

# PostgreSQL

## Chapter 4

# Make Applications

各種アプリケーション  
を作成する

# PostgreSQL

## 4-0

アプリケーション  
作成の前に

本章では、PostgreSQL の各種プログラミングインターフェースを使ったアプリケーションをご紹介します。

本章は各種言語用のインターフェース別のパートに分かれています（表4.0.1）。各パートでは、PostgreSQL のプログラミングインターフェースの基本的な使い方がわかるような例題を取り上げ、アプリケーションの使い方からソースコードの内容まで詳しく解説します。また、例題アプリケーションの中で使わなかったインターフェース関数についても、別途できるだ

け解説するようにしました。これからPostgreSQL を使ってアプリケーションを開発しようとしての方のお役に立てば幸いです。

ここに取り上げたサンプルアプリケーションは、以下の環境で開発とテストを行いました<sup>注1</sup>。

表 4.0.1 本章で取り上げる言語インターフェース

節	言語	ページ
4.1	Tcl/Tk	p.129
4.2	C	p.161
4.3	PHP	p.175
4.4	Perl	p.215
4.5	Java	p.232

- ハードウェア：Apple PowerBook 2400c/180（メモリ80Mバイト）
- OS：LinuxPPC（Linux 2.1.24 『PowerPC Linux Release4 CD-ROM』からインストール）

また、FreeBSD 2.2.6-RELEASE でもできるだけ確認を行いました。Intel 版のLinuxでは確認を行っていませんが、筆者の経験ではLinuxPPCで動いたPostgreSQLのアプリケーションがIntel版のLinuxで動作しなかった、ということはほとんどありませんでした。

なお、今回取り上げたアプリケーションのソースコードや関連プログラムは、可能なかぎり付属CD-ROMに収録しています。これらの具体的なインストール方法については、各節の解説をご覧ください。

## 注 1

同じLinuxPPCでも、2.1.103 / 2.1.115 / 2.1.125のバージョンのカーネルではPostgreSQLを動かすとシステムがクラッシュします。これ以外のバージョンではどうか、またPowerBook 2400c以外の機種でもやはりクラッシュするかどうかなどについては確認していません。ちなみに、LinuxPPCとバイナリ互換性のあるMkLinuxでは、ほぼPostgreSQLは問題なく動くようで、筆者が知るかぎりトラブルは報告されていません。

## 4-1 Tcl/Tk インターフェースを使って Mail/News の全文検索システム

### 4.1.1 Tcl/Tk とは

Tcl/Tk<sup>注1</sup>は、Perl と並んで、今最も広く使われているスクリプト言語のひとつです。Tcl/Tk には以下のような特長があります。

注1  
「ティクル ティーケー」  
と読みます。

文法が簡単で覚えやすい

インタプリタ言語なのでコンパイルする必要がない。そのわりには動作が速く、十分実用的なアプリケーションを構築可能

容易にGUIを構築できる

処理系がコンパクトで移植性が高く、UNIX やWindows、それにMacでも動く  
(ただしlibpgtclの利用は原則UNIXのみ)

拡張が容易

フリーである

PostgreSQL にはlibpgtcl というTcl/Tk用のインターフェースが付属しています。libpgtcl はC言語で記述されており、内部的にlibpqを呼び出すことによってバックエンドとコミュニケーションします。つまり、libpgtcl はTcl/Tk との間を取り持つ存在であると言えます。

本節では、libpgtcl の利用方法とlibpgtclを使ったアプリケーションを紹介します。

## 4.1.2 Tcl/Tk のインストール

Tcl/Tk とひと口に言いますが、実際にはTclとTkは別々のパッケージです。Tclは言語インタプリタ本体を含む基本的なパッケージです。TkはTclの上に実装され、アプリケーションがGUIを構築するためのライブラリを提供します。ですから、GUIが必要なければTclだけでアプリケーションを作成することができます。

Tcl/Tkのインストールそのものは比較的容易ですが、注意すべき点は、Tcl/Tkのバージョンと日本語化です。Tcl/Tkの最新バージョンは8ですが、やや日本語の扱いに難があるようで、本書ではその1つ前のバージョンであるTclの7.6とTkの4.2を使います。このバージョンには日本語対応のパッチがありますので、それも併用します。

なお、FreeBSDやLinuxなどでは、すでにTcl/Tkがインストールされている場合もあるのでご注意ください。たまたまインストールされているバージョンがTcl7.6の日本語版とTk4.2の日本語版の組み合わせなら問題ないのですが、最近はTcl8や日本語版でないTcl7.6/Tk4.2がインストールされている場合もあります。この場合、事前にこれらのTclを削除しておくのが無難です。もちろん共存させることも可能ですが、やや技術的に難しいこと、またプラットフォームによって対応方法が異なるので、腕に自信のある方以外にはお勧めしません。

ちなみに、筆者が確認用に使ったLinuxPPC (Version 4) では、以下のパッケージを事前に削除する必要がありました。

- tclx-8.0.2
- tclx-8.0.2
- tk-8.0p2

FreeBSDの場合は日本語対応Tcl/Tkパッケージであるjp-tcl-7.6とjp-tk-4.2が用意されていますので、こちらをインストールしてもよいでしょう。

### Tcl のインストール

Tcl/Tkは実際にはTcl処理系の本体であるTclと、それを応用したTkに分かれています。インストールの順序はまずTclからになります。

付属CD-ROMのarchives/tcl/ディレクトリにすでに日本語パッチが適用済みのソースが収録されていますので、これを利用します。

ここでは、コンパイルは/usr/local/src以下で行うものとします。また、以下はす

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

べてroot（スーパーユーザ）で操作を行います．

```
# cd /usr/local/src
# tar xvf /mnt/cdrom/archives/tcl7.6jp.tar.gz
# cd tcl7.6/unix
# ./configure --enable-shared
```

ここで，筆者のシステムでは図4.1.1のエラーが出ました．原因は，Linuxに定義済みのmatherr()と，tclMtherr.cで定義している同名の関数の定義が矛盾しているためです．こちらは不要なので，tclMtherr.cの46行目

```
#ifndef NEED_MATHERR
```

の前に，

```
#ifdef NOTUSED
```

を挿入し，ファイルの最後に

```
#endif
```

を追加し対応します．これで後は問題なくコンパイルが通るはずです．後は

```
# make install
```

により，/usr/local/以下にTclがインストールされます．

図 4.1.1 コンパイル時のエラー

```
cc -c -O -fPIC -I./../generic -I. -DHAVE_UNISTD_H=1 -DHAVE_SYS_TIME_H=1 -
DTIME_WITH_SYS_TIME=1 -DHAVE_TM_ZONE=1 -DHAVE_TM_GMTOFF=1 -DHAVE_TIMEZONE_VAR=1 -
DSTDC_HEADERS=1 -DHAVE_SYS_IOCTL_H=1 -DKANJI -DTCL_SHLIB_EXT=".so"
./tclMtherr.c
./tclMtherr.c:50: warning: `DOMAIN' redefined
/usr/include/math.h:123: warning: this is the location of the previous definition
./tclMtherr.c:51: warning: `SING' redefined
/usr/include/math.h:124: warning: this is the location of the previous definition
./tclMtherr.c: In function `matherr':
./tclMtherr.c:76: argument `xPtr' doesn't match prototype
/usr/include/math.h:118: prototype declaration
make: *** [tclMtherr.o] Error 1
```

## Tk のインストール

無事にTclがインストールできたので、続いてTkをインストールします。

```
# cd /usr/local/src
# tar xzf /mnt/cdrom/archives/tck4.2jp.tar.gz
# cd tk4.2/unix
# ./configure --enable-shared
# make
# make install
```

### 4.1.3 libpgtclのインストール ~ PostgreSQL 6.3.2 の場合

libpgtclのインストール方法は6.3.2と6.4でかなり違います。ここでは6.3.2の場合について説明します。なお、すでに第2章の手順でPostgreSQL 6.3.2がインストール済みであるものとします。

以後のステップはpostgresユーザになって実行します。いったんログアウトしてからpostgresでログインし直すか、suコマンドを使います。

```
# su - postgres

$ cd /usr/local/src/postgresql-6.3.2/src
```

#### ▶ Linuxの場合

ここでLinuxの場合、interfaces/libpgtcl/Makefile.inにちょっとした修正が必要です。src/interfaces/libpgtcl/Makefile.inの34行目付近にある

```
LIBPQ                = -L $(SRCDIR)/interfaces/libpq -lpq
```

を

```
LIBPQ                = -L $(SRCDIR)/interfaces/libpq -lpq -lcrypt
```

に変更してください。次に、36行目と37行目の間

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

```
ifeq ($(PORTNAME), linux)
    ifdef LINUX_ELF
```

にLINUX\_ELF=trueを追加し,

```
ifeq ($(PORTNAME), linux)
    LINUX_ELF=true
    ifdef LINUX_ELF
```

とします。そして、/usr/local/src/postgresql-6.3.2/srcでconfigureを実行します。

```
$ ./configure --with-mb=EUC_JP --with-tcl --with-
includes="/usr/local/include" --with-lib="/usr/local/lib"
$ make
$ make install
```

次に、環境変数LD\_LIBRARY\_PATHに/usr/local/libを追加します。

```
export LD_LIBRARY_PATH=/usr/local/pgsql/lib:/usr/local/lib
```

### ▶ FreeBSD の場合

jp-tcl-7.6 とjp-tk-4.2 を使用する場合、configureの実行は以下のようになります。

```
% ./configure --with-mb=EUC_JP --with-tcl --with-includes="/usr/local
/include/tcl7.6/jp /usr/local/include/tk4.2jp" --with-libs="/usr/local
/lib"
% gmake
% gmake install
```

#### 4.1.4 libpqとlibpq-tclのインストール ~ PostgreSQL 6.4 の場合

前項と同様に、すでに第2章の手順でPostgreSQL 6.4がインストール済みであるものとします。

以後のステップはpostgresユーザになって実行します。いったんログアウトしてからpostgresでログインし直すか、suコマンドを使います。

```
# su - postgres
$ cd /usr/local/src/postgresql-v6.4/src
```

▶ Linuxの場合

Linuxの場合、6.3.2と同様、interfaces/libpq/Makefile.inに修正が必要です。src/interfaces/libpq/Makefile.inの30行目付近にある

```
SHLIB_LINK= -L../libpq -lpq
```

を

```
SHLIB_LINK= -L../libpq -lpq -lcrypt
```

に変更してください。そして

```
$ cd /usr/local/src/postgresql-v6.4/src
```

とし、configureを実行します。

```
$ ./configure --with-mb=EUC_JP --with-tcl --with-tclconfig=/usr/local/lib
```

次に、

```
$ make
$ make install
```

でコンパイル&インストールします。

次に環境変数LD\_LIBRARY\_PATHに/usr/local/libを追加します。

```
export LD_LIBRARY_PATH=/usr/local/pgsql/lib:/usr/local/lib
```

▶ FreeBSDの場合



## 4.1 Tcl/Tk インターフェースを使って ~ Mail/News の全文検索システム

6.4 の場合 , Linux と同様に interfaces/libpgtcl/Makefile.in に修正が必要になってしまいました . src/interfaces/libpgtcl/Makefile.in の30 行目付近にある

```
SHLIB_LINK= -L../libpq -lpq
```

を

```
SHLIB_LINK= -L../libpq -lpq -lcrypt
```

に変更してください . そして

```
% cd /usr/local/src/postgresql-v6.4/src
```

configure の5260 行目付近の

```
for dir in $library_dirs
```

を

```
for dir in $library_dirs /usr/local/lib/tk4.2jp
```

に変更し , configure を実行します .

```
% ./configure --with-mb=EUC_JP --with-tcl --with-tclconfig=/usr/  
local/lib/tcl7.6jp --with-includes="/usr/local/include/tcl7.6jp  
/usr/local/include/tk4.2jp"
```

```
gmake
```

```
gmake install
```

以上でインストールが完了です . /usr/local/lib に libpgtcl.a と libpgtcl.so がインストールされていること , そして , /usr/local/pgsql/bin に pgtclsh と pgtksh がインストールされていることを確認しておいてください .

ちなみに , pgtclsh は Tcl のインタプリタである tclsh に PostgreSQL の拡張を加えたもの , また pgtksh は , Tcl の GUI 付きインタプリタである wish に PostgreSQL の拡張を加えたものです . Tcl/Tk は , シェアードライブラリ (つまり libpgtcl.so) を load コマンドで実行時にロードすることにより , 拡張部分のモジュールをオリジナルの tclsh や wish から呼び出すことが可能になるため , pgtclsh や pgtksh のように静的に PostgreSQL の機能を組み込んだツールは本来必要ないのですが , load コマンドがうまく働かないときに備えて , 捨てないで取っておきましょう .

## 4.1.5 pgaccess を試してみる

PostgreSQL には、pgaccess という Tcl/Tk で作成された DB 管理ツールが付属しています。これを使ってみましょう。

pgaccess の起動は、6.3.2 の場合は

```
$ cd /usr/local/src/postgresql-6.3.2/bin/pgaccess
$ wish4.2jp -f pgaccess.tcl
```

6.4 の場合は

```
$ cd /usr/local/src/postgresql-6.4/bin/pgaccess
$ wish4.2jp -f pgaccess.tcl
```

のように行います。

これでうまく起動できない場合は、

```
load libpgtcl.so
```

という行を

```
load /usr/local/pgsql/lib/libpgtcl.so
```

に変更してみてください。

それでもだめな場合は、この行を pgaccess.tcl から削除し、

```
$ pgtksh -f pgaccess.tcl
```

として起動してみてください。

pgaccess が正常に起動されるとメイン画面が表示されます(図4.1.2)。最初はまだ何もデータベースが選択されていません。データベースを選ぶには、Database メニューから Open を選びます。そこでホスト名/ポート番号/データベース名を入力し Open を押すと、目的のデータベースに接続し、テーブルの一覧を表示します(図4.1.3)。ちなみにここで選んだデータベースは、次回

図 4.1.2  
pgaccess のメイン画面

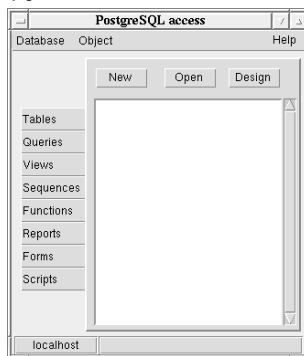
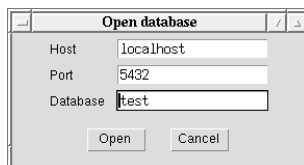


図 4.1.3  
テーブル一覧表示



## 4.1 Tcl/Tk インターフェースを使って ~ Mail/News の全文検索システム

pgaccess を起動したときに最初に接続するデータベースになります。この情報は、ホームディレクトリの下に pgaccess というファイルに格納されます。

テーブル名の一覧からテーブルを選び、Open ボタンを押すと、Table View 画面にテーブルの内容が200 レコードだけ表示されます（図4.1.4）。なおこの200 という値は、メイン画面の Database Preferences で変更できます。

表示結果はソートすることができません。Sort field にソートの対象とするフィールド名を入力しReload を押すと、ソートされて表示されます（図4.1.5）。

特定のレコードだけを選択したい場合はFilter Conditions に検索条件を入力してReload します（図4.1.6）。

新しいテーブルを作成するときは、メイン画面のNew ボタンを押すとテーブル定義用の画面が表示されます（図4.1.7）。この画面では新しいテーブルの名前やカラムなどを設定しCreate Table ボタンを押してテーブルを作成することができます。

レコードの追加 / 変更 / 削除もできます。メイン画面でテーブルを選んでTable View を表示します。\* 印のところに新しいデータを入力し、リターンキー  を押すと、データが追加されます。Reload を押すと、再び\* が表示され、新しいデータが追加できるようになります（図4.1.8）。テキスト型のデータ型では、日本

図 4.1.4

Table View に表示されたレコード

pid	pref	kana_pref
19	長野県	ナガノケン
36	山梨県	ヤマノリケン
22	東京都	トウキョウト
18	埼玉県	サイタマケン
15	千葉県	チバケン
21	静岡県	シズオカケン
11	愛知県	アイチケン
4	岐阜県	ギフケン
7	富山県	トヨカミケン
35	石川県	イハラクケン
12	福井県	フクイケン
46	山梨県	ヤマノリケン
13	長野県	ナガノケン
27	新潟県	ニガタケン
33	秋田県	アキタケン
2	岩手県	イワテケン
38	宮城県	ミヤギケン
37	山形県	ヤマノリケン
48	福島県	フクシマケン
32	茨城県	イバラクケン
41	栃木県	トチギケン
1	群馬県	グンマケン

図 4.1.5

ソートされたレコード

pid	pref	kana_pref
0	北海道	ホッカイドウ
1	青森県	アオモリケン
2	岩手県	イワテケン
3	宮城県	ミヤギケン
4	山形県	ヤマノリケン
5	福島県	フクシマケン
6	茨城県	イバラクケン
7	栃木県	トチギケン
8	群馬県	グンマケン
9	埼玉県	サイタマケン
10	千葉県	チバケン
11	東京都	トウキョウト
12	神奈川県	カンガワケン
13	新潟県	ニガタケン
14	山梨県	ヤマノリケン
15	長野県	ナガノケン
16	静岡県	シズオカケン
17	愛知県	アイチケン
18	岐阜県	ギフケン
19	富山県	トヨカミケン
20	石川県	イハラクケン
21	福井県	フクイケン

図 4.1.6

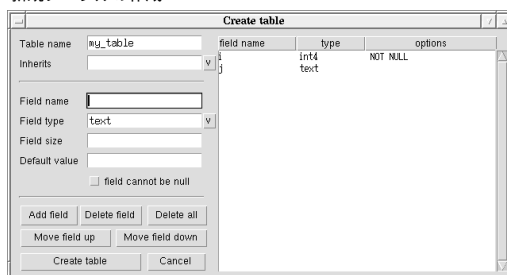
pid < 10 という条件のレコードを表示させたところ

pid	pref	kana_pref
0	北海道	ホッカイドウ
1	青森県	アオモリケン
2	岩手県	イワテケン
3	宮城県	ミヤギケン
4	山形県	ヤマノリケン
5	福島県	フクシマケン
6	茨城県	イバラクケン
7	栃木県	トチギケン
8	群馬県	グンマケン
9	埼玉県	サイタマケン
*		*

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

図 4.1.7  
新規テーブルの作成



語入力用のフロントエンドプログラムkinput2を起動しておけば日本語も入力できます。  
CTRL + [でkinput2の入力画面が表示されます。レコードの変更は、該当フィールドを変更した後[ ]を押します。削除はgを押します。

興味深い機能に“visual query designer”があります。

図 4.1.8  
新しいデータの追加

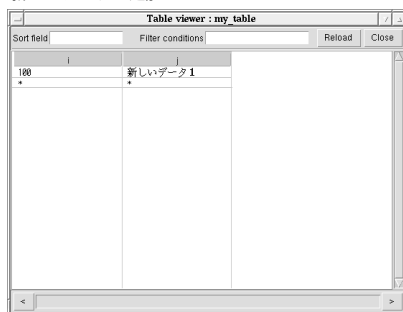
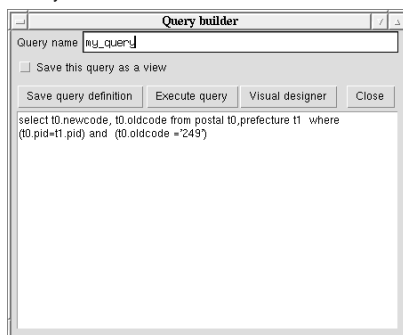


図 4.1.9  
Query builder



これはselect文をGUIを使って生成するものです。メイン画面のQueriesボタンを押し、Newを選択すると“Query builder”画面になります(図4.1.9)。Query Builderでは、問い合わせを入力してそれに名前を付けて保存することができます。

Visual Designerを押すとVisual query designerが起動され、GUIを使って視覚的にSELECT文を作成することができます。図4.1.10は、postalとprefectureをpidカラムでJOINする記述を作成したところです。操作は簡単でAdd tableにテーブル名を入力してテーブルを表す箱を表示させ、pidカラムを別のテーブルのpidカラムの上にDrag & Dropすると線が繋がってJOINが表現されます。また、カラムを下部の表にDrag & Dropすると、そのカラムに対してsortやさまざまな検索条件を設定できます。Show SQLを押すと、生成されたSQL文が確認できます。Execute SQLを押すと、その場で問い合わせを実行し、結果を表示できます

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

(図4.1.11). Save to query builderを押すと、生成されたSQL文がQuery builderのパツファに表示されます。Query builderに戻り、必要ならば修正を加えて名前を付けて保存します。保存したクエリーはいつでも呼び出して実行できます。ただしVisual query designerで作成した絵が保存されないところはちょっと残念です。

このほか、pgaccessには簡単なアプリケーションビルダがあります(図4.1.12)。もっとも処理の部分をTclで記述しなければならないため、完全にプログラムレスとはいきません。

このようになかなか高機能なpgaccessですが、まだバグや機能的に不十分な点も多く、改良の余地があります。しかし、ちょっとテーブルの中味を見たり修正するときにはとても重宝します。とくに多彩なGUIはTcl/Tkであればこそ、と言えると思います。

図 4.1.10  
postalとprefectureをJOINする

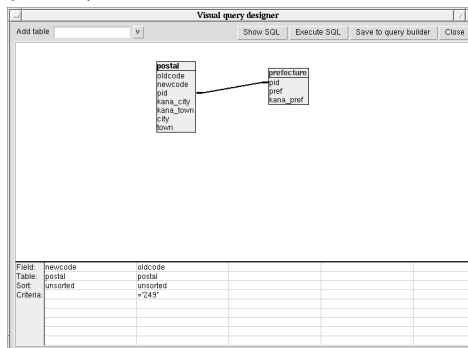


図 4.1.11  
結果表示

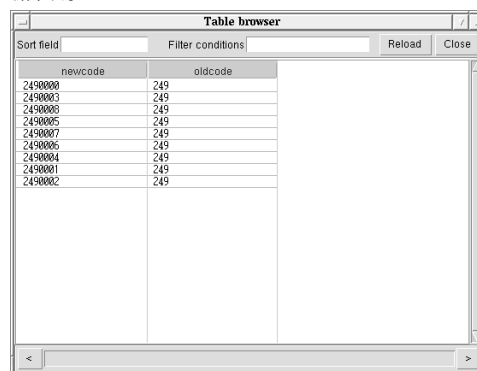
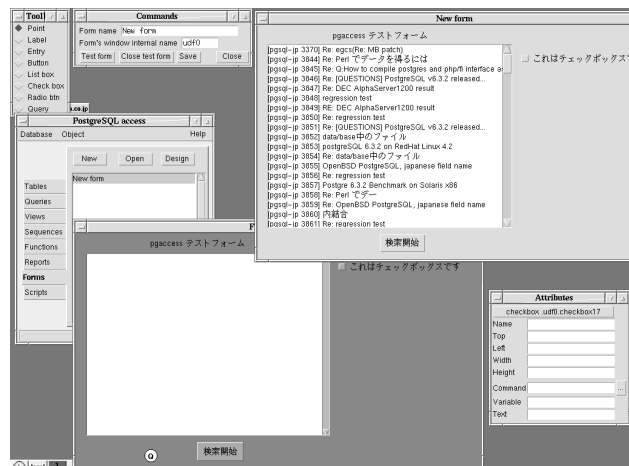


図 4.1.12  
pgaccessのアプリケーションビルダ



## 4.1.6 Mail/News の全文検索システムの作成

### 全文検索システムとは

あらかじめ登録しておいたタイトルやキーワードではなく、テキストデータの中味そのもので検索を行うシステムを全文検索システムと呼びます。同様のことはgrepなどを使ってもできそうですが、データ量が増えるにつれて処理速度が実用的でなくなります。このため、全文検索システムではあらかじめコンテンツからインデックスを生成しておき、高速な検索を実現しています。全文検索システムについては参考文献10や、

「フリーソフトで構築する全文検索システム」/馬場 肇 /『Software Design』1997年8月号 / 技術評論社

[http://www.kusastro.kyoto-u.ac.jp/~baba/Sbarticle/sd1997\\_08.html](http://www.kusastro.kyoto-u.ac.jp/~baba/Sbarticle/sd1997_08.html)

をご覧ください。

本節では、メールやニュースのアーカイブを全文検索するシステムをPostgreSQLのTcl/Tk インターフェースで作ってみます<sup>注2</sup>。

RDBMSを使った全文検索システムでも、数千件程度のデータ量なら専用の全文検索システムに負けないくらいの速度が出ますし、全文検索専用システムではできないような正規表現による検索も可能です。また、データがプログラムから独立していますから、一度作ったデータをいろいろなプログラムで使い回すこともできますし、データの追加 / 削除も容易です。

もっとも、データ件数が数十万件に達するような本格的な使い方では、やはり専用の全文検索システムを使った方が効率がよいようです。

### 全文検索システムの概要

今回作成するシステムは、

メールから単語やメールヘッダを抽出するツール：makeindex.pl

GUIを備えた検索ツール：smst (Simple Mail Search Tool)

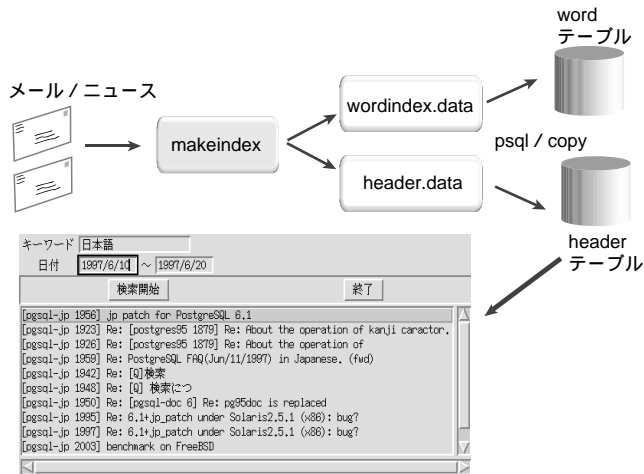
から構成されます (図4.1.13)。はkakasiというツールを起動し、単語を抽出するPerlプログラムです。makeindex.plが抽出したデータはタブ区切りのテキストデータで、PostgreSQLのcopyを使ってデータベースに登録されます。はTcl/Tkベース

#### 注2

このシステムは『Software Design』1997年10月号 (技術評論社)に掲載されたものをベースにしています。当時、PostgreSQLのバージョンは6.1.1で、現在のバージョンとは仕様が異なっているところがあり、その部分を書き換えました。また検索キーとして、キーワード、日付だけでなく、FromおよびSubjectでも検索できるようにしました。

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

図 4.1.1.3  
今回作成する全文検索  
システム



のGUI プログラムです。

なお、makeindex.pl はPerl5 の日本語版であるjperl5 とコード変換プログラムnkfを使いますので、あらかじめインストールしておいてください。

また、kakasi のインストールについては参考文献に詳しいのですが、ここに簡単にインストール方法を示します。なお、FreeBSD ではパッケージが存在するので、そちらを使ってもよいでしょう。

### nkf のインストール

kakasi を使うためにはnkf が必要です。nkf はLinux やFreeBSD の場合、コンパイル済みのパッケージが用意されているので、これからインストールするのが簡単です。もちろんソースからインストールしてもかまいません。

### kakasi のインストール

kakasi のインストールに必要なファイルは、以下から入手できます。

```
ftp://ftp.kusastro.kyoto-u.ac.jp/pub/baba/wais/kakasi-2.2.5.tar.gz
ftp://ftp.kusastro.kyoto-u.ac.jp/pub/baba/wais/kakasidict.940620.gz
ftp://ftp.kusastro.kyoto-u.ac.jp/pub/baba/wais/kakasi-ext.tar.gz
```

本書付属のCD-ROM に同じものがありますので、ここではそれを使います。kakasi 本体を展開し、パッチを当てます。なお、ここから先はroot で作業します。

```
$ su
# cd /usr/local/src
# tar xzf /mnt/cdrom/packages/kakasi/kakasi-2.2.5.tar.gz
# tar xzf /mnt/cdrom/packages/kakasi/kakasi-ext.tar.gz
# cd kakasi-2.2.5/src
# patch -p1 < ../../kakasi-ext/patch.kakasi-2.2.5
# make
# gunzip -c /mnt/cdrom/packages/kakasi/kakasidict.940620.gz > kakasidict
# mkdir /usr/local/lib/kakasi
# make install
```

