Cocoa はやっぱり!
時計とカレンダーの巻
2002.7.30 (1st Edition)

■ 今回のテーマ

今回のテーマは、**日時の扱い**です。日時を扱うためのクラスの説明と、それを使ってカレンダーを作ります。その中で、修飾属性付きの文字列にも少し触れます。

◎ 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

- 2002.07.30 : 新規作成

■ 日時を扱う 2 つのクラス NSDate, NSCalendarDate

Cocoa には、日時を扱うクラスがいくつかありますが、代表的なものが **NSDate** と **NSCalendarDate** です。NSObject を直接継承しているのが NSDate で、さらにそれを継承しているのが NSCalendarDate になります。

この 2 つの違いですが、NSDate は「 **日時を、ある基準時からの経過秒数として扱う** 」クラスです。利便性のために秒以外のインターフェイスも持ちますが、基本的には「 **秒を扱うクラス** 」と思っていただければよいと思います。「 秒 」といっても浮動小数値なので、秒よりも細かい時間を扱うことも可能です。NSCalendarDate は「 **日時を、年／月／日／時／分／秒で扱うもので、曜日や閏年、タイムゾーン、夏時間も考慮した** 」クラスです。NSDate は、時間を科学的な秒の積み重ねとして扱い、人間のカレンダーという概念をかぶせたのが NSCalendar ということになります。

■ 秒を扱うクラス NSDate

● 特定の日時を生成する

では、具体的に NSDate の中身を見ていきましょう。最も基本的、かつ、よく使うと思われるのが、現在の日時を得るためのメソッドです。これはクラスメソッドの **date** です。

```
NSDate *d = [ NSDate date ];  // 現在の日時を得る
```

NSDate : 現在の日時を表すインスタンスを生成する

書式
+ (id) date
出力
返り値 : 現在の日時を表すインスタンス

インスタンスメソッドでも現在の日時を返すものがあります。**init** メソッドです。以下のコードで date メソッドと同じ結果を得ることが出来ます。

```
NSDate *d = [ [ [ NSDate alloc ] init ] autorelease ];  // 現在の日時を得る
```

現在から 3 分後の日時を得たいときは、クラスメソッドでは「 **dateWithTimeIntervalSinceNow :** 」、インスタンスメソッドでは「 **initWithTimeIntervalSinceNow :** 」を使います。パラメータは、NSTimeInterval なので秒を浮動小数値で指定します。負の値だと過去の日時になります。

```
NSDate* d = [ NSDate dateWithTimeIntervalSinceNow : 3*60 ];  // 3分後の日時
```

NSDate : 現在からのずれを指定してインスタンスを生成

書式
```
+ (NSDate *) dateWithTimeIntervalSinceNow : (NSTimeInterval) sec
```
入力
```
sec    : 現在からのずれを秒で指定。正の数なら未来、負の数なら過去。
```
出力
```
返り値 : 現在からの指定のずれを計算した日時のインスタンス
```

好きな日時を得たいときには、クラスメソッドでは「 **dateWithString :** 」、インスタンスメソッドでは「 **initWithString :** 」を使います。以下のような書式に従った文字列を渡します。最後の「 +0900 」は世界標準時からの時差です。日本標準時は、世界標準時より「 09 時 00 分 」進んでいるので「 +0900 」になります。最初の符号が世界標準時より進んでいるかを符号で書いて (一般に東経なら+、西経なら-)、上位 2 桁に時間、下位 2 桁に分を書きます。

```
NSDate* d = [ NSDate dateWithString : @"2001-03-24 10:45:32 +0900" ];
```

NSDate : 文字列で指定した日時を表すインスタンスを生成

書式
```
+ (NSDate *) dateWithString : (NSString *) sTime
```
入力
```
sTime : 生成したい日時を文字列で指定。"YYYY-MM-DD hh:mm:ss +HHMM"の形式
      :  YYYY - 年、 MM - 月、 DD - 日、
      :    hh - 時、 mm - 分、 ss - 秒、
      :  +or- - 世界標準時(GMT)より進んでいるなら+、後れているなら-
      :    HH - 時 (GMTからのずれ)、 MM - 分 (GMTからのずれ)
```
出力
```
返り値 : sTimeで指定した日時を表すインスタンス
```

NSDate にはちょっと不思議なメソッドがあります。NSDate を生成するのに、自然言語を使うことができます。といっても、簡単なルールに基づいて文字列から日時に関するものを見つけているようですが。例えば、" yesterday " とか " sunday " というような文字列から NSDate を作ることが出来ます。

