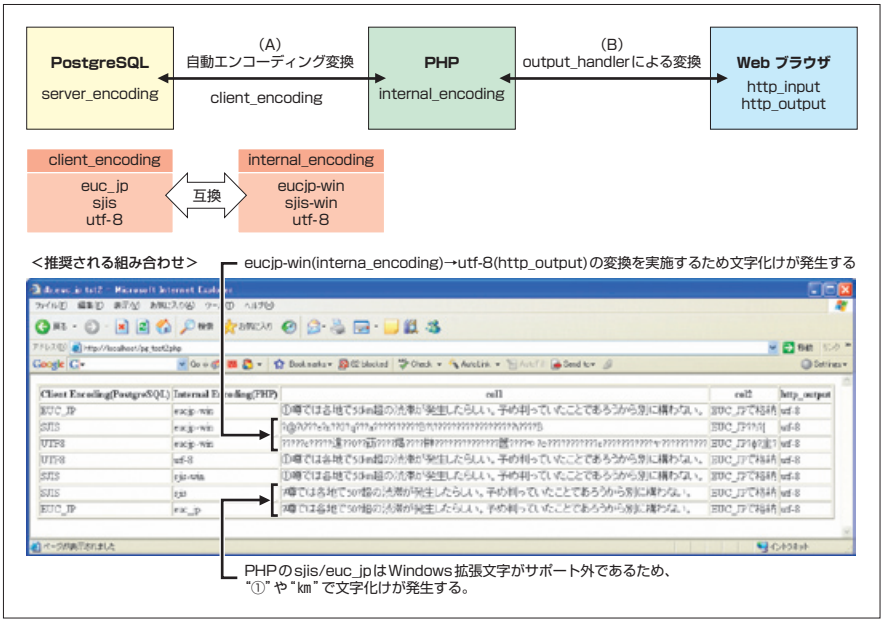


図4：client_encodingとinternal_encodingの組み合わせ

set client_encoding指令を実行する。

例えば、Windows環境用に提供されているpsqlODBC（バイナリ）であれば、実行環境に応じてSQL_ASCII、SJIS、BIG5などが指定される。接続後に明示的にset client_encoding指令を実行して変更することも可能である。

PHP

PHPは独自のコード変換の機能を備えている。つまり、PostgreSQLのサーバープロセスが変換して出力したデータ（**図4**の（A））が、クライアント（PHP）において別のロジックで再度コード変換される（図4の（B））。PHP変換機能の設定は任意だが、ここでは変換の実施を前提で説明する。

PHPはデータをWebブラウザに出力するときに、PHPの内部文字コード（以下、internal_encoding）からWebブラウザへの出力文字コード（以下、http_output）への変換を行なう。PostgreSQLがPHPに渡したデータは、PHPの内部ではinternal_encodingに設定されている文字

コードとして認識され、Webブラウザへ出力するときにしかるべき文字コードに変換されるため、PHPのinternal_encodingとPostgreSQLのclient_encodingは一致している必要がある。

図4はclient_encodingとinternal_encodingの組み合わせで、それぞれ日本語がどのように表示されるかを示している。不一致の場合、図4の結果が示すように不適切な変換を行なって文字化けが発生する。

互換性の観点から、PostgreSQLのエンコーディングEUC_JP、SJISおよびUTF-8は、それぞれPHPのeucJP_win、SJIS-winおよびUTF-8との対応が推奨される。なお、PHPのPostgreSQLインターフェイスも、libpqが提供しているAPIに対応している。PHPスクリプトからset client_encoding指令やshow client_encoding指令と同等の処理を行なうことができる(図3)。

日本語と文字列関数

データ型characterで扱える文字は、ASCII文字集合、漢字および絵文字まで多岐に渡り、かつその格納サイズはエンコーディングによってさまざまである。そのため、関数の戻り値や引数に指定される「長さ」や「位置」が文字数なのかバイト数なのかを意識して使用する必要がある。

