Cocoa はやっぱり!
ボリュームリストの巻

2002.7.9 (1st Edition)

■ 今回のテーマ

今回のテーマは、**ボリュームリスト**です。マウントされているボリュームのリストの取得や変化したときのリストの更新などについて説明します。ファイルブラウザのようなものを作る際には必須のテーマです。また、後半では、ボリュームリストをメニューに表示するための、ダイナミックなメニューの生成方法についても説明します。

◎ 推奨環境

この解説は、以下の環境を前提にしていますので、ご確認ください。これを書いている段階では、Developer Tools の最新版がベータ版なのですが、これをそのまま使用しています。

- Mac OS X 10.1.5
- Project Builder Version 2.0 (April 2002 Developer Tools Beta)
- Interface Builder 2.2.1 (v263.2)

◎ 更新履歴

- 2002.07.09 : 新規作成

■ローカルボリュームリストの取得

まず、ローカルボリュームのリストを取得する方法から説明していきます。ローカルボリュームのリストの取得は、まさにそのためのメソッドが NSWorkspace にあります。**mountedLocalVolumePaths** メソッドです。

```
NSArray *localVols = [ [ NSWorkspace sharedWorkspace ]
                          mountedLocalVolumePaths ];
```

NSWorkspace : マウントしている全ローカルボリュームのマウントパスを取得

書式
```
- (NSArray *) mountedLocalVolumePaths
```
出力
返り値 : マウントしている全ローカルボリュームのマウントパス

配列で値が返ってきますが、各ボリュームがマウントされているパスが文字列で入っています。例えば以下のようになっています。

```
0 : "/"
1 : "/Volumes/ExternalHDD"
2 : "/Volumes/AudioCD"
```

"/"が起動ボリュームで、それ以外が起動ボリューム以外のボリュームです。Mac OS X は、UNIX ベースですので起動ボリュームの"/Volumes"の下に、起動ボリューム以外のボリュームのディレクトリツリーがぶら下がります。しかしながら、ユーザーが目にするファイルパスは、Mac OS 9 以前のスタイルを引き継いでいますので、以下のようになります (デリミタにスラッシュが使われる場合もありますが...)。

```
"ボリューム名:フォルダー名:...:ファイル名"
```

そのため、画面に表示する際には「 生のファイルパス 」を使わずに「 **表示用の名前** 」に変換する必要があります。これには、NSFileManager の **displayNameAtPath :** メソッドを使います。ファイルパスを渡すだけです。

```
NSString *sName = [ [ NSFileManager defaultManager ]
                       displayNameAtPath : [ localVols objectAtIndex : i ] ];
```

NSFileManager : 実際のパスから表示用の名前を取得

書式
```
- (NSString *) displayNameAtPath : (NSString *) fullPath
```
入力
```
fullPath : 実際のファイルのフルパス
```
出力
```
返り値     : 表示用の名前
```

これで、以下のような文字列が取得できます。

```
0 : "BootDrive"
1 : "ExternalHDD"
2 : "AudioCD"
```

一般のファイルに displayNameAtPath : を使用するのは、拡張子を表示するかどうかの設定を反映したファイル名を得るためという理由もありましたが、ボリュームの場合はマウントパスを隠すというのが理由の 1 つと言えます。また、Mac OS は、**同じボリューム名を持つ異なるボリュームを同時にマウントできる**という、ちょっと特殊な仕様になっているのも理由の 1 つです。

例えば、「 SameName 」というボリューム名の 2 つのボリュームを同時にマウントした場合、mountedLocalVolumePaths の返り値は以下のようになります。

```
0 : "/"
1 : "/Volumes/SameName"
2 : "/Volumes/SameName 1"
```

このように、数字が付いて内部的には見分けられるようになっています。ただし、Finder などの上では、2 つのボリュームは同じ名前で表示されます。displayNameAtPath : を必ず使った結果を使用する必要があります。

■ボリュームアイコンの取得

ボリュームのリストを表示する際には、ドライブのアイコンの画像が必要になることがあります。これも、NSWorkspace にメソッドが用意されています。**iconForFile :** メソッドがそれです。

```
NSImage *imgIcon = [ [ NSWorkspace sharedWorkspace  ] iconForFile : @"/" ];
```

この 1 行で、起動ボリュームのアイコンが得られます。iconForFile : メソッドはボリューム以外のファイルやフォルダーのアイコンの取得にも使用できます。