NSDate : 自然言語から生成

書式
```
+ dateWithNaturalLanguageString : (NSString *) string
```
入力
```
string : 自然言語による指定。"yesterday"などが使用できる。
```
出力
```
返り値  : 自然減をを解釈した結果のインスタンス。解釈できない場合はnull。
```

「 yesterday 」は「 昨日の正午 」の日時を生成します。さらに、「 yesterday morning 」にすると「 昨日の午前 10:00 」を生成します。「 sunday 」 は「 次の日曜日の正午 」を生成し、「 last sunday 」は「 直前の日曜日の正午 」を生成します。もっと普通に 「 8.13 5:20 」とすると、「 次の 8 月 13 日 5 時 20 分 」になります。

このように、「 sunday 」というように「 いつの日曜日 」かを指定していない場合は、「 次の日曜日 」になります。日の指定に時刻がなければ「 正午 」になります。

```
next xxx    : 次の
last xxx    : 直前の
```

```
morning     : 10:00
noon        : 12:00
lunch       : 12:00
dinner      : 19:00
```

```
tomorrow          : 明日
yesterday         : 昨日
sun, mon..        : 各曜日 ( ex. "next sun" )
sunday, monday..  : 各曜日 ( ex. "next sunday" )
```

● 日時の計算

既存の NSDate のインスタンスの 1 時間後を求めたい場合は、**addTimeInterval :** メソッドを使います。メソッド名のまんまですが、ある NSDate にある秒数を加えた NSDate のインスタンスを生成します。

```
  NSDate *d1 = [ xxxx ];                          // 何かの値が入っている
  NSDate *d2 = [ d1 addTimeInterval : 60*60 ]; // d1の1時間後になる
```

NSDate : 指定の秒数を足したインスタンスを生成

書式
```
  - (id) addTimeInterval : (NSTimeInterval) sec
```
入力
  sec　: 足したい時刻を秒で指定。正の数なら未来、負の数なら過去。
出力
  返り値 : 指定の秒数を足した日時のインスタンス

ここで気づいた方もいらっしゃるかもしれませんが「 **NSDate 自身は一度生成すると内容の変更が出来ません** 」。ですので、既存の NSDate に秒数を加えるというメソッドは存在しなくて、ある秒数を加えたインスタンスを**新たに生成する**というメソッドになっています。

以下のように addTimeInterval : メソッドを使って、もとのインスタンスに代入するような書き方をすると、**古いインスタンスを捨てながら処理を進める**ことになります。もちろん、このコードでは古いインスタンスは自動破棄されますので心配はいりません。alloc したり retain した場合は注意が必要です。

```
  NSDate *d = [ NSDate date ];        // 現在の日時を得る
  d = [ d addTimeInterval : 60*60 ]; // 1時間後になる ( 新しいインスタンスになる )
```

2 つの NSDate がどれだけ離れているかを知るには、**timeIntervalSinceDate :** メソッドを使います。

```
NSDate *d1 = [ NSDate date ];                 // 現在の時刻
NSDate *d2 = [ d1 addTimeInterval : 60*60 ]; // 現在の1時間後

NSTimeInterval ti = [ d1 timeIntervalSinceDate : d2 ]; // d1 - d2 = -3600
```

「 d1 - d2 」の値が返ってきますので、timeIntervalSinceDate : をマイナス符号と置き換えて考えると分かりやすいでしょう。

NSDate : 指定の日時との差を求める

書式
- (NSTimeInterval) timeIntervalSinceDate : (NSDate *) anotherDate
入力
anotherDate : 比較したい日時。
出力
返り値 : 「 このインスタンス - anotherDate 」の秒数。

● 日時の比較

2 つの日時の大小比較は、**compare :** メソッドを使います。NSComparisonResult で値が返ってきます。

```
NSDate *d1 = [ NSDate date ];                 // 現在の時刻
NSDate *d2 = [ d1 addTimeInterval : 60*60 ]; // 現在の1時間後

switch( [ d1 compare : d2 ] ) {
  case NSOrderedAscending  : NSLog( @"d1 < d2" ); break; // これになる
  case NSOrderedSame       : NSLog( @"d1 = d2" ); break;
  case NSOrderedDescending : NSLog( @"d1 > d2" ); break;
}
```