戻り値として文字数を返す関数にはchar_length、positionなどがある。char_lengthは文字列の長さを、positionは任意の文字列の開始位置を文字数で返す。また、関数substringは開始位置と長さを文字数で指定し文字列を切り出す。一方、octet_length/bit_lengthは文字列のバイト数／ビット数を返す(**LIST3**)。

運用と文字コード

ここでは、PostgreSQLの運用管理に関連するコマンドや周辺ソフトウェアと文字コード、日本語の取り扱いについて解説する。

PostgreSQLを使用してシステムを構築する場合、そのDBの管理業務も必要となる。DB本体では、日本語データが自由に扱えていても、DBを管理するうえで日本語対応が不十分であれば、必要以上の手間暇がかかる可能性があるからである。例えば、テーブル名など日本語のDBオブジェクトを使用した場合、管理ツールにおいても日本語のテーブル名が正しく表示される必要がある。ここでは、運用管理に関するコマンド、周辺ソフトウェア、ツールなどと文字コードについて紹介する。

バックアップ／リストア

DBを運用するなかでバックアップとリストアは、障害からDBを守るために欠かせない作業である。PostgreSQLの標準コマンドでは、バックアップはpg_dumpまたはpg_dumpallコマンド、リストアはpg_restoreコマンドが用意されている。

文字コードとの関連で注目してみるとpg_dumpには、オプション-Eまたは--encodingに文字コードを指定することによって、データをダンプするときのエンコーディングを指定できる。多くの場合、データをバックアップしてリストアするPostgreSQLサーバーは同一のサーバーのことが多いが、DBの文字コードが異なるプラットフォームに移行するときに、このエンコーディングオプションが必要となる。

一方、リストアするためのpg_resotreコマンドには、エンコーディングを指定できない。そのため、リストア先が異なるエンコーディングのDBの場合、pg_dumpコマンドのオプションによって、あらかじめ目的の文字コードにしてDBごとにダンプしておく必要がある。

DBクラスタ全体をダンプするコマンドpg_dumpallには、エンコーディングのオプションを指定できない。理由は、グローバルオブジェクト(ロールとテーブル空間)もダンプする必要があり、それは初期構築のエンコーディングに依存しているからである。そのため、pg_dumpallを使用してクラスタ全体をエンコーディングを変えてリストアすることはできない。

pg_dumpでエンコーディングを指定する例

```
pg_dump --encoding=SJIS  jp_test  ⇒
> /home/pgsql/jp_test.dump
```

※誌面の都合により⇒で改行

インポート／エクスポート

PostgreSQLのDBにデータを論理的にインポートするコマンドがcopyである。

ほかのシステムからデータを使用して、Postgr

LIST3：octet_length/bit_lengthの戻り値

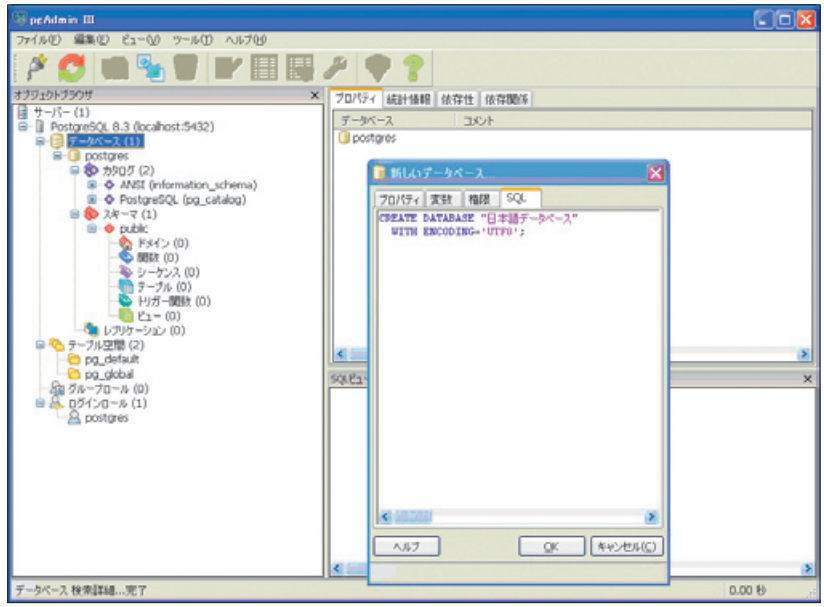
```
euc_jp_db=# select octet_length('データベース管理者(DBA)'),bit_length('データベース管理者(DBA)');
 octet_length | bit_length
--------------+------------
           23 |        184
(1 row)

euc_jp_db=# select * from mytable;
                                          col1                                          |     col2
----------------------------------------------------------------------------------------+--------------
 ①噂では各地で50km超の渋滞が発生したらしい。予め判っていたことであろうから別に構わない。  | EUC_JPで格納
(1 row)

euc_jp_db=# select substring(col1 from 1 for 10), char_length(col1), position('構' in col1) from mytable;
     substring      | char_length | position
--------------------+-------------+----------
 ①噂では各地で50km    |          45 |       41
(1 row)
```

LIST4：set client_encoding toの使用例

```
test=# set client_encoding to 'SJIS';
SET
test=# copy t1 from '/home/pgsql/sjisinput.txt';
COPY 5
test=# set client_encoding to 'UTF8';
SET
test=# select * from t1;
name
--------------------------
 あああああ
 いいいいい
 ううううう
 えええええ
 おおおおお
```



画面1：日本語名のDBに接続

eSQLのDBにデータを挿入する場合などは、テキストファイルを入力しcopyコマンドを使用する。

例えば、Excelで作成したデータをPostgreSQLのDBに大量にインサートする場合、いったんCSV形式のテキストファイルに出力した後でPostgreSQLに転送し、copyコマンドでDBにデータを流し込むということになる。

Excelで作ったデータの文字コードは、SJISであることが多いだろう。しかし、copyコマンドにエンコーディングを直接指定するオプションはない。psqlからcopyコマンドを実行するには、事前にset client_encoding toコマンドを実行し、入力ファイルのエンコーディング（この場合sjisを指定）して実行する必要がある。これにより、クライアントと異なるエンコーディングで作られた文字コードデータファイルを使用して、データをDBに書き込むことができる。set client_encoding toコマンドを使用すれば、テキストファイルを一般的な文字変換ツールで事前に変換しておく手間もなくなるというわけだ。

SJISのファイルのデータをLinuxプラットフォームのPostgreSQLにコピーする例を**LIST4**に示す。当然のことながら、クライアントのエンコーディングを指定しなかった場合、DBでは、期待どおりの文字コードに変換されない。

また、同様にpsqlから出力ファイルを指定することで、表データをエクスポート可能である。インポートと同様にset client_encoding toコマンドを使用してエンコーディングを変えれば良い。

管理ツール「pgAdmin」

PostgreSQLには、DB管理ツールとして「pgAdmin」がある。pgAdminは、GUIベースでDBを管理できる。Linux、MAC OS Xなどのプラットフォームで実行可能である。

pgAdminは、以下のURLよりダウンロードできる。

http://www.pgadmin.org/index.php

ここでは、原稿執筆時の最新版である1.10.0をダウンロードしてみる。以降では、Windowsにインストールしたpgadmin からLinux上のPostgreSQLのDB管理を前提に文字コード対応機能に絞って紹介する。

pgAdminの文字コード機能

pgAdminは、ツールのメニューなどはすべて日本語化されている。**画面1**は、pgAdminを使用して、日本語名のDBに接続した例である。

目的のDBをクリックすると、CREATE DATABASEコマンドがウィンドウに現われる。このことから、デフォルトでどんなオプションが指定され実行されたが分かる。そのため、実行された内部的なencodingの指定も確認できる。

これで分かるように、PostgreSQLは日本語名のオブジェクトが使用可能である。Linux上で作成した日本語オブジェクトを持つDBもpgAdminで正しく表示／操作可能だ。DBを管理するという観点では、非常に重宝するツールである。

また、pgAdminはDB名、テーブル名、カラム名など、すべて日本語で作成しても正しく表示できる。Linux上で直接表などを作成した場合は問題ないが、pgAdminからも日本語オブジェクトを作成できる。

しかし、気をつけなければならないのは、pgAdminから日本語オブジェクトを作成するときに意図せず後方に全角スペースが入ってしまう点である。実際には、その名前を使用してSELECTコマンドなどを実行しようとすると「オブジェクト名がみつかりません」と表示される。

全角スペースは、画面上では視認できないため、作成時に全角スペースが入ってしまったかどうか分かりにくい。実際に日本語オブジェクトを作成するときには「""」（ダブルコーテーション）で囲ったほうが全角スペースが含まれたかどうかが明確になる。なお、英数字も当然全角の英数字と半角の英数字は区別されるので、注意が必要だ。

よくPostgreSQLユーザーは、オブジェクトを管理するのにpgAdminではなくコンソール端末

から日本語を使用することも多い。そのため、VIEW表で日本語オブジェクト名を使用し、管理のための実表は英数字でオブジェクトを作成する方法をとることもある。ただし、現時点（メジャーバージョン8.4）でVIEW表は読み込み専用となり、VIEW表を通した更新はできない。

また、pgAdminから直接SQLを実行することもできる。メニューから［ツール］－［クエリーツール］を選択すると、SQLを書き込み実行するテキスト画面が現われる。メンテナンスなどちょっとしたデータの更新や検索ならばこの画面から実行できる。なお、このときのクライアントエンコーディングは、WindowsでのpgAdminであったとしても、UNICODEに設定されている（**画面2**）。

文字コードとソースコード

PostgreSQLを使用する理由として、オープンソースだからという点が挙げられる。ソースを読むことができるので、RDBMSの動きがホワイトボックスとなり理解しやすい。また、エラーや障害発生時には原因追究の手がかりとなる。

ただし、どのように読んで良いか検討がつかない方もいるだろう。PostgreSQL 8.4.1の場合、ソースコードは約12万行もある。いきなり全部を理解しようとせずに、読む（目に触れる）ソースコードの量を徐々に増やしていけば良いだろう。

ソースコードの読み方としては、次の2つの方法がある。

① SELECTコマンドが文法のパーシングからその結果を返すまでの概略を流れに応じて読んでいく
② ある機能や関数に特化して詳細部分を読み理解していく

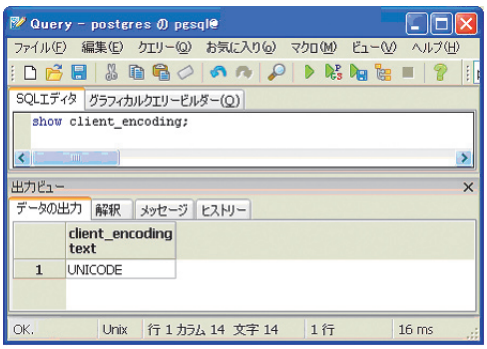
今回は、②の文字コードに関する関数（特に文字コードの変換部分）に注目して説明する。この説明を参考に、実際のソースコードを参照してほしい。なお、ここで取り上げているソースコードはバージョン8.4.0を基にしており、以降のLISTに示したソースコードは説明のため一部編集してある。

文字コードに関するソースコードを読む

文字コードを明示的にソースコード上で意識しなければならない箇所は、文字のエンコーディングの変換、関数LIKEなど文字を意識する必要がある処理、全文検索処理などである。

エンコーディングの変換処理はSQLの種類にかかわらず、クライアントとサーバーのエンコーディングが異なるときに必要となる。ここでは、文字コードの変換処理についてソースコードを俯瞰してみる。文字コード変換は、前述したようにサーバー側ですべて行なわれる。