これで/usr/local 以下にkakasi がインストールされました。

## スキーマ定義

本システムでは、以下のように2つのテーブルを定義します。

- words テーブルは抽出した単語を格納し、全文検索の対象となります(表4.1.1)。
- header テーブルはメールヘッダからの抽出情報を格納します。2つのテーブルはfname カラムによって関係付けられます(表4.1.2)。

PostgreSQL では8Kバイト以上の長さのレコードを扱うことができないので、メールの本体自身はDBには格納せず、fname カラムにパス名を入れておくことにします。厳

表 4.1.1 words テーブル

カラム名	型	備考
word	text	抽出された単語
count	int	単語の出現回数
fname	text	オリジナルのメールメッセージが格納されているパス

表 4.1.2 header テーブル

カラム名	型	備考
hfrom	text	メールヘッダのFrom:行
subject	text	メールヘッダのSubject:行
date	datetime	メールヘッダのDate:行
fname	text	オリジナルのメールメッセージが格納されているパス



## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

密に言うとlarge object を使えば可能ですが、制約が多いので今回は使用しません。

### 索引作成ツール makeindex.pl

makeindex.pl はperl で書かれたツールで、From やSubject などのメールヘッダの  
情報と単語情報を抽出します。

起動時のオプションは以下のようになります。

```
makeindex.pl [-h header_data_file] [-b] [-s] [-n] file1 file2...
```

- -h : メールヘッダのインデックスの生成を行う
- -b : メールボディの単語インデックスの生成を行う
- -s : X-Sequence:ヘッダからシーケンス番号情報を取り出す
- -n : 重複したmessage-id のファイルは無視する

header\_data\_file は、メールヘッダの抽出情報を格納するファイルです。すでにファイルが存在する場合は、データは末尾に追加されます。メールの本体から抽出した単語は標準出力に書き出されます。file1, file2... は抽出の対象となるメールファイルのパスで、1 ファイル / 1 メールメッセージであることを前提にしています。単語抽出の際には、英数字の大文字 / 小文字、全角 / 半角の統一および平仮名のみからなる単語など、検索に不要な単語の削除処理を行っています。リスト4.1.1 とリスト4.1.2 にメールから抽出した情報の例を示します。

makeindex.pl は簡単なプログラム (リスト4.1.3) なので、ポイントだけ説明します。なおmakeindex.pl は、CD-ROM のexamples/smst/makeindex.pl にありますので、コピーして使ってください。

```
jperl
```

1行目のjperl のパスは必要ならば書き換えてください。

リスト 4.1.1 header.data

```
Tatsuo Ishii <t-ishii@sra.co.jp> [pgsql-jp 5323] performance of 6.4 Thu  
Sep 10 1998 13:46:16 +0900      inbox/3
```

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

リスト 4.1.2 wordindex.data

速く	1	inbox/3	明る	1	inbox/3
real	2	inbox/3	#	2	inbox/3
t-ishii@sra.co.jp	1	inbox/3	石井	2	inbox/3
分か	1	inbox/3	---	1	inbox/3
達夫	1	inbox/3	fix	1	inbox/3
期待	1	inbox/3	mklinux	1	inbox/3
株	1	inbox/3	)	1	inbox/3
スケジュール	1	inbox/3	切れ	1	inbox/3
プラットフォーム	1	inbox/3	比べ	1	inbox/3
バグ	2	inbox/3	潰れ	1	inbox/3
(	1	inbox/3	昨日	1	inbox/3
benchmark	1	inbox/3	:-)	1	inbox/3
遅れ	1	inbox/3	0.040	1	inbox/3
見通し	1	inbox/3	0.340	1	inbox/3
6.3.2:	1	inbox/3	0.620	1	inbox/3
vadim	1	inbox/3	6.3.2	1	inbox/3
)sra	1	inbox/3	兇悪	1	inbox/3
6.4:	1	inbox/3	0.407	1	inbox/3
9:	2	inbox/3	9/1	1	inbox/3
7600/120+g3	1	inbox/3	カード	1	inbox/3
0.070	1	inbox/3	待ち	1	inbox/3
pre-dr3(mac	1	inbox/3	出る	1	inbox/3
query	2	inbox/3	wisconsin	1	inbox/3
0.711	1	inbox/3	sys	2	inbox/3
来ま	1	inbox/3	user	2	inbox/3
6.4	1	inbox/3			

リスト 4.1.3 makeindex.pl

```

1: #! /usr/bin/perl
2: # usage: makeindex [-h header_data_file] [-b] [-s] file1 file2...\n"
3: # -h: メールヘッダインデックスの生成を行う
4: # -b: メールボディの単語インデックスの生成を行う
5: # -s: X-Sequence: ヘッダからシーケンス番号情報を取り出す
6: # -n: 重複した message-id のファイルは無視する
7:
8: $body = "/tmp/BODY$$";
9: $result = "/tmp/RESULT$$";
10: $kakasi = "kakasi";
11: $nkf = "nkf";
12:
13: $header_flag = 0;      # メールヘッダの単語情報を生成するなら 1
14: $body_flag = 0;       # メール本体の単語情報を生成するなら 1
15: $seq_flag = 0;        # X-Sequence ヘッダ生成フラグ
16: $ignore_dup = 0;      # 重複 message-id ファイル無視フラグ
17:
18: $header = "";          # ヘッダ情報
19:
20: while ($ARGV[0] =~ /^~/) {
21:     $_ = shift;
22:
23:     if (/^-h$/) {
24:         $header = shift;
25:         $header_flag = 1;
26:         open(HEADER,">>$header") || die "Can't open $header\n";
27:     }

```

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

```
28:     elseif (/^b/) {
29:         $body_flag = 1;
30:     }
31:     elseif (/^s/) {
32:         $seq_flag = 1;
33:     }
34:     elseif (/^n/) {
35:         $ignore_dup = 1;
36:     }
37:     else {
38:         &usage;
39:         die;
40:     }
41: }
42:
43: if ($#ARGV < 0) {
44:     while (<>) {
45:         chop;
46:         $fname = $_;
47:         &do_it;
48:     }
49: } else {
50:     foreach $fname (@ARGV) {
51:         &do_it;
52:     }
53: }
54:
55: if ($header != "") {
56:     close HEADER;
57: }
58:
59: sub do_it {
60:     open(FILE,"$nkf -e -m $fname|") || die "Can't open $fname\n";
61:
62:     printf STDERR "processing $fname...";
63:
64:     if ($header_flag == 1) {
65:         $from = -1;
66:         $dup = 0;
67:         &gen_header;
68:         if ($from == -1) {
69:             printf STDERR "no From: header. skipped $i\n";
70:             next;
71:         } elseif ($dup == 1) {
72:             printf STDERR "Duplicate message. skipped $i\n";
73:             next;
74:         }
75:     }
76:
77:     if ($body_flag == 0) {
78:         close FILE;
79:     } else {
80:         &gen_body;
81:     }
82:
83:     printf STDERR "done.\n";
84: }
85:
```

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```

86: sub gen_body {
87:   if ($header_flag == 0) {
88:     # メールヘッダをスキップ
89:     while (<FILE>) {
90:       last if (/^\n/);
91:     }
92:   }
93:
94:   open(BODY,">$body") || die "Can't open $body\n";
95:   while(<FILE>) {
96:     s/\s+/ /g;          # 連続したタブ、空白、改行を ASCII スペースに置換
97:     s/ / /g; # 全角スペースを ASCII スペースに置換
98:     print BODY;
99:   }
100:  close BODY;
101:  close FILE;
102:  system("$kakasi -w -c'\<\<' -o euc < $body |tr ' ' '\012' > $result");
103:  open(RESULT,$result);
104:  undef %words;
105:  while (<RESULT>) {
106:    chop;
107:
108:    y/A-Z/a-z/;          # case を統一
109:    y/A-Z/a-z/;
110:    y/0-9/0-9/;
111:
112:    s/\\/\\\\/g;        # PostgreSQL は backslash の escape を必要とする
113:
114:    next if (length() > 64); # 64 バイト以上の長い単語は外す
115:    next if (/^\n/); # 空行は無視
116:    next if (/^ *$/); # 全角スペースだけの行も無視
117:    next if (/^[\., ]*$/); # 句読点その他だけの行は無視
118:    next if (/^[あ-ん]*$/); # 平仮名だけの行は無視
119:    next if (/^[a-zA-Z0-9]*$/); # 英数 1 文字の行は無視
120:    $words{$_}++;
121:  }
122:  close RESULT;
123:  foreach $word (keys %words) {
124:    if ($seq_flag == 1) {
125:      printf "$seq %s      $words{$word}      $fname\n", $word;
126:    } else {
127:      printf "%s      $words{$word}      $fname\n", $word;
128:    }
129:  }
130:  unlink $body;
131:  unlink $result;
132: }
133:
134:
135: sub gen_header {
136:   $msgid = "dummy";
137:
138:   while (<FILE>) {
139:     s/\\/\\\\/g;
140:     s/\s+//g;
141:     $seq = $1 if (/^X-Sequence: .* ([0-9]*)$/);
142:     $from = $1 if (/^From: (.*?)$/);
143:     $subject = $1 if (/^Subject: (.*?)$/);

```

147

### わかち書き

102 行目では、`kakasi` を起動してテキストを「わかち書き」に分解しています。わかち書きになった単語は空白文字で区切られているので、区切り文字を `tr` コマンドで改行に変更しています。

### 単語の削除

114 ~ 119 行目では、`kakasi` が抽出した単語のうち、不要な単語を削除しています。ここに条件を追加することにより、さらに不要な単語を削除できます。words テーブルは非常に大きくなります。データにもよりますが、5,000 件のメールから単語を抽出すると、40 万レコード以上にもなります。ですから、単語の削除条件を変えてみると大きな効果があるかもしれません。

### Date:フィールド

159 ~ 185 行目までは `Date:` フィールドを解析しています。一応ほとんどのフォーマットに対応しているはずですが、メーラによってはほとんどないフォーマットの `Date:` フィールドを生成するものがあります（利用者の設定ミスもありますが）。解析エラーになる場合は、ここを修正してみてください。

## データベースへの登録

`makeindex.pl` で抽出したデータは、`psql` を使ってデータベースに登録します。今回は `/somewhere/inbox/` の下にメールファイルが格納されているものとします。

```
$ cd /somewhere
$ makeindex.pl -h header.data -b inbox/* > wordindex.data
```

としてデータを抽出します。inbox の下のメールファイルの文字コードは、EUC/JIS/SJIS のどれでもかまいません。`makeindex.pl` から起動される `nkf` により、自動的に EUC にコード変換されます。

また、メールヘッダのデータを格納する `header.data` ですが、このファイルが存在すると、上書きせずにデータを追加します。なんらかの理由でやり直しをする場合は、前もって `header.data` を削除しておいてください。

inbox の下のファイルの数が非常に多いとシェルのエラーになる場合もあります。そのときは処理したいファイル名を改行で区切ったファイルを用意し、それを読み込ませます。たとえば、

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

### リスト 4.1.4 create.sql

```
drop table words;
create table words (word text, count int, fname text);
drop table header;
create table header (hfrom text, subject text, date datetime, fname text);
```

inbox/1

inbox/2

inbox/3

という内容のファイルを “lists” という名前で作っておいたとします。この場合、以下のようにmakeindex.pl を起動します。

```
$ cat lists|makeindex.pl -h header.data -b > wordindex.data
```

次にテーブルを作成します。この時点ではまだインデックスを作りません。これは、大量にデータを登録する場合はインデックスなしの方がずっと速いからです。インデックスはデータを登録した後に作成します。テーブル作成に必要なSQL 文をリスト 4.1.4 に示します<sup>注3</sup>。

```
$ psql -f create.sql test
```

この時点ではこのSQL を実行したユーザ以外はテーブルがアクセスできません。他人にもアクセスを許す場合は、

```
$ psql -c 'grant select on words,header to public' test
```

を実行しておいてください。

データの登録はpsql のcopy 文を使います。以下を実行してください。

```
$ cat header.data |psql -c 'copy header from stdin' test
```

```
$ cat wordindex.data |psql -c 'copy words from stdin' test
```

最後に、インデックス作成用のSQL 文（リスト4.1.5）を実行します<sup>注4</sup>。

```
$ psql -f create_index.sql test
```

#### 注 3

同じものがCD-ROMのexamples/smsst/create.sqlにあるので、コピーして使ってください。

#### 注 4

こちらも同じものがCD-ROMのexamples/smsst/create\_index.sqlにあるので、コピーして使ってください。

リスト 4.1.5 create\_index.sql

```
drop index words_word_index;
create index words_word_index on words using btree (word);
drop index words_fname_index;
create index words_fname_index on words using btree (fname text_ops);
drop index header_date_index;
create index header_date_index on header using btree (date);
drop index header_hfrom_index;
create index header_hfrom_index on header using btree (hfrom);
drop index header_subject_index;
create index header_subject_index on header using btree (subject);
drop index header_fname_index;
create index header_fname_index on header using btree (fname);
vacuum header;
vacuum words;
select count(*) from words;
```

## 検索用 GUI smst

smst は CD-ROM の `examples/srst/srst` にありますので、コピーして使ってください。

smst の起動オプションは次の通りです。

```
smst [-host DBhost] [-db DBname] [-dir directory]
```

- -host DBhost

データベースサーバ (postmaster) の動いているホスト名。ローカルホストで postmaster が動いている場合は省略可能です。

- -db DBname

words, header テーブルの登録されているデータベース名。ユーザ名と同じ名前のデータベースを使う場合は省略可能です。

- -dir directory

smst はメール本体を表示する際に「ここで指定したディレクトリ + “/” + makeindex.pl の引数で与えたパス名」をファイル名としてメールファイルを開きます。省略すると、カレントディレクトリからの相対パスでメールファイルを開きます。

今回の例では、

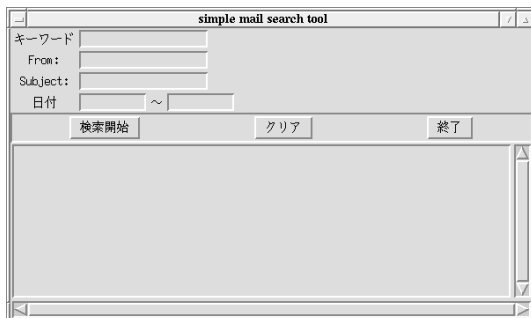
```
$ smst -dir /somewhere -db test
```

でよいはずです。あるいはすでに /somewhere にいるなら、



## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

図 4.1.14  
smst の起動



```
$ smst -db test
```

となります。

smst を起動すると検索画面が表示され、全件検索用のキーワード / From: / Subject: および日付の範囲指定ができるようになります (図 4.1.14)。複数の検索条件を入力すると、AND 条件で検索を行います。キーワード / From: / Subject: の文字列検索の仕様は以下の通りです。

- キーワード...前方一致かつ大文字 / 小文字の区別なし
- From: ...部分一致かつ大文字 / 小文字の区別なし
- Subject: ...部分一致かつ大文字 / 小文字の区別なし

キーワードで部分一致検索をしたい場合は、キーワードの前に \* を追加します。たとえば

```
free      .*free
```

のようにします。検索自体は PostgreSQL の正規表現検索を利用していますので、正規表現も使えます。

日付データのフォーマットは PostgreSQL が理解できるものでなければなりません。“1999/8/14” や “Thu Aug 14 19:01:51 1999 JST” などです。そのほか予約語として “epoch” “now” などがあります。epoch は 1970/1/1 で、now は現在時刻です。たとえば “1999/1/1 ~ now” では、1999 年 1 月 1 日以降のデータが指定されることになります。詳しくは p.88 や PostgreSQL 付属のマニュアル<sup>注5</sup>をご覧ください。

検索キーワードを入力してから「検索開始」ボタンを押すと、下のリストボックスに検索されたメールのサブジェクトと From: の一覧が表示されます。サブジェクト

注 5

6.3.2 の場合は /usr/local/pgsql/man/man3/pgbuiltin.3、6.4 の場合は doc の下のドキュメントです。

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

図 4.1.15  
smst で検索され  
たメールの本体



覧は日付順に並びます。その中からメール本体を表示するものを選び、マウスの第1ボタンでダブルクリックすると、メール本体が別のウィンドウに表示されます（図 4.1.15）。

## smst の構造

smst の使い方がわかったところで、プログラムの構造を見ていくことにしましょう（リスト 4.1.6）。ここでは Tcl/Tk の文法を説明するのが目的ではないので、必要がな  
いぎりプログラムの細かい点には触れません。Tcl/Tk の詳細については、以下の参  
考文献などをご覧ください。

### 参考文献：

『Tcl & Tk ツールキット』 / John K. Ousterhout / ソフトバンク

1 行目は wish4.2jp のパスを指定しています。wish4.2jp が違う場所にある場合は、  
ここを書き換えてください。また、パスが非常に長い場合、この方法では起動できな  
いことがあります、そのときは

```
$ wish4.2jp -f smst
```

で起動できます。なお、smst は libpgtcl.so を load コマンドでダイナミックロードして  
いますが、これがうまく動かない場合は、192 行目の

```
load "libpgtcl.so"
```

を

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

### リスト 4.1.6 smst

```
1: #! /usr/local/bin/wish4.2jp -f
2: #
3: # 超シンプルメールアーカイブ検索ツール
4:
5: # 初期画面を作成する
6: proc make_widgets {} {
7:     global Key From Subject Sdate Edate
8:
9:     wm title . "simple mail search tool"
10:    wm iconname . "smst"
11:    frame .k
12:
13:    set f [frame .k.keyword]
14:    pack [label $f.l -width 10 -text "キーワード"] \
15:    [entry $f.e -width 20 -textvariable Key] -side left
16:    pack $f -side top -fill x -anchor w
17:
18:    set f [frame .k.from]
19:    pack [label $f.l -width 10 -text "From:" ] \
20:    [entry $f.e -width 20 -textvariable From] -side left
21:    pack $f -side top -fill x -anchor w
22:
23:    set f [frame .k.subject]
24:    pack [label $f.l -width 10 -text "Subject:" ] \
25:    [entry $f.e -width 20 -textvariable Subject] -side left
26:    pack $f -side top -fill x -anchor w
27:
28:    set f [frame .k.date]
29:    pack [label $f.l1 -width 10 -text "日付"] \
30:    [entry $f.e1 -width 10 -textvariable Sdate] \
31:    [label $f.l2 -text "~"] \
32:    [entry $f.e2 -width 10 -textvariable Edate] -side left
33:    pack $f -side top -fill x -anchor w
34:
35:    set f [frame .k.buttons -relief sunken -borderwidth 2]
36:    set b1 [button $f.b1 -text "検索開始" -command "start_select"]
37:    set b2 [button $f.b2 -text "クリア" -command "clear"]
38:    set b3 [button $f.b3 -text "終了" -command "cleanup"]
39:    pack $b1 $b2 $b3 -side left -expand yes
40:    pack $f -side top -fill x
41:
42:    set f [frame .k.list]
43:    set w [listbox $f.listbox -xscroll "$f.xscroll set" \
44:    -yscroll "$f.yscroll set" -relief sunken -setgrid 1 \
45:    -selectmode single -width 80 -font a14 -kanjifont k14]
46:    bind $w <Double-1> "show"
47:    set xs [scrollbar $f.xscroll -orient horizontal \
48:    -relief sunken -command "$w xview"]
49:    set ys [scrollbar $f.yscroll -relief sunken -command "$w yview"]
50:    pack $xs -side bottom -fill x -expand yes
51:    pack $ys -side right -fill y -expand yes
52:    pack $w -side top -fill both -expand yes
53:    pack $f -side top -fill both -expand yes
54:
55:    pack .k -fill both -expand yes
56: }
57:
58: # 終了ボタンを押された時に呼び出される後始末 procedure
59: proc cleanup {} {
```

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
60:     global con
61:     pg_disconnect $con
62:     destroy .
63: }
64:
65: # 検索キーをクリア
66: proc clear {} {
67:     global Key From Subject Sdate Edate
68:     set Key ""
69:     set From ""
70:     set Subject ""
71:     set Sdate ""
72:     set Edate ""
73: }
74:
75: # 検索開始ボタンを押された時に呼び出されて実際に検索を実行する
76: proc start_select {} {
77:     global Key From Subject Sdate Edate con SeqList
78:
79:     set hasKey 0
80:
81:     set query "select distinct b.subject, b.hfrom, b.fname, b.date \
82: from words a, header b "
83:     if {$Key != ""} {
84:         append query "where a.word ~* '^$Key' "
85:         set hasKey 1
86:     }
87:     if {$From != ""} {
88:         if {$hasKey != 0} {
89:             append query "and "
90:         } else {
91:             append query "where "
92:         }
93:         append query "b.hfrom ~* '$From' "
94:         set hasKey 1
95:     }
96:     if {$Subject != ""} {
97:         if {$hasKey != 0} {
98:             append query "and "
99:         } else {
100:             append query "where "
101:         }
102:         append query "b.subject ~* '$Subject' "
103:         set hasKey 1
104:     }
105:
106:     if {$Sdate != "" && $Edate != ""} {
107:         if {$hasKey != 0} {
108:             append query "and "
109:         } else {
110:             append query "where "
111:         }
112:         append query "b.date >= '$Sdate' and b.date <= '$Edate'"
113:         set hasKey 1
114:     }
115:
116:     append query " and a.fname = b.fname order by b.date"
117:
118:     puts $query
119: }
```

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

```
120:     if {$hasKey == 0} {
121:         set msg "検索キーが入力されてません"
122:         tk_dialog .dialogue "selection failed" $msg error -1 "そうですか"
123:         return
124:     }
125:
126:     catch {set res [pg_exec $con $query]}
127:     if {[info exist res] == 0 || \
128:         [string range $res 0 4] != "pgsql" || \
129:         [pg_result $res -numTuples] == 0} {
130:         set msg "該当データがありません。"
131:         tk_dialog .dialogue "selection failed" $msg error -1 "そうですか"
132:     } else {
133:         set n [pg_result $res -numTuples]
134:         set SeqList ""
135:         .k.list.listbox delete 0 end
136:
137:         for {set i 0} {$i < $n} {incr i} {
138:             set l [pg_result $res -getTuple $i]
139:             set subject [lindex $l 0]
140:             set from [lindex $l 1]
141:             set seq [lindex $l 2]
142:             .k.list.listbox insert end "$subject $from"
143:             lappend SeqList $seq
144:         }
145:         pg_result $res -clear
146:     }
147: }
148:
149: # 検索結果リストのダブルクリックで呼び出され、メール本体を表示する
150: proc show {} {
151:     global SeqList MyArgs
152:
153:     catch {destroy .d};          # すでにウィンドウが存在したらそれを消してから表示
154:     toplevel .d;                 # 独立したウィンドウにする
155:     wm title .d "Contents"
156:     set f [frame .d.t]
157:     set w [text $f.text -width 80 -wrap none -xscrollcommand "$f.xscroll set" \
158:         -yscrollcommand "$f.yscroll set"]
159:     set xs [scrollbar $f.xscroll -orient horizontal \
160:         -relief sunken -command "$w xview"]
161:     set ys [scrollbar $f.yscroll -relief sunken -command "$w yview"]
162:     pack $xs -side bottom -fill x
163:     pack $ys -side right -fill y
164:     pack $w -side top -fill both -expand yes
165:     pack $f -side top -fill both -expand yes
166:
167:     set f [frame .d.buttons -relief sunken -borderwidth 2]
168:     set b1 [button $f.b1 -text "閉じる" -command "destroy .d"]
169:     pack $b1 -side left -expand yes
170:     pack $f -side top -fill x
171:
172:     set fname $MyArgs(-dir)
173:     if {$fname != ""} {append fname "/"}
174:     append fname [lindex $SeqList [lindex $k.list.listbox curselection]]
175:     set fd [open $fname]
176:     while {[gets $fd line] >= 0} {
177:         $w insert end [format "%s\n" $line]
178:     }
179:     close $fd
```

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
180: }
181:
182: #-----
183: # ここからメインプログラム
184: #-----
185:
186: # 漢字コードを EUC-JP にする
187: kanji internalCode EUC
188: kanji defaultOutputCode EUC
189: kanji defaultInputCode ANY
190:
191: # libpgtcl をロード
192: load "libpgtcl.so"
193:
194: global MyArgs env
195: set MyArgs(-host) "localhost"
196: set MyArgs(-db) $env(USER)
197: set MyArgs(-dir) ""
198:
199: # コマンド引数の処理
200: set argexpect ""
201: foreach i $argv {
202:     if {$argexpect != ""} {
203:         set MyArgs($argexpect) $i
204:         set argexpect ""
205:     } else {
206:         case $i in
207:             {-host}) {
208:                 set argexpect $i
209:             }
210:             {-db}) {
211:                 set argexpect $i
212:             }
213:             {-dir}) {
214:                 set argexpect $i
215:             }
216:             {default}) {
217:                 puts stderr {usage: smst [-- -host DBhost -db DBname -dir directory]}
218:                 exit
219:             }
220:         }
221:     }
222: }
223:
224: global con; # PostgreSQL サーバへの接続ハンドル
225: catch {set con [pg_connect $MyArgs(-db) -host $MyArgs(-host)]}
226: if {[info exist con] == 0 || \
227:     [info exist con] == 1 && \
228:     [string range $con 0 4] != "pgsql"} {
229:     set msg [format "データベースサーバ %s のデータベース %s と接続できません。" \
230:         $MyArgs(-host) $MyArgs(-db)]
231:     tk_dialog .dialogue "smst Dialogue" $msg error -1 "そうですか"
232:     exit
233: }
234:
235: make_widgets;          # 初期画面の作成
236:
```

## 4.1 Tcl/Tk インターフェースを使って～ Mail/News の全文検索システム

```
load /usr/local/pgsql/lib/libpgtcl.so
```

に変更してみてください。それでもだめな場合はこの行を削除し、1行目を

```
#!/usr/local/pgsql/bin/pgtksh -f
```

に書き換えてください。

プログラムの前半はprocキーワードではじまるprocedure（サブルーチン）の宣言部です。

make\_widgets（6～56行目）

make\_widgetsは、検索用の画面を作ります。make\_widgetsでは、主なwidget（ウィジェット：GUI部品）として、

- キーワード入力用entry widget（15行目）
- From:入力用entry widget（18行目）
- Subject:入力用entry widget（23行目）
- 日付入力用entry widget（28行目と32行目）
- 検索開始 / 検索キークリア / 終了用button widget（36，37，38行目）
- 検索結果表示用のlistbox widget（42行目）

を作っています。入力されたキーワードは大域変数の“Key”，From:は“From:”，Subject:は“Subject:”，日付は“Sdate”または“Edate”に自動的に設定されます。また、リストボックスに表示された検索結果をダブルクリックすることによりメール本体を表示する“show”procedureが起動されますが、これは46行目のbindコマンドで設定されます。

cleanup（59～63行目）

終了ボタンが押されると、終了処理を行います。

- pg\_disconnectによりPostgreSQLサーバとの接続を切る
- destroyによりsmstのウィンドウを消去する

clear（66～73行目）

各検索キー入力エリアを消去します。

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

start\_select (76 ~ 147 行目)

「検索開始」ボタンが押されるとSQL文を生成し、結果をリストボックスに表示します。SQL文を生成する処理は81 ~ 116行目でを行っています。たとえばキーワードに「検索」を指定すると、生成されるSQL文は

```
select distinct b.subject, b.hfrom, b.fname, b.date from words a,
header b where a.word ~* '^検索' and a.fname = b.fname order by
b.date
```

となります。118行目でSQL文を表示していますので、いろいろな検索キーに対してどのようなSQL文が生成されるか確認してください。

生成されたSQL文は、126行目のpg\_exec コマンドにより実行されます。もし該当データが見つければ133行目のpg\_result-numTuples コマンドにより件数を確認した後、138行目のpg\_result-getTuples コマンドにより1件ずつデータを取り出します。

取り出したデータはサブジェクト、From:、ファイル名...などがリストになっていますので、139行目と140行目でサブジェクトとFrom:を取り出した後、142行目でリストボックスに追加します。

検索結果を表示する際、表示されたデータのfname カラムは大域リスト変数の“SeqList”に格納されます。これは後述のshow procedure が参照します。

show (150 ~ 180 行目)

検索結果リストでダブルクリックがあると、show が呼び出されます。ダブルクリックされた行番号は、リストボックスwidgetのcurselection コマンドで取得できます(174行目)。これをキーにしてstart\_select が作成したSeqListを検索すれば、表示すべきファイル名がわかるわけです。

メインプログラム (182 行目 ~)

procの内側に書かれていない部分はメインプログラムとみなされ、ただちに実行されます。メインプログラムの処理は以下の通りです。

使用する漢字コードをEUCにする (187 ~ 189 行目)

引数の解析 (200 ~ 222 行目)

PostgreSQL データベースサーバへの接続 (225 ~ 233 行目)

初期画面の作成 (235 行目)



### データの更新について

以上でメールやニュース記事のデータをデータベースに登録 / 検索できるようになりましたが、新たにデータを追加したい場合や、不要なデータを削除する方法を説明します。

#### データの追加

まず、追加分のメールデータを別ディレクトリに入れます。そして、そのディレクトリのデータを対象にmakeindex.plでデータを作成します。作成したデータはデータの新規登録とまったく同じ要領で追加できます。

たとえばinbox.newに追加分のデータがある場合は、