NSDate : 指定の日時との大小比較

書式
- (NSComparisonResult) compare : (NSDate *) anotherDate
入力
anotherDate : 比較したい日時。
出力
返り値 : 「 このインスタンス < anotherDate 」ならNSOrderedAscending。
: 「 このインスタンス = anotherDate 」ならNSOrderedSame。
: 「 このインスタンス > anotherDate 」ならNSOrderedDescending。

同じかどうかだけ比較したい場合は、**isEqualToDate :** メソッドを使います。

```
    if ( [ d1 isEqualToDate : d2 ] ) NSLog( @"d1 = d2" );
```

NSDate : 指定の日時との比較

書式
- (BOOL) isEqualToDate : (NSDate *) anotherDate
入力
anotherDate : 比較したい日時。
出力
返り値 : 同じならYES、違うならNO。

2 つの日時で**古い方を選択する**には「 **earlierDate :** 」、**新しい方を選択する**には「 **laterDate :** 」を使います。

```
  NSDate *d1 = [ NSDate date ];                 // 現在の時刻
  NSDate *d2 = [ d1 addTimeInterval : 60*60 ]; // 現在の1時間後

  NSDate *d3 = [ d1 earlierDate : d2 ]; // 古い方を選択   d3 <- d1
  NSDate *d4 = [ d1 laterDate   : d2 ]; // 新しい方を選択 d4 <- d2
```

NSDate : 指定の日時と比較して古い方を選択

書式
- (NSDate *) earlierDate : (NSDate *) anotherDate
入力
anotherDate : 比較したい日時。
出力
返り値 : 古い方の日時。

NSDate : 指定の日時と比較して新しい方を選択

書式
- (NSDate *) laterDate : (NSDate *) anotherDate
入力
anotherDate : 比較したい日時。
出力
返り値 : 新しい方の日時。

沢山の日時の中から、最も古い日時を求めるようなことがあります。例えば、スケジューラーで次にアラームを鳴らす時刻を求めるような場合です。この場合、最初の比較の基準値として、遠い未来の日時をまず作っておくと思います。逆に遠い過去の日時が必要になることもあります。このためのメソッドも用意されています。**distantFuture** と **distantPast** メソッドです。

```
  NSDate *dFuture = [ NSDate distantFuture ]; // 遠い未来 4001-01-02 00:00:00 GMT
  NSDate *dPasr   = [ NSDate distantPast   ]; // 遠い過去 0001-01-17 00:00:00 GMT
```

Mac OS X 10.1.5 では、上に書いているような日時が返ってきます。

NSDate : 遠い未来の日時を取得

書式

```
- (NSDate *) distantFuture
```

出力

返り値　　: 遠い未来の日時 (4001-01-02 00:00:00 GMT)。

NSDate : 遠い過去の日時を取得

書式

```
- (NSDate *) distantPast
```

出力

返り値　　: 遠い過去の日時 (0001-01-17 00:00:00 GMT)。

■ カレンダー形式で日時を扱う NSCalendarDate

NSCalendarDate は、先程も説明しましたが、時間にカレンダー的概念をかぶせていますので、NSDate にタイムゾーン等の情報が付加されています。NSDate と同様に日時の部分は変更できませんが、タイムゾーンの箇所については変更ができます。

● 指定日時のインスタンス生成

まずは、現在の日時を取得するメソッド **calendarDate** から。使い方は、NSDate の date と同じです。

```
NSCalendarDate *cd = [ NSCalendarDate calendarDate ];
```

NSCalendarDate : 現在の日時を取得

書式
+ (id) calendarDate
出力
返り値 : 現在の日時。

NSDate は、秒を基準にしていますが、NSCalendarDate は、年／月／日／時／分／秒での処理もできます。例えば、**dateWithYear : month : day : hour : minute : second : timeZone :** メソッドを使うと「 年月日時分秒＋タイムゾーン 」の指定でインスタンスを生成できます。タイムゾーンに nil を指定すると、**システム環境設定から取得したタイムゾーンの値**が使用されます。

```
NSCalendarDate *cd = [ NSCalendarDate dateWithYear : 2002
                                        month : 4
                                          day : 20
                                         hour : 1
                                       minute : 0
                                       second : 0
                                     timeZone : nil ];
```

タイムゾーンの扱い方は後程説明します。

NSCalendarDate : 指定日時のインスタンスの取得

書式

```
- (id) dateWithYear : (int         ) year
               month : (unsigned    ) month
                 day : (unsigned    ) day
                hour : (unsigned    ) hour
              minute : (unsigned    ) minute
              second : (unsigned    ) second
            timeZone : (NSTimeZone *) timeZone
```