文字コードのコンバージョン関連の関数は、backend/utils/mb/mbutils/cmbutils.cというファイルにまとめられている。このファイルに含まれている関数名を読んでいくだけでも、文字コードに関してどのような関数が実装されているのかを知ることができる。また、例えば“encoding”というキーワードでソースコードをgrepしても、いろいろな箇所で“encoding”の考慮が必要なことが分かる。



画面2：クライアントエンコーディングが「UNICODE」に設定されている

SetClientEncoding

クライアントのエンコーディングを設定する処理である。クライアントとサーバーのエンコーディングを調べて異なるようであれば、エンコーディングごとに用意されているコンバージョン関数を実際に使用できるように設定する。文字変換用の関数は、システムカタログpg_conversion表に登録される（**LIST5**）。SetClientEncoding関数の処理の中で、この表から対応するコンバージョン関数（列名conprocで保存されている）の名前を得る。

実際のコンバージョンのルーチンは、backend/utils/mb/conversion_procs/ディレクトリの下にある。

GetDatabaseEncoding

CREATE DATABASEで作成したDBのエンコーディングを知るための関数である。実際には、PostgreSQLがすでに実装しているエンコーディングリストを持つ構造体pg_enc2name_tblに対してインデックスが示す値を返すだけである。

InitializeClientEncoding

PostgreSQLにアクセスしたときにセッションを確立するためにpostgresプロセスが作られる。そのクライアントの初期化処理の一部として、クライアントのエンコーディングを設定する関数である（**LIST6**）。

pg_client_to_serverとpg_server_to_client

クライアントからサーバー、サーバーからクライアントのデータの受け渡しのために呼ばれる関数である。ここから関数perform_default_encoding_conversionを呼び出し、具体的なコンバージョンが行なわれる。ほかのRDBMSでは、クライアント側のドライバで変換することもあるが、PostgreSQLの場合はサーバー側で変換を行な

LIST5：pg_conversion表

```
test=# select * from pg_conversion where conname like '%sjis%' limit 10;
    conname     | connamespace | conowner | conforencoding | contoencoding |    conproc     | condefault
----------------+--------------+----------+----------------+---------------+----------------+------------
 euc_jp_to_sjis |           11 |       10 |              1 |            34 | euc_jp_to_sjis | t
 sjis_to_euc_jp |           11 |       10 |             34 |             1 | sjis_to_euc_jp | t
 sjis_to_mic    |           11 |       10 |             34 |             7 | sjis_to_mic    | t
 mic_to_sjis    |           11 |       10 |              7 |            34 | mic_to_sjis    | t
 sjis_to_utf8   |           11 |       10 |             34 |             6 | sjis_to_utf8   | t
 utf8_to_sjis   |           11 |       10 |              6 |            34 | utf8_to_sjis   | t
```

うことがLIST7のソースコードから分かる。

perform_default_encoding_conversion

入力と出力のエンコーディングを設定し、変換領域としてのメモリを用意して実際に変換を行なう。変換は、backend/utils/mb/conversion_procs/ディレクトリに配置されたCの関数を使用する。

pg_wcha.h

文字コードの考慮が必要なSQL関数が定義されたCプログラムのヘッダーファイルである。その関数名を読むだけでどんな関数がPostgreSQLの中で用意されているのかを知ることができる。

createdb

pg_char_to_encodingの中で-Eオプションで指定されたencodingをチェックし、保存する。

文字コード変換の評価

前項では、PostgreSQLのソースコードにみられる文字コードについて少し解説した。クライアントとサーバーのエンコーディングが異なっていれば、PostgreSQLが対応している変換処理によってデータの文字コード変換が行なわれることが分かっただろう。変換処理が必要であるということは、同じSELECTコマンドであっても処理すべきソースコードが増えるということである。

ここでは、無変換に比べてその変換処理がどれだけの負荷となるのかをPostgreSQLの一般的なベンチマークツール「pg_bench」を改変して、測定した結果を報告する。漠然と負荷が多少は増えるだろうということは分かっていても、それが無視できないほどの負荷であれば、クライアントとサーバーのエンコーディングを統一することが望ましくなるからである。