NSWorkspace : 指定パスのファイル／フォルダー／ボリューム等のアイコンを取得

書式
```
- (NSImage *) iconForFile : (NSString *) fullPath
```
入力
fullPath : 取得したいファイル等のフルパス
出力
返り値 : 指定パスのファイル等のアイコンのイメージ。
解像度別の複数のイメージが格納されていることが多いです。

■ ボリュームリストの更新

CD-ROM や MO のようなリムーバブルなメディアの挿入やイジェクトが起きたときには、ボリュームリストを更新する必要があります。そこで、Cocoa のフレームワークには、**ボリュームリストに変化があった場合に通知を受ける仕組み**が用意されています。

● 通知依頼を行う

通知を受けるには、**通知センター（ Notification Center ）に通知依頼を行っておく**必要があります。以下のように行います。

マウント／アンマウントの通知を受けるための依頼

```
[ [ [ NSWorkspace sharedWorkspace ] notificationCenter ]
  addObserver : self                                  // 2. このインスタンスの
     selector : @selector( didMount : )               // 3. didMount : をコール
         name : NSWorkspaceDidMountNotification       // 1. マウントされたら
       object : nil ];                                // 4. 誰が発した通知でもOK

[ [ [ NSWorkspace sharedWorkspace ] notificationCenter ]
  addObserver : self                                  // 6. このインスタンスの
     selector : @selector( didUnmount : )             // 7. didUnmount : をコール
         name : NSWorkspaceDidUnmountNotification     // 5. アンマウントされたら
       object : nil ];                                // 8. 誰が発した通知でもOK
```

通知センターのインスタンスを得るには、NSWorkspace の **notificationCenter** メソッドを使います。この通知センターに対して、**addObserver : selector : name : object :** メソッドで通知依頼が出来ます。

マウントやアンマウントを検知したら、検知したインスタンスは通知センターに報告を行います (このインスタンスは今回の場合はシステム側にあります) 。この場合は、「 CD-ROM ドライブに AudioCD という名前のメディアが挿入されてマウントされました 」というような報告がなされます。この時点で、通知センターに「 ボリュームのマウントの通知を受けたい 」という依頼をしていた**全ての**インスタンスに対して、その情報を通知します。

addObserver : selector : name : object : には、4 つのパラメータがありますが、name のパラメータに通知の種類 (名前) を指定します。「 **NSWorkspaceDidMountNotification** 」が**マウントの通知**で、「 **NSWorkspaceDidUnmountNotification** 」が**アンマウントの通知**です。通知というのは、「 **指定されたインスタンスの指定されたメソッドを呼び出す** 」ことによって行われますので、呼び出し先のインスタンスとメソッドを知らせておく必要があります。インスタンスの指定は **addObserver** のパラメータで、メソッドの指定が **selector** のパラメータになります。「 Observer 」という言葉が使われていますが、通知を受けるインスタンスのことを「 オブザーバ (Observer) 」といいます。

残りの object のパラメータは、この通知の発信者を絞り込むときに使います。**nil にしておくと、誰が発したものに関係なく通知を受ける**ことが出来ます。例えば、「 ファイルをバックグラウンドで読み込ませておいて、読み込み完了したら通知する 」というような通知もあります。この場合は、同時に 2 つのファイルを読み込ませることもできますが、どちらか片方だけの通知だけを受けたいということもあるでしょう。そういう場合は、通知の発信者のインスタンスを取得しておいて、それを object のパラメータに指定しておけばよいわけです。

NSNotificationCenter : オブザーバー登録

書式

```
- (void) addObserver : (id        ) anObserver
             selector : (SEL       ) aSelector
                 name : (NSString *) notificationName
               object : (id        ) anObject
```

入力

```
addObserver : 通知先のインスタンス。
selector    : 通知先のインスタンスの実行するメソッド。
name        : 通知して欲しい通知の種類。
object      : 通知元のインスタンス指定。nilだと全てのインスタンスからの通知を受ける。
```

● 通知を受けるメソッド

上のコードでは、通知先を self にしましたので、上のコードを含むクラスには、didMount : メソッドと didUnmount : メソッドが必要になります。以下のコードで、マウントされたりアンマウントされたボリュームのパスを表示できます。