```
$ makeindex.pl -h header.data -b inbox.new/* > wordindex.data
```

でheader.dataとwordindex.dataを生成した後、

```
$ cat header.data |psql -c 'copy header from stdin' test
$ cat wordindex.data |psql -c 'copy words from stdin' test
```

データを大量に追加したときは、vacuumをやり直すほうがよいでしょう。すなわち、

```
$ psql -c 'vacuum header' test
$ psql -c 'vacuum words' test
```

を実行します。

#### 不要データの削除

SQLを用いて不要データを削除します。たとえば、1ヵ月以上前のデータを削除するにはまず、

```
delete from words where header.date < ('now'::datetime - '1 month'::interval)
and header.fname = words.fname;
```

でwordsテーブルから単語データを削除し、次に

```
delete from header where date < ('now'::datetime - '1 month'::interval);
```

headerテーブルからデータを削除します。削除する順番を逆にするとwordsテーブルのデータを日付を元にして削除することができなくなるのでご注意ください。

問題はテキストファイルで保存されているオリジナルデータですが、

```
select fname from header where date < ('now'::datetime - '1 month'::interval);
```

で削除対象のファイル名を求めておいて、UNIXのrmコマンドで個別に消すしかありません。将来、PostgreSQLが8Kバイトよりも大きなデータをtext型に格納できるようになればこのような手間もなくなるので、それに期待しましょう。

#### 4.1.7 感想など

最近ユーザインターフェースが必要なものはPHPで作ることが多く、今回本書に収録するにあたり、久しぶりにTcl/Tkに触ってみました。改めて感じることは「Tcl/Tkは軽くてよい」です。近ごろでは、GUIをJavaで書く機会も増えてきましたが、最新型の強力なマシンでもまだまだJavaは重く、それに比べてTcl/Tkはなんと軽く感じることか！また、コンパイルが必要ないので、ちょっとソースをいじってはその場で気軽に試すことができ、大いにプログラム作成の生産性が上がります。

半面、厳密な型付がないことは誤りを誘発しやすく、またコンパイルする必要がないということは逆に実行するまでバグの存在がわからないということでもあります。したがって、巨大なシステムをすべてTcl/Tkで作るのは無理がありますが、ちょっとしたツールを書くのにはTcl/Tkは十分すぎるほど強力です。そういう意味で、PostgreSQLのような小回りの効くデータベースとは非常に相性がよいと言えるのではないのでしょうか。

## 4-2

# C インターフェース を使って アイコン管理コマンドの作成

## 4.2.1 基本ライブラリとしての libpq

libpq は C 言語で書かれた PostgreSQL 用のインターフェースライブラリで、C 言語から呼び出して使います。前章で解説したように、libpq は JDBC や ODBC を除き、Tcl や PHP などの他の言語のインターフェースからも呼び出される非常に基本的なライブラリです。それだけに、アプリケーションが直接 libpq を使う局面はあまりないかもしれません。

本節では、その数少ないケースのひとつである、BINARY CURSOR を使うアプリケーションを例に取りながら、libpq の使い方を説明します。

## 4.2.2 アイコン管理コマンド pgicons

large object を使って PostgreSQL で大きなバイナリを扱うことができることはすでに説明しました。ここでは、それほど大きくないバイナリデータ、たとえば Web ページで使われている小さなアイコンデータを PostgreSQL で扱うことを考えましょう。このようなデータで、大きさが 8K バイトを超えないものは、もう 1 つのバイナリ用のデータ型である bytea で扱うことができます。

本節で取り上げる “pgicons” は、libpq を使って C 言語で書かれたシンプルなコマンドで、アイコンデータのファイル名の一覧を表示し、選択されたものを xv を使って表示します<sup>注1</sup>。

pgicons は以下のテーブルを使用します。

### 注 1

xv は汎用のグラフィックス表示プログラムです。非常に多くのグラフィックスフォーマットを扱うことができます。ただしフリーソフトではありませんので、使い続ける場合はシェアウェア料金を払ってください。

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
create table icons(
    name text,
    icon bytea
);
```

name はアイコンのファイル名です。icon にはアイコンデータを格納します。xv で表示でき、大きさが2Kバイトくらいまでのものなら<sup>※2</sup>、GIF やJPEG などほとんどのデータを扱うことができます。

ところで考えてみると、アイコンデータはバイナリですから、INSERT 文にはそのまま書けません。そこでPostgreSQL では、バイナリデータを8進数で表現します。たとえば、10進で20、すなわち8進で24は、

```
\024
```

と書きます。INSERT 文で使うときは、実際には\ をエスケープするために、

```
\\024
```

#### 注 2

バイナリデータを後述のような文字列に展開すると、約4倍の大きさになります。したがって、8Kバイトの1/4の2Kバイトが上限になるわけです。

#### リスト 4.2.1 icons2sql.c

```
#include <stdio.h>
main(int argc, char **argv)
{
    int i;
    FILE *fd;
    char *fname;
    int c;

    if (argc < 2) {
        fprintf(stderr, "usage: icons2sql file1 file2...\n");
        exit(1);
    }

    for (i=1; i<argc; i++) {
        fname = argv[i];
        fd = fopen(fname, "r");
        if (!fd) {
            fprintf(stderr, "couldn't open %s\n", fname);
            exit(1);
        }
        printf("insert into icons values ('%s', '\\', fname);
        while ((c = fgetc(fd)) != EOF) {
            printf("\\\\%03o", c);
        }
        fclose(fd);
        printf("\\');\n");
    }
    exit(0);
}
```

## 4.2 C インターフェースを使って～アイコン管理コマンドの作成

とします。リスト4.2.1は指定されたアイコンファイルの内容を、このエスケープ表現を使ってINSERT文に変換するプログラムです。同様のことはPerlでもできますが、今回はC言語にこだわってCで書いてみました。

さて、byteaで格納されたデータはSELECT文で取り出したときにも\024のような「エスケープ表現」になります。それをいちいちバイナリに変換してもよいのですが、BINARY CURSORという機能を使えば、データベースに格納されているバイナリ値をそのまま取り出すことができます。そこで、pgiconsではこのBINARY CURSORを使ってみることにします。

### 4.2.3 インストール

概要がわかったところで、さっそくpgiconsをインストールしましょう。付属CD-ROMのexamples/pgicons.tar.gzにプログラム一式がありますので、それを適当なディレクトリに展開します。

```
$ tar xzf /mnt/CD-ROM/examples/pgicons.tar.gz
$ cd pgicons
```

ここで、Makefile（リスト4.2.2）の調整をします。まず、1行目のPOSTGRESHOMEはPostgreSQLがインストールされているディレクトリです。/usr/local/pgsqlでない場合は、適当に変更してください。

リスト 4.2.2 Makefile

```
1: POSTGRESHOME = /usr/local/pgsql
2: XV = xv
3: CFLAGS = -DXV=\"$$(XV)\" -g -O2 -I$(POSTGRESHOME)/include
4: OPTLIBS = -lcrypt
5:
6: all:: pgicons icons2sql icons.sql
7:
8: pgicons: pgicons.o
9:     $(CC) -o pgicons pgicons.o -L$(POSTGRESHOME)/lib -lpq $(OPTLIBS)
10:
11: icons2sql: icons2sql.o
12:     $(CC) -o icons2sql icons2sql.o
13:
14: icons.sql: icons2sql
15:     (cd icons;../icons2sql *.gif) > icons.sql
16:
17: clean:
18:     rm -f pgicons icons2sql icons.sql *.o
19:
```

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

次は4行目のOPTLIBSです。ここにはlibpq以外に必要なライブラリを記述します。  
Linux では`-lcrypt` が必要ですが、FreeBSD ではここは

```
OPTLIBS =
```

でよいはずです。

以上、用意ができれば

```
$ make
```

を実行します。これで、カレントディレクトリに以下のファイルができます。

- create.sql : テーブル生成 SQL 文
- icons.sql : データ登録 SQL 文
- icons2sql : icons.sql を生成したプログラム
- pgicons : アイコンデータ管理ツール

次にテーブルを作成します (リスト4.2.3 参照)。

```
$ psql -f create.sql test
```

これでiconsというテーブルができるので、icons.sql を使ってデータを登録します。今回、Apache のソースに付属するアイコンデータをサンプルに使い、icons.sql を生成しています。独自のアイコンデータを使用する場合は、以下のようにしてicons.sql を生成します。

```
$ icons2sql アイコンファイル1 アイコンファイル2 ... > icons.sql
```

icons.sql のデータは、

```
insert into icons values ('a.gif', '\\107\\111\\106...');
```

のような感じで、バイナリのアイコンデータがエスケープ表現になっているのがわかります。これを登録します。

```
$ psql -f icons.sql test
```

リスト 4.2.3 create.sql

```
1: drop table icons;  
2: create table icons(name text, icon bytea);  
3:
```



## 4.2.4 pgicons の起動

データの準備ができたので、さっそく pgicons を起動してみましょう。

```
$ pgicons test
```

すると、図4.2.1のように端末に番号、アイコンのファイル名、サイズが表示されます。

ここで表示したい番号を入力すると、xv が起動され、アイコン画像が表示されます。

たとえば“18”を入力すると  が、“55”を入力すると  が表示されます。画像を確認した後 xv を終了すると、再び画像の一覧表が表示されます。終了したいときは

**CTRL + D** を押してください。

図 4.2.1 pgicons 起動後の表示

表示したいアイコンを番号で指定して下さい。

No.	ファイル名	サイズ(バイト)
1	a.gif	246
2	alert.black.gif	242
3	alert.red.gif	247
4	back.gif	216
5	ball.gray.gif	233
6	ball.red.gif	205
7	binary.gif	246
8	binhex.gif	246
9	blank.gif	148
10	bomb.gif	308
11	box1.gif	251
12	box2.gif	268
13	broken.gif	247
14	burst.gif	235
15	c.gif	242
16	comp.blue.gif	251
17	comp.gray.gif	246
18	compressed.gif	1038
19	continued.gif	214
20	dir.gif	225
21	down.gif	163
22	dvi.gif	238
23	f.gif	236
24	folder.gif	225
25	folder.open.gif	242
26	folder.sec.gif	243
27	forward.gif	219
28	generic.gif	221
29	generic.red.gif	220
30	generic.sec.gif	249
31	hand.right.gif	217
32	hand.up.gif	223
33	image1.gif	274
34	image2.gif	309
35	image3.gif	286

36	index.gif	268
37	layout.gif	276
38	left.gif	172
39	link.gif	249
40	movie.gif	243
41	p.gif	237
42	patch.gif	251
43	pdf.gif	249
44	pie0.gif	188
45	pie1.gif	198
46	pie2.gif	198
47	pie3.gif	191
48	pie4.gif	193
49	pie5.gif	189
50	pie6.gif	186
51	pie7.gif	185
52	pie8.gif	173
53	portal.gif	254
54	ps.gif	244
55	quill.gif	267
56	right.gif	172
57	screw1.gif	258
58	screw2.gif	263
59	script.gif	242
60	sound1.gif	248
61	sound2.gif	221
62	sphere1.gif	285
63	sphere2.gif	264
64	tar.gif	219
65	tex.gif	251
66	text.gif	229
67	transfer.gif	242
68	unknown.gif	245
69	up.gif	164
70	uu.gif	236
71	uuencoded.gif	236
72	world1.gif	228
73	world2.gif	261

## 4.2.5 ソースコードについて

pgicons のソース (リスト4.2.4) を例に取りながら, libpq の使い方を解説します。pgicons.c は, PostgreSQL に付属するサンプルコード ( /usr/local/src/postgresql-6.3.2/test/examples/testlibpq3.c ) を改造したものです。

リスト 4.2.4 pgicons.c

```
1: #include <stdio.h>
2: #include "postgres.h"
3: #include "libpq-fe.h"
4:
5: static void
6: exit_nicely(PGconn *conn)
7: {
8:     PQfinish(conn);
9:     exit(1);
10: }
11:
12: int
13: main(int argc, char **argv)
14: {
15:     char          *pgghost,
16:                 *pgport,
17:                 *pgoptions,
18:                 *pgtty;
19:     char          *dbName;
20:     int           i;
21:     PGconn        *conn;
22:     PGresult      *res;
23:     int n;
24:     char buf[128];
25:
26:     /*
27:      * begin, by setting the parameters for a backend connection if the
28:      * parameters are null, then the system will try to use reasonable
29:      * defaults by looking up environment variables or, failing that,
30:      * using hardwired constants
31:      */
32:     pgghost = NULL;          /* host name of the backend server */
33:     pgport = NULL;          /* port of the backend server */
34:     pgoptions = NULL;       /* special options to start up the backend
35:                             * server */
36:     pgtty = NULL;           /* debugging tty for the backend server */
37:
38:     if (argc == 1) {
39:         dbName = getenv("USER");
40:     } else if (argc == 2) {
41:         dbName = argv[1];
42:     } else {
43:         fprintf(stderr, "usage: pgicons [dbname]\n");
44:         exit(1);
45:     }
```



## 4.2 C インターフェースを使って~アイコン管理コマンドの作成

```
46:
47: /* make a connection to the database */
48: conn = PQsetdb(pghost, pgport, pgoptions, pgtty, dbName);
49:
50: /* check to see that the backend connection was successfully made */
51: if (PQstatus(conn) == CONNECTION_BAD)
52: {
53:     fprintf(stderr, "Connection to database '%s' failed.\n", dbName);
54:     fprintf(stderr, "%s", PQerrorMessage(conn));
55:     exit_nicely(conn);
56: }
57:
58: /* start a transaction block */
59: res = PQexec(conn, "BEGIN");
60: if (PQresultStatus(res) != PGRES_COMMAND_OK)
61: {
62:     fprintf(stderr, "BEGIN command failed\n");
63:     PQclear(res);
64:     exit_nicely(conn);
65: }
66:
67: /*
68:  * should PQclear PGresult whenever it is no longer needed to avoid
69:  * memory leaks
70:  */
71: PQclear(res);
72:
73: /*
74:  * fetch instances from the pg_database, the system catalog of
75:  * databases
76:  */
77: res = PQexec(conn, "DECLARE mycursor BINARY CURSOR FOR select * from icons order by
name");
78: if (res == NULL ||
79:     PQresultStatus(res) != PGRES_COMMAND_OK)
80: {
81:     fprintf(stderr, "DECLARE CURSOR command failed\n");
82:     if (res)
83:         PQclear(res);
84:     exit_nicely(conn);
85: }
86: PQclear(res);
87:
88: res = PQexec(conn, "FETCH ALL in mycursor");
89: if (res == NULL ||
90:     PQresultStatus(res) != PGRES_TUPLES_OK)
91: {
92:     fprintf(stderr, "FETCH ALL command didn't return tuples properly\n");
93:     if (res)
94:         PQclear(res);
95:     exit_nicely(conn);
96: }
97:
98: for (;;) {
99:     printf("表示したいアイコンを番号で指定して下さい。 \n\n");
100:    printf("No. \t ファイル名 \t サイズ(バイト) \n\n");
101:
102:    for (i = 0; i < PQntuples(res); i++)
```

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
103:     {
104:     char *name;
105:
106:     name = (char *) PQgetvalue(res, i, 0);
107:     printf("%d\t%s\t\t%d\n", i+1, name, PQgetlength(res, i, 1));
108:     }
109:
110:     printf("\n");
111:
112:     if (fgets(buf, sizeof(buf), stdin) == NULL) {
113:         break;
114:     }
115:     n = atoi(buf);
116:     if (n > 0 && n <= i) {
117:         int len;
118:         char *image;
119:         char fname[1024];
120:         FILE *fd;
121:         char cmd[1024];
122:
123:         n--;
124:         len = PQgetlength(res, n, 1);
125:         image = (char *) malloc(len);
126:         if (!image) {
127:             fprintf(stderr, "malloc failed\n");
128:             break;
129:         }
130:         memmove(image, PQgetvalue(res, n, 1), len);
131:         sprintf(fname, "/tmp/pgicons%d", getpid());
132:         fd = fopen(fname, "w");
133:         if (!fd) {
134:             fprintf(stderr, "couldn't open temp file %s\n", fname);
135:             break;
136:         }
137:         fwrite(image, len, 1, fd);
138:         fclose(fd);
139:         sprintf(cmd, "%s %s", XV, fname);
140:         system(cmd);
141:         unlink(fname);
142:     }
143: }
144:
145: PQclear(res);
146:
147: /* close the portal */
148: res = PQexec(conn, "CLOSE mycursor");
149: PQclear(res);
150:
151: /* end the transaction */
152: res = PQexec(conn, "END");
153: PQclear(res);
154:
155: /* close the connection to the database and cleanup */
156: PQfinish(conn);
157: }
158:
```

## 4.2 C インターフェースを使って～アイコン管理コマンドの作成

libpq を使ったプログラムで必ず include するのが postgres.h と libpq-fe.h です (2, 3 行目)。これらのヘッダファイルは標準では /usr/local/pgsql/include にインストールされます。ここでは、Makefile の中で -I オプションを使って、ソース中にはヘッダファイルのフルパスを書かないようにしています。

8 行目で後始末用に PQfinish( ) を呼んでいますが、PQfinish( ) については後述します。

pgicons は、引数でデータベース名を与えられたときはそれを、引数がないときはユーザ名と同じ名前のデータベースに接続します (38 ~ 45 行目)。データベースへの接続は、PQsetdb( ) で行います (48 行目)。引数は表 4.2.1 の通りです。

データベースへの接続がうまくいったかどうかは PQstatus( ) で調べます (51 行目)。CONNECTION\_OK が返れば正常ですが、CONNECTION\_BAD が返る場合はエラーなので、54 行目で PQerrorMessage( ) を使ってエラーメッセージを表示して終了しています。

pgicons では、BINARY CURSOR を使いますが、PostgreSQL は BINARY CURSOR にかぎらず普通の CURSOR を使う場合でも、トランザクションの内側で実行する必要があるので、まず SQL の BEGIN を実行します (59 行目)。

SQL の実行はすべて PQexec( ) で行います。PQexec( ) が正常終了したかどうかは PQresultStatus( ) でチェックします。PQresultStatus( ) の返却値は表 4.2.2 のうちのどれかです。

表 4.2.1 PQsetdb( ) の引数一覧

pghost	接続ホスト名。NULL を渡すと UNIX ドメインソケットでローカルホストに接続
pgport	接続ポート番号。NULL を渡すとデフォルトのポート番号 (5432) を使う
pgoptions	バックエンドへの接続オプション。必要なければ NULL を渡す
pgtty	デバッグ用 TTY。必要なければ NULL を渡す
dbName	データベース名

表 4.2.2 PQresultStatus( ) の返却値

PGRES_EMPTY_QUERY	empty query (空白文字列の問い合わせ) が終了した
PGRES_COMMAND_OK	(SELECT のように) 返却値がないコマンドが正常終了した
PGRES_TUPLES_OK	SELECT が正常に終了した
PGRES_COPY_OUT	COPY コマンド (出力) が終了した
PGRES_COPY_IN	COPY コマンド (入力) が終了した
PGRES_BAD_RESPONSE	バックエンドが異常な値を返却した
PGRES_NONFATAL_ERROR	(致命的でない) エラー
PGRES_FATAL_ERROR	致命的なエラー

63行目では、`PQclear()`を呼び出しています。`PQexec()`を呼んだ後、そのSQLコマンドの実行結果が不要になったら必ず`PQclear()`を呼んでメモリを解放してください。

77行目では、`BINARY CURSOR`を宣言しています。これでバイナリ値が取得できるようになります。バイナリ値が取得できる点、また文字列への変換のオーバーヘッドがない点において`BINARY CURSOR`は優れていますが、気を付ける点もあります。`BINARY CURSOR`はバックエンドのデータベースの中のデータをそのまま取り出すため、バイトオーダーや浮動小数点の表現など、プラットフォームに依存するデータをプラットフォームが異なるクライアントで受け取った場合にはデータが無意味になることもあります。

88行目では`SELECT`で選択したデータをすべて`FETCH`しています。この場合、`FETCH`は単独の`SELECT`同様、タプルを取得するコマンドですから、`PQresultStatus()`のチェックも`PGRES_TUPLES_OK`が返っているかどうかになります。

102行目からのループは、端末にアイコンファイル名の一覧表を出力する処理を行います。ループを回る回数は、`PQntuples()`が返す取得したタプルの数分です。

106行目の`PQgetvalue()`は、指定タプル/カラムのデータを返す関数です。`PQgetvalue()`の引数は以下です。

- `PQexec()`の返り値
- 取得したデータ中のタプル番号
- カラム番号

タプル番号、カラム番号は0から数えます。

107行目では、`printf()`で行番号、アイコンファイル名、アイコンファイルの長さを出力しますが、アイコンファイルの長さは`PQgetlength()`で取得しています。

112行目でユーザが入力した番号を取り込みます。115行目で文字列表現の番号を`int`に変換し、その値が適切かどうかチェックした後、まず124行目で該当アイコンファイルの大きさを知ります。そしてその分のメモリを`malloc()`で取得し、130行目でそのアイコンのバイナリ値をメモリに取り出します。

131行目からは`xv`コマンドを使ってアイコンを表示するための処理です。`xv`では表示データをファイルに置いておく必要があるため、そのファイルを`/tmp`の下に作ります。適当なファイル名にすると名前がぶつかる可能性があるので、`getpid()`で取り出した自分のプロセス番号を使ってランダムな名前を生成します。

132～138行目で一時ファイルにアイコンの中身を書き込んでいます。

そして140行目で、`system()`関数を使って`xv`を起動し、アイコンを表示します。`xv`

## 4.2 C インターフェースを使って～アイコン管理コマンドの作成

が終了すれば、system( ) から戻って来ますので、その後 unlink( ) で不要になった一時ファイルを削除します。

ユーザが行番号の代わりに **CTRL** + **D** で EOF (End of file) を入力した場合は 113 行目で for ループを脱出し、145 行目に行きます。

148 行目でカーソルを閉じ、152 行目でトランザクションを終了します。最後に、PQfinish( ) を呼んでデータベースとの接続を切断します。

### 4.2.6 その他の libpq 関数

前章のサンプルアプリケーションではいくつかの PostgreSQL 用の関数について述べましたが、ここではそれ以外の libpq 関数を紹介します。

```
.....
PGconn *PQsetdbLogin(char *pghost,
                     char *pgport,
                     char *pgoptions,
                     char *pgtty,
                     char *dbName,
                     char *login,
                     char *pwd);
```

➡ データベースに接続するための関数です。PQsetdb( ) と違い、ユーザ認証が設定されている場合に使用します。pghost から dbName までは PQsetdb( ) と同じです。違いは login にユーザ名、pwd にパスワードを設定するところです。

```
.....
char *PQdb(PGconn *conn)
```

➡ 現在接続しているデータベース名を返す関数です。

```
.....
char *PQhost(PGconn *conn)
```

➡ 現在接続しているホスト名を返す関数です。

```
.....
PQOptions(PGconn *conn)
```

➡ 現在の接続のオプションを返します。

```
.....
char *PQport(PGconn *conn)
```

➡ 現在接続しているポート番号を返します。

```
.....
char *PQtty(PGconn *conn)
```

⇒ 現在接続している tty を返します .

```
.....
int PQnfields(PGresult *res);
```

⇒ 問い合わせ結果のカラム数を返します .

```
.....
char *PQfname(PGresult *res, int field_index);
```

⇒ カラム番号 ( 0 から開始 ) に対応したカラム名を返します .

```
.....
int PQfnumber(PGresult *res, char* field_name);
```

⇒ カラム名に対応するカラム番号を返します .

```
.....
Oid PQftype(PGresult *res, int field_num);
```

⇒ カラム番号に対応したデータの型 ( oid : object id ) を返します . oid は単なる数字なので、実際どのような型に対応しているのかは、システムカタログのひとつである pg\_type テーブルを検索します . たとえば oid 18 に対応する型名を探すには、  
 select typename from pg\_type where oid = 18;  
 を実行します<sup>注3</sup> .

```
.....
int2 PQfsize(PGresult *res, int field_num)...6.3.2の場合
```

```
short PQfsize(PGresult *res, int tuple_index, int field_index)...6.4の場合
```

⇒ PQfsize( ) も PQgetlength( ) 同様、指定タプル / カラムの大きさを返しますが、text や varchar 型のように可変長データ型の場合は -1 が返ります . PQgetlength( ) では、可変長データ型の場合は実際のデータ長が返ります .

```
.....
int PQfmod(PGresult *res, int field_index);
```

⇒ そのカラムに対応した型修飾データ ( type-specific modification data ) を返します . 残念ながら、これがどのように使われるのか筆者にはよくわかりません .

```
.....
int PQgetisnull(PGresult *res, int tup_num, int field_num);
```

⇒ 指定タプル / カラムのデータが NULL なら 1 を、そうでないなら 0 を返します .

```
.....
char* PQoidStatus(PGresult *res);
```

⇒ INSERT 文を実行すると、新しく作られたタプルに oid が割り当てられますが、その oid を返す関数です .

注3  
 この場合 “ char ” が返  
 ってきます .

## 4.2 C インターフェースを使って～アイコン管理コマンドの作成

```
.....
void PQprint(FILE* fout,PGresult* res,PQprintOpt *ps);
    ➔ PQprintOpt *ps の指定に従い、検索結果を fout に文字列で出力します。
    PQprintOpt はリスト 4.2.5 の構造体です。
.....

int PQrequestCancel(PGconn *conn);
    ➔ 6.4 から新しく追加された関数で、実行中の問い合わせを途中でキャンセルします。
    キャンセルに成功すれば TRUE、失敗すれば NULL が返ります。
.....
```

ここで紹介した以外にも表 4.2.3 の機能を持つ関数があります。これらについてはオンラインマニュアルかドキュメントをご覧ください。

リスト 4.2.5

```
typedef struct _PQprintOpt {
    bool header;          /* ヘッダとタブル数を印字するかどうか */
    bool align;           /* カラムを印字する際に桁そろえを行うかどうか */
    bool standard;        /* 古い形式で出力する (align も true にすること) */
    bool html3;           /* HTML のテーブル形式で出力 */
    bool expanded;        /* 端末のスクリーンいっぱい出力 */
    bool pager;           /* pager を使う */
    char *fieldSep;        /* フィールドの区切り文字 */
    char *tableOpt;        /* HTML 形式の際、<table ...> に挿入する
                           文字列 */
    char *caption;         /* HTML で出力する際、caption を
                           <centre><h2>caption</h2></centre> として
                           追加する (NULL なら出力しない) */
    char **fieldName;      /* カラム名のリスト (NULL なら出力しない) */
} PQprintOpt;
```

表 4.2.3 その他の libpq 関数

Fast Path	バックエンドの関数を呼び出す。セキュリティホールになる可能性があるため、使用は推奨されていない
非同期通知	あらかじめイベントを登録しておき、そのイベントが起こった際に通知を受ける
COPY コマンドサポート	psql の \copy で使われているもの
デバッグ支援機能	フロントエンド/バックエンドのコミュニケーションをトレースする
ユーザ認証	認証に関する細かな制御を行う
large object	large object のハンドリング

## 4.2.7 非同期処理関数

PQexec( )を実行すると、バックエンドの処理が終了するまで関数から戻って来ません。場合によっては、バックエンドの終了を待たずにアプリケーション側で他の処理を行いたい場合もありますが、このような場合に非同期処理関数を使います。

```
.....
PQsendQuery(PGconn *conn, const char *query);
```

➡ PQexec( )の代わりに使用する非同期処理関数です。PQexec( )と違い、PQsendQuery( )はただちに関数から戻って来ます。PQsendQuery( )の結果は必ずPQgetResult( )で取得します。

```
.....
PGresult *PQgetResult(PGconn *conn);
```

➡ PQgetResult( )は、PQsendQuery( )の結果が返ってくるのを待ちます。PQgetResult( )からの戻りは、前述の関数で処理できます。気を付けなければならないのは、PQgetResult( )は一度にすべての結果を返すとは限らないことです。PQgetResult( )がNULLを返すまで、繰り返しPQgetResult( )を呼ばなければなりません。また、PQgetResult( )がNULLを返すまでは次のPQsendQuery( )の呼び出しを行ってはいけません。

ところでPQgetResult( )だけでは、処理結果を得る際にブロックされてしまう可能性があります。そこで以下の関数を使うと、ブロックされることなくバックエンドの処理結果を取り出すことができます。

```
int PQsocket(PGconn *conn)が、返すソケットをselect(2)で調べ、データがあるかどうか確認する。
```

```
データがあれば、void PQconsumeInput(PGconn *conn)を呼び出す。
```

```
int PQisBusy(PGconn *conn)がFALSEを返せばPQgetResult()を呼び出す。
```

```
.....
```

## 4.2.8 感想など

実際には使う機会の少ないlibpqですが、PostgreSQL フロントエンドの基礎とも言える大事なライブラリです。パイナリを直接扱ったり、特殊な機能やパフォーマンスが要求される場合はやはりlibpqの出番となります。ただ、libpqのインターフェースはお世辞にも使いやすいとは言えないので、本書では触れていませんが、同じC言語を使うのならecpgを試してみるとよいかもしれません。



# 4-3 PHP インターフェース を使って Apache など WWW サーバとの連携

### 4.3.1 WWW とは

Netscape Navigator などの Web ブラウザは、今ではインターネットをアクセスするのに欠かせないツールとなっています。Web ブラウザは WWW サーバと呼ばれるサーバにネットワークを経由してアクセスし、必要な情報を取得します。取得されたデータには文字や画像など多彩な形式がありますが、すべて HTML という言語により統一された指示に従っているので、どんな Web ブラウザ製品でもほぼ同じ外見でデータを表示することができます。

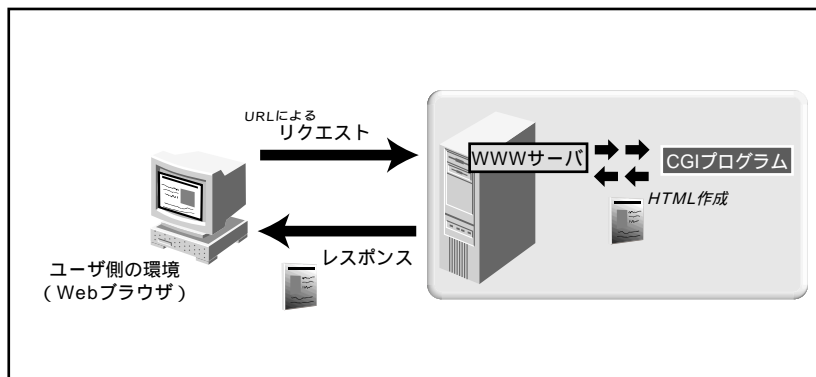
### 4.3.2 CGI

このように便利な HTML ですが、あらかじめファイルとして WWW サーバに置いておく方法では、いつも決まったデータしか表示できません。アクセスするときに動的に HTML のデータを生成する方法がいくつかあり、そのひとつが CGI (Common Gateway Interface) です。CGI では、ブラウザからデータの住所である URL (Uniform Resource Locator) を指定する際にプログラムを指定します。指定されたプログラムは WWW サーバから起動され、HTML データを生成して WWW サーバに返します。そして WWW サーバはそのデータを Web ブラウザに送ります。このデータも HTML 形式になっているので、Web ブラウザで表示することができます (図 4.3.1)。したがって、PostgreSQL を使ってデータベースをアクセスする CGI を作れば、Web ブラウザを使ってデータベースを検索することができるようになります。

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

図 4.3.1 CGI の仕組み



### 4.3.3 サーバサイドスクリプトとしての PHP

CGI を使う方式では、HTML と CGI のプログラムがまったく別に管理されるので、Web ブラウザに表示される結果を変更しようとしても、どこを直したらよいのかわかりにくいことがあります。また、CGI プログラムが C 言語などで書かれている場合、ソースプログラムを修正したらそのつどコンパイル / リンクする必要があります。

そこで、HTML ファイルの中にプログラム自体を埋め込んでしまおうというのが「サーバサイドスクリプト」と呼ばれる方式です。これなら HTML とプログラムの関係が一目瞭然です。

サーバサイドスクリプトを埋め込んだ HTML ファイルは、WWW サーバによって適当な解釈プログラムが呼び出されることによってスクリプトが実行され、結果として「普通の」HTML データが生成されます。したがって、サーバサイドスクリプトが埋め込まれた HTML であっても、Web ブラウザの利用者からは特別に意識することはありません。

サーバサイドスクリプトにもいくつかの種類がありますが、ここでは最も普及していると思われる PHP を取り上げます。

### 4.3.4 PHP とは

PHP (PHP: Hypertext Preprocessor) は、HTML ファイルに埋め込んで使えるスクリプト言語です。無償で提供され、ライセンス条件に従うかぎり再配布も可能で

## 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

す。ライセンス条件は、GNU General Public License (GPL) か、PHP License のどちらかを選択できます。ライセンス条件の詳細は、ソース付属のファイルに書いてあります。GPL についてはCOPYING、PHP License についてはLICENSE をそれぞれご覧ください。

本書執筆時点（1998年10月中旬）でのPHPの最新バージョンは3.0.5です。旧バージョンのPHP/FI (Personal Home Page Construction Kit/Form Interpreter) は、現在では開発が終了しており、サポートも行われていません。PHPは文字コードとしてEUCコードを使うかぎり、正規表現やファイルのアップロードなどの一部の機能を除いて日本語が使用できますし、安定性や機能、そして性能面でPHP/FIに比べてメリットがあるようです。そこで、本書でもPHP/FIではなくPHPを採用します。

PHPには以下のような特徴があります。

### 初心者にも覚えやすい簡単な構文

PHP/FIの構文はPerlをずっと簡単にしたようなものです。PerlがCを知っている人なら、すぐにでも使いはじめることができるでしょうし、そうでない人でも容易に習得できるはずです。

### デバッグが楽

構文エラーも含め、エラーが発生するとエラーメッセージがブラウザに表示されるので原因がすぐにわかります。CGIの開発で苦労された方はこのありがたみがわかると思います。

### PostgreSQLをサポート

PHPはPostgreSQLをはじめとする各種データベースをサポートします。Oracle、Sybase、Infomixなどの商用データベースへのインターフェースも備えています。

### 使いやすく高性能な Apache モジュール

PHPは単独のCGIプログラムとしても利用できますが、Apacheの「モジュール」として組み込むことにより効果的に使うことができます。ApacheはフリーなWWWサーバで、現在稼働中の世界中のWWWサーバの4割以上がApacheであると言われており、事実上WWWサーバの標準ソフトと言えます（図4.3.2）。モジュールとして組み込まれたPHPはApacheから関数として呼び出されるので、CGIのように外部プロセスを起動するオーバーヘッドが発生しません。また、モジュールとしてのPHPはCGIよりも使い方が簡単です。PHPスクリプトを埋め込んだHTMLファイルは“.php3”

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

図 4.3.2  
Apache の公式 Web サイト  
<http://www.apache.org/>



## 注 1

拡張子はApacheの設定  
ファイルで.php3以外に  
することもできます。

という拡張子持つ以外は<sup>注1</sup>，通常のHTMLファイルとまったく同じように扱うことができます。

### 4.3.5 PHP スクリプトの書き方

それでは，PHP スクリプトの書き方を簡単に紹介します。

#### 開始タグと終了タグ

PHPはHTMLファイル中にスクリプトを記述します。HTMLの普通のタグと区別するために，特別なタグを使用します。PHP スクリプトの開始と終了を表すタグには以下の3種類があります。

```
<? ... ?>
<?php ... ?>
<script language="php"> ... </script>
```

どれを使ってもよいのですが，本書では `<?php ... ?>` を使います。開始タグと終了タグのペアは1つのHTMLファイル中に何回現れてもよいのですが，開始タグと終了タグは必ず対応していなければなりません。

開始タグと終了タグは同じ行に書いてある必要はありません。たとえば次の例はす

## 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

べて同じように扱われます<sup>注2</sup>。

```
<? echo ("こんにちは"); echo ("こんばんは"); ?>
```

```
<? echo ("こんにちは");  
echo ("こんばんは"); ?>
```

```
<?  
    echo ("こんにちは");  
    echo ("こんばんは");  
?>
```

注 2

PHP では、プログラム  
行の終わりは、(セミコ  
ロン) です。

### 大文字 / 小文字

PHP は、言語のキーワード、関数名の大文字 / 小文字を区別しません。たとえば、

```
echo ("こんにちは");  
Echo ("こんにちは");  
eCho ("こんにちは");  
ECH0 ("こんにちは");
```

はどれも同じです。ただし、変数名は大文字 / 小文字が区別されます。

### コメント

コメントは C 言語式の `/* ... */`、または C++ 式の `//` の両方が使えます。

```
/* ここはコメントです */
```

```
//ここはコメントです
```

### 変数

PHP の変数は `$` で始まります。変数には `int` (整数) / `double` (浮動小数点) / `string` (文字列) / 配列などの型がありますが、その型は代入時に自動的に決定され

まず、明示的に型を指定する場合は、型名を( )で囲ったキャストを使います。また、`settype( )`という関数を使って型を変数に設定することもできます。

基本的に変数を使うための宣言は前もって必要ありません。宣言なしに使われた変数はグローバル変数となり、そのPHP ファイル内のどこからもアクセスできます。ただし、後述のユーザ定義関数内で宣言された変数はその関数の中だけで有効なローカル変数になります。同じ名前のグローバル変数があった場合はローカル変数が優先されます。関数の中でグローバル変数を使う場合は、その変数を明示的に宣言する必要があります。

関数の中では、`static` 変数を宣言することができます。`static` 変数はC 言語の `static` 変数と同じで、関数が終了してもその値を失いません。

変数名自体を変数とすることもできます。たとえば、

```
$a = "hello";
```

とすると、`$a` という変数が生成されてその内容は `"hello"` となりますが、さらに、

```
$a = "world";
```

とすると、`$hello` という変数が生成され、その内容は `"world"` となります。つまり `#{ $a }` という表現は、まず `{ $a }` が評価されて `"hello"` となり、次に `$hello` が評価されるので、結局 `"world"` となります。

## 配列

PHP は、多次元の配列をサポートします。配列の添字は数字でも文字列でもかまいません<sup>注3</sup>。

```
$a[0] = 5;
```

```
$a[1]["test"] = 17;
```

配列に配列を代入することもできます。

```
$b["foo"] = $a;
```

配列は関数 `array( )` を使って初期化できます。たとえば

```
$week = array("月","火","水","木","金","土","日");
```

この例では

### 注 3

配列の添字に文字列を使ったものは特に「連想配列」と呼ばれます。

## 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

```
$week[0] = "月"  
$week[1] = "火"  
...
```

となります。配列の1から初期化したい場合は、

```
$week = array(1=>"月","火","水","木","金","土","日");
```

とします。ほかにも=> は

```
$week = array("Mon"=>"月","Tue"=>"火","Wed"=>"水","Thu"=>"木"  
,"Fri"=>"金","Sat"=>"土","Sun"=>"日");
```

のような使い方もできます。

arrayのほか、いくつか配列に関する関数が提供されています(表4.3.1)。

表 4.3.1 配列に関する関数

asort	連想配列をソートする
arsort	連想配列を逆順にソートする
count	配列の要素数を返す
current	現在の配列要素を返す
end	内部的なポインタを配列の最後の要素に移す
key	連想配列からキーを得る
ksort	連想配列をキーでソートする
list	変数のリストに一度に値をセットする
next	内部的なポインタを配列の次の要素に移す
pos	currentと同様
prev	内部的なポインタを配列の前の要素に移す
reset	内部的なポインタを配列の先頭の要素に移す
rsort	配列を逆順にソートする
sizeof	countと同様
sort	配列をソートする

## 関数

PHP の関数は以下の構文です。

```
function foo($bar,$buz) {  
    return $bar.$buz;  
}
```

この関数は引数を結合した文字列を返します。ちなみに、`.` は文字列を結合する演算子です。

関数の呼び出しは

```
foo("acb","def");
```

のように行いますが、「関数へのポインタ」も使えます。

```
$myfunc = "foo";  
$myfunc("acb","def");
```

## 演算子

C 言語と同じような演算子がひと通り揃っています。以下に例を挙げておきます。

```
$a+$b  
$a++;  
$a--;  
++$a;  
--$a;  
$a += 2;  
$a *= 3;  
$a = $b = 10;
```

## if 文

if 文の書き方も C 言語と似ています。



## 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

```
if ($a > $b)
    echo $a;

if ($a > $b) {
    echo $a;
} elseif ($a < $b) {
    echo $b;
} else {
    echo "What?";
}
```

if 文のブロックを、別々のPHP のスクリプトのブロックに書くこともできます。

```
<?
if ($a > $b):
?>
<font color="red">a is bigger than b.</font>
<?
elseif ($a < $b):
?>
<font color="blue">a is smaller than b.</font>
<?
else:
?>
<font color="green">What?</font>
<?
endif;
?>
```

この書き方は後述の while... と for でも使えます。

### while 文

条件が真の間繰り返します。

```
while ($a > 1) {
```

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
    echo "a is bigger than 1"
    $a--;
}

```

以下のような書き方もできます。

```
<?
while ($a > 1): ?>
<h1>a is bigger than 1</h1>
<? $a--;
endwhile; ?>

```

ループを脱出する `break` , 継続する `continue` もあります。なお, `break` には脱出するレベルを指定する引数が指定できます。たとえば,

```
break 2

```

は二重のループを脱出します。

### do...while 文

条件が偽になるまで繰り返します。

```
do {
    echo "a is bigger than 1"
    $a--;
} while ($a > 1);

```

### for 文

C の `for` 文と同じです。

```
for ($i=0;$i<10;$i++) {
    echo "\$i is $i<br>";
}

```

`endfor` を使うこともできます。

## 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

```
<? for ($i=0;$i<10;$i++): ?>
    <font color="green">i is smaller than 10</font>
<? endfor; ?>
```

### case 文

これもCのcase文と同じです。

```
switch ($i) {
case 0:
    print "0";
    break;
case 1:
    print "2";
    break;
default:
    print "other";
}
```

### class 文

PHP はclassをサポートするオブジェクト指向言語でもあります<sup>注4</sup>。

```
class myclass {
    var $a;
    var $b = "initialized";

    function show() {
        printf("\$a is %s \$b is %s\n", $this->a, $this->b);
    }
}
```

クラスmyclassは変数\$aと\$bを持ち、メンバ関数のshow()を持ちます。\$thisは自身のインスタンスオブジェクトを指します。クラスのインスタンスはnewで作ります。

---

注 4  
と言い切るにはちょっと抵抗がありますが...

```
$obj = new myclass;
$obj->show();
$obj->a = "assigned";
$obj->show();
```

PHP は継承もサポートします。

```
class anotherClass extends myclass {
    function show() {
        printf("<font color='red'>\$a is %s \$b is %s\n</font>", $this->a, $this->b);
    }
}
```

PHP では、派生したクラスに同じ名前の変数やメンバ関数があると、それは派生クラスのものにオーバーライドされます。

## 組み込み関数

PHP には各種組み込み関数が用意されています。たいへん数が多いので、ここでは一例を紹介するにとどめます。詳しくはドキュメントをご覧ください。なおPostgreSQL 関係の関数については後述します。

### print 関係の関数

Web ブラウザに結果を表示するための関数として、echo、print、printf が用意されています。

echo は、カンマ区切りのstring 型の引数を出力します。引数の数に制限はありません。

```
echo("abc", $a);
```

print は、echo と同じように文字列を出力しますが、引数の数は1 個だけです。printf は、C 言語のprintf と同じです。フォーマット文字列と、可変個のデータを引数としてとります。

```
printf("%d %s\n", 10, "abc");
```

このほか、文字を出力するのではなく、編集した文字列を返すsprintf() という関数があります。引数はprintf と同じです。

### 4.3.6 PHP のインストール

ここでは執筆時点（1998年10月中旬）で最新のバージョンPHP（バージョン3.0.5）をインストールします。PHPは単独で使うものではなく、CGIとして、あるいは前述したApacheと一緒に「モジュール」として使います。CGIがWebサーバ本体とは別のプロセスとして動作するのに対し、モジュールとしてのPHPはApacheのプロセスに組み込まれて関数として呼び出されるため、プロセス起動のオーバーヘッドがなく、高い性能が期待できます。そこで本書でもモジュールとしてPHPをインストールすることにします。Apacheの方も、最新バージョンの1.3.2を使います。

#### 準備

PostgreSQL 6.3.2または、6.4をあらかじめ /usr/local/pgsql/ にインストールしておきます。

FreeBSD とSolarisの場合はgdbmもあらかじめインストールしておきます。FreeBSDならパッケージがありますし、Solarisの場合はftp://ftp.iiij.ad.jp/pub/GNU/あたりから最新と思われるgdbmを取ってきてコンパイルすればいいでしょう。

Solarisの場合、GNU tarをインストールしておくとう便利です。以下、本書で使用するtarはGNU tarであると考えてください。

#### 入手とコンパイル

準備ができたならさっそくインストールに取りかかりましょう。まずはsu コマンドなどでrootユーザになってください。

```
% su
```

PHP とApacheのソースは、本書付属のCD-ROMのarchivesディレクトリに以下のファイル名であります。

```
apache_1.3.2.tar.gz
php-3.0.5.tar.gz
```

インターネット上の入手先は以下です。

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

- PHP...<http://www.php.net/>
- Apache...<http://www.apache.org/>

/usr/local/src/の下にApacheのソースを展開し、configure します。ここでのconfigureは仮のもので、後でもう一度configureし直します。

```
# cd /usr/local/src
# tar xzf /mnt/cdrom/archives/apache_1.3.2.tar.gz
# cd apache_1.3.2
# ./configure
```

次にPHPをコンパイルします。

```
# cd /usr/local/src
# tar xzf /mnt/cdrom/archives/php-3.0.5.tar.gz
# cd php-3.0.5
# ./configure --with-pgsql --with-apache=../apache_1.3.2 --enable-track-vars
# make
# make install
```

make installで、/usr/local/src/apache\_1.3.2/src/modules/php3/ というディレクトリができ、その中に以下のファイルがコピーされるはずです。

```
Makefile.libdir  libmodphp3.a      mod_php3.c      php_version.h
Makefile.tmpl    libphp3.module  mod_php3.h
```

## Apache のコンパイル&インストール

環境変数LD\_LIBRARY\_PATHを設定します。cshやtcshなら

```
# setenv LD_LIBRARY_PATH /usr/local/pgsql/lib
```

shやbashなら

```
# LD_LIBRARY_PATH=/usr/local/pgsql/lib
# export LD_LIBRARY_PATH
```

としてください。

```
# cd /usr/local/src/apache_1.3.2
```

## 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

図 4.3.3 Apache インストール時のエラーメッセージ

```
modules/php3/libphp3.a(dns.o): In function `php3_checkednsrr':
/hda10/local/src/php-3.0.5/functions/dns.c:212: undefined reference to `__res_search'
modules/php3/libphp3.a(dns.o): In function `php3_getmxrr':
/hda10/local/src/php-3.0.5/functions/dns.c:279: undefined reference to `__res_search'
/hda10/local/src/php-3.0.5/functions/dns.c:288: undefined reference to `__dn_skipname'
/hda10/local/src/php-3.0.5/functions/dns.c:294: undefined reference to `__dn_skipname'
/hda10/local/src/php-3.0.5/functions/dns.c:306: undefined reference to `__dn_expand'
collect2: ld returned 1 exit status
```

```
# ./configure --activate-module=src/modules/php3/libphp3.a
# make
```

ここで、筆者の環境では図4.3.3のようなエラーメッセージが出ました。これはPHP側のバグのようです。\_\_res\_searchのような関数は、libresolv.aに定義されており、PHPのconfigureがこのことを検出してくれるはずなのですが、それがうまくいってません。そこで、apache\_1.3.2/src/Makefileの42行目付近の

```
LIBS1=-Lmodules/php3 -L../modules/php3 -L../../modules/php3 -
lmodphp3 -lgdbm -L/usr/local/pgsql/lib -lpq
-lm -ldl -lcrypt -lnsl -lm -lcrypt
```

の最後に-lresolvを追加します<sup>注5</sup>。

```
LIBS1=-Lmodules/php3 -L../modules/php3 -L../../modules/php3 -
lmodphp3 -lgdbm -L/usr/local/pgsql/lib -lpq
-lm -ldl -lcrypt -lnsl -lm -lcrypt -lresolv
```

そして、

```
# make
# make install
```

を実行します。今度はうまくいくはずですが、make installで/usr/local/apache/にApacheがインストールされます。

別の方法としては、/usr/local/src/php-3.0.5/libphp3.moduleを修正する方法があります。

```
LIBS="-Lmodules/php3 -L../modules/php3 -L../../modules/php3 -
lmodphp3 -lgdbm -L/usr/local/pgsql/lib -lpq
```

注 5

42行目の内容はプラットフォームによって異なります。

## 注 6

これも細かい内容はプラットフォームによって異なります。

```
-lm -ldl -lcrypt -lnsl $LIBS"
```

のような行<sup>注6</sup>の\$LIBSの前に-lresolvを入れ、次のようにします。

```
LIBS="-Lmodules/php3 -L../modules/php3 -L../../modules/php3 -
lmodphp3 -lgdbm -L/usr/local/pgsql/lib -lpq
-lm -ldl -lcrypt -lnsl -lresolv $LIBS"
```

そして/usr/local/src/php-3.0.5でmake installした後、前述のconfigureからやり直します。今後モジュールの追加などでApacheのコンパイルをやり直す可能性がある場合は、こちらの方法のほうがよいでしょう。

## 設定ファイルの調整

```
# cd /usr/local/src/php-3.0.5
# cp php3.ini-dist /usr/local/lib/php3.ini
```

/usr/local/apache/etc/srm.confの以下の2行の先頭のコメント#を外し、.phtmlになっているのを.php3に変更します<sup>注7</sup>。

つまり、

```
#AddType application/x-httpd-php3 .phtml
#AddType application/x-httpd-php3-source .phps
```

を以下のようにしてください。

```
AddType application/x-httpd-php3 .php3
AddType application/x-httpd-php3-source .phps
```

これで.php3という拡張子が付いたファイルは、PHPのスクリプトを含んだファイルと見なされます。

一方.phpsもPHPのスクリプトを含んだファイルと解釈されますが、.php3のと違って、PHPスクリプトが実行されるのではなく、PHPのソースをそのまま表示します。このとき、PHP言語のキーワードがハイライトされて表示されます。.phpsは、PHPのソースを公開したいときなどに使用します。

次に、同じくsrm.confのDirectoryIndexにindex.php3を追加します。

```
DirectoryIndex index.html index.php3
```

## 注 7

PHP3のドキュメントをphtmlの拡張子のままで使いたい場合は変更しません。



## 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

これにより, `http://www.foobar.co.jp/` のように URL としてディレクトリを指定したとき, まず `index.html`, 次に `index.php3` の順にファイルがあるかどうか調べ, あればそのファイルを表示するようになります.

また Solaris では, 同じディレクトリにある `httpd.conf` の `ServerName` の先頭のコメント `#` を外す処理が必要でした.

```
ServerName foo.bar.co.jp
```

### Apache の起動

さっそく Apache を起動してみましょう. 起動は, Apache 付属のコマンドである `apachectl` を使います.

```
# /usr/local/apache/sbin/apachectl start
apachectl start: httpd started
```

このようになれば Apache の起動はうまくいっています. 適当なブラウザで `http://localhost/` という URL を表示して, 図 4.3.4 のような画面が表示されれば, とりあえず Apache のインストールは成功です.

図 4.3.4  
Apache の起動画面

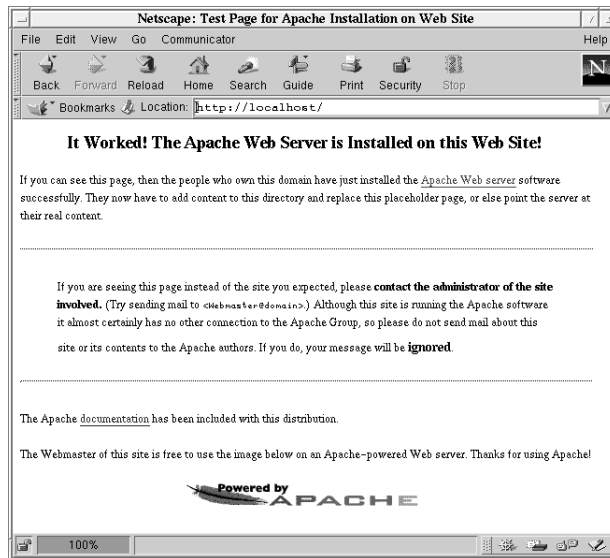
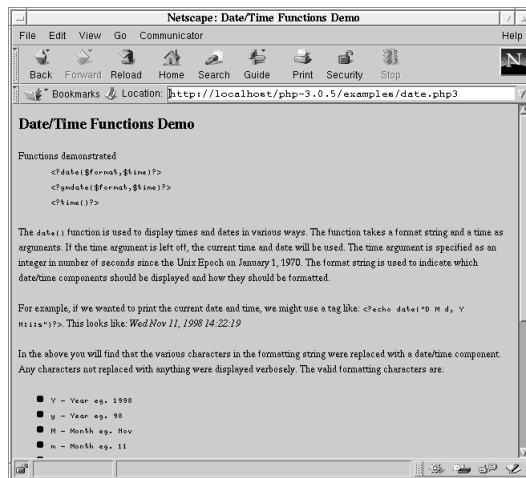


図 4.3.5  
PHP の動作確認



## サンプル

PHP3 にはサンプルスクリプトが付属しています (src/examples/ )。これを表示するためには、このディレクトリがApache から<sup>たど</sup>辿れるようにしなければなりません。一番簡単な方法はリンクを張ることです。

```
# cd /usr/local/apache/share/htdocs/
# ln -s /usr/local/src/php-3.0.5 .
```

とりあえず、適当なブラウザで <http://localhost/php-3.0.5/examples/date.php3> を開いてみましょう。図4.3.5 のような画面が出ればPHP3 は動いています。

## 4.3.7 ドキュメント

Apache のドキュメントはApache 初期画面の下の方に “ documentation ” というリンクがあるのでここから参照できます。

PHP のドキュメントはソースにも付属していますが ( doc ), SGML で書いてあり、HTML に翻訳するためにはjade やdocbook の環境を整える必要があってちょっと面倒です。てっとり早いのはPHP ページのオンラインマニュアルを参照することですが、

## 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

ローカルにマニュアルを持ちたい場合は、PHP のサイトからHTML やPDF など、各種形式のドキュメントが入手できます。

また、PHP の日本語ページ<sup>注8</sup>では、PHP マニュアルの翻訳プロジェクトが進行中ですのので、チェックしてみるとよいでしょう。

注 8

<http://www.city.fujisawa.ne.jp/%7Elouis/apps/phpfi/>

### 4.3.8 新旧郵便番号検索システム

本節では、PHP を使ったアプリケーションとして、5桁の旧郵便番号から7桁の郵便番号を検索するシステムを作ってみましょう。

#### 新旧郵便番号検索システムについて

郵政省では、郵便番号が7桁に移行するのに伴い、11万件に及ぶデータベースを無償で公開しています。このデータベースには新旧郵便番号のほか、住所なども含まれており、郵便番号を調べるだけでなく、読みのわからない地名を調べるのにも役立ちます。また、全国の「銀座」が付く地名を探すといった変わった使い方もできます。

このシステムは以前PostgreSQLのメーリングリストで公開したのですが、そのときはPHP/FIで作成していました。そこで今回、PHP3とApache1.3.2に対応するように改造しました。また、新たな試みとして、ユーザ認証機能を盛り込んでみました。これにより、nobody以外のユーザでデータベースにアクセスできるようになります。今回のシステムでは検索のみなのであまり意味はありませんが、データの更新を伴うようなアプリケーションではユーザ認証が必須なので、ここで使われている技術が役に立つと思います。

新旧郵便番号検索システムの画面を表示すると、図4.3.6のように検索キーを入力する画面になります。「旧郵便番号」に古い郵便番号を入力し「検索開始」をクリックすると検索結果が次の画面に表示されます（図4.3.7）。検索結果は非常に多数表示される可能性があるため、一定件数で表示を打ち切っています<sup>注9</sup>。同じ要領で新郵便番号から検索することもできます。

郵便番号が不明なときは、都道府県（プルダウンメニュー）、市町村名、町域名でも検索できます。複数の項目を指定すれば、AND条件で検索します。市町村名は全部入力する必要はありません。たとえば、横浜市都築区なら「横浜市」まででも検索できます。町域名についても同じことが言えます。「都築区」のように、途中の文字列で検索したい場合は、\*都築と入力します。

注 9

この打ち切り件数は変更できます。

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

図 4.3.6 郵便番号検索画面

新／旧郵便番号検索システム

旧郵便番号: 249

新郵便番号:

都道府県: 指定なし

市町村名:

市町村名(カナ):

市町村名(カナ):

検索結果表示上関件数: 100

検索開始 [検索キャンセル]

データベース: postal

接続ホスト:

接続ポート番号: 5432

ユーザ認証: ☐

デバッグオプション: ☐

図 4.3.7 検索結果の表示

検索結果

9件中9件を表示しました。

旧郵便番号	新郵便番号	都道府県	市町村	町域名	市町村(カナ)	町域名(カナ)
249-0000	249	神奈川県	鎌倉市	以下に該当がない場合	ズシシ	イカニワサイガイバパイ
249-0001	249	神奈川県	鎌倉市	大井	ズシシ	ヒサギ
249-0002	249	神奈川県	鎌倉市	山の根	ズシシ	ヤマノネ
249-0003	249	神奈川県	鎌倉市	辻子	ズシシ	イタゴ
249-0004	249	神奈川県	鎌倉市	辻間	ズシシ	タママ
249-0005	249	神奈川県	鎌倉市	鶴山	ズシシ	サカサヤマ
249-0006	249	神奈川県	鎌倉市	新堀	ズシシ	ズシ
249-0007	249	神奈川県	鎌倉市	新堀	ズシシ	シンジュク
249-0008	249	神奈川県	鎌倉市	小坪	ズシシ	コツボ

接続先のホスト, データベース, ポート番号を設定することもできます。デフォルトは,

ホスト = 指定なし (UNIX ドメインソケットによる接続)

データベース = "postal"

ポート番号 = 5432

となっています。

チェックボックス項目は2つあります。「ユーザ認証」チェックボックスは, PostgreSQL 接続時にパスワードを認証を行うかどうかの指定です。認証を行わない場合, Apache の実行ユーザ (通常 nobody) でパスワードなしで PostgreSQL に接続できるように設定されていなければなりません。「デバッグ」チェックボックスは, 実行した SQL 文を表示するかどうかの指定です。デフォルトは不表示です。

## スキーマ定義について

使用するテーブルは2つです。

```
create table prefecture (
    pid int2,          -- 都道府県コード
    pref char(8),      -- 都道府県名
    kana_pref char(16) -- 都道府県名(カナ)
```

## 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

```
);
```

このテーブルは都道府県名を格納します。

```
create table postal (
    oldcode varchar(5),    -- 旧郵便番号
    newcode char(7),       -- 新郵便番号
    pid int2,             -- 都道府県コード . prefecture 参照
    kana_city text,       -- 市町村名(カナ)
    kana_town text,       -- 町域名(カナ)
    city text,            -- 市町村名
    town text             -- 町域名
);
```

このテーブルが郵便番号データの本体です。都道府県コードから prefecture テーブルの都道府県名を引けるようになっています。

### インストール

まず付属のCD-ROM の examples/pgpost/ 以下を適当なディレクトリにコピーしてください<sup>注10</sup>。

```
$ tar xzf /mnt/cdrom/examples/pgpost.tar.gz
```

次にデータベースを構築します。pgpost/data ディレクトリに元データと登録用のSQL文があります。

```
$ cd pgpost/data
$ make
```

create.sql ができるので、これを使ってデータを登録します。

その前に Apache が使用するユーザ “nobody” を PostgreSQL にも登録しておきましょう<sup>注11</sup>。

```
$ createuser nobody
Enter user's postgres ID or RETURN to use unix user ID: 99  ENTER
Is user "nobody" allowed to create databases (y/n) n
Is user "nobody" allowed to add users? (y/n) n
```

注 10

郵便番号データが大きいので、20M バイトほどあります。

注 11

nobody の unix user ID はシステムによって異なります。

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
createuser: nobody was successfully added  
don't forget to create a database for nobody
```

また、今回は使用するデータベース “ postal ” も作りましょう。

```
$ createdb postal
```

データの登録を行います。

```
$ psql -f create.sql postal
```

データ量が多いので、かなり時間がかかります。筆者のマシンの場合 (LinuxPPC / PowerPC 603e 180MHz / メモリ80M バイト), 12分ほどかかりました。また、ディスクも大量に消費します。postal のデータベースエリア ( /usr/local/pgsql/data/base/postal/ ) 以下だけで43M バイトほど使いますので、ディスクの空き容量には十分注意してください。

次に、php3 ファイルをmake します。

```
$ cd ../pgpost  
$ make
```

これでindex.php3 ができたので、必要なPHP3 ファイルなどをインストールします。デフォルトでは、 /usr/local/etc/apache/share/htdocs/postal にインストールしますので、 /usr/local/etc/apache/share/htdocs/postal を作成し、書き込み権限を与えておいてください。

root になり、以下を実行します。“ foo ” は自分のアカウントに読み替えてください。

```
# mkdir /usr/local/apache/share/htdocs/postal  
# chown foo /usr/local/apache/share/htdocs/postal
```

一般ユーザに戻ってインストールを行います。

```
$ make install
```

これで以下のファイルがインストールされます。

- index.php3 : 初期画面、検索条件の入力フォーム
- help.html : ヘルプファイル
- select.php3 : 実際に検索を行うPHP3 スクリプト
- pref.php3 : 都道府県データ

## 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

適当なブラウザで

```
http://localhost/postal/
```

と入力して検索条件入力フォームの画面が表示されれば成功です。「説明を読む」をクリックして使い方をひと通り読んだら検索を行ってみてください。

### ソースコードについて

index.php3 (リスト 4.3.1)

16行目のincludeはファイルを読み込む機能です。ここで読み込んでいるpref.php3は、

```
<?
$pref_tbl[] = "北海道  ";
$pref_tbl[] = "青森県  ";
$pref_tbl[] = "岩手県  ";
:
```

のようになっていて、\$pref\_tblという配列への代入が全国の都道府県分羅列されています。このpref.php3ファイルは、実はmakeprefというPerlスクリプトで作成したものです。このシステムでは、prefectureという都道府県データのテーブルを持っていますが、makeprefは

```
select pid,pref from prefecture order by pid;
```

というSQL文でpref.php3を作成します。phpスクリプトの中でprefectureテーブルをアクセスせずに、ファイルに落としている理由は、都道府県のデータはめったに変化しないこと、そしてわずかですが処理が高速になるからです。

19行目の\$showdbという変数が1の場合、データベース、接続ホスト、ポート番号、ユーザ認証の入力エリアを表示します。これらのエリアを非表示にして、ユーザが特定のホスト以外に接続できないようにするときは、直接この変数を0にするか、Makefile.globalのSHOWDBを0にし、makeし直してください。

同じように、20行目の\$debugという変数が1の場合、「デバッグ」のチェックボックスを表示します。これを非表示にする場合は、直接この変数を0にするか、Makefile.globalのDEBUGを0にし、makeし直してください。

23行目からはHTMLのフォームの記述です。このフォームではいくつかの変数名が

表 4.3.2 このシステムで使用する変数一覧

oldcode	旧郵便番号
newcode	新郵便番号
prefecture	都道府県名
city	市町村名
town	町域名
kana_city	市町村名(カナ)
kana_town	町域名(カナ)
ulimit	検索結果上限件数
dbname	データベース名
hostname	接続ホスト
port	接続ポート番号
auth	ユーザ認証有無
sqlprint	SQL文の表示有無

使われていますが、action で指定されているselect.php3 が起動されると、このフォームで設定された値がそのまま入っている同じ名前の変数として利用することができます(表4.3.2)。

注 12  
プルダウンメニューで都道府県の選択ができます。

49 行目からはフォームのselect<sup>注12</sup>を設定しています。53 行目のcount は、配列の要素数を返します。

#### リスト 4.3.1 index.php3

```

1: <!doctype html public "-//w3c//dtd html 3.2//ja">
2: <html>
3: <head>
4: <title>postal code search</title>
5: <meta http-equiv=content-type content=text/html; charset=x-euc-jp>
6: </head>
7: <body bgcolor=white>
8: <h1>新 / 旧郵便番号検索システム </h1>
9:
10: <a href="help.html">説明を読む </a>
11:
12: <? /* $Id: index.php3.source,v 1.1.1.1 1998/10/12 01:23:15 t-ishii Exp $
13:   お願い: このファイルは Makefile により自動生成されます。
14:   修正は index.php3.source の方に行なって下さい。*/ ?>
15:
16: <? include "pref.php3" ?>
17:
18: <?
19: $showdb = 1;
20: $debug = 1;
21: ?>
22:

```



### 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

```
23: <form method="POST" action="select.php3">
24: <table>
25:
26: <tr>
27: <td>
28: 旧郵便番号
29: </td>
30: <td>
31: <input type="text" name="oldcode">
32: </td>
33: </tr>
34:
35: <tr>
36: <td>
37: 新郵便番号
38: </td>
39: <td>
40: <input type="text" name="newcode">
41: </td>
42: </tr>
43:
44: <tr>
45: <td>
46: 都道府県
47: </td>
48: <td>
49: <select name=prefecture>
50: <option selected value=-1>指定なし
51:
52: <?
53: for($i=0;$i < count($pref_tbl);$i++) {
54:   print("<option value=$i>$pref_tbl[$i]\n");
55: }
56: ?>
57:
58: </select>
59: </td>
60:
61: <tr>
62: <td>
63: 市町村名
64: </td>
65: <td>
66: <input type="text" name="city">
67: </td>
68: </tr>
69:
70: <tr>
71: <td>
72: 町域名
73: </td>
74: <td>
75: <input type="text" name="town">
76: </td>
77: </tr>
78:
79: <tr>
80: <td>
```

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
81: 市町村名(カナ)
82: </td>
83: <td>
84: <input type="text" name="kana_city">
85: </td>
86: </tr>
87:
88: <tr>
89: <td>
90: 町域名(カナ)
91: </td>
92: <td>
93: <input type="text" name="kana_town">
94: </td>
95: </tr>
96:
97: <tr>
98: <td>
99: 検索結果表示上限件数
100: </td>
101: <td>
102: <input type="text" name="ulimit" value="100">
103: </td>
104: </tr>
105:
106: <tr>
107: <td colspan=2>
108: <input type="submit" value="検索開始">
109: <input type="reset" value="検索キークリア">
110: </td>
111: </tr>
112:
113: </table>
114:
115: <? if ($showdb || $debug): ?>
116: <hr>
117: <table>
118: <? endif; ?>
119:
120: <? if ($showdb): ?>
121:
122: <tr>
123: <td>データベース</td>
124: <td><input type="text" name="dbname" value="postal">
125: </td>
126: </tr>
127:
128: <tr>
129: <td>
130: 接続ホスト
131: </td>
132: <td>
133: <input type="text" name="hostname" value="">
134: </td>
135: </tr>
136:
137: <tr>
138: <td>
```

### 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

```
139: 接続ポート番号
140: </td>
141: <td>
142: <input type="text" name="port" value="5432">
143: </td>
144: </tr>
145:
146: <tr>
147: <td>
148: ユーザ認証
149: </td>
150: <td>
151: <input type="checkbox" name="auth" >
152: </td>
153: </tr>
154:
155: <?else: ?>
156:
157: <input type="hidden" name="dbname" value="postal">
158: <input type="hidden" name="hostname" value="">
159: <input type="hidden" name="port" value="5432">
160: <input type="hidden" name="auth" value="">
161:
162: <?endif; ?>
163:
164: <?if ($debug): ?>
165: <tr>
166: <td>
167: デバッグオプション
168: </td>
169: <td>
170: <input type="checkbox" name="sqlprint">
171: </td>
172: </tr>
173:
174: <? else: ?>
175:
176: <input type="hidden" name="sqlprint">
177:
178: <?endif; ?>
179:
180: <? if ($showdb || $debug): ?>
181: </table>
182: <?endif; ?>
183:
184: </form>
185: <hr>
186:
187: </body>
188: </html>
189:
```

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

注 13  
検索フォームで「認証あり」が設定されている場合、\$auth がセットされます。

select.php3 (リスト 4.3.2)

select.php3 は、index.php3 から起動されます。

検索条件 \$auth が設定されている場合<sup>注13</sup>、ユーザ認証を行います。最初は \$PHP\_AUTH\_USER が設定されていないので、6 行目を実行します。Header コマンドは、ユーザ認証ダイアログの表示を行うプロトコルを発行します。すると Web ブラウザ上に、ユーザ名とパスワードの入力ダイアログが表示されます (図 4.3.8)。

ここでユーザ名とパスワードを入力し「OK」を押すと、7 行目が実行されるのではなく、いったん実行が打ち切れ、新たに select.php3 が最初から実行されます。ただし今度は \$PHP\_AUTH\_USER が設定されていますので、11 行目から実行します。

一方「Cancel」を押した場合は、7 行目を実行します。ここでは、ユーザ認証が失敗したことを伝えるプロトコルを発行します。そして 8 行目のメッセージが表示されず (図 4.3.9)。

11 行目から 17 行目は、接続ホスト名 (\$hostname) があるかどうかによってメ

図 4.3.8  
ユーザ名とパスワードの入力画面

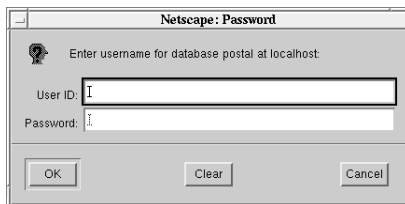
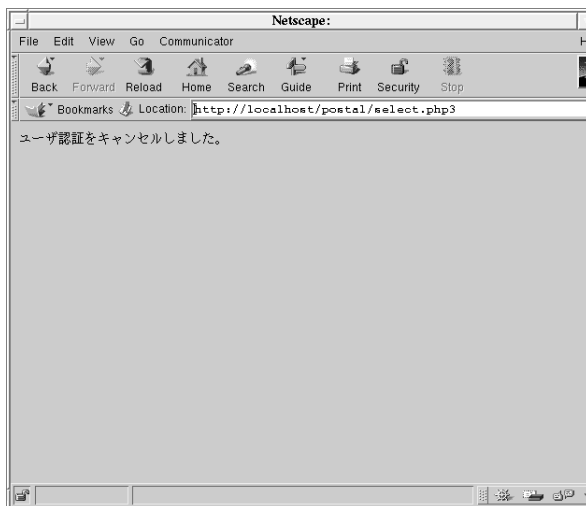


図 4.3.9  
ユーザ認証をキャンセルした画面



## 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

表 4.3.3 pgconnect のキーワード

hostname	接続先のホスト名。ローカルホストにUNIX ドメインのソケットで接続する場合はこの情報を渡さない
dbname	データベース名
port	ポート番号
user	ユーザ名
password	パスワード

ッセージを切り替えています。接続ホスト名がない場合、PostgreSQL はUNIX ドメインのソケットを使って接続します。

18 行目の `pg_connect` は PostgreSQL に接続するための関数です。PostgreSQL を使うためには、まず `pg_connect` を呼び出しておかなければなりません。`pg_connect` の前の `@` は、エラーメッセージの出力を抑止する働きがあります。エラーメッセージに限らず、`header` 関数を呼び出す前にメッセージが出力されると、これ以後の `header` を使った処理がうまくいかなくなってしまいます。

`pg_connect` の引数は接続に必要な情報を表す文字列です。「キーワード=値」の形で1つの情報を表現し、複数の情報はスペースで区切ります。キーワードには表4.3.3のものがあります。

今回、`user` と `password` にはそれぞれ `$PHP_AUTH_USER` と `$PHP_AUTH_PW` を与えていますが、これらの変数にはユーザ認証ダイアログで入力した値があらかじめ設定されてます。これらのユーザ名とパスワードを PostgreSQL でも受け付けるようにするには、前もってユーザ名とパスワードを PostgreSQL に登録しておく必要があります。`createuser` で該当ユーザを登録した後、第2章で解説した「セキュリティ機能」の手順に従い、flat file password authentication (p.62) または crypt authentication (p.64) によりパスワードを設定してください。

また、これだけでは `pg_connect` による接続はできても、`select` などの操作ができません。そこで、`grant` 文を使って `select` 権限を与えます。仮にユーザ名を `foo` とすると、

```
grant select on postal to foo;
grant select on prefecture to foo;
```

を実行しておいてください。

`pg_connect` は接続に成功すると、接続用のハンドルを表す文字列を返します。失敗した場合は20行目でユーザ認証が失敗したものとします。こうしておかないと、パスワードの誤りなどで PostgreSQL がユーザ認証に失敗しても、ブラウザを終了しない限り認証がうまくいっているものとして扱われてしまい、パスワードなどの再入力

ができなくなってしまう。

26行目の`pg_connect`は、認証を行わないときの呼び出し形式で、以下の形になります。

```
pg_connect(hostname, port, options, tty, dbname);
```

#### 注 14

`/usr/local/apache/etc/httpd.conf`の“User”ディレクティブで指定されるユーザです。

#### 注 15

`tty`はデバッグ用のメッセージを出力する端末ですが、通常使うことではないので、“”（空文字列）を与えておきます。

この場合、PostgreSQL に対するユーザ名としては、Apacheの実行ユーザID<sup>注14</sup>が採用されます。通常これは`nobody`に設定されているはずですので、`nobody`のユーザアカウントを`createuser`コマンドを使って作成しておかなければなりません（今回は`create.sql`で行っています）。

`pg_connect`にはほかに2つの呼び出し形式があります<sup>注15</sup>。これらは上の呼び出し形式の簡略形で、認証を伴わないときに使用します。

```
pg_connect(hostname, port, tty);
```

```
pg_connect(hostname, port, options, dbname);
```

ところで、35行目からは説明のためのコメントですが、本来こうしたコメントはファイルの先頭に持って行きたいところです。ところが、そうするとユーザ認証がうまく働きません。おそらくPHPのバグだと思うのですが...

68行目は`index.php3`でも使った都道府県データの取り込みです。

76行目からは関数定義です。

`makeup1`は、旧または新の郵便番号が検索キーとして与えられたときの処理です。80行目の`ereg_replace`は、第1引数の正規表現文字列を第3引数の文字列から探し、第2引数文字列で置き換えた結果を返します。第2引数は空白文字なので、結果的に文字が削除されます。`[^0-9]`は、数字以外を表す正規表現です。つまり`makeup1`は数字以外が与えられると削除します。これは検索キーとして「100-11」のように、-などを含む文字列が与えられても、エラーにならないようにするためです。`$f`は複数の検索条件があるかどうかのフラグです。つまり、ある検索条件が先頭にある場合は、

```
select * from foo where bar = ...
```

のように、検索条件は`where`の次に来ますが、複数の検索条件の2番目以降の場合は、

```
select * from foo where ... and bar = ...
```

のように`and`の次にこななければなりません。`$f`はその判定に使用します。

`makeup2`は`makeup1`とよく似ていますが、検索キーは文字列型ではなく、都道府県を表す数字のコードになります。

## 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

表 4.3.4 table 関係のタグ

<table border>	テーブルの開始宣言．終了は </table>．border は表の枠線を表示する
<tr>	表の1行の開始．終了は </tr>
<th>	表の列タイトル開始．終了は </th>．nowrap があると，長いデータでも改行しない
<td>	表示データの開始．終了は </td>

makeup3 は，市町村名などの文字列フィールドの問い合わせ用で，無条件に前方一致検索を行うようになっています．そのため substr を使って，先頭の文字が ^ でなければ，前方一致検索を表す ^ を付加しています．

133 行目から 139 行目までで SQL 文の主要な部分を作成し，142 行目で order by 句を設定して，検索結果が郵便番号でソートされるようにしています．

150 行目では，index.php3 にて「デバッグ」スイッチがオンのときに SQL 文を表示しています．

155 行目からが検索結果の表示です．ご覧のように table を使っています．ここで HTML に馴染みがない方のために，本スクリプトで使われている table 関係のタグを表 4.3.4 で説明しておきます．

171 行目では，pg\_exec でトランザクション開始の SQL 文である “begin” を発行しています．pg\_exec は，第 1 引数が pg\_connect の返したコネクション ID，第 2 引数が SQL 文です．

ところで begin を使う理由ですが，検索のときにカーソルを使いたいからです<sup>注 16</sup>．カーソルを使わないと，検索した結果をすべて取り込まなければなりません．本システムでは，指定した都道府県のすべての郵便番号を検索することが可能で，この場合，2000 件以上の結果がサーバから返却されることもあります．これらがすべて Web ブラウザに転送され，メモリに取り込まれるわけですから，遅いだけでなく，プラットフォームによってはメモリ不足で異常終了する可能性もあります．そこでカーソルを使い，\$ulimit に設定した「検索上限件数」で指定した件数だけを取り込むようにしています．

172 行目ではカーソルを宣言しています．具体的には，

```
declare c cursor for select newcode,oldcode,pid,city,town,
kana_city,kana_town from postal where ....
```

の形の SQL 文です．この時点で実際に postgres が検索を実行します．

175 行目で，fetch を発行しデータを取り込んでいます．fetch の構文は，

注 16

PostgreSQL では，トランザクションの内側でないとカーソルが使えません．

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
fetch [取得希望件数] in カースル名
```

です。取得希望件数を省略すると、1 と見なされます。ここでは、

```
while (1件fetchしてみてデータがある間) {
    表示データを作成
    :
}
```

というアルゴリズムも可能で、その方がプログラムも簡単になるのですが、この方法では1件ずつデータを転送するオーバーヘッドのため、大変遅いものになってしまいます。そこで本スクリプトでは、1回で必要なデータをすべて転送するようにしています。

176行目のpg\_numrows は、検索の結果得られた件数を返すので、179行目～195行目のループをその件数分だけ回します。

ここで中心となるのはpg\_result です。この関数は、pg\_exec が返すディスクリプタ、行番号、列番号を引数にしてデータを取り出します<sup>注17</sup>。

182行目では、substr を使って郵便番号の3桁目と4桁目の間にハイフンを入れ、見やすくしています。substr には2つの呼び出し形式があります。

- 文字列, 開始位置, 文字数: 文字列を開始位置から文字数分取り出して返します
- 文字列, 開始位置: 文字列を開始位置から最後まで取り出して返します

なお、開始位置は0から始まります。

旧郵便番号は、3桁の場合はそのまま、5桁の場合は“123-45”のようにハイフンを間に入れるので、桁数の判断が必要です。ここではstrlen を使って桁数を得ています(184行目)。

189行目は、都道府県名を入れた配列 \$pref\_tbl を使って都道府県コードから都道府県名に変換しています。その他のカラムに関してはとくに編集が必要ないので、pg\_result の返す値をそのままテーブルで使います。

197行目のpg\_freeresult は、pg\_exec によって得られた結果を格納するメモリを解放します。PHP の場合、スクリプトが終了するときに自動的にメモリも解放されるので、pg\_freeresult を呼び出さなくても動作には支障がありませんが、pg\_exec の結果をもう利用しないことがわかっているときには、 unnecessary メモリを解放するほうがいいのは言うまでもありません。

ここまでで検索結果の表示がすべて終わったのですが、もうひとつ仕事が残っています。それは、実際にヒットした件数を数えることです。たとえば、100件しか表示していないが、実際には2000件ヒットしているというようなことは、どうすればわか

注17  
行番号、列番号は0から始まります。



## 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

るでしょうか。

確実な方法は実際にfetch してみることです。201 行目からの処理がそれです。まず fetch 1000 と大雑把にデータを取得してみます。実際に何件取得できたかを pg\_numrows でチェックします。ここでは取得したデータを使うわけではなくて、単に件数を知りたいだけです。直後に pg\_freeresult を呼び出しています。取得件数が1000 以下の場合は、もうデータが残っていないことになりますので、ここまで数えた件数が実際にヒットした件数になるわけです。215 行目で件数を表示しています。

217 行目の close c でカーソルを閉じ、次の end でトランザクションを終了します。220 行目の pg\_close は pg\_connect と対になる関数で、PostgreSQL との接続を切断します。

リスト 4.3.2 select.php3

```
1: <?
2: if ($auth) {
3:     /* ユーザ認証を行う */
4:
5:     if (!$PHP_AUTH_USER) {
6:         Header("WWW-authenticate: basic realm=\"database $dbname\"");
7:         Header("HTTP/1.0 401 Unauthorized");
8:         print("ユーザ認証をキャンセルしました。");
9:         exit;
10:    } else {
11:        if ($hostname) {
12:            $constr = "host=$hostname";
13:            $msg = "$hostname の";
14:        } else {
15:            $constr = "";
16:            $msg = "";
17:        }
18:        @ $con = pg_connect("$constr dbname=$dbname port=$port user=$PHP_AUTH_USER
password=$PHP_AUTH_PW");
19:        if (!$con) {
20:            Header("HTTP/1.0 401 Unauthorized");
21:            print("ユーザ $PHP_AUTH_USER で $msg データベース $dbname (ポート番号 $port)に接続出来ませ
んでした。<br>");
22:            exit;
23:        }
24:    }
25: } else {
26:     $con = pg_connect("$hostname", $port, "", "", $dbname);
27:     if (!$con) {
28:         print("$hostname のデータベース $dbname (ポート番号 $port)に接続出来ませんでした。");
29:         exit;
30:     }
31: }
32: ?>
33:
34: <?
35: /* $Id: select.php3,v 1.3 1998/11/01 11:55:23 t-ishii Exp $
36: * 新旧郵便番号検索システム "pgpost" version 1.2
```

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
37: * index.php3 から起動され、実際に検索を行い、結果を
38: * 表示する。以下の変数がフォームで設定されているものとする。
39: *
40: * oldcode          旧郵便番号
41: * newcode          新郵便番号
42: * prefecture       都道府県名
43: * city             市町村名
44: * town            町域名
45: * kana_city        市町村名(カナ)
46: * kana_town        町域名(カナ)
47: * ulimit           検索結果上限件数
48: * dbname           データベース名
49: * hostname         接続ホスト
50: * port            接続ポート番号
51: * auth            ユーザ認証有無
52: * sqlprint         SQL文の表示有無
53: *
54: * 注意！このコメントをこのファイルの先頭に持って行くと
55: * authentication がうまくいかなくなります！
56: */
57: ?>
58:
59: <!doctype html public "-//w3c//dtd html 3.2//ja">
60: <html>
61: <head>
62: <title>searching result of postal code</title>
63: <meta http-equiv=content-type content=text/html; charset=x-euc-jp>
64: </head>
65: <body bgcolor="white">
66:
67: <?
68: include "pref.php3"; /* 都道府県データを取り込む */
69:
70: /*
71: * 問い合わせ作成用の関数
72: * global変数として $query(問い合わせ文字列)と$f(検索条件の有無フラグ)
73: * を使用する
74: */
75:
76: /* 郵便番号フィールドの問い合わせ作成。数字以外の文字は無視する */
77: function makeup1($name, $value) {
78:     global $f,$query;
79:     if ($value != "") {
80:         $value = ereg_replace("[^0-9]","", $value);
81:         if ($f) {
82:             $query = $query . " and";
83:         } else {
84:             $query = $query . " where";
85:         }
86:         $query = $query . " $name = '$value'";
87:         $f++;
88:     }
89: };
90:
91: /* 都道府県コードフィールドの問い合わせ作成 */
92: function makeup2($name,$value) {
93:     global $f,$query;
94:     if ($value != -1) {
```

### 4.3 PHP インターフェースを使って～ Apache など WWW サーバとの連携

```
95:     if ($f) {
96:         $query = $query . " and";
97:     } else {
98:         $query = $query . " where";
99:     }
100:     $query = $query . " $name = '$value'";
101:     $f++;
102: }
103: };
104:
105: /* 市町村名などの文字列フィールドの問い合わせ作成。
106:  * 無条件に前方一致検索を行う
107:  */
108: function makeup3 ($name,$value) {
109:     global $f,$query;
110:     if ($value != "") {
111:         if (substr($value,0,1) != "^") {
112:             $value = "^" . $value;
113:         }
114:         if ($f) {
115:             $query = $query . " and";
116:         } else {
117:             $query = $query . " where";
118:         }
119:         $query = $query . " $name ~ '$value'";
120:         $f++;
121:     }
122: };
123:
124:
125: /* 問い合わせ文字列格納用の変数 */
126: $query = "declare c cursor for " .
127: "select newcode,oldcode,pid," .
128: "city,town,kana_city,kana_town from postal ";
129:
130: /* 検索条件があるかどうかのフラグ */
131: $f = 0;
132:
133: makeup1("oldcode",$oldcode);      /* 旧郵便番号 */
134: makeup1("newcode",$newcode);      /* 新郵便番号 */
135: makeup2("pid",$prefecture);       /* 都道府県コード */
136: makeup3("city",$city);            /* 市町村 */
137: makeup3("town",$town);            /* 町域名 */
138: makeup3("kana_city",$kana_city);  /* 市町村(カナ) */
139: makeup3("kana_town",$kana_town);  /* 町域名(カナ) */
140:
141: if ($f) {
142:     $query = $query . " order by newcode";
143: } else {
144:     print("検索キーが設定されていません。\\n");
145:     exit;
146: }
147:
148: /* デバッグオプションがオンなら SQL 文を表示する */
149: if ($sqlprint) {
150:     print("以下の SQL 文を発行しました。<br>$query<br>\\n");
151: }
152:
```

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
153: ?>
154:
155: <h1>検索結果 </h1>
156: <table border>
157: <tr>
158: <th nowrap>新郵便番号 </th>
159: <th nowrap>旧郵便番号 </th>
160: <th nowrap>都道府県 </th>
161: <th nowrap>市町村 </th>
162: <th nowrap>町域名 </th>
163: <th nowrap>市町村(カナ) </th>
164: <th nowrap>町域名(カナ) </th>
165: </tr>
166:
167: <?
168: /* PostgreSQL では、cursor の利用は、トランザクションの内側で
169:  * 行う必要がある
170:  */
171: $rtn = pg_exec($con,"begin");
172: $rtn = pg_exec($con,$query);
173:
174: /* 検索上限件数分データを取得 */
175: $rtn = pg_exec($con,"fetch $ulimit in c");
176: $cnt = pg_numrows($rtn);
177:
178: /* テーブル形式でデータを表示 */
179: for($i=0;$i < $cnt;$i++) {
180:     print("<tr>\n");
181:     $str = pg_result($rtn,$i,0);          /* 新郵便番号 */
182:     echo "<td nowrap>" . substr($str,0,3) . "-" . substr($str,3) . "</td>\n";
183:     $str = pg_result($rtn,$i,1);          /* 旧郵便番号 */
184:     if (strlen($str) > 3) {
185:         print("<td nowrap>" . substr($str,0,3) . "-" . substr($str,3) . "</td>\n");
186:     } else {
187:         print("<td nowrap>" . $str . "</td>\n");
188:     }
189:     print("<td nowrap>" . $pref_tbl[pg_result($rtn,$i,2)] . "</td>\n");
190:     print("<td nowrap>" . pg_result($rtn,$i,3) . "</td>\n");
191:     print("<td nowrap>" . pg_result($rtn,$i,4) . "</td>\n");
192:     print("<td nowrap>" . pg_result($rtn,$i,5) . "</td>\n");
193:     print("<td nowrap>" . pg_result($rtn,$i,6) . "</td>\n");
194:     print("</tr>\n");
195: }
196:
197: pg_freeresult($rtn);
198:
199: /* 検索件数を求めるために、残りのデータを空読みする */
200: if ($cnt >= $ulimit) {
201:     for(;;) {
202:         $rtn = pg_exec($con,"fetch 1000 in c");
203:         $n = pg_numrows($rtn);
204:         pg_freeresult($rtn);
205:         $cnt += $n;
206:         if ($n < 1000) {
207:             break;
208:         }
209:     }
210: }
```

## 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

```
211:
212: if ($cnt < $ulimit) {
213:     $ulimit = $cnt;
214: }
215: print("$cnt 件中 $ulimit 件を表示しました。");
216:
217: pg_exec($con,"close c");
218: pg_exec($con,"end");
219:
220: pg_close($con);
221:
222: ?>
223:
224: </table>
225: </body>
226: </html>
227:
```

### 4.3.9 その他の PostgreSQL 用関数

前章のサンプルアプリケーションではいくつかのPostgreSQL用の関数について述べましたが、ここではそれ以外のPostgreSQL関連の関数を紹介します。なお、intは整数型、stringは文字列型であることを表しています。

.....

int pg\_cmdTuples(int result\_id)

➡ INSERT、UPDATE、DELETEのSQL文を発行した際に影響を受けたタブルの数を返します。

.....

string pg\_dbname(int connection)

➡ pg\_connectで接続したデータベース名を返します。引数はpg\_connectの返却する接続ハンドルです。

.....

string pg\_errorMessage(int connection)

➡ エラーが生じた際のエラーメッセージを返します。

.....

int pg\_fieldIsNull(int result\_id, string fieldname)

➡ pg\_resultの返したresult\_idとカラム名から、そのデータがNULLであるかどうかを返します。

.....

```

.....
string pg_fieldName(int result_id, int field_number)
    ➔ 指定したカラム番号のカラム名を返します。カラム番号は0から始まります。
.....
int pg_fieldNum(int result_id, string field_name)
    ➔ 指定したカラム名に対応するカラム番号を返します。カラム番号は0から始ま
    ります。
.....
int pg_fieldPrLen(int result_id, int row_number, string
field_name)
    ➔ 指定した行、カラムのデータを実際に表示するのに必要なキャラクタ数(バイト
    数)を返します。
.....
int pg_fieldSize(int result_id, string field_name)
    ➔ 指定したカラムのデータベース上での大きさをバイト数で返します。表示上の大
    きさでないことに注意してください。
.....
string pg_fieldType(int result_id, int field_number)
    ➔ 指定したカラムのPostgreSQL上での型名を返します。
.....
int pg_getlastOid()
    ➔ pg_execでINSERTを実行した後に、挿入された新しいタブルのoid(object id)
    を返します。oidはPostgreSQLがデータベース内のオブジェクトを識別するユニー
    クIDです。
.....
string pg_host(int connection_id)
    ➔ 接続先のホスト名を返します。
.....
void pg_loClose(int fd)
    ➔ large objectを閉じます。large objectについては、第3章の「C関数」を参照
    してください。
.....
int pg_loCreate(int connection_id)
    ➔ large objectを新規に作成し、そのoidを返却します。
.....
int pg_loOpen(int connection_id, int oid, string mode)
    ➔ oidで指定したlarge objectを開き、そのファイルディスクリプタを返します。
    modeはlarge objectをどのような目的で開くかを指定します。
    • r...読み込み

```

## 4.3 PHP インターフェースを使って ~ Apache など WWW サーバとの連携

- w ...書き込み
- rw ...読み込みと書き込みの両方

.....  
void pg\_loRead(int fd, string bufvar, int len)

➡ large object を読み込みます。fd は pg\_loOpen が返したもの、bufvar は large object を読み込み変数の名前です。読み込みは最大でも len バイト分だけ行われます。  
.....

void pg\_loReadAll(int fd)

➡ large object の全データを読み込み、ブラウザに表示します。画像やサウンドデータを表示するときなどに使います。  
.....

void pg\_loUnlink(int connection\_id, int oid)

➡ large object を削除します。  
.....

void pg\_loWrite(int fd, string buf, int len)

➡ pg\_loRead と反対に、large object にデータを書き込みます。  
.....

int pg\_numFields(int result\_id)

➡ pg\_exec の返した result\_id からカラム数を返します。  
.....

string pg\_options(int connection\_id)

➡ pg\_connect で指定した option を返します。  
.....

string pg\_port(int connection\_id)

➡ pg\_connect で指定した port を返します。  
.....

string pg\_tty(int connection\_id)

➡ pg\_connect で指定した tty を返します。  
.....

### 4.3.10 感想など

PHP/FI の時代から PostgreSQL との関係では定評のある PHP ですが、PHP/FI から PHP へと名称が変わっても、そのよさは少しも損なわれていないようです。アプリケーションを構築する際に、プラットフォームからの独立性、メンテナンスの容易性、そしてシンプルな文法などを考えると、やはり現時点では Apache + PHP + PostgreSQL は最強の解と言えるでしょう。

問題があるとするれば、ドキュメントの日本語化ですが、現在ボランティアの方々が精力的に取り組んでおられますので、本書が刊行されるころには完成しているかもしれません。ときどき

<http://www.cityfujisawa.ne.jp/%7Elouis/apps/phpfi/index.html>

をチェックしてみましょう。



## 4-4 Perl インターフェース スを使って

### 4.4.1 PostgreSQL Perl インターフェース

PostgreSQL Perl インターフェース (`pgsql_perl5`) は、Perl5 から PostgreSQL にアクセスするためのモジュールです。 `pgsql_perl5` は `libpq` を使って実装されており、またインターフェースのデザインもできるだけ `libpq` に似るように意図されているそうです。

### 4.4.2 `pgsql_perl5` のインストール

`pgsql_perl5` のインストールの方法には2つあります。

`configure` 時に Perl インターフェースを構築することを指定する

PostgreSQL のインストール後に単独で `pgsql_perl5` をインストールする

の方法は 6.3.2 では使えず、また結局、`make install` 時に `root` にならなければならないので、あまり便利とは言えません。

そこで本節では の方法でインストールすることにします。なお、`pgsql_perl5` は Perl5、さらにできれば日本語対応の Perl5 が必要です。あらかじめインストールしておいてください。

`pgsql_perl5` のインストールの前に、環境変数 `POSTGRES_HOME` に `/usr/local/pgsql` をセットします。

`bash` の場合なら

```
$ export POSTGRES_HOME=/usr/local/pgsql
```

とし、`csh/tcsh` の場合なら

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

図 4.4.1 筆者の環境でのテスト結果

```

PERL_DL_NONLAZY=1 /usr/local/bin/perl -I./blib/arch -I./blib/lib -I/usr/lib/perl5/ppc-
linux/5.00404 -I/usr/lib/perl5 test.pl
Pg::conndefaults ..... ok
Pg::connectdb ..... ok
$conn->exec ..... ok
$conn->errorMessage ..... ok
$conn->db ..... ok
$conn->user ..... ok
$conn->port ..... ok
$conn->cmdStatus ..... ok
$conn->oidStatus ..... ok
$conn->getline ..... ok
$conn->endcopy ..... ok
$result->cmdTuples ..... ok
$result->fname ..... ok
$result->ftype ..... ok
$result->fsize ..... ok
$result->fnumber ..... ok
$result->fetchrow ..... ok
test sequence finished.

```

図 4.4.2 pgsql\_perl5 のインストール

```

Installing /usr/lib/perl5/site_perl/ppc-linux/./auto/Pg/Pg.so
Installing /usr/lib/perl5/site_perl/ppc-linux/./auto/Pg/Pg.bs
Installing /usr/lib/perl5/site_perl/./auto/Pg/autosplit.ix
Installing /usr/lib/perl5/site_perl/./Pg.pm
Installing /usr/lib/perl5/man/man3/./Pg.3
Writing /usr/lib/perl5/site_perl/ppc-linux/auto/Pg/.packlist
Appending installation info to /usr/lib/perl5/ppc-linux/5.00404/perllocal.pod

```

```
% setenv POSTGRES_HOME /usr/local/pgsql
```

とします。pgsql\_perl5 の作成そのものは非常に簡単です。

注 1  
6.4 を使われている場合  
は postgresql-6.3.2 のと  
ころを postgresql-v6.4  
として実行してくださ  
い。

```
$ cd /usr/local/src/postgresql-6.3.2/src/interfaces/perl5注1
```

```
$ perl Makefile.PL
```

```
$ make
```

pgsql\_perl5 にはテストスイートが付属しているので、それを実行してみましょう。

```
$ make test
```

筆者の環境では図 4.4.1 のように出力されました。

テスト結果がよさそうならば、インストールに移ります。前述のように、インストールは root で行います。

```
# make install
```

図4.4.2のような一連のファイルがインストールされます。

### 4.4.3 Perl/CGI による Mail/News の全文検索システムの作成

#### Perl と CGI

Perl は非常に機能が豊富で応用範囲の広い言語ですが、WWW サーバと連携する CGI<sup>注2</sup> としてもよく使われています。本章では、4.1 節で作成したデータベースを用い、CGI による Mail/News の全文検索システムを作成してみます。

注 2

CGI については前節を参照してください。

#### システム概要

検索フォームを開くと、図4.4.3の画面になります。機能的には4.1 節の smst と同じです。全文検索のキーワード、メールヘッダ From:/Subject:、および日付で検索ができます。複数の項目が検索キーとして指定されると、AND 検索になるところも smst と同じです。

検索キーを入力し終わったら「Go!」ボタンを押します。すると検索が開始され、該

図 4.4.3  
検索フォームを開く

The screenshot shows a Netscape browser window with the title 'Netscape: Simple mail/news search by psql\_perl5'. The address bar shows 'http://localhost/~t-inhii/'. The main content area contains a search form titled 'psql\_perl5 によるメールデータベース検索'. The form includes input fields for 'キーワード' (Keyword), 'From:', 'Subject:', and '日付' (Date), which is split into two parts with a tilde '~' between them. There is also a 'データベース' (Database) dropdown menu currently set to 'news'. At the bottom of the form are 'Go!' and 'Clear' buttons.

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

図 4.4.4  
検索結果の表示

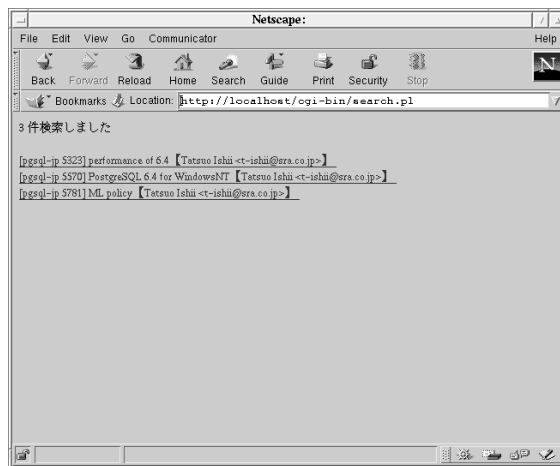
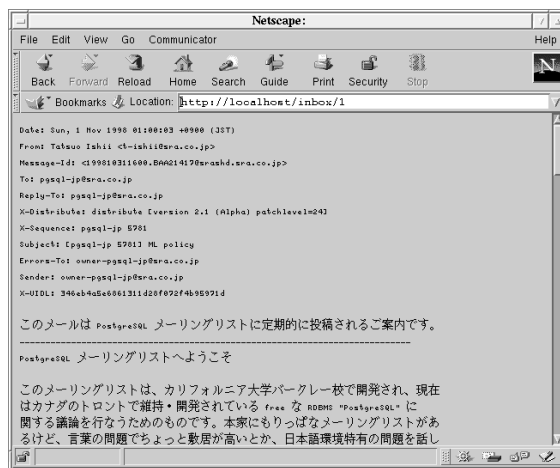


図 4.4.5  
見たいデータを表示させたところ



当データがあれば図4.4.4の画面になります。各行はサブジェクトとFrom:を表しています。見たいデータに対応する行をクリックすると、図4.4.5のようにメールの内容が表示されます。

今回作成するシステムは以下の要素から構成されます。

検索キーワードの入力フォーム

から起動されるPerl プログラム (CGI)

PostgreSQL データベース (4.1 節のものをそのまま使用)

では、`pgsql_perl5`を使うのはもちろんのことですが、Perl5には標準で“CGI”という、まさにその名の通りCGIの作成を便利してくれるクラスが付属しているので、それも使います。

## データベースのセットアップ

前述のように、使用するデータベースは4.1節のものをそのまま使いますので、その手順にしたがってデータベースを準備してください。これ以外に必要な作業は以下の通りです。

### nobody ユーザの登録

ApacheがCGIを起動する際、そのユーザ名は“nobody”になります。そこで、PostgreSQLにnobodyというユーザを登録します<sup>注3</sup>。

```
$ createuser nobody
Enter user's postgres ID or RETURN to use unix user ID: 99 
Is user "nobody" allowed to create databases (y/n) n
Is user "nobody" allowed to add users? (y/n) n
createuser: nobody was successfully added
don't forget to create a database for nobody
```

注 3  
nobodyのunix user ID  
はシステムによって異  
なります。

なお、4.3節ですでにnobodyユーザを登録済みの方は、もちろん再度“nobody”を登録する必要はありません。これだけではnobodyユーザがテーブルを検索する権限を与えられていません。そこで以下のようなgrant文でSELECT権限を付与します。

```
$ psql -c 'grant select on words, header to nobody' test
```

## Apache (WWW サーバ) のセットアップ

Apacheのインストールは前節の「PHPのインストール」に書かれているので、それに従ってApacheのインストールを済ませてください。前節ではPHPもインストールしていますが、PerlのCGIと問題なく併用できます。

ただ、前節の設定では、ApacheでCGIが使えるようになっていません。そこで、Apacheの設定でCGIを有効にする方法を説明します。

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
/usr/local/apache/etc/srm.conf
```

184行目付近の

```
#AddHandler cgi-script .cgi
```

を

```
AddHandler cgi-script .cgi
```

に変更します。また、164行目の

```
#ScriptAlias /cgi-bin/ /usr/local/apache/share/cgi-bin/
```

を

```
ScriptAlias /cgi-bin/ /usr/local/apache/share/cgi-bin/
```

に変更します。そして最後に

```
SetEnv LD_LIBRARY_PATH /usr/local/pgsql/lib
```

を追加します。

```
/usr/local/apache/etc/access.conf
```

58行目付近の

```
<Directory /usr/local/apache/share/cgi-bin>
```

を

```
<Directory /cgi-bin>
```

に変更します。また、60行目の

```
Options None
```

を

```
Options ExecCGI
```

に変更します。

以上で、/usr/local/apache/share/cgi-bin/に登録したCGIプログラムが、

/cgi-bin/ CGI プログラム名

というURLで実行できるようになります。

### メールデータのセットアップ

smst もそうでしたが、今回のシステムでも、メール本体のファイルはPostgreSQLのデータベースには格納されていません。あらかじめ /usr/local/apache/share/htdocs/ 以下にコピーしておいてください。たとえば、4.1 節に出てきたように、

```
$ makeindex.pl -h header.data -b inbox/* > wordindex.data
```

としてデータを作成した場合は、/usr/local/apache/share/htdocs/inbox にメールファイルがコピーされていなければなりません。また、nobody ユーザがそのファイルを読むことができるように、read permission を設定してください。

### index.html (リスト 4.4.1)

検索キーの入力フォーム用HTMLファイルです。本書付属CD-ROMのexamples/perl\_cgi/index.htmlに収録されているので、適当な場所にコピーしてください。コピーする場所は、URLとして指定可能な場所ならどこでも結構です。たとえば、/usr/local/apache/share/htdocs/perl\_cgi\_example/index.htmlにインストールすれば、

```
http://localhost/perl_cgi_example/index.html
```

でアクセスできますし、ユーザfooのホームディレクトリの下のpublic\_html/perl\_cgi\_example/index.htmlにインストールすれば、

```
http://localhost/~foo/perl_cgi_example/index.html
```

でアクセスできます。

CGIではユーザがフォームに入力したデータは、フォームで指定した変数に格納されてCGIプログラムに引き渡されます。たとえば全文検索用のキーワードは15行目の

```
<input type="text" name="keyword">
```

により、keywordという変数に格納する指定がされています。表4.4.1に変数名と使

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

リスト 4.4.1 index.html

```
1: <!doctype html public "-//w3c//dtd html 3.2//ja">
2: <html>
3: <head>
4: <title>Simple mail/news search by psql_perl5</title>
5: <meta http-equiv=content-type content=text/html; charset=x-euc-jp>
6: </head>
7: <body bgcolor="white">
8: <form method="post" action="/cgi-bin/search.pl">
9: <table>
10:
11: <h1>psql_perl5 によるメールデータベース検索 </h1>
12:
13: <tr>
14: <td>キーワード </td>
15: <td><input type="text" name="keyword"></td>
16: </tr>
17:
18: <tr>
19: <td>From:</td>
20: <td><input type="text" name="from"></td>
21: </tr>
22:
23: <tr>
24: <td>Subject:</td>
25: <td><input type="text" name="subject"></td>
26: </tr>
27:
28: <tr>
29: <td>日付 </td>
30: <td>
31: <input type="text" name="sdate"> ~ <input type="text" name="edate">
32: </td>
33: </tr>
34:
35: <tr>
36: <td>データベース </td>
37: <td><input type="text" name="dbname" value="test"></td>
38: </tr>
39:
40: </table>
41:
42: <input type="submit" value="Go!">
43: <input type="reset" value="Clear">
44:
45: </body>
46: </html>
47:
```

用目的を示します。



search.pl (リスト 4.4.2)

CGI プログラム本体が search.pl です。本書付属 CD-ROM の examples/perl\_cgi/



## 4.4 Perl インターフェースを使って

表 4.4.1 検索に使用する変数と使用目的

変数名	使用目的
keyword	キーワードによる検索
from	From:による検索
subject	Subject:による検索
sdate	日付指定（開始日時）
edate	日付指定（終了日時）
dbname	データベース名（デフォルト値はtest）

search.pl を /usr/local/apache/share/cgi-bin/search.pl にコピーします。次に、お使いのPerl のパスに合わせ、1 行目を書き換えます。

それでは、search.pl を説明しながらpgsql\_perl5 の使い方について説明しましょう。

2 ~ 4 行目は使用するクラスの宣言です。Pg はpgsql\_perl5 のクラスですが、まず CGI クラスについて触れておきます。

CGI クラスはCGI に関するツールを集めたクラスです。CGI クラスを使うには、まずCGI クラスのオブジェクトを作成します（11 行目）。

CGI クラスにはたくさんの機能がありますが、便利なのが「offline mode」機能です。CGI はエラーが発生してもそれが見えないため、とかくデバッグが面倒になりがちですが、offline mode 機能を使えば、普通のPerl プログラムと同様に端末で実行してデバッグできます。offline mode で実行するには、単にsearch.pl を起動するだけです。

```
$ /usr/local/apache/share/cgi-bin/search.pl
(offline mode: enter name=value pairs on standard input)
```

次に、フォームから渡されるはずのデータを指示に従って入力します。たとえば入力フォームで、「キーワード」に6.4、「From:」にt-ishii、データベースにtest と入力したことを想定する場合は、

```
keyword=6.4&from=t-ishii&dbname=test
```

と入力し、改行してから **CTRL** + **D** など EOF (End of file) を入力します。すると、CGI クラスはそれらのキーワードを取り込んだ後、CGI プログラムの実行を開始します。

4 行目は、構文エラーそのほかのエラーメッセージをブラウザの画面で確認できるようにする宣言です。これがないと例の “ documents contains nodata ” というエラーメッセージがブラウザから表示されるだけで、Apache のログを確認しない限り何が起

こっているのかわからなくなります。

CGI クラスの機能を使うには、まずオブジェクトを作ります (11 行目)。

12 行目では、Content-Type: ヘッダを出力しています。ふだん HTML ファイルの中に記述しているデータに先立ち、http プロトコルではドキュメントの性質などを表すヘッダを送信することになっています。これを忘れるとデータがまったく表示されなくなります。このあたりのことは、PHP では自動的にしてくれていますが、Perl などの CGI ではこのように明示的に実行しなければなりません。

14 行目 ~ 19 行目も CGI クラスの機能を使って、フォームにユーザが入力した値を取り出しています。

CGI クラスのお世話になるのはここまでで、21 行目からはいよいよ `pgsql_perl5` の出番です<sup>注4</sup>。まず `Pg::connectdb("dbname=$dbname")` で、データベースへ接続します。このケースでは、引数としてデータベース名しか渡していませんが、ポート番号など他のパラメータも渡せます。その場合のパラメータの形式は、

```
option1=value option2=value ...
```

となります。option として指定できるのは `dbname` のほか、`host`、`user`、`password`、`authtype`、`port`、`tty`、`options` があります。これ以外にデータベースに接続するための関数として、`setdb($host, $port, $options, $tty, $dbname)` があり、`libpq` の `PQsetdb()` に相当するものです。ただし、この関数では `PQsetdb()` と同様、パスワードを渡すことができません。

`connectdb()` での接続が成功したかどうかは `$conn->status` で確認できます。接続に成功していないのに `$conn` を使うと後で問題が起きますので、必ず確認してください。失敗した際のエラーメッセージは、`$conn->errorMessage` で取得できます。

27 行目からは、ユーザ入力キーワードから検索用の SQL 文を作る処理です。アルゴリズムや生成される SQL 文は 4.1 節の `smst` とまったく同じです。たとえばキーワードに「検索」を指定すると、生成される SQL 文は

```
select distinct b.subject, b.hfrom, b.fname, b.date from words a,
header b where a.word ~* '^検索' and a.fname = b.fname order by
b.date
```

となります。ポイントとしては、全文検索用のテーブル `words` の検索に対しては `~*` で大文字 / 小文字を無視した正規表現を使いつつも、性能を考慮して<sup>注5</sup>、`^` を付加して前方一致検索を強制していることです。PostgreSQL では、`LIKE` や正規表現の検

#### 注 4

`pgsql_perl5` は、ここで使っている Perl5 の “new style” のほか、Perl4 式の “old style” 形式での呼び出しも可能ですが、本書では触れません。

#### 注 5

`words` テーブルは巨大になります。

## 4.4 Perl インターフェースを使って

索にインデックスが使われないため、検索速度が劣化します。前方一致検索だけは例外で、この場合だけはインデックスが使われて高速検索が可能になります。

生成したSQL文は、63行目の`$conn->exec`で実行します。今回実行しているのはSELECT文ですが、その場合、検索が成功したかどうかは`$result->resultStatus`がPGRES\_TUPLES\_OKであるかどうかで判断できます。なお、検索結果が0件の場合でも正常扱いです。検索件数は`$result->ntuples`で取得できます(69行目)。

78行目～83行目のループは、検索結果からSubject:とFrom:のリストを作っています。なお、リストをクリックした際に、該当ファイルにリンクするように、`<a>`タグを作ってファイル名を指すようにしています。たとえば、以下のようになります。

```
<a href="/inbox/3">[pgsql-jp 5323] performance of 6.4【Tatsuo Ishii
<lt;t-ishii@sra.co.jp>】</a>
```

85行目からのサブルーチン`html_escape`は、HTMLにおける以下の特殊文字をエスケープ表現に置き換えます。

```
& < > "
```

以上でCGIプログラムは終わりですが、`libpq`や`Tcl/Tk`のときには必ず使われていたメモリ資源を解放する関数や、接続を切断する関数が使われていないことに気付いたでしょうか。これは`pgsql_perl5`が、検索結果や接続情報を持つオブジェクトへのリファレンスを管理しているからです。これらのオブジェクトへのリファレンスがなくなれば、検索結果を保持するメモリや、PostgreSQLへの接続は自動的に解放されます。ただし、この方法が使えるのは`new style`のときだけです。この意味でも`new style`を採用したほうがよさそうです<sup>注6</sup>。

### 注 6

`pgsql_perl5`のREADMEによれば、`old style`はPostgreSQLの次のバージョンからサポートされなくなるかもしれないとのことです。

### リスト 4.4.2 search.pl

```
1: #! /usr/local/bin/perl -w
2: use CGI;
3: use Pg;
4: use CGI::Carp qw(fatalsToBrowser);
5:
6: my $sstr = "select distinct b.subject, b.hfrom, b.fname, b.date from words a, header b
7: ";
8: my $hasKey = 0;
9:
10: my ($query, $conn, $result, $i, $n);
11:
12: $query = new CGI;
13: print $query->header; # Content-type: header
14:
15: my $keyword = $query->param('keyword');
```

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
16: my $from = $query->param('from');
17: my $subject = $query->param('subject');
18: my $s_date = $query->param('sdate');
19: my $e_date = $query->param('edate');
20: my $dbname = $query->param('dbname');
21:
22: $conn = Pg::connectdb("dbname=$dbname");
23: if (PGRES_CONNECTION_OK ne $conn->status) {
24:     printf("データベース %s に接続できませんでした。理由: %s", $dbname, $conn->errorMessage);
25:     exit;
26: }
27:
28: if ($keyword ne "") {
29:     $qstr .= "where a.word ~* '^$keyword' ";
30:     $hasKey = 1;
31: }
32:
33: if ($from ne "") {
34:     ($hasKey != 0)
35:     and $qstr .= "and "
36:     or $qstr .= "where ";
37:     $qstr .= "b.hfrom ~* '$from' ";
38:     $hasKey = 1;
39: }
40:
41: if ($subject ne "") {
42:     ($hasKey != 0)
43:     and $qstr .= "and "
44:     or $qstr .= "where ";
45:     $qstr .= "b.subject ~* '$subject' ";
46:     $hasKey = 1;
47: }
48:
49: if ($s_date ne "" && $e_date ne "") {
50:     ($hasKey != 0)
51:     and $qstr .= "and "
52:     or $qstr .= "where ";
53:     $qstr .= "b.date >= '$s_date' and b.date <= '$e_date'";
54:     $hasKey = 1;
55: }
56:
57: if ($hasKey == 0) {
58:     printf("検索キーが入力されていません。");
59:     exit;
60: }
61:
62: $qstr .= " and a.fname = b.fname order by b.date";
63:
64: $result = $conn->exec($qstr);
65: if ($result->resultStatus ne PGRES_TUPLES_OK) {
66:     printf("検索に失敗しました。理由: %s", $conn->errorMessage);
67:     exit;
68: }
69:
70: $n = $result->ntuples;
71:
72: if ($n == 0) {
73:     printf("該当するデータはありませんでした。");
```

```

74:     exit;
75: }
76:
77: printf ("%d 件検索しました <br><br>\n", $n);
78:
79: for ($i=0; $i<$n; $i++) {
80:     printf("<a href=\"%s\">", $result->getvalue($i, 2));
81:     printf("%s", &html_escape($result->getvalue($i, 0)));
82:     printf("【%s】", &html_escape($result->getvalue($i, 1)));
83:     printf("</a><br>\n");
84: }
85:
86: sub html_escape {
87:     my($str) = @_;
88:     $str =~ s/&/&amp;/g;
89:     $str =~ s/</&lt;/g;
90:     $str =~ s/>/&gt;/g;
91:     $str =~ s/'/'&quot;/g;
92:     return $str;
93: }
94:

```

#### 4.4.4 その他の関数

ここでは、前項で解説した以外の関数について説明します。old style のみで使用できる関数には触れません。まず、connectdb( ) の戻り値 ( \$conn ) 関係の関数です。

.....

Pg::conndefaults( )

➡ connectdb( ) で使用できるオプションのハッシュへのリファレンスを返します。

.....

\$conn->reset

➡ バックエンドへのコネクションをいったん切断し、新しいコネクションを張り直します。

.....

\$conn->db

➡ 現在のコネクションのデータベース名を返します。

.....

\$conn->user

➡ 現在のコネクションのユーザ名を返します。

.....

\$conn->host

➡ 現在のコネクションのホスト名を返します。

.....

.....  
\$conn->options

➡ 現在のコネクションのオプションを返します .

.....  
\$conn->port

➡ 現在のコネクションのポート番号を返します .

.....  
\$conn->tty

➡ 現在のコネクションのttyを返します .

.....  
\$conn->status

➡ 現在のコネクションのステータス名を返します .

.....  
このほか、デバッグ用の関数などがありますが、ここでは省略します .

次は、\$conn->execの戻り値(\$result) 関係の関数です .

.....  
\$result->nfields

➡ カラム数を返します .

.....  
\$result->fname(\$field\_num)

➡ \$field\_num 番目のカラムの名前を返します .

.....  
\$result->fnumber(\$field\_name)

➡ カラム名が \$field\_name であるカラム番号を返します .

.....  
\$result->ftype(\$field\_num)

➡ \$field\_num 番目のカラムの型に対応するoidを返します . oidは単なる数字なので、実際どのような型に対応しているのかは、システムカタログのひとつであるpg\_typeテーブルを検索します . たとえば、oid 18に対応する型名を探すには、

```
select typename from pg_type where oid = 18;
```

を実行します (この場合 “ char ” が返って来ます) .

.....  
\$result->fsize(\$field\_num)

➡ 指定カラム番号の大きさをバイト数で返します . 可変長データの場合は-1 が返ります .

## 4.4 Perlインターフェースを使って

.....  
\$result->getlength(\$tup\_num, \$field\_num)

➡ 指定タプル / カラムの大きさをバイト数で返します .

.....  
\$result->getisnull(\$tup\_num, \$field\_num)

➡ 指定タプル / カラムのデータが NULL なら 1 を , そうでないなら 0 を返します .

.....  
\$result->oidStatus

➡ INSERT 文を実行すると , 新しく作られたタプルに oid が割り当てられますが , その oid を返す関数です .

.....  
\$result->cmdTuples

➡ INSERT あるいは DELETE を実行したときに影響を受けたタプル数を返します .

.....  
\$result->print(\$fout,\$header,\$align,\$standard,\$html3,\$expanded,  
\$pager,\$fieldSep,\$tableOpt,\$caption, ...)

➡ カラム内容を \$fout に出力します . \$header 以降は出力形態の指定です . \$header から \$pager までは bool 値で , \$fieldSep から \$caption までは文字列です ( 表 4.4.2 ) . これらの引数の後に , 出力する際に表示するカラム名を連ねます .

表 4.4.2 \$fout への出力形態 ( 引数 )

\$header	ヘッダとタプル数を印字するかどうか
\$align	カラムを印字する際に桁そろえを行うかどうか
\$standard	古い形式で出力する ( align も true にすること )
\$html3	HTML のテーブル形式で出力
\$expanded	端末のスクリーンいっぱいに出る
\$pager	pager を使う
\$fieldSep	フィールドの区切り文字
\$tableOpt	HTML 形式の際 , <table ...> に挿入する文字列
\$caption	HTML で出力する際 , caption を <centre><h2>caption</h2></centre> として出力

## 4.4.5 large object 用の関数

pgsql\_perl5 でも, large object を使うことができます.

```

.....
$lobjId = $conn->lo_create($mode)
    ➔ large object を生成して oid を返します. $mode は, PGRES_INV_WRITE (書き込み) または PGRES_INV_READ (読み込み) を指定します.
.....
$conn->$conn->lo_unlink($lobjId)
    ➔ large object を削除します. 失敗すると -1 が返ります.
.....
$lobj_fd = $conn->lo_open($lobjId, $mode)
    ➔ large object を開きます.
.....
$conn->lo_close($lobj_fd)
    ➔ large object を閉じます. 成功すると 0, 失敗すると -1 が返ります.
.....
$bytes = $conn->lo_read($lobj_fd, $buf, $len)
    ➔ large object からデータを読み込み, 読み込んだバイト数を返します. 失敗すると -1 が返ります.
.....
$bytes = $conn->lo_write($lobj_fd, $buf, $len)
    ➔ large object にデータを書き込み, 書き込んだバイト数を返します. 失敗すると -1 が返ります.
.....
$conn->lo_seek($lobj_fd, $offset, $whence)
    ➔ fseek() と同様の関数で, large object 中のオフセットを移動します. $whence は 0 のみ指定してください.
.....
$location = $conn->lo_tell($lobj_fd)
    ➔ ftell() と同様の関数で, large object 中のオフセットを報告します.
.....
$lobjId = $conn->lo_import($filename)
    ➔ UNIX ファイルのデータから large object を作ります.
.....

```



```
.....  
$conn->lo_export($lobjId, $filename)
```

➡ large object のデータ内容を UNIX ファイルに吐き出します。失敗すると -1 ,  
成功すると 1 が返ります。

### 4.4.6 感想など

筆者はどうも喰わず嫌いというか、Perl は極力避けるようにしてきました。今回、本書を執筆するにあたり、やむを得ず Perl5 に取り組んだわけですが、思ったよりは扱いやすくて安心しました。とくに、pgsql\_perl5 の「不要になった資源が自動的に解放される」というところが気に入りました。機能的には Perl はいろいろなスクリプト言語の中でも最強の部類だという点は筆者も認める場所ですので、今後は少しずつ Perl にも親しんでいきたいと思います。

## 4-5

JDBC ドライバ  
を使って

## 4.5.1 JDBC ドライバとは

## 注 1

一般的には “Java Data Base Connectivity” の略とされていますが、Sun は「JDBC は Sun の登録商標であって何かの略称ではない」と主張しています。

## 注 2

Applet : Web ブラウザの中で実行される Java プログラムのことで “小さなアプリケーション” という意味です。

## 注 3

Open Data Base Connectivity : 主に PC で使われているデータベースへのアクセスプロトコルです。

## 注 4

C 言語など、Java 以外の言語で書かれたモジュールを呼び出すメソッドのことです。

JDBC<sup>※1</sup> は、Java で書かれたアプリケーションやアプレット<sup>※2</sup> が、PostgreSQL のようなリレーショナルデータベースにアクセスできるように用意された Java のクラスの集まりです。

JDBC には 2 つのアプリケーションインターフェース (API) があり、1 つは通常のアプリケーションプログラムへのインターフェース (JDBC API) で、もう 1 つは「JDBC ドライバ」へのインターフェース (JDBC ドライバ API) です。

JDBC API はデータベース製品の種類にかかわらず基本的には同一なので、アプリケーションの作成者はデータベース製品の違いにあまり煩わされることなく、アプリケーションロジックに専念することができます。

一方、JDBC ドライバはデータベースに直接アクセスします。個々のデータベース製品による違いは JDBC ドライバが吸収します。したがって、JDBC ドライバはデータベース製品ごとに用意しなければなりません。

JDBC ドライバには 4 種類のタイプがあり、ODBC<sup>※3</sup> ドライバを経由するもの、そうでないものなど、実装方法に違いがあります。

PostgreSQL に付属する JDBC ドライバは、このうち「ダイレクトドライバ」と呼ばれるもので、Java だけで記述されており、しかも ODBC などに頼らず直接データベースに接続できるタイプです。このタイプの JDBC ドライバの利点は、native メソッド<sup>※4</sup> を含まないのでアプレットからも利用できること、プラットフォームを選ばず可搬性が高いこと、ODBC などを経由しないので性能がよいことなどが挙げられます。欠点としては、すべてを Java で記述しなければならないので、クラスライブラリが大きくなりがちなことです。

PostgreSQL の JDBC ドライバは JDBC 1.0 という規格に対応しています。最近、JDBC 2.0 という新しい規格が制定されつつありますが、こちらには対応していないの

ご注意ください。ちなみに、JDK 1.2（およびそのバージョン）に含まれている JDBC のバージョンは 2.0 です。

## 4.5.2 JDBC ドライバのインストール

では、さっそく PostgreSQL の JDBC ドライバをインストールしてみましょう。その前に、お手持ちの Java 環境を確認してください。筆者が利用した環境は以下です。

- JDK : JDK-1.1.7\_v1 + Symantec JIT compiler
- Swing : swing-1.1 beta3

このほか、Sparc/Solaris 2.6（JDK-1.1.6 + swing-1.1 beta2）および FreeBSD 2.2.6-RELEASE（JDK-1.1.6 + swing-1.1 beta2）の環境でも動作が確認できました。また、今回作成するプログラムはすべて Java で記述されていますので、JDK、Swing の適当なバージョンがインストールされていれば、クライアントプログラムは UNIX 以外のプラットフォームでも稼働します。実際、後述のサンプルアプリケーション “jpgpost” は Windows NT 4.0 でも動作しました。

なお、Swing は JDBC ドライバのインストールには必要ありませんが、jpgpost の実行に使います。ちなみに、JDK-1.2 beta に付属の Swing はバージョンが古いので、このサンプルプログラムは動作しませんし、そもそも前述のように JDBC のバージョンが違うので、JDK 1.2 は利用できません。

## Java 環境の設定

環境変数の設定はリスト 4.5.1 のようにしています（bash 用）。参考までに、筆者の環境での JDK と Swing のインストール方法を説明します。

### リスト 4.5.1 環境変数の設定

```
sw=/usr/local/swing-1.1beta3
export JDK_HOME=/usr/local/jdk117_v1
export CLASSPATH=./usr/local/pgsql/lib/postgresql.jar:$JDK_HOME/lib/
classes.zip:$sw/swing.jar:$sw/motif.jar
export PATH=$PATH:$JDK_HOME/bin
export JAVA_HOME=$JDK_HOME
export JAVA_COMPILER=mwjit-981013
```

### JDK の入手

インターネット上からの入手先は以下になります。筆者が入手したパッケージは `jdk117_v1.tar.gz` です。

```
http://business.tyler.wm.edu/mklinux/index.html
```

### JDK の展開

`tar` をほどきます。

```
# cd /usr/local
# tar xzf jdk117_v1.tar.gz
```

### JIT の入手

```
http://business.tyler.wm.edu/mklinux/dl/mwjit.bin-981013.tar.gz
```

### JIT のインストール

```
# cd /tmp
# tar xzf mwjit.bin-981013.tar.gz
# cp mwjit.bin/libmwjit-*.so
/usr/local/jdk117_v1/lib/ppc/green_threads/
```

### Swing の入手

本書を執筆時点（1998 年 11 月上旬）では、Swing の最新バージョンは 扱いであり、Sun の主催する JDC（Java Developers Connection）のメンバーでないと入手できません。とはいえ、JDC のメンバーになるためには、入会審査も会費もいりませんのでご安心ください。入会の申し込みは、

```
http://developer.java.sun.com/developer/
```

で行います。

入会すると Swing-1.1beta3 のダウンロードができるようになります。URL は

```
http://java.sun.com/products/jfc/index.html
```

となっています。

### Swing のインストール

```
cd /usr/local
tar xzf swing11-beta3.tar.Z
```

## JDBC ドライバのインストール

以上、確認ができたところで、インストールを始めましょう。PostgreSQL JDBC ドライバのソースディレクトリに移動します。

```
$ cd /usr/local/src/postgresql-6.3.2/src/interfaces/jdbc
```

6.4 の場合は

```
$ cd /usr/local/src/postgresql-v6.4/src/interfaces/jdbc
```

とします。Solaris の場合、Makefile の87行目をコメントにし、88行目のコメントを外します。

```
postgresql.jar: $(OBSJ)
#      $(JAR) -cOf $@ $$($(FIND) postgresql -name "*.class" -print)
      $(JAR) -cOf $@ `$(FIND) postgresql -name "*.class" -print`

$ make
```

postgres.jar を適当なところ<sup>注5</sup>にコピーし、CLASSPATH に追加します。

```
export CLASSPATH="$CLASSPATH" :/usr/local/pgsql/lib/postgresql.jar
```

これでJDBCが使えるようになります。

以上の方法でうまくコンパイルできない場合は、付属CD-ROMのJava/6.3.2/postgresql.jar またはJava/6.4/postgresql.jar をご利用ください。

注5

たとえば/usr/local/pgsql/libなどです。

### 4.5.3 付属 example プログラムの実行

PostgreSQL JDBC ドライバに付属のサンプルを動かしてみましょう。コンパイルが終わると、図4.5.1のようなメッセージが表示されます。とりあえずexample.basicを試してみましょう（図4.5.2）。このように使い方が表示されますが、実行はされません。実行するためにはデータベース、ユーザ名などの指定を行う必要があります。基本的には、

```
java example.basic jdbc:postgresql:database user password
```

のように、実行するクラス名（この場合basic）の後に、データベースを指定するURL、ユーザ名、パスワードの順に指定します。

URLの部分はjdbc:postgresql:までは固定で、その後にホストやデータベースなどの指定をします。指定方法は3種類あります。

自ホストのデータベースを使用する場合

```
jdbc:postgresql:database
```

図 4.5.1 サンプルプログラムを実行したところ

```
$ make example

javac example/basic.java
javac example/blobtest.java
javac example/datestyle.java
javac example/psql.java
javac example/ImageViewer.java

-----
The examples have been built.

For instructions on how to use them, simply run them. For example:

    java example.blobtest

This would display instructions on how to run the example.
-----
Available examples:

example.basic      Basic JDBC usage
example.blobtest   Binary Large Object tests
example.datestyle  Shows how datestyles are handled
example.ImageViewer Example application storing images
example.psql       Simple java implementation of psql
```

図 4.5.2 example.basic : これでは実行できない

```
$ java example.basic

PostgreSQL basic test v6.3 rev 1

This example tests the basic components of the JDBC driver, demonstrating
how to build simple queries in java.

Usage:
java example.basic jdbc:postgresql:database user password [debug]

The debug field can be anything. It's presence will enable DriverManager's
debug trace. Unless you want to see screens of items, don't put anything in
here.
```

database の部分がデータベース名になります。なお JDBC では、データベースへの接続に必ず INET ドメインのソケットを使うため、postmaster を起動する際に必ず -i オプションを付けて INET ドメインの接続を有効にしておく必要があります。

#### ホスト名とデータベース名を指定する場合

```
jdbc:postgresql://host/database
```

このように、// の後にホスト名、続いて / で区切ってデータベースを指定します。

#### ホスト名とポート番号とデータベース名を指定する場合

```
jdbc:postgresql://host:port/database
```

このように、ホスト名の後に: で区切ってポート番号を指定します。

では、もっとも基本的な例題である basic を実行してみましょう。basic は、basic というテーブルを作った後、データを insert 文で登録し、select を実行します。そして最後に basic テーブルを削除します。

図 4.5.3 の例では、ホスト名として localhost、ポート番号として 5432、データベース名として test を指定しています。また、ユーザ名は t-ishii で、パスワードの設定はしていないので "" となっています。うまく実行できたでしょうか。

図 4.5.3 example.basic を実行したところ

```
$ java example.basic jdbc:postgresql://localhost:5432/test t-ishii ""
PostgreSQL basic test v6.3 rev 1

Connecting to Database URL = jdbc:postgresql://localhost:5432/test
Connected...Now creating a statement

Running tests:
performing a query
a=1 b=1
a=2 b=1
a=3 b=1
a=4 b=2
a=4 b=3
a=4 b=4
performing another query
a=4 b=2
a=4 b=3
a=4 b=4
Now closing the connection
```

## 4.5.4 ImageViewer

次はもう少し興味深いexampleであるImageViewerを試してみましょう。ImageViewerはPostgreSQLのlarge objectを使って画像をデータベースに格納し、管理するアプリケーションです。ImageViewerを使うには、JPEGやGIFフォーマットなどの適当な画像ファイルが必要です。お手元に適当な画像データがない場合は、付属CD-ROMのJava/images/にあるファイルをご利用ください<sup>注6</sup>。

ImageViewerを最初に起動すると、図4.5.4のような画面になります。これはエラーではなく、データベースの初期化を求めているのです。そこでPostgreSQLというメニューを選ぶと、Initializeという項目があるのでこれを選んでください。すぐに終了すると思いますが、これで初期化が完了です。

次に用意した画像をデータベースに登録しましょう。Image Importを選びます。図4.5.5のようなダイアログが表示されるので、画像ファイルを選んでOKを押します。すると、図4.5.6のようなメッセージが端末に表示され、PostgreSQLのデータベースに画像が格納されるとともに、画像が表示されます（図4.5.7）。画面が狭い場合は適当にマウスで大きくしてみてください<sup>注7</sup>。

同じ要領で次々に画像が追加できます。取り込んだ画像のファイル名は左のメニューに表示されます。ファイル名をクリックすれば、その画像が右側のエリアに表示されます。不要になった画像はImage Removeで削除できます。

### 注 6

筆者が手持ちの低解像度のデジタルカメラで撮影したもので、あまりきれいではありませんが。



## 4.5 JDBC ドライバを使って

図 4.5.4 ImageViewer を起動



図 4.5.5 画像をデータベースに登録



注 7

図 4.5.7 の花は筆者が自宅の庭で栽培している「コモンマロー」というハーブで、赤紫色の美しい花は目を楽ませしてくれるだけでなく、ハーブティーとして飲用でき、気管支などに薬効があります。ハーブティーにすると、鮮やかなブルーになりますが、レモンを垂らすとピンクに色が変わるというおもしろい性質を持っています。

図 4.5.6 登録時のメッセージ

```
Importing file
Opening file /hda10/home/t-ishii/doc/book/CD-ROM/Java/images//DSC00001.JPG
Gaining LOAPI
creating blob
Opening 264203
Importing file
Block s=2048 t=0
Block s=2048 t=2048
Block s=2048 t=4096
[中略]
Block s=2048 t=61440
Block s=1457 t=63488
Closing blob
Selecting oid for DSC00001.JPG
Got oid 264203
Import complete
```

図 4.5.7  
登録した画像を開く



## 4.5.5 PostgreSQL JDBC APIの使い方

### はじめに

ここでは、PostgreSQL JDBC APIの使い方を説明します。本書はJavaの解説書ではありませんので、Java言語自体については説明しません。また、JDBCについても、PostgreSQL独自の部分については解説しますが、JDBC自体についても必要最低限の言及しかしませんので、必要ならば参考文献<sup>11</sup>などをご覧ください。

PostgreSQL JDBC ドライバに関するドキュメントは、

```
/usr/local/src/postgresql-6.3.2/interfaces/jdbc/README
/usr/local/src/postgresql-v6.4/interfaces/jdbc/README
```

に簡単なものがあります。個々のAPIに関する詳細なドキュメントは、javadoc形式で書いてあるので、以下の方法で生成します。

```
$ cd /usr/local/src/postgresql-6.3.2/interfaces/jdbc
$ make doc
```

カレントディレクトリにHTML形式のドキュメントが生成されますので、適当なWebブラウザで参照してください。

### JDBC ドライバのロード

JDBC APIを使うためには、JDBCのクラスをimportしておかなければなりません。これは、

```
import java.sql.*;
```

とファイルの先頭に書いておけばOKです。

これだけではまだJDBCは使えません。なぜなら、実際にデータベースにアクセスするJDBCドライバがまだ存在していないからです。

そこで次にPostgreSQL JDBC ドライバをロードします。いくつか方法がありますが、PostgreSQL JDBC ドライバ以外は使わないのであれば、以下のClass.forNameを使うのが簡単です。

```
try {
    Class.forName("postgresql.Driver");
    // Connect to database
} catch (ClassNotFoundException ex) {
    // 例外処理を記述
}
```

## URL の指定方法

JDBC ドライバがロードできたら、次にデータベースに接続します。これは、JDBC API の `DriverManager.getConnection()` メソッドを使います。

```
Connection db = DriverManager.getConnection(url, user, pwd);
```

引数の `user`、`pwd` は `String` 型でユーザ名、パスワードを渡します。`url` も `String` ですが、これは WWW の URL のような形式で接続先のデータベースに関する情報を表現します。これについては前述のように、

```
jdbc:postgresql:database
jdbc:postgresql://host/database
jdbc:postgresql://host:port/database
```

の3つの形式があります。

なお、`DriverManager.getConnection()` もそうですが、以下で説明するほとんどのメソッドは `SQLException` を `throw` するので、`try-catch` で捕捉するようにしておいてください。

## query の実行

query を実行するには、まず `Connection` クラスのメソッドである `createStatement()` を実行して `Statement` オブジェクトを作ります。

```
Statement st = db.createStatement();
```

`SELECT` 文を実行するには、`executeQuery()` メソッドを使います。

```
ResultSet rs = st.executeQuery(sql);
```

そしてResultSet オブジェクトを使い、検索結果を取り出します。SQL データ型をJava のデータ型に変換するためのメソッドがいろいろ用意されていますので、それらを使って適当なデータ型のデータを得ます。たとえば、String 型ならgetString()、int 型ならgetInt() というメソッドを使います。具体的な使い方は、後述のサンプルプログラムの解説をご覧ください。

なおgetBytes() ではちょっと気を付けなければならない点があります。getBytes() はbyte[]を返しますが、該当カラムの型がOID 型の場合、それをlarge object とみなしてlarge object の全データの読み込みを行います。これは便利なのですが、large object が非常に大きい場合は、メモリ不足を起こす可能性もあります。

SELECT 以外の、INSERT、UPDATE、DELETE およびCREATE TABLE などの query はexecuteUpdate() を利用します。

## large object のサポート

large object は、getBytes() または getAsciiStream、getBinaryStream、getUnicodeStream などを使ってアクセスできますが、とくに細かな制御を行いたい場合はlargeobject パッケージを使います。パッケージのimport は以下に行います。

```
import postgresql.largeobject.*;
```

large object をアクセスするには、まずLargeObjectManager のオブジェクトを作成する必要があります。LargeObjectManager のオブジェクトはConnection クラスのインスタンスメソッドgetLargeObjectAPI() を使って作成します。概略は以下のようになります。

```

:
:
db = DriverManager.getConnection(url, user, password);
LargeObjectManager lobj = ((postgresql.Connection)db).getLargeObjectAPI();

```

いったんLargeObjectManager が作られると、それを使ってlarge object の作成、open、read/write などが可能となります。たとえば新たにlarge object を作成し、ローカルファイルの内容をlarge object に格納する処理は以下のようになります。

```

int oid = lom.create();

LargeObject blob = lom.open(oid);

int s,t=0;
while((s=fis.read(buf,0,buf.length))>0) {
    blob.write(buf,0,s);
}

blob.close();

```

詳しくは、LargeObject および LargeObjectManager のドキュメントをご覧ください。

## サポートされていないメソッド

PostgreSQL がサポートしていない SQL 構文は、当然 JDBC でもサポートされませんが、ほかにもサポートされていないメソッドがあります。リスト 4.5.2 の DatabaseMetaData クラスのメソッドは実装されていません。

## 日本語対応

PostgreSQL の JDBC ドライバは、日本語のことは何も考慮していないので、普通に使うとデータベース内に格納された日本語の文字列は化けてしまいます。ひとつの方法としては、プロパティの file.encoding を指定する方法です。具体的には Java の実行時に

リスト 4.5.2 実装されない DatabaseMetaData クラスのメソッド

```

getNumericFunctions()
getStringFunctions()
getSystemFunctions()
getTimeDateFunctions()
getMaxSchemaNameLength()
getMaxCatalogNameLength()
getTablePrivileges()
.getVersionColumns()
getImportedKeys()
getExportedKeys()
getCrossReference()

```

```
java -Dfile.encoding=EUCJIS
```

とします。より完全な方法としては、持田氏の作成されたパッチを使用することです。このパッチでは、`-Dpostgresql.databaseencoding=EUC_JP` とすることにより、日本語が使えるようになります。また日本語だけでなく、EUC\_JP の代わりに EUC\_KR, EUC\_CN, EUC\_TW とすることにより、韓国語、中国語、中国語（台湾）が使用できるようになります<sup>注8</sup>。

注8

もちろん、データベースのエンコーディングがそれと合っている必要があります。

パッチの当て方は、6.3.2 では、

```
$ cd /usr/local/src/postgresql-6.3.2/src/interfaces/jdbc
$ patch -p1 < /mnt/CD-ROM/Java/6.3.2/pgjdbc-mb-1998930.patch
```

です。パッチに関する説明は、`/usr/local/src/postgresql-6.3.2/src/interfaces/jdbc/README.mb.eucjp` にあります。

## 4.5.6 サンプルアプリケーション jpgpost

### jpgpost について

jpgpost は、4.3 節の新旧郵便番号検索システムと同様の機能を Java で実現したものです。データベースについても 4.3 節のものがそのまま使えます。jpgpost では、データベースへのアクセスを PostgreSQL の JDBC ドライバを使って行い、ユーザーインターフェースは Swing で実現しています。

Swing は AWT と同様、GUI を記述するためのクラスライブラリですが、以下のような違いがあります。

高機能のコンポーネントが用意されている

複数の Look & Feel を切り替えることができる

「軽量」コンポーネントを使っている

とくに注目されるのは、表形式のデータを表示できる「JTable」クラスが用意されていることです。RDBMS の行 / 列データモデルは表形式と相性がよいため、jpgpost でも JTable を利用してみました。

なお、Swing はいまだ開発中であり、バージョンによって微妙に違いがあります。jpgpost では swing-1.1beta2 および swing-1.1beta3 というバージョンで動作を確認し

ています<sup>注9</sup>。これ以外のバージョン、たとえばJDK 1.2のバージョンに付属のSwingでは動作しません。

## jpgpost の起動方法

jpgpost のクラスはpostal.jar という名前のjar ファイルで提供されます。postal.jar のあるディレクトリに移動して起動する<sup>注10</sup>か、postal.jar をCLASSPATH に追加しておいてください。たとえば、/usr/local/pgsql/lib にpostal.jar が置いてあるなら、

```
$ CLASSPATH="$CLASSPATH":/usr/local/pgsql/lib/postal.jar
```

となります。

jpgpost のパッケージ名はpostal、起動するクラスはPostal です。実際の起動は以下ようになります。