LIST6：InitializeClientEncoding

```
InitializeClientEncoding(void)
{    backend_startup_complete = true;
     if (SetClientEncoding(pending_client_encoding, true) < 0)
     ereport(FATAL,   ～)
}
```

LIST7：関数pg_server_to_clientの一部

```
pg_server_to_client(const char *s, int len)
{    if (len <= 0)
return (char *) s;
if (ClientEncoding->encoding == DatabaseEncoding->encoding ||
ClientEncoding->encoding == PG_SQL_ASCII ||
DatabaseEncoding->encoding == PG_SQL_ASCII)
return (char *) s;              /* assume data is valid */
return perform_default_encoding_conversion(s, len, false);
}
```

pg_benchの改変

今回、変換処理の負荷を測定するために改変したpg_benchのポイントを簡単に説明する。

pg_benchは、PostgreSQLユーザーにとって一般的な効率測定ツールで、contribにソースコードが内包され気軽に使用できるようになっている。

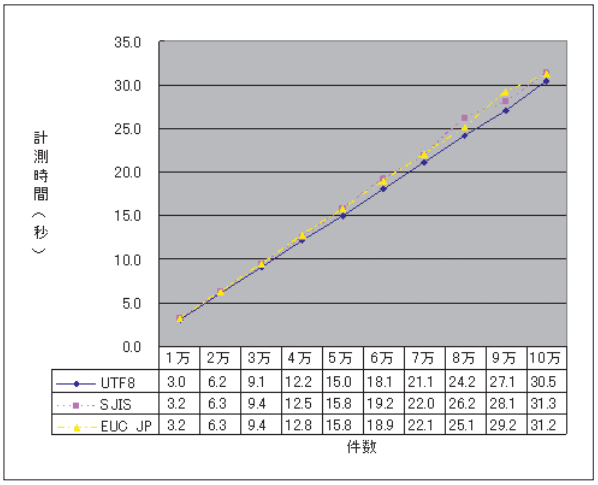
オリジナルのpg_benchはデータがASCIIであり、変換が必要となるデータを作り出さないため、エンコーディング処理が発生しない。そのため、日本語データを含むDBを作成し、サーバーと異なるエンコーディングでアクセスすることができるように改変した。

ソースコードは、contribにある1ファイルをmake実行するだけで、複数のファイルを修正する必要はない。PostgreSQL本体を改変するよりも簡単なので、本体のソースコードを修正し動かしたことがないようなCプログラミング初心者にとっても親しみやすいはずである。

改変した内容は、次のとおりである。

① サーバーのエンコーディングは、PostgreSQLで一般的なUTF-8で固定とする
② クライアントがSQLを実行する際に、SetClientEncodingコマンドを追加し、クライアントのエンコーディングを任意に変えられるようにする
③ 変換するための日本語データ用として保存する列を表定義に1列追加する
④ 初期DBを構築するCOPY文の元データに日本語データを追加して、DBに日本語データが含まれるようにする
⑤ 追加した日本語列を検索（ただし、クライアントで表示はしない）するようにSELECTコマンドの日本語列を修正する
⑥ エンコーディングを変えて測定するために、pg_benchの起動パラメータに-eを追加し、変換するエンコーディングを指定できるようにする
⑦ 変更できるエンコーディングは、代表的なUTF-8（無変換）、SJIS、EUC_JPの3種類で測定する

測定では、メモリでヒット確率を高くするようにわざとDBを小さく作った。DBの容量を大きくし、



	1万	2万	3万	4万	5万	6万	7万	8万	9万	10万
UTF8	3.0	6.2	9.1	12.2	15.0	18.1	21.1	24.2	27.1	30.5
SJIS	3.2	6.3	9.4	12.5	15.8	19.2	22.0	26.2	28.1	31.3
EUC_JP	3.2	6.3	9.4	12.8	15.8	18.9	22.1	25.1	29.2	31.2

画面3：測定結果

I/Oの回数が増えるとSQL処理の中でI/Oがボトルネックになるため、I/Oの負荷に文字コード変換によるCPU負荷の差がなくなると考えたからである。そこで、PostgreSQLを起動したときにリソースの使用率のなかでI/O waitを比較し、極端にI/O waitが増加しないようにDBの初期サイズを調整した。

測定は、pg_benchの検索オプションを使用した。検索だけ測定とすることで、DBを更新することによる内部的なメモリの排他制御の影響などを最小限にしたつもりである。

3種類のエンコーディング、それぞれの計測の開始ではPostgreSQLを立ち上げ直し、PostgreSQLが使用するメモリを初期化している。一般的にベンチマークを開始するときは、初期状態のDBのバックアップをリストアする必要があるが、今回は更新がないので、DBを毎回作り直したり、バックアップから戻したりすることは行なっていない。

測定結果

測定結果を、**画面3**に示す。これを見ると、やはり1回1回のエンコーディング処理による負荷は少なく、数多くのSQLを実行してもそれほど負荷の影響がないことが分かる。

積極的にエンコーディングを一致させる必要はないが、一致させると前述したような文字コード変換による留意点を考える必要がないのに加えて、今回のようなエンコーディングの負荷を減らせるメリットもあることが実証できた。理想を言えば、このように初期DBクラスタはプラットフォームのエンコーディング、そしてクライアントに応じたエンコーディングでCREATE DATABASEコマンドでDBを作成したい。

COPYコマンドなど、大量のI/Oを伴う処理におけるエンコーディングの負荷はほとんど無視できるであろう。しかし、OLTP系のレコード長で少ないレコード件数を頻繁にアクセスする場合などはI/O処理にCPU処理が隠れないため、負荷が多少なりとも顕在化しやすいと言える。

考察

筆者は、かつて「クライアント側のエンコーディングとサーバー側のエンコーディングは同じにしたほうが良い」という通説にこだわっていた時期があった。文字化けの心配がないし、高価なCPUを効率的に利用するためだ。

しかし、近年のシステムのDBアクセス形態は、直接検索はもちろんJavaやPHPといったさまざまな言語およびプラットフォームからアクセスされる。そのため、このようにクライアントとサーバーのエンコーディングを一致させるほうが難しい。とはいうものの、DB管理者ならC/S間のエンコーディング不一致によるシステムリソースへの影響を考えることは必要であろう。また、実際にはなかなか実証できなかったエンコーディングのリソースへの負荷を今回明らかにできた。

DBの物理設計では、当然、文字コードを考える必要がある。その場合の1つのTipsとして参考にしてほしい。

おわりに

本パートでは全般にわたって繰り返し触れてきたが、PostgreSQLのエンコーディング変換に関する考え方は極めてシンプルである。ただし、ODBCドライバやpgAdminといった外部ツールやPHPなど、PostgreSQLにとってはクライアント側にあるアプリケーションの動作と組み合わせたときに、より複雑さが生じることに気づいたはずだ。

こういった外部ツール群やソフトウェアは、プリプロセッサの条件文の影響を受けたり、パラメータの設定によって変換処理が制御されたりするため、ツールやソフトウェアの生成環境や実行環境によって動作が異なることもある。業務システムへの適用に際して日本語を使用する場合は、PostgreSQL本体よりもむしろ、連携する個々のソフトウェアの動作を確認することのほうが重要であることに留意してほしい。

最後に、本パートがPostgreSQLの日本語処理を検討されている方々の参考となれば幸いである。 DBM

米田健治（よねだけんじ）
ユニアデックス株式会社に勤務。入社以来、メインフレームからオープン系まで、一貫してDBのサポート業務に従事。最近は、流行のDWHアプライアンスサーバーにも参画。社内ではDBスペシャリスト育成のため、実践的なDB設計の講座も持っている。

中川 浩（なかがわひろし）
ユニアデックス株式会社に勤務。入社以来、メインフレーム系のデータベースの主管業務に携わり現在に至る。近年は、メインフレームに加え、OSSデータベースのサポートを主要な業務としている。