マウント／アンマウントの通知を受けるメソッド

```
- (void) didMount : (NSNotification *) aNote {
  NSString *sNewDrive = [ [ aNote userInfo ] objectForKey : @"NSDevicePath" ];
  NSLog( @"didMount %@", sNewDrive );
}

- (void) didUnmount : (NSNotification *) aNote {
  NSString  *sRemDrive = [ [ aNote userInfo ] objectForKey : @"NSDevicePath" ];
  NSLog( @"didUnmount %@", sRemDrive );
}
```

通知を受けるメソッドは、NSNotification のパラメータを 1 つ持つ必要があります。このパラメータに通

知情報が詰まっています。**userInfo** メソッドで、この通知独自の情報が辞書形式（NSDictionary）で取り出せます。"**NSDevicePath**" というキーを使って、objectForKey : で値を取り出すとボリュームのマウントパスが得られます。

■ ネットワークボリュームリストの取得

リムーバブルなボリュームだけを得る mountedRemovableMedia というメソッドもありますが、ネットワーク経由で接続しているドライブを得るメソッドは、Mac OS X 10.1.5 の時点では存在しません。

ただ、ネットワークボリュームでも、**マウントとアンマウントの通知はローカルと同様にやってきます**。さらに、ネットワークボリュームについては、**通知依頼を出したときに、その時点でマウントされている全ネットワークボリュームのマウント通知が届きます**。ですので、その通知結果を保持しておけば、ネットワークボリュームのリストを得ることが出来ます。

■ ボリューム情報の取得

ボリュームのリストを得ることが出来ましたが、そのボリュームの情報が必要になることがあるでしょう。ここでは、そのいくつかの方法を紹介します。

NSWorkspace の **getFileSystemInfoForPath : isRemovable : isWritable : isUnmountable : description : type :** では、そのボリュームが**リムーバブル**か、**書き込み可能**か、**アンマウント可能**か、**説明**、**タイプ情報**についての情報が得られます。

NSWorkspace : ボリュームの各種情報を取得

書式

```
- (BOOL) getFileSystemInfoForPath : (NSString  *) fullPath
                       isRemovable : (BOOL      *) removableFlag
                        isWritable : (BOOL      *) writableFlag
                     isUnmountable : (BOOL      *) unmountableFlag
                       description : (NSString **) description
                              type : (NSString **) fileSystemType
```

入力

```
fullPath        : 情報取得したいボリュームのフルパス
removableFlag   : リムーバブルかどうかのフラグの格納先 ( → 出力 )
writableFlag    : 書き込み可能かどうかのフラグの格納先 ( → 出力 )
unmountableFlag : アンマウント可能かどうかのフラグの格納先 ( → 出力 )
description     : その他の情報の格納先 ( → 出力 )
fileSystemType  : ファイルシステムの種別情報の格納先 ( → 出力 )
```

出力

```
返り値          : fullPathがファイルシステムのマウントポイントならYES。
```

NSFileManager の **fileSystemAttributesAtPath :** メソッドでは、**ボリュームの容量**や**空き容量**などを知ることが出来ます。以下のコードでは、起動ボリュームの容量と空き容量が得られます。

```
NSDictionary *dicVol = [ [ NSFileManager defaultManager ]
                           fileSystemAttributesAtPath : @"/" ];
long volSize  = [ [ dicVol objectForKey : NSFileSystemSize     ] longValue ];
long freeSize = [ [ dicVol objectForKey : NSFileSystemFreeSize ] longValue ];
```

辞書から総容量を取り出すキーが **NSFileSystemSize** で、空き容量を取り出すキーが **NSFileSystemFreeSize** です。

NSFileManager : ボリュームの各種情報を取得

書式

```
- (NSDictionary *) fileSystemAttributesAtPath : (NSString *) fullPath
```

入力

```
fullPath : 情報取得したいボリュームのフルパス
```

出力

```
返り値     : 各種情報が詰まった辞書。キーと意味は以下のとおり。
             NSFileSystemSize     : ボリュームの総容量 (byte)  ( NSNumber )
             NSFileSystemFreeSize : ボリュームの空き容量(byte) ( NSNumber )
```

■ ボリュームメニューの作成

アプリケーションのメニューに「 ボリュームメニュー 」というものをつけたいとします。現在マウントされているボリュームがメニュー項目として表示されて、選択すると Finder で表示されるというようなメニューです。ボリュームリストは変化するものですので、いつものように nib ファイルを編集するというスタイルではなく、プログラム的に生成する必要があります。

● 動的なメニューの生成

生成の処理の流れとしては、以下の大きく 2 つのステップが必要です。

1. メニューを生成する
2. 生成したメニューをメニューバーに登録する

まずは、メニューを生成するステップについて。1 つの NSMenu インスタンスが親になり、これに複数個の NSMenuItem インスタンスが子供としてぶら下がる構造になっています。今回は、最も単純な以下のようなメニューを作ります。

サンプルのメニュー

メニューを生成

```
NSMenu     *menuVol;
NSMenuItem *menuItem;

// NSMenuを生成

menuVol =  [ [ [ NSMenu alloc ] init ] autorelease ]; // 生成
[ menuVol setTitle : @"Volumes" ];                    // タイトル設定

// NSMenuItemを生成

menuItem = [ [ [ NSMenuItem alloc ] init ] autorelease ]; // 生成
[ menuItem setTitle  : @"abc" ];                          // タイトル設定
[ menuItem setAction : @selector( showVolume : ) ];       // アクション設定
[ menuItem setTarget : self                      ];       // ターゲット設定

// NSMenuItemをNSMenuに登録
[ menuVol addItem : menuItem ];
```

メニュー本体は、NSMenu に対して、alloc / init / autorelease を行って生成します。setTitle : メソッドでメニューバー上に表示されるタイトルを設定します。

メニュー項目も、同様に NSMenuItem に対して、alloc / init / autorelease を行って生成します。

setTitle : でタイトル、**setAction :** と **setTarget :** でメニューが選択されたときに実行するターゲットのインスタンスとアクションを設定します。これは、いつもは Interface Builder 上で行っているアクションの接続に相当します。自分自身の showVolume : というメソッドが呼ばれるようになります。これを、**addItem :** でメニューの末尾に追加していきます。

今回は、メニュー項目は 1 つですが、NSMenuItem の生成と、メニューへの登録を繰り返すと複数個のメニュー項目を登録できます。

これでメニューが出来ました。この状態では、まだメニューバーに登録していませんので画面には表示されていません。2 つ目のステップに進んで、メニューバーに登録します。

メニューバーへ登録

```
menuItem = [ [ [ NSMenuItem alloc ] init ] autorelease ]; // 生成
[ [ NSApp mainMenu ] addItem : menuItem ];                // メニューバーへ登録

[ menuItem setSubmenu : menuVol ]; // ぶら下げる
```

メニューバーに登録するには、まず、メニューバー上の登録したい場所にメニュー項目 (NSMenuItem) を **addItem :** メソッドで追加します (メニューバー全体は、NSMenu でできており、メニューバー上の「 アップルマーク 」や「 ファイル 」は NSMenuItem でできています)。追加することで、先程作ったメニューをぶら下げる場所を確保できます。そのメニュー項目に **setSubmenu :** メソッドでぶら下げることが出来ます。これでメニューバーに登録され、画面に表示されます。

addItem : メソッドでは、メニューバーの一番右に追加されますが、**insertItem : atIndex :** メソッドを使うと、好きな場所に追加することも出来ます。

```
[ [ NSApp mainMenu ] insertItem : menuItem atIndex : 2 ];
```

アップルメニューの右側が 0、ファイルメニューの右側が 1 というように、隙間に番号が振られています。

● VolumeMenu クラスの作成

これまでのノウハウを使って、VolumeMenu というクラスを作ってみましょう。簡単にメニューバー上にボリュームメニューを作ることが出来るクラスを目指します。

“VolumeMenu.h”

```
#import <Cocoa/Cocoa.h>

@interface VolumeMenu : NSObject {
    NSMenu          *menuVols;    // メニューを保持
    NSMutableArray *networkVols; // ネットワークボリュームを保持
}
- (void) addToMenubarAt : (int) aiLoc;  // メニューバーに登録
@end
```

変数としては、生成したメニューとネットワークボリュームのリストを持ちます。先程説明しましたが、ネットワークボリュームは、マウント通知とアンマウント通知を使ってこのリストを更新しておきます。addToMenubarAt : というのが、メニューバーの指定位置にボリュームメニューを登録するメソッドです。外部から呼ぶメソッドはこれだけで、ボリュームリストの更新や、メニューが選ばれたときの処理も自動的にこのクラスによって実現されるようにします。

まずは、この addToMenubarAt : から見ていきます。

“VolumeMenu.m” > addToMenubarAt :

```
#import "VolumeMenu.h"

@implementation VolumeMenu

- (void) addToMenubarAt : (int) aiLoc {

  networkVols = [ [ NSMutableArray array ] retain ];
  menuVols = [ [ [ NSMenu alloc ] init ] autorelease ]; // メニューを生成
  [ menuVols setTitle : @"Volumes" ];                   // タイトル設定

  { // メニューバーに登録
    NSMenuItem *menuItem = [ [ [ NSMenuItem alloc ] init ] autorelease ];
    [ [ NSApp mainMenu ] insertItem : menuItem
                            atIndex : aiLoc ];
    [ menuItem setSubmenu : menuVols ];
    [ self updateMenu ];
  }

  [ self entryNotificationCenter ];    // オブザーバ登録

}
```

menuVols というのが、ボリュームメニュー本体ですので、alloc / init / autorelease で生成します。autorelease しても平気なのは、メニューバーに登録することで retain されるためです。もし、途中でメニューバーから切り離すようなことがある場合は、autorelease はしない方がよいでしょう。

そして、その後の処理でこのメニューをメニューバーに登録しています。メニュー項目が登録されていませんが、それは、[self updateMenu] というメソッドの中で行っています。後程説明します。そして、オブザーバ登録も行っています。オブザーバ登録に関しては、先程と全く同じです。

通知は、この VolumeMenu のインスタンス自身が受けるため、メニューの更新も外部に面倒をかけずに自動的に行われることになります。

“VolumeMenu.m” > entryNotificationCenter

```
//// マウントとアンマウントのオブザーバ登録

- (void) entryNotificationCenter {

  [ [ [ NSWorkspace sharedWorkspace ] notificationCenter ]
                addObserver : self
                   selector : @selector( didUnmount : )
                       name : NSWorkspaceDidUnmountNotification
                     object : nil ];

  [ [ [ NSWorkspace sharedWorkspace ] notificationCenter ]
                addObserver : self
                   selector : @selector( didMount : )
                       name : NSWorkspaceDidMountNotification
                     object : nil ];

}
```

そして、メニュー項目を最新の状態にする updateMenu メソッドです。

“VolumeMenu.m” > updateMenu

```
///// メニューを更新する

- (void) updateMenu {

  NSArray    *vols;     // ドライブリスト
  NSMenuItem *menuItem; // メニューアイテム
  NSString   *sPath;    // ボリュームのパス
  NSImage    *img;      // ボリュームのアイコン
  int i;

  vols = [ self allVolumes ];  // ドライブリストを取得
  [ self removeAllMenuItems ]; // 全メニュー項目を削除

  for ( i = 0 ; i < [ vols count ] ; i++ ) {

    menuItem = [ [ [ NSMenuItem alloc ] init ] autorelease ];
    sPath = [ vols objectAtIndex : i ];

    /// アイコン
    img = [ [ NSWorkspace sharedWorkspace  ] iconForFile : sPath ];
    [ img setSize : NSMakeSize( 32, 32 ) ];
    [ menuItem setImage : img ];

    [ menuItem setTitle : [ [ NSFileManager defaultManager ]
                                          displayNameAtPath : sPath ] ];

    /// アクション関係
    [ menuItem setAction : @selector( showVolume : ) ]; // アクション設定
    [ menuItem setTarget : self                      ]; // ターゲット設定
    [ menuItem setRepresentedObject : sPath          ]; // 付随データ設定

    [ menuVols addItem : menuItem ]; // メニューへ登録

  }

}
```

ほとんどは説明済みのものですので、新しく出てきたものを中心に説明をしていきます。最初の方の、[self allVolumes] というのは、ローカルとネットワークのボリュームを合わせた全ボリュームのリストを得るためのメソッドです。[self removeAllMenuItems] というのは、メニュー項目を一旦全て削除するというものです。どちらも、後程出てきます。

全メニュー項目を削除していますので、全ボリュームについてメニュー項目を作って登録していけばよいことになります。1 つ新しいのは、**setRepresentedObject :** メソッドです。これは、メニューに自由になに

かしらのオブジェクトを繋いでおくためのメソッドです。setTag : メソッドや Interface Builder 上でメニュー項目に数値を記憶させておくことが出来ましたが、数値では物足りない場合は、文字列などのオブジェクトを登録しておくことも出来ます。今回は、ボリュームのパスを登録してありますので、メニューが選ばれたときに簡単にパスを取り出すことが出来ます。

“VolumeMenu.m” > allVolumes

```
//// ローカルドライブとネットワークドライブを合わせてソートして返す

- (NSArray *) allVolumes {

   NSMutableArray *allVols;

  allVols = [ [ [ [ NSWorkspace sharedWorkspace ] mountedLocalVolumePaths ]
                      mutableCopy ] autorelease ];  // ローカルボリュームを取得

  [ allVols addObjectsFromArray : networkVols  ];   // ネットワークボリュームを追加
  [ allVols sortUsingSelector   : @selector( compare : ) ];   // パス順にソート

  return( allVols );

}
```

先程出てきました、allVolumes メソッドです。ローカルボリュームを取得して、それにネットワークボリュームのリストを追加して、ソートして返しています。

“VolumeMenu.m” > removeAllMenuItems

```
//// メニューアイテムを全て削除する

- (void) removeAllMenuItems {

  int iNum = [ menuVols numberOfItems ]; // メニュー項目の数を取得
  int i;

  for ( i = 0 ; i < iNum ; i++ )
    [ menuVols removeItemAtIndex : 0 ]; // 先頭のメニュー項目を繰り返し削除

}
```

同じく、先程出てきた removeAllMenuItems です。現在のメニュー項目の数を取得して、先頭のメニュー項目を繰り返し削除しています。

ここまでで、まずはメニューをメニューバーに登録するところまでは出来ました。後は、ネットワークボリュームとマウント／アンマウント通知の処理です。

“VolumeMenu.m” > isLocalVolume : , didMount :

```
//// 指定のパスのドライブがローカルドライブかどうかを判断する

- (BOOL) isLocalVolume : (NSString *) aPath {

  NSArray *localVols = [ [ NSWorkspace sharedWorkspace ]
                              mountedLocalVolumePaths ];

  if ( [ localVols containsObject : aPath ] ) return YES;
  return NO;
}

///// マウント時

- (void) didMount : (NSNotification *) aNote {

  NSString  *sMountVol = [ [ aNote userInfo ] objectForKey : @"NSDevicePath" ];

  if ( ![ self isLocalVolume : sMountVol ] )
    [ networkVols addObject : sMountVol ]; // ネットワークボリューム更新

  [ self updateMenu ]; // メニュー更新
}
```

didMount : メソッドの中から、マウントされたボリュームがローカルボリュームかどうかを判断するメソッド isLovalVolume : を呼んでいます。このメソッドの中では、マウントされたパスがローカルボリュームの中に含まれるかを NSArray の containsObject : で調べているだけです。そして、ローカルボリュームでないときのみ、ネットワークボリュームの配列に登録を行って、updateMenu でメニューを更新します。

“VolumeMenu.m” > didMount :

```
- (void) didUnmount : (NSNotification *) aNote {

  NSString  *sUnmoutVol = [ [ aNote userInfo ] objectForKey : @"NSDevicePath" ];

  [ networkVols removeObject : sUnmoutVol ]; // ネットワークボリューム更新

  [ self updateMenu ]; // メニュー更新
}
```

こちらは、アンマント時の処理です。無条件に removeObject : で配列からアンマウントされたボリュームのパスを配列から削除しています。その後、メニューを更新しています。これで、ネットワークボリュームもメニューの更新もできるようになりました。残るは、メニューが選択されたときのアクションメソッドのみです。

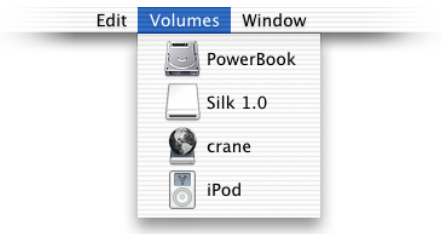
“VolumeMenu.m” > showVolume :

```
- (IBAction) showVolume : (id) sender {
```

```
  NSString * sVolume = [ sender representedObject ]; // メニュー項目からパスを取得

  [ [ NSWorkspace sharedWorkspace ]
                    selectFile : sVolume
      inFileViewerRootedAtPath : sVolume ]; // Finderで指定パスを表示

}

@end
```

setupMenu メソッドでメニュー項目を作成しているところで、setRepresentedObject : でボリュームのパスをメニュー項目にセットしていました。この showVolume : の sender パラメータには選択されたメニュー項目が渡ってきますので、このインスタンスに対して representedObject メソッドを実行することで、セットしておいたボリュームのパスが得られます。

これで完成です。以下のようなメニューが表示されるはずです。



以上