```
$ java -Duser.language=ja -Dfile.encoding=EUCJIS postal.Postal
```

前述のパッチを当てている場合は、

```
$ java -Duser.language=ja -Dpostgresql.databaseencoding=EUC_JP postal.Postal
```

となります。Java プログラムは一般的にそうですが、とくにSwingを使ったJava アプリケーションはプロセスサイズが巨大になります。筆者の環境では34M バイトを超えていました<sup>注11</sup>。メモリが少ない環境ではかなり厳しいものになるのでご注意ください。

jpgpost が正常に起動すると、図4.5.8のような画面になります。上半分は検索キーの設定部です。下半分は検索結果の表示部で、JTable を使った表形式でデータが表示されます。検索キーの設定方法は、ほぼ前節のシステムと同じです。

図 4.5.8  
jpgpost 起動時の画像



注 9

swing-1.1beta2を使う場合は、ソース中のコメントに従ってimport文を変更してください。

注 10

これはCLASSPATHにカレントディレクトリが含まれている場合です。

注 11

JITを使っているせいもあります。JITを使わなければもっとプロセスサイズが小さくなりますが、今度は実行速度が落ちます。

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

図 4.5.9  
検索結果の表示

The screenshot shows a Java Swing window titled "Postal Code Search System". It has input fields for "旧郵便番号" (Old Postal Code) with value "249", "新郵便番号" (New Postal Code), and a "郵便区別" (Postal District) dropdown set to "選択なし". Below these are fields for "市町村名" (City/Town/Village Name), "町域名" (Municipality Name), "市町村名(カナ)" (City/Town/Village Name in Kana), and "町域名(カナ)". There are buttons for "検索開始" (Start Search), "検索条件クリア" (Clear Search Conditions), and "終了" (End). Below the form is a table with columns: "郵便番号" (Postal Code), "旧郵便番号" (Old Postal Code), "郵便区別" (Postal District), "市町村" (City/Town/Village), "町域名" (Municipality Name), "市町村(カナ)" (City/Town/Village in Kana), and "町" (Town). The table displays results for postal code 249, listing various municipalities in Kanagawa Prefecture.

郵便番号	旧郵便番号	郵便区別	市町村	町域名	市町村(カナ)	町
249-0000	249	神奈川県	逗子市	以下に掲載がない場合	ズシシ	イカニケイ▲
249-0001	249	神奈川県	逗子市	久木	ズシシ	ヒサギ
249-0002	249	神奈川県	逗子市	山の根	ズシシ	ヤマノネ
249-0003	249	神奈川県	逗子市	池子	ズシシ	イケコ
249-0004	249	神奈川県	逗子市	沼間	ズシシ	ヌママ
249-0005	249	神奈川県	逗子市	根山	ズシシ	サクラヤマ
249-0006	249	神奈川県	逗子市	逗子	ズシシ	ズシ
249-0007	249	神奈川県	逗子市	新宿	ズシシ	シンジュク
249-0008	249	神奈川県	逗子市	小坪	ズシシ	コツボ

検索キーを入力して「検索開始」ボタンを押すと、検索結果が下半分に表示されます。

表示が隠れている部分は、スクロールバーを使って見るができます。また、好みのカラム幅に調節することもできます(図4.5.9)。カラムとカラムの間の線にマウスポインタを合わせ、ドラッグしてみてください。

## インストール

jpgpost は4.3節で使ったデータベースを利用しますので、p.195の手順に従ってあらかじめデータベースを作成しておいてください。jpgpost のプログラムも4.3で使ったexamples/pgpost.tar.gzにあります。これを展開してできるpgpost/jpgpostでmakeを行います。

```
$ cd pgpost/jpgpost
```

```
$ make
```

カレントディレクトリにできたpostal.jar を適当なディレクトリにコピーし、CLASSPATH に追加してください。

## ソースコードについて

jpgpost は、2つのクラスから構成されています。

- Postal...このクラスはメイン関数を含み、Swing を使った画面を構成します。



## 4.5 JDBC ドライバを使って

- dbTableModel ... このクラスは、Swing の AbstractTableModel を継承した JTable クラスの「データモデル」です。JDBC を使ったデータベースへのアクセスはすべてこのクラスで行われます。

### Postal (リスト 4.5.3)

6 ~ 15 行目は Swing パッケージの import です。swing 1.1beta3 ではなく、swing 1.1beta2 を使う場合は、6 ~ 9 行目をコメントアウトし、12 ~ 15 行を生かしてください。

18 行目で Swing の JFrame クラスを継承した Postal クラスを宣言しています。JFrame はトップレベルのウィンドウになります。

23 行目では、getContentPane() メソッドを使い、Pane コンテナを得ています。Swing ではすべてのコンポーネント (GUI 部品) は JFrame に添付することはできず、必ず Pane にセットします。

25 ~ 37 行目はコンポーネントの宣言です。表 4.5.1 のように対応しています。

46 行目からは Postal クラスのコンストラクタの記述です。ところで前述したように、Swing では Look & Feel (GUI の外見) を変更することができます。デフォルトではプラットフォームに依存しない “Metal” という Look & Feel になっていますが、49 ~ 55 行目を生かせば Motif 風の Look & Feel になります (図 4.5.10)。違いがわかりますか？

61 ~ 264 行目まではすべてコンポーネントの設定です。

276 行目の execQuery() は「検索開始」ボタンが押されたときに呼ばれるコールバ

表 4.5.1 コンポーネントの宣言

oldPcode	旧郵便番号
newPcode	新郵便番号
city	市町村名
town	町域名
kana_city	市町村名 (カナ)
kana_town	町域名 (カナ)
host	ホスト名
port	ポート番号
user	ユーザ名
pwd	パスワード
db	データベース名
pre	都道府県名メニュー

図 4.5.10 Motif に Look & Feel を変更



## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

ックです。execQuery() の主な処理はSQL 文を組み立てることです。データベースへのアクセスおよびJTable コンポーネントへの表示は、myTableModel.doSelect() にすべてまかせています。

SQL 文の組み立ては、フィールドごとに “makeup\* ” という名前の付いた、お互いによく似た関数を使います。これらのメソッドの呼び出し形式は同じです。

```
public String makeup1 (String sqlstr, String name, String value);
```

sqlstr は組み立て中のSQL 文、name はカラム名、value はフィールドへのユーザの入力値です。このほか、大域変数のisQuery を参照 / セットします。isQuery はフラグで、SQL 文にwhere を付ける必要があるかどうかを判断するのに使っています。

- makeup1 : 郵便番号フィールドの処理を行います。数字以外の文字は無視します。
- makeup2 : 都道府県コードを処理します。
- makeup3 : 市町村名などの文字列フィールドの処理を行います。無条件に前方一致検索を行います。

396 行目はJTable で検索結果を表示するテーブルをJTable で作っています。ちょっとやっかいだったのは、デフォルトのカラム幅がやや狭いので、それを広げる処理でした。ピクセル数を指定してカラム幅を設定する方法もありますが、この方法ではフォントを変更したときにうまくいきません。そこで、「いかにけいさいがないばあい」というデータの中で最長の文字列から必要な表示幅を得る方法を採用しました。

## リスト 4.5.3 Postal.java

```
1: package postal;
2:
3: import java.awt.*;
4: import java.awt.event.*;
5:
6: import javax.swing.*;
7: import javax.swing.table.*;
8: import javax.swing.event.*;
9: import javax.swing.border.*;
10:
11: /* for Swing 1.1 beta2
12: import com.sun.java.swing.*;
13: import com.sun.java.swing.table.*;
14: import com.sun.java.swing.event.*;
15: import com.sun.java.swing.border.*;
16: */
17:
18: public class Postal extends JFrame {
19:
20:     // デフォルトフォント
```

## 4.5 JDBC ドライバを使って

```
21: Font font = new Font("Serif",Font.PLAIN,12);
22:
23: Container pane = getContentPane();
24:
25: JTextField oldPcode;
26: JTextField newPcode;
27: JTextField city;
28: JTextField town;
29: JTextField kana_city;
30: JTextField kana_town;
31: JTextField host;
32: JTextField port;
33: JTextField user;
34: JPasswordField pwd;
35: JTextField db;
36: JComboBox pre;
37: JTable table;
38: dbTableModel myTableModel;
39:
40: boolean isQuery;
41:
42: Dimension horizontalPadding = new Dimension(10, 1);
43: Border raisedBorder = new BevelBorder(BevelBorder.RAISED);
44:
45:
46: Postal() {
47:     super("Postal Code Search System");
48:
49:     /* Motif Look & Feel
50:     try {
51:         UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
52:     } catch (Exception ex) {
53:         System.out.println("Unable to set Motif Look & Feel");
54:     }
55:     */
56:
57:     addWindowListener(new WindowAdapter() {
58:         public void windowClosing(WindowEvent e) {System.exit(0);}
59:     });
60:
61:     pane.setLayout(new BoxLayout(pane,BoxLayout.Y_AXIS));
62:
63:     //-----
64:     // pcodes
65:     //
66:     JPanel pcodes = new JPanel();
67:     pcodes.setBorder(new BevelBorder(BevelBorder.LOWERED));
68:
69:     // 旧〒番号
70:     JLabel l = new JLabel("旧郵便番号");
71:     l.setFont(font);
72:     pcodes.add(l);
73:
74:     oldPcode = new JTextField(10);
75:     oldPcode.setToolTipText("旧郵便番号を123-45の形で入力します");
76:     oldPcode.setFont(font);
77:
78:     pcodes.add(oldPcode);
```

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
79:
80:     // 新干番号
81:     l = new JLabel("新郵便番号");
82:     l.setFont(font);
83:     pcodes.add(l);
84:     newPcode = new JTextField(10);
85:     newPcode.setFont(font);
86:     newPcode.setToolTipText("新郵便番号を 123-4567 の形で入力します");
87:     pcodes.add(newPcode);
88:
89:     //-----
90:     // pre
91:     //
92:     l = new JLabel("都道府県");
93:     l.setFont(font);
94:     pcodes.add(l);
95:     pre = new JComboBox(Prefecture.preList);
96:     pre.setFont(font);
97:     pre.setSelectedItem(Prefecture.preList[0]);
98:     pcodes.add(pre);
99:
100:    pane.add(pcodes);
101:
102:    //-----
103:    // labelPanel
104:    //
105:    JPanel labelPanel = new JPanel(new GridLayout(0, 1));
106:    // 市町村名
107:    l = new JLabel("市町村名");
108:    l.setFont(font);
109:    labelPanel.add(l);
110:    // 町域名
111:    l = new JLabel("町域名");
112:    labelPanel.add(l);
113:    l.setFont(font);
114:    // 市町村名(カナ)
115:    l = new JLabel("市町村名(カナ)");
116:    labelPanel.add(l);
117:    l.setFont(font);
118:    // 町域名(カナ)
119:    l = new JLabel("町域名(カナ)");
120:    labelPanel.add(l);
121:    l.setFont(font);
122:
123:    //-----
124:    // textPanel
125:    //
126:    JPanel textPanel = new JPanel(new GridLayout(0, 1));
127:    city = new JTextField(30);
128:    city.setFont(font);
129:    textPanel.add(city);
130:
131:    town = new JTextField(30);
132:    town.setFont(font);
133:    textPanel.add(town);
134:
135:    kana_city = new JTextField(30);
136:    kana_city.setFont(font);
```

## 4.5 JDBC ドライバを使って

```
137:     textPanel.add(kana_city);
138:
139:     kana_town = new JTextField(30);
140:     kana_town.setFont(font);
141:     textPanel.add(kana_town);
142:
143:     JPanel areaPanel = new JPanel();
144:     areaPanel.add(LabelPanel);
145:     areaPanel.add(textPanel);
146:
147:     //-----
148:     // ButtonPanel
149:     //
150:     JPanel buttonPanel = new JPanel();
151:     buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));
152:
153:     // 検索開始ボタン
154:     JButton execQueryButton = new JButton("検索開始");
155:     execQueryButton.setFont(font);
156:     execQueryButton.addActionListener(new ActionListener() {
157:         public void actionPerformed(ActionEvent e) {
158:             setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
159:             execQuery();
160:         }
161:     });
162:
163:
164:     execQueryButton.setBorder(raisedBorder);
165:
166:     buttonPanel.add(execQueryButton);
167:
168:     buttonPanel.add(Box.createRigidArea(horizontalPadding));
169:
170:     // 検索条件クリアボタン
171:     JButton clearButton = new JButton("検索条件クリア");
172:     clearButton.setFont(font);
173:     clearButton.addActionListener(new ActionListener() {
174:         public void actionPerformed(ActionEvent e) {
175:             clear();
176:         }
177:     });
178:
179:     clearButton.setBorder(raisedBorder);
180:
181:     buttonPanel.add(clearButton);
182:
183:     JPanel subPanel = new JPanel();
184:     subPanel.add(buttonPanel);
185:
186:     JPanel middlePanel = new JPanel();
187:     middlePanel.add(areaPanel);
188:     middlePanel.add(subPanel);
189:
190:     //-----
191:     // ButtonPanel
192:     //
193:     JPanel exitPanel = new JPanel(new GridLayout(0, 1));
194:     exitPanel.setBorder(new BevelBorder(BevelBorder.LOWERED));
```

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
195:
196:     JButton exitButton = new JButton("終了");
197:     exitButton.setFont(font);
198:     exitButton.addActionListener(new ActionListener() {
199:         public void actionPerformed(ActionEvent e) {
200:             int opt = JOptionPane.showConfirmDialog(null,
201:                 "終了してよろしいですか?",
202:                 "Question",
203:                 JOptionPane.OK_CANCEL_OPTION,
204:                 JOptionPane.QUESTION_MESSAGE
205:             );
206:             if (opt == 0) {
207:                 System.exit(0);
208:             }
209:         }
210:     });
211:     exitButton.setBorder(raisedBorder);
212:     exitPanel.add(exitButton);
213:
214:     middlePanel.add(exitButton);
215:
216:     pane.add(middlePanel);
217:
218:     //-----
219:     // host, port, user etc.
220:     //
221:     JPanel url = new JPanel();
222:     url.setBorder(new BevelBorder(BevelBorder.LOWERED));
223:
224:     l = new JLabel("ホスト名");
225:     l.setFont(font);
226:     url.add(l);
227:     host = new JTextField("localhost", 8);
228:     host.setFont(font);
229:     url.add(host);
230:
231:     l = new JLabel("データベース");
232:     l.setFont(font);
233:     url.add(l);
234:     db = new JTextField("postal", 8);
235:     db.setFont(font);
236:     url.add(db);
237:
238:     l = new JLabel("ユーザ名");
239:     l.setFont(font);
240:     url.add(l);
241:     user = new JTextField(System.getProperty("user.name"), 8);
242:     user.setFont(font);
243:     url.add(user);
244:
245:     l = new JLabel("パスワード");
246:     l.setFont(font);
247:     url.add(l);
248:     pwd = new JPasswordField(8);
249:     pwd.setFont(font);
250:     url.add(pwd);
251:
252:     l = new JLabel("ポート番号");
```

```

253:     l.setFont(font);
254:     url.add(l);
255:     port = new JTextField("5432", 5);
256:     port.setFont(font);
257:     url.add(port);
258:
259:     pane.add(url);
260:
261:     //-----
262:     // 検索結果表示テーブル
263:     //
264:     pane.add(createTable());
265: }
266:
267: public void clear() {
268:     oldPcode.setText("");
269:     newPcode.setText("");
270:     city.setText("");
271:     town.setText("");
272:     kana_city.setText("");
273:     kana_town.setText("");
274: }
275:
276: public void execQuery() {
277:     isQuery = false;
278:     String sql = "select newcode,oldcode,pid,city,town,kana_city,kana_town from postal";
279:     sql = makeup1(sql,"oldcode", oldPcode.getText());
280:     sql = makeup1(sql,"newcode", newPcode.getText());
281:     sql = makeup2(sql,"pid", pre.getSelectedIndex()-1);
282:     sql = makeup3(sql,"city",city.getText());
283:     sql = makeup3(sql,"town",town.getText());
284:     sql = makeup3(sql,"kana_city",kana_city.getText());
285:     sql = makeup3(sql,"kana_town",kana_town.getText());
286:
287:     if (isQuery == false) {
288:         Object msg[] = {"検索キーが入力されていません"};
289:         JOptionPane.showMessageDialog(pane, msg, "Info",
290:             JOptionPane.WARNING_MESSAGE);
291:     return;
292:     }
293:
294:     sql = sql.concat(" order by newcode");
295:     String password = new String(pwd.getPassword());
296:
297:     Object msg[] = myTableModel.doSelect(sql,
298:                                         host.getText(),
299:                                         port.getText(),
300:                                         db.getText(),
301:                                         user.getText(),
302:                                         password
303:                                         );
304:
305:     if (msg != null) {
306:         JOptionPane.showMessageDialog(pane, msg, "Error",
307:             JOptionPane.ERROR_MESSAGE);
308:     }
309:     myTableModel.fireTableDataChanged();
310: }

```

## Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
311:
312: /**
313:  * 郵便番号フィールドの問い合わせ作成。数字以外の文字は無視する。
314:  * @param sqlstr SQL文
315:  * @param name カラム名
316:  * @param value ユーザ入力の検索キー
317:  */
318: public String makeUp1 (String sqlstr, String name, String value) {
319:
320:     StringBuffer sql = new StringBuffer(sqlstr);
321:     StringBuffer v = new StringBuffer();
322:
323:     if (value.length() != 0) {
324:         for (int i=0;i<value.length();i++) {
325:             String s = value.substring(i,i+1);
326:             try {
327:                 Integer idigit = new Integer(s);
328:                 v.append(s);
329:             } catch (NumberFormatException ex) {
330:             }
331:         }
332:         if (isQuery == true) {
333:             sql.append(" and");
334:         } else {
335:             sql.append(" where");
336:         }
337:         sql.append(" "+name+" = '"+v+"'");
338:         isQuery = true;
339:     }
340:     return new String(sql);
341: }
342:
343: /**
344:  * 都道府県コードフィールドの問い合わせ作成
345:  * @param sqlstr SQL文
346:  * @param name カラム名
347:  * @param value ユーザ入力の検索キー(都道府県コード)
348:  */
349: public String makeUp2 (String sqlstr, String name, int value) {
350:     StringBuffer sql = new StringBuffer(sqlstr);
351:
352:     if (value >= 0) {
353:         if (isQuery == true) {
354:             sql.append(" and");
355:         } else {
356:             sql.append(" where");
357:         }
358:         sql.append(" "+name+" = '"+value+"'");
359:         isQuery = true;
360:     }
361:     return new String(sql);
362: }
363:
364: /**
365:  * 市町村名などの文字列フィールドの問い合わせ作成。
366:  * 無条件に前方一致検索を行う
367:  * @param sqlstr SQL文
368:  * @param name カラム名
```



```

369:  * @param value ユーザ入力の検索キー
370:  */
371:  public String makeup3 (String sqlstr, String name, String value) {
372:
373:      StringBuffer sql = new StringBuffer(sqlstr);
374:      StringBuffer v = new StringBuffer();
375:
376:      if (value.length() != 0) {
377:          if (value.substring(0,1).equals("^") == false) {
378:              v.append("^");
379:          }
380:          v.append(value);
381:          if (isQuery == true) {
382:              sql.append(" and");
383:          } else {
384:              sql.append(" where");
385:          }
386:          sql.append(" "+name+" = '"+v+"'");
387:          isQuery = true;
388:      }
389:      return new String(sql);
390:  }
391:
392:  /**
393:   * 検索結果表示テーブルを作る。
394:   *
395:   */
396:  public JScrollPane createTable() {
397:      myTableModel = new dbTableModel();
398:      table = new JTable(myTableModel);
399:      table.setFont(font);
400:      table.getTableHeader().setFont(font);
401:      table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
402:      JScrollPane scrollpane = new JScrollPane(table);
403:
404:      // 市町村名(カナ)と町域名(カナ)のカラム幅を広げる。
405:      Component c = table.getDefaultRenderer(myTableModel.getColumnClass(5)).
406:          getTableCellRendererComponent(
407:              table, "いかにけいさいがないばあい",
408:              false, false, 0, 5);
409:      int width = c.getPreferredSize().width;
410:      TableColumn tc = table.getColumnModel().getColumn(5);
411:      tc.setPreferredWidth(width);
412:      tc = table.getColumnModel().getColumn(6);
413:      tc.setPreferredWidth(width);
414:      return scrollpane;
415:  }
416:
417:  public static void main(String s[]) {
418:      Postal postal = new Postal();
419:      postal.pack();
420:      postal.setVisible(true);
421:  }
422: }
423:

```

dbTableModel (リスト 4.5.4)

21行目でテーブルの行数を設定しています。ここでは簡単のために固定の行数にしていますが、画面から行数を設定できるようにしたほうがよいでしょう。

23行目は「データモデル」のコンストラクタです。データモデルはJTable特有の概念で、テーブルコンポーネントを操作するのに必要な操作の集合をJavaのクラスで表現したものです。ここでは、doSelect以外のメソッドはすべてテーブルモデルのメソッドです。コンストラクタ以外には表4.5.2のメソッドがあります。

56行は実際に検索を行うdoSelectです。引数は以下です。

sql : SQL文  
host : ホスト名  
port : ポート番号  
dbname : データベース名  
user : ユーザ名  
pwd : パスワード

まず、67行目でJDBCドライバをロードします。次に74～85行目まででURLを作成し、87行目でデータベースと接続します。98行目で問い合わせを実行します。検索結果があれば、103行目から検索結果を取り出します。

next()は、検索結果のポインタを次の行に移動します。データがなくなればfalseを返すので、これを利用してループを構成できます。なお、検索結果が多すぎる場合は、getRowCount()の返す値(100)で打ち切ります。

104行目からの内側のループは、カラムの数だけ回ります。

最初のカラムは新郵便番号です。109行目で郵便番号の3桁目と4桁目の間に-を挿入しながらsetValueAtを使ってテーブルにセットします。

次のカラムは旧郵便番号です。旧郵便番号は、3桁の場合は-をセットしないので、length()でその判断が必要です(113行目)。

都道府県カラムは、コードでデータが格納されていますが、見やすくするために文

表 4.5.2 コンストラクタ以外のメソッド

getColumnCount	列数を返す
getColumnName	列名を返す
getRowCount	行数を返す
getValueAt	指定行、列の値を返す
setValueAt	指定行、列に値を設定する
getColumnClass	指定列のクラスを返す

字列に変換します (120 行目)。

これ以外のカラムでは、検索結果の文字列をそのままセットします。

130 行目からのループは、検索結果が100 件に満たない場合に空白文字をセットします。

すべての処理が終わったら、142 行目と143 行目でclose() を呼び出し、後処理をします。この時点でデータベースとの接続は切断されます。

リスト 4.5.4 dbTableModel.java

```

1: package postal;
2:
3: import java.sql.*;
4:
5: import javax.swing.table.AbstractTableModel;
6: import javax.swing.event.TableModelEvent;
7:
8: /* for swing 1.1beta2
9: import com.sun.java.swing.table.AbstractTableModel;
10: import com.sun.java.swing.event.TableModelEvent;
11: */
12:
13: class dbTableModel extends AbstractTableModel {
14:
15:     static String columnName[] = {
16:         "新郵便番号", "旧郵便番号", "都道府県", "市町村", "町域名",
17:         "市町村(カナ)", "町域名(カナ)"};
18:
19:     String[][] data;
20:
21:     int maxRows = 100;
22:
23:     public dbTableModel() {
24:         data = new String[maxRows][columnName.length];
25:         for (int i = 0; i < maxRows; i++) {
26:             for (int j = 0; j < columnName.length; j++) {
27:                 data[i][j] = new String();
28:             }
29:         }
30:     }
31:
32:     public int getColumnCount() {
33:         return columnName.length;
34:     }
35:
36:     public String getColumnName(int column) {
37:         return columnName[column];
38:     }
39:
40:     public int getRowCount() {
41:         return maxRows;
42:     }
43:
44:     public Object getValueAt (int row, int column) {
45:         return new String(data[row][column]);

```

# Chapter 4

## Make Applications ~ 各種アプリケーションを作成する

```
46: }
47:
48: public void setValueAt (Object value, int row, int column) {
49:     data[row][column] = (String)value;
50: }
51:
52: public Class getColumnClass (int columnIndex) {
53:     return getValueAt(0,columnIndex).getClass();
54: }
55:
56: public Object[] doSelect(String sql,
57:                           String host,
58:                           String port,
59:                           String dbname,
60:                           String user,
61:                           String pwd) {
62:
63:     Connection db;
64:     Statement st;
65:
66:     try {
67:         Class.forName("postgresql.Driver");
68:         // Connect to database
69:     } catch (ClassNotFoundException ex) {
70:         Object msg[] = {"JDBCドライバをロードできませんでした"};
71:         return msg;
72:     }
73:
74:     String url = "jdbc:postgresql:";
75:
76:     try {
77:         if (host.length() != 0) {
78:             url = url.concat("///"+host+":");
79:         }
80:         if (port.length() != 0) {
81:             url = url.concat(port+"/");
82:         }
83:         if (dbname.length() != 0) {
84:             url = url.concat(dbname);
85:         }
86:         System.out.println("Connecting to Database URL = " + url);
87:         db = DriverManager.getConnection(url, user, pwd);
88:         st = db.createStatement();
89:
90:     } catch (SQLException ex) {
91:         Object[] msg = {"URL: "+url, "にユーザ: "+user+" で接続できませんでした",
92:                         "理由: "+ex};
93:         return msg;
94:     }
95:
96:     try {
97:         System.out.println("sql..." + sql);
98:         ResultSet rs = st.executeQuery(sql);
99:
100:         int j = 0;
101:
102:         if (rs != null) {
103:             while (rs.next() && j < getRowCount()) {
```

## 4.5 JDBC ドライバを使って

```
104:     for (int i=0;i < 7;i++) {
105:         String s;
106:         switch (i) {
107:             case 0:
108:                 s = rs.getString(i+1);
109:                 setValueAt(s.substring(0,3)+"-"+s.substring(3,7),j,i);
110:                 break;
111:             case 1:
112:                 s = rs.getString(i+1);
113:                 if (s.length() > 3) {
114:                     setValueAt(s.substring(0,3)+"-"+s.substring(3,5),j,i);
115:                 } else {
116:                     setValueAt(s,j,i);
117:                 }
118:                 break;
119:             case 2:
120:                 setValueAt(Prefecture.preList[rs.getInt(3)+1],j,2);
121:                 break;
122:             default:
123:                 setValueAt(rs.getString(i+1),j,i);
124:                 break;
125:         }
126:     }
127:     j++;
128: }
129: }
130: while (j < getRowCount()) {
131:     for (int i=0;i < 7;i++) {
132:         setValueAt(new String(),j,i);
133:     }
134:     j++;
135: }
136:
137: } catch (SQLException ex) {
138:     Object[] msg = {"SQL エラーが発生しました",ex};
139:     return msg;
140: }
141: try {
142:     st.close();
143:     db.close();
144: } catch (SQLException ex) {
145: }
146: Object msg[] = null;
147: return msg;
148: }
149: }
150:
```

### 4.5.7 感想など

実は今回作成したjpgpostは、筆者がはじめて作ったJavaアプリケーションです。「どうせはじめて作るのなら最初からSwingを使ってみよう」と思い立ったまではよかったのですが、まだまだSwingの解説書は少なく、考えていたよりも完成までに時間がかかってしまいました。しかし苦勞したのはそれくらいで、後はJavaによるオブジェクト指向プログラミング言語を十分楽しむことができました。オブジェクト指向言語に関する筆者の乏しい経験の中から判断するに、少なくとも、JavaはC++よりははるかに扱いやすいようです。これなら実用サイズの大きなシステムも十分作れそうな気がします。

Javaの問題点は実行系にあります。なんせプロセスサイズが簡単に30Mバイトを超えるのでは、いくらメモリが安くなったとはいえ、負担が大きすぎます。この点に関しては、JVM (Java Virtual Machine) の進歩に期待するしかないわけですが、本書が刊行されるころにはたぶんSunのHotSpotも登場しているでしょうし、もっと進歩したJVMがサードパーティからリリースされるかもしれません。長い目で見ていきたいと思います。

また、今のところPostgreSQLのバックエンドにはJavaのインターフェースはないのですが、今後ユーザ定義関数などからJavaが使えるようになればおもしろいこともできそうで、こちらも楽しみです。