入力

```
year     : 生成したい日時の年。
month    : 生成したい日時の月。
day      : 生成したい日時の日。
hour     : 生成したい日時の時。
minute   : 生成したい日時の分。
second   : 生成したい日時の秒。
timeZone : 生成したい日時の場所のタイムゾーン。nilならシステム環境設定値に従う。
```

出力

```
返り値    : 指定の日時のインスタンス。
```

NSDate では、例えば、年の値を数値として取り出すためのメソッドもありませんでしたが、NSCalendarDate には、年以外にも様々な取得用メソッドが用意されています。

```
NSCalendarDate *cd = [ NSCalendar calendarDate ];

int iYear   = [ cd yearOfCommonEra ]; // 年 ( 西暦 )
int iMonth  = [ cd monthOfYear     ]; // 月 ( 1 - 12 )

int iDay    = [ cd dayOfMonth      ]; // 日 ( 1 - 31 )
int iDayY   = [ cd dayOfYear       ]; // その年の1月1日からの日数 ( 1 - 366 )
int iDayC   = [ cd dayOfCommonEra  ]; // 西暦紀元からの日数
int iWeek   = [ cd dayOfWeek       ]; // 曜日 ( 0 = Sunday .. 6 = Saturday )

int iHour   = [ cd hourOfDay       ]; // 時 ( 0 - 23 )
int iMinute = [ cd minuteOfHour    ]; // 分 ( 0 - 59 )
int iSecond = [ cd secondOfMinute  ]; // 秒 ( 0 - 59 )
```

NSCalendarDate : 西暦を取得

書式

```
- (int) yearOfCommonEra
```

出力

```
返り値 : 西暦。
```

サンプル

```
int iYear = [ [ NSDate calendarDate ] yearOfCommonEra ]; // 現在の西暦
```

NSCalendarDate : 月を取得

書式
```
- (int) monthOfYear
```
出力
返り値 : 月 (1 - 12)。
サンプル
```
int iMonth = [ [ NSDate calendarDate ] monthOfYear ]; // 現在の月
```

NSCalendarDate : 日を取得

書式
```
- (int) dayOfMonth
```
出力
返り値 : 日 (1- 31)。
サンプル
```
int iDay = [ [ NSDate calendarDate ] dayOfMonth ]; // 現在の日
```

NSCalendarDate : その年の1月1日からの日数を取得

書式
```
- (int) dayOfYear
```
出力
返り値 : その年の1月1日からの日数 (1 - 366)。
サンプル
```
int iDay = [ [ NSDate calendarDate ] dayOfYear]; // 現在の1.1からの日数
```

NSCalendarDate : 西暦紀元からの日数を取得

書式
```
- (int) dayOfCommonEra
```
出力
返り値 : 西暦紀元からの日数 (1 -)。
サンプル
```
int iDay = [ [ NSDate calendarDate ] dayOfCommonEra ]; // 現在の西暦紀元からの日数
```

NSCalendarDate : 曜日を取得

書式
```
- (int) dayOfWeek
```
出力
返り値 : 曜日 (0 - Sunday ... 6 - Saturday)。
サンプル
```
int iWeek = [ [ NSDate calendarDate ] dayOfWeek  ]; // 現在の曜日
```

NSCalendarDate : 時を取得

書式
```
- (int) hourOfDay
```
出力
返り値 : 時 (0 - 23)。
サンプル
```
int iHour = [ [ NSDate calendarDate ] hourOfDay ]; // 現在の時
```

NSCalendarDate : 分を取得

書式
```
- (int) minuteOfHour
```
出力
返り値 : 分 (0 - 59)。
サンプル
```
int iMinute = [ [ NSDate calendarDate ] minuteOfHour ]; // 現在の分
```

NSCalendarDate : 秒を取得

書式
```
- (int) secondOfMinute
```
出力
返り値 : 秒 (0 - 59)。
サンプル
```
int iSecond = [ [ NSDate calendarDate ] secondOfMinute ]; // 現在の秒
```

● 日時の演算

NSCalendarDate の **dateByAddingYears : months : days : hours : minutes : seconds :** メソッドを使うことで、年月日時分秒のどれでも好きなもので日時の加算が行えます。これは、なかなか便利なメソッドです。

NSCalendarDate : 日時の加算

書式

```
- (NSCalendarDate *) dateByAddingYears : (int) year
                                months : (int) month
                                  days : (int) day
                                 hours : (int) hour
                               minutes : (int) minute
                               seconds : (int) second
```

入力

```
year   : 加算したい年数。
month  : 加算したