AppleScript Working》
4 – AppleScript から Java を呼び出す

AppleScript Studio で作成するアプリケーションは、クラスやあるいはオブジェクトのメソッドを呼び出すという機能が利用できる。AppleScript に対するインタフェースを特別に定義したわけではないようなクラスを直接呼び出せるのだ。AppleScript Stduio のチュートリアルにも、Objective-C での事例が掲載されており、Cocoa のクラスにあるメソッド等を呼び出す方法が書いてある。Cocoa だから Objective-C だけかという雰囲気もあるが、実は、Java で作ったクラスも AppleScript から直接呼び出すことができる。これらは、機能拡張の手法に大きな柔軟性を与えるものだ。

AppleScript からのメソッド呼び出し

AppleScript Studio の Application Suite には、call method というコマンドがある。アプリケーションの中なら、そのまま使えるわけだが、用語辞典を日本語におおまかになおすと次のようなものだ。

```
call method　メソッド名
  [of class　クラス名]
  [of object　オブジェクト]
  [with parameter　引数]
  [with parameters　引数のリスト]
```

つまり、いきなり call method コマンドを使えるのだが、クラスないしはオブジェクトの一方を指定し、呼び出すメソッドがあれば、引数あるいは引数リストのいずれか一方を指定することになる。このメソッドを使って、Objective-C のクラスを呼び出す方法が、以下の文書に掲載されている。

◇More on AppleScript Studio
http://developer.apple.com/techpubs/macosx/CoreTechnologies/AppleScriptStudio/applescriptstudio/chapter3/More_on_App_ript_Studio.html

Objective-C の場合はクラスメソッドであれば、クラス名とメソッドを与えればいいが、

ウインドウ上のボタンへの参照をオブジェクトとして指定してもかまわないのである。また、自分でソースプログラムを供給したクラスを指定し、そこに定義されているメソッドを呼び出すこともできる。この場合、一般にはクラスメソッドを定義して、そこでさまざまな処理を組み込むことになるだろう。Objective-C で作成するのが基本となっているが、クラス定義部分はもちろん、Objective-C での文法に従う必要があるが、メソッド内の処理では C の記述も可能であるため、その意味では Toolbox の API をそのまま呼び出したり、あるいは Cocoa のクラスを利用するということは可能である。しかしながら、後で説明するように、Objective-C だけでなく、Java でも同様にプログラミングはできる。それぞれ長所や短所はあるが、両方できるという点は選択肢の上でも広くなり好ましいことだと言えるだろう。なお、Carbon の API を使うには C 言語の API であるため、Objective-C で作る方が何かと楽だろう。一方、Java だと Pure Java のさまざまなライブラリが使えるという点が大きい。Java だと実行速度の上では若干不利であるのだが、AppleScript 自体の処理スピードのことを考慮すれば、Java の処理能力低下はほとんど考慮外になることもあるだろう。

Objective-C のクラスを定義して呼び出す

自分で作った Objective-C のクラスのメソッドを呼び出す方法をまずは説明しよう。Project Builder で、AppleScript Application を選択して作ったプロジェクトがあるとする。そのプロジェクトに、次のようにして、Objective-C のソースファイルを追加する。まず、「ファイル」メニューから「新規ファイル」（Command+N）を選択する。ダイアログボックスが表示されるので、ここでは Cocoa の Objective-C クラスを選択する。

新規ファイルで Objective-C Class を選択する

すると、クラス名やこのクラスを追加するターゲットを指定するダイアログボックスになる。ここでは、SoundPlay.m というファイルに加えて、ヘッダのファイルも作成し、最初からあるターゲットに追加するように指定した。

ファイル名とターゲットを指定する

これで、SoundPlay.m と SoundPlay.h という 2 つのファイルが追加された。ここで、プログラムを追加するが、文字通り、サウンドを鳴らすプログラムを追加したい。システム環境設定の「サウンド」で、警告音の一覧が出るが、これは/System/Library/Sounds および~/Library/Sounds にある AIFF ファイル（拡張子は.aiff）の一覧である。NSSound

の機能を使うと、これらのサウンドファイルを、ファイル名を指定するだけで鳴らすことができる。どんな名前でいいのかは、システム環境設定で見る方が確実だろう。
それぞれのリストは次のようにした。まず、ヘッダファイルは次の通りだ。

```
//  SoundPlay.h

#import <Cocoa/Cocoa.h>

@interface SoundPlay : NSObject {            }

+ (void)playSound;
+ (void)playSound:(NSString *)soundName;
+ (void)playSound:(NSString *)soundName async:(BOOL)isAsync;

@end
```

ヘッダでは、とりあえず、クラスメソッドのインタフェースを記述する。ここでは、3種類の playSound メソッドを定義するが、インポートするヘッダは、初期状態の Foundation.h から Cocoa.h にしておくのが何かと便利だろう。
最初の playSound は、Temple という名前のサウンドを無条件に鳴らす。2 つ目は 1 つの引数を指定し、その引数にサウンド名を指定して、そのサウンドを鳴らす。3 つ目のものは、2 つの引数を取り、サウンド名と非同期で鳴らすかどうかを論理値でそれぞれ指定する。2 つ目の引数は同期で鳴らすなら、true を指定する。なお、最初の 2 つのメソッドはサウンドを非同期で鳴らすため、サウンドを慣らしている最終に次の処理に移動する。
そして、インプリメントを記述する SoundPlay.h は次のようにした。

```
//  SoundPlay.m

#import "SoundPlay.h"


@implementation SoundPlay
```

```
+ (void)playSound;
{
	NSSound * theSound = [NSSound soundNamed:@"Temple"];
	[theSound play];
}
+ (void)playSound:(NSString *)soundName;
{
	NSSound * theSound = [NSSound soundNamed:soundName];
	[theSound play];
}
+ (void)playSound:(NSString *)soundName async:(BOOL)isAsync;
{
	NSSound * theSound;
	theSound    = [NSSound soundNamed:soundName];
	[theSound play];
	if(isAsync)
		while([theSound isPlaying])
			sleep(1);
}

@end
```

いずれも、NSSound というクラスにあるクラスメソッド、soundNamed で指定した名前のサウンドのインスタンスを得て、play メソッドで実際に音を鳴らしている。なお、同期で鳴らすために、isPlaying メソッドで鳴らしている途中かどうかを判断しながら、鳴らしている途中であれば、sleep で 1 秒待って鳴り終わるのを待つというわけだ。
これらのメソッドを AppleScript で使うプログラムをまず見ていただきたい。

```
call method "playSound" of class "SoundPlay"
call method "playSound:" of class "SoundPlay" with parameter "Frog"
call method "playSound:async:" of class "SoundPlay" with parameters {"Temple",
true}
```

まず、クラスメソッドの利用なので、of class "SoundPlay" がいずれのコマンドにもつけられている。1 行目は引数のない playSound メソッドを呼んでいる。2 行目は引数が 1 つの playSound メソッドを呼んでいるが、結果的にコロンが 1 つメソッド名についている。3 つ目は引数が 2 つの playSound を呼び出しているが、「playSound:async:」となっている点に中止してもらいたい。いずれにしても、メソッドのインタフェース定義の部分にあるメソッド名に加えて、メッセージキーワードとコロンを含めたものをメソッド名として指定する必要がある。

ここで、まず注意したいのは、正しくないメソッド名を指定してもエラーは出なという点だ。だから、間違えた AppleScript の記述をしても「何もおこらない」かのような現象になってしまう。

それから、引数のあるものは、with parameter(s)で指定を行う。ここで、文字列を指定すると、Objective-C 側のプログラムでは（NSString *）で受ければいいし、論理値なら BOOL、整数なら int のように、いずれにしても、素直に対応付けられる型で受け取ることができる。また、戻り値がある場合も同様だ。複数の引数があるときには、リスト型で順序を間違えないように指定する。ただし、前のプログラム例で、with parameter {"Temple", true}　つまり「s」が 1 つないだけで、これは、「配列の引数が 1 つ指定されている」とみなされてしまう。そして、1 番目の引数に大カッコの中身の値が（NSArray *）で引き渡されてしまうので、2 つ目の引数には値は渡らないのである。

なお、Cocoa のクラスのクラスメソッドも同様に呼び出すことができる。クラス名に「NSNumber」などと指定をすれば良い。

Java のクラスを呼び出す

Java の場合は少し実験をしながら作ったクラスで例を示そう。やはり、AppleScript Application をプロジェクトのテンプレートとして選択したアプリケーションで、「ファイル」メニューの「新規ファイル」（Command+N）を選択して、新しいファイルを作るが、Java class を選択して現在のターゲットにそのファイルを追加するようにしておく。ここでは、JavaObj.java というファイルを作ることにした。そして次のように、クラスメソッドを作った。

```
//   JavaObj.java

import com.apple.cocoa.foundation.*;
import com.apple.cocoa.application.*;


public class JavaObj {

      static public int actionA()        {
            return 99;
      }
      static public int actionB(int a)  {
            return a*99;
      }
      static public int actionC(int a, int b)      {
            return (a+b)*99;
      }
      static public NSArray actionD()            {
            Object obj[] = {new Integer(19), "Sender OK", new Float(0.876)};
            return new NSArray(obj);
      }
      static public NSDictionary actionE()      {
            Object obj[] = {new Integer(19), "Sender OK", new Float(0.876)};
            Object key[] = {"Age", "Message", "The Number"};
            return new NSDictionary(obj, key);
      }
      static int stock = 0;
      static public void setStock(int n)          {          stock = n;          }
      static public int getStock()      {          return stock;       }
}
```

引数がいくつもある場合や、リスト、レコードを戻せるかといった点を中心にチェックしたわけだが、これらの呼び出しを行う AppleScript のプログラムは次のようになる。まず、引数に数に応じて、メソッド名をどのように指定しないといけないかを探るシ

リーズ、actionA〜ActionC の呼び出し結果を見てみよう。以下のステートメントは問題なく動くことをもちろん確認している。メソッドを呼び出した結果を、display dialog でダイアログ表示してみた。

```
set x to call method "actionA" of class "JavaObj"
display dialog x as string
set x to call method "actionB:" of class "JavaObj" with parameter 10
display dialog x as string
set x to call method "actionC::" of class "JavaObj" with parameters {10, 4}
display dialog x as string
```

引数のない actionA メソッドはそのままメソッド名を書けばいいが、引数のあるものの場合は、コロンがその引数の数に応じて必要になる。つまり、引数の数だけコロンを続けたものをメソッド名として指定する必要がある。これは、Java と Objective-C のブリッジ部分の仕様であるのだろう。この場合は、Objective-C のメソッド呼び出しの機能で、Java のメソッドを呼んでいる。Java のメソッドは、メッセージキーワードのない状態で定義されていると考えれば、Objective-C のクラスを呼ぶ場合の記述と対応がとれるだろう。
続いて、actionD は NSArray 型のデータを戻してみた。actionE は NSDictionary 型のデータを戻してみた。

```
set x to call method "actionD" of class "JavaObj"
display dialog (item 2 of x) as string

set x to call method "actionE" of class "JavaObj"
display dialog (item 2 of x) as string
display dialog (Age of x) as string
```

ここで、actionD の戻り値は、確かに AppleScript でのリストになっているのだが、actionE の戻り値はレコードにはなっていない。理由は分からないが、NSDictionary ではだめなのかあるいは AppleScript システム側に問題があるかもしれない。
続いて、結論を言えば当たり前なのだが、ここでの JavaObj クラスは static であること

から、値を覚えるかどうかを試してみたのが、setStock、getStock のメソッドだ。いずれも、static 変数の stock への値の設定や取り出しを行っている。setStock で設定した値を、getStock で取り出されるのは以下のプログラムで確認できる。

```
call method "setStock" of class "JavaObj" with property 123
set x to call method "getStock" of class "JavaObj"
display dialog x as string
```

したがって、static な変数を定義しておけば、call method をまたがって、同じ値を共有できるということになる。複雑な処理をさせる場合には、こうした手法も有効だろう。なお、Java のライブラリを使うには、たとえば次のようにプログラムを作成すればよい。Math クラスに max というメソッドがあり、引数を 2 つ取る。したがって、メソッド名は「max::」となる。また、クラス名は、フルパスで指定するのが基本のようである。

```
set x to call method "max::" of class "java.lang.Math" with parameters {3, 4}
```

JDBC を利用したサンプル

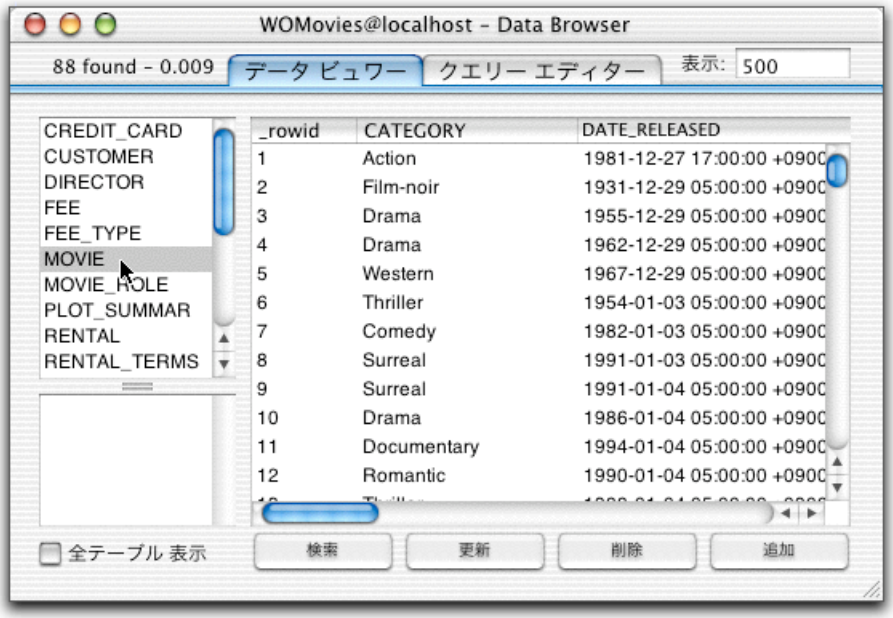
Objective-C のクラスを作るメリットは、比較的示しやすいかもしれない。たとえば、キーチェーンの処理をさせたりといった、システムの API 呼び出しができるからだ。一方、Java のメリットとしては、やはり純正 Java ライブラリの利用ということになるが、格好のサンプルとして、JDBC を使って、データベースアクセスを行う例を示したい。Java には JDBC（Java DataBase Connection）として、SQL データベースへのアクセスを行う機能が用意されている。サーバサイドが Java にシフトしていることもあって、JDBC に対応したデータベースが一般的である。最近では、Microsoft の SQL Server までも JBDC ドライバをまともに作るということがニュースになったほどだ。JDBC ドライバを用意すれば、SQL ステートメントの違いなどはデータベースエンジンごとにあるものの、接続や SQL 実行、そこからの値の取り出しのプログラムはほとんど共通のものが使える。つまり、データベースエンジンに依存しないデータベースアクセスが JDBC によって可能になっているというわけだ。

いろいろなデータベースが結果的に使えるのであるが、今回のサンプルでは、OpenBase SQL を使用しよう。OpenBase SQL は、WebObjects に付属しているのでなじみがあるかもしれない。もし、WebObjects を持っていなくても、OpenBase の評価版は無償でダウンロードして利用できるので、Mac OS X にインストールして使ってみよう。GUI の管理ツールがあるなど使い勝手もよく、日本語にも対応している。今回は、OpenBase がインストールされた状態であるとして、以下の例を示したい。OpenBase には、「WOMovies」という映画のデータベースのサンプルがある。そのデータベースを起動した状態で、以下の作業を行うものとする。

◇OpenBase International
http://www.openbase.com/

この WOMovies というサンプルデータベースには、MOVIE というテーブルがあって、映画の一覧表が用意されている。以下は、OpenBase Manager で見たそのテーブルである。

サンプルデータベースの WOMovies にある MOVIE テーブル

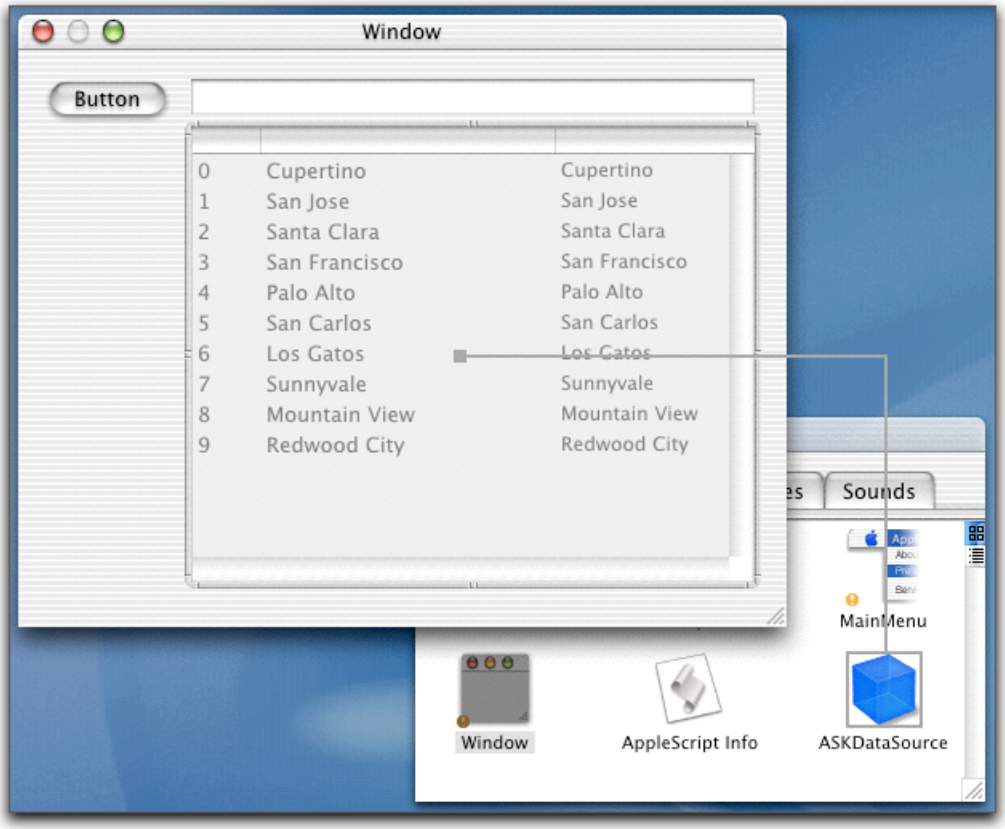


それでは、このデータベースから値を取り出して、テーブル（NSTableView）のコントロールに取り出し結果を表示するといったプログラムを作成してみよう。まず、ユーザインタフェースの部分は、もちろん、Interface Builder で作成する。ここでは、プロジェクトとして AppleScript アプリケーションとして作ったものを利用するが、そこ

には最初から MainMenu.nib という nib ファイルが用意されているので、それをダブルクリックし、最初から用意されているウインドウにユーザインタフェースを作り込むことにしよう。以下の図のようなものを作成したが、要は、ボタンが 1 つあり、このボタンは、Info パレットで、clicked イベントを発生するようにしておき、そのプログラムをデフォルトのプログラムファイルである Application.applescript に設定してある。また、テキストフィールド、テーブルをそれぞれ配置した。同じ種類のコンポーネントがないので、今回は、AppleScript Name の設定を行わない方法でプログラムを組んでみよう。

それから、ツールパレットの AppleScript のカテゴリから、青い四角いボックスを配置しておき、ASKDataSource というインスタンスを作っておく。そして、NSTextView の DataSource とつなげておく。これは前回説明した通りのデータソースの確保である。

作成したウインドウとデータソースへの接続



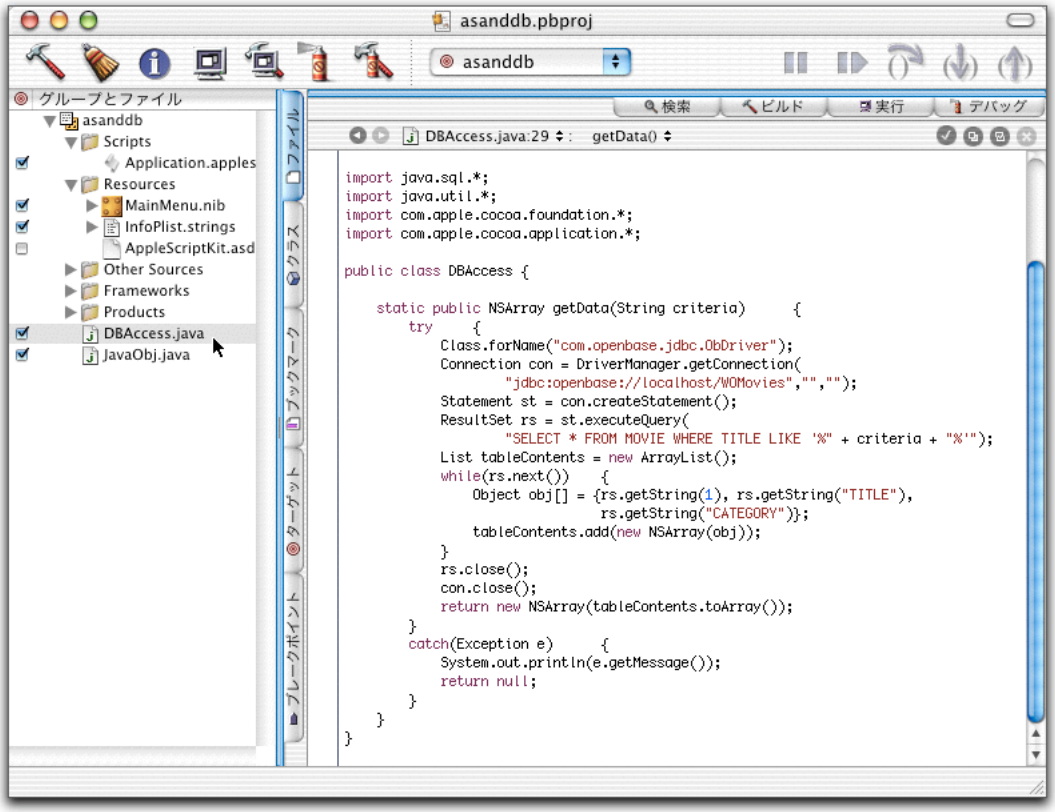
なお、NSTableView は 3 列を表示できるようにしておき、列幅は適当にドラッグして設定しておこう。

これでユーザインタフェースは作成できた。ちなみに、テキストフィールドに検索条件を記入すると、その文字を含む題名の映画をテーブルの一覧するという動作をさせ

たい。

続いて、データベースアクセスをおこなうクラスを Java で作る。「ファイル」メニューの「新規ファイル」（Command+N）を選択して、Java class を選び、ここでは DBAccess というクラスを定義した。プログラムはテキストでも示そう。

Java の DBAccess クラスを定義する




```
//  DBAccess.java

import java.sql.*;
import java.util.*;
import com.apple.cocoa.foundation.*;
import com.apple.cocoa.application.*;

public class DBAccess {

    static public NSArray getData(String criteria)    {
        try    {
```

```
                Class.forName("com.openbase.jdbc.ObDriver");
                Connection con = DriverManager.getConnection(
                        "jdbc:openbase://localhost/WOMovies","","");
                Statement st = con.createStatement();
                ResultSet rs = st.executeQuery(
                        "SELECT * FROM MOVIE WHERE TITLE LIKE '%"
                            + criteria + "%'");
                List tableContents = new ArrayList();
                while(rs.next())        {
                    Object obj[] = {rs.getString(1), rs.getString("TITLE"),
                                    rs.getString("CATEGORY")};
                    tableContents.add(new NSArray(obj));
                }
                rs.close();
                con.close();
                return new NSArray(tableContents.toArray());
            }
            catch(Exception e)        {
                System.out.println(e.getMessage());
                return null;
            }
        }
    }
```

プログラムの細かな点は JDBC に立ち入ることなので、ここでは概要を説明したい。getData メソッドは 1 つの引数を取るが、その引数を検索条件として、MOVIE テーブルからレコードを取り出す。SELECT ステートメントとして、LIKE 演算子やワイルドカードを使って SQL コマンドを作っているが、つまりは、TITLE カラムのデータに、引数に指定された文字列が含むものであれば、すべて抽出するとうい SELECT 文となっている。

データベースの接続には、JDBC ドライバのクラスのフルパス記述や、あるは接続文字列の指定が必要だが、これは、OpenBase のドキュメントを参考に、プログラムにあるように決めた。なお、これは、接続先として localhost を指定している。つまり、実

行するマシンで稼働している OpenBase を対象にデータベースアクセスするわけだ。このあたりのプログラムは、JDBC の定番的な流れである。

そして、取り出した結果を、2 次元配列として戻したいのだが、NSArray を要素に持つ NSArray を作れば、それは 2 次元配列として AppleScript 側に伝わる。1 レコード分はまとめて NSArray を作ってしまえばいいが、レコードを順に調べていき、それらを逐次記憶するために ArrayList クラスを使っている。このクラスを使って最終的に toArray メソッドで配列を得て、そこから、NSArray を得ているのである。

続いて、AppleScript 側のプログラムを紹介しよう。ボタンをクリックすると、以下のようなハンドルが呼び出される。

```
on clicked theObject
    set crit to text field 1 of window of theObject
    set dbData to call method "getData:" of class "DBAccess" with ツ
        parameter string value of crit

    set theTable to table view 1 of scroll view 1 of window of theObject
    set ds to data source 1 of theTable
    if (count data columns of ds) is 0 then
        make new data column at the end of data columns of ds
        make new data column at the end of data columns of ds
        make new data column at the end of data columns of ds
    end if
    delete data rows of ds
    repeat with aRow in dbData
        set newRow to make new data row at the end of data rows of ds
        set contents of data cell 1 of newRow to item 1 of aRow
        set contents of data cell 2 of newRow to item 2 of aRow
        set contents of data cell 3 of newRow to item 3 of aRow
    end repeat
end clicked
```

まず、最初に、Java の DBAccess クラスの getData メソッドを呼び出している。テキストフィールドの値を、メソッドの引数として指定しているわけだ。これで、call method

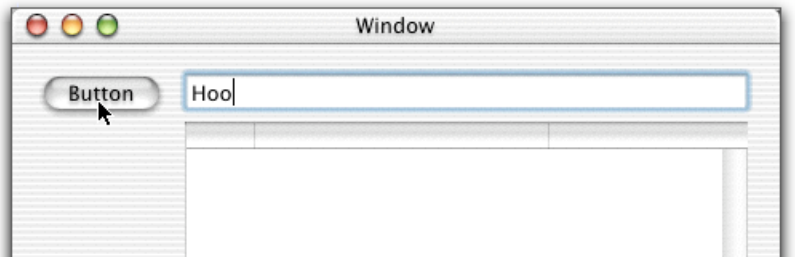
ステートメントの戻り値は、データベースから取り出した結果が含まれた 2 次元の配列となる。

あとは、前回のプログラムとかなり近い。データソースにカラムが用意されているかどうかを調べて、なければ作成する。そして、まず、行を全部削除してから、各行に、各カラムのデータを設定している。

実際の動きを見るのが早いかもしれない。

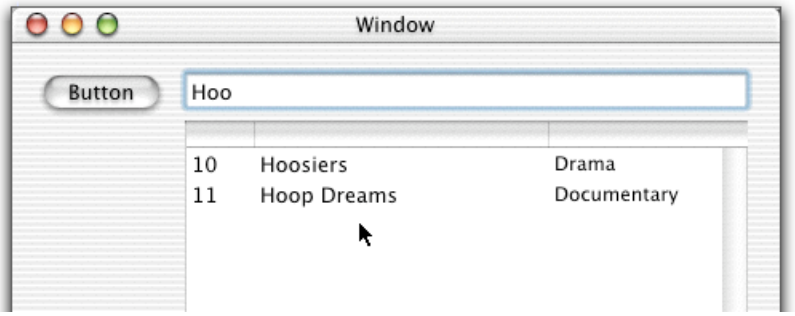
まず、アプリケーションを起動すれば、ウインドウが表示される。テキストフィールドに、ここでは条件として「Hoo」を指定してボタンをクリックしてみた。

検索条件を入れてボタンをクリック

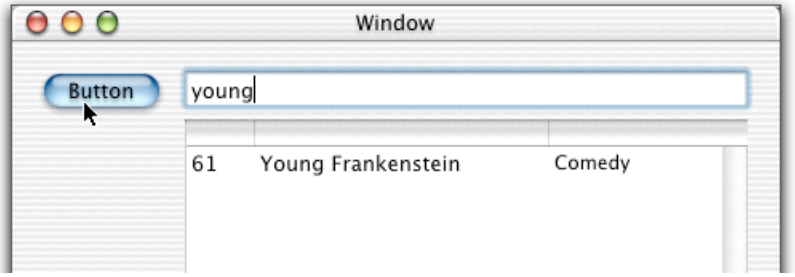


すると、テーブルの部分に、条件に合ったレコードが取り出さされる。繰り返し、検索を行ってもかまわないようにしてある。

検索結果が表示された



別の検索を行った結果



ここで、検索条件に何も指定しないで、ボタンをクリックしてみよう。その場合、すべてのレコードが取り出されて、100 行以上に渡ってテーブルへの書き込みが行われる。データベースのアクセスはローカルだとすぐに終わるのだが、テーブルのデータソースの行の追加といったしょりに時間がかかるあたりが分かるだろう。

このように、データベースアクセスのためのクラスを Java で用意しておくことができるが、もちろん、汎用的なフェッチメソッドを作るのもいいし、特定の用途のものをつくるのもいいだろう。クラス変数を使えば、コネクションをキープしたままいくつかのメソッドで処理することもできるし、取り出したデータをクラス内にキャッシュするということもできるだろう。こうした Java でのデータベースアクセスの形態は、まさに 3 階層システムになっているというわけだ。つまり、ユーザインタフェースを AppleScript で組み、ビジネスロジックを Java で組むというわけである。
call method はある意味ではアドホック的な手段ではあるが、AppleScript のソースがベタな感じになるという気はするものの、手軽にこうして AppleScript 以外の手段とリンクできるようになっている点は発展性を強く感じるところだ。もちろん、フレームワークとしてしっかりしたものを作り、AppleScript の用語集などを備えた、AppleScript でのクラスやコマンドとして使えるものを用意するのが最終的な手段ではあるだろうけど、必要な機能の Objective-C や Java で作られたソースを自分のプロジェクトに組み込んで、call method で使うというのが、Mac OS X 時代のお手軽 OSAX ではないかと思われる。
（この項、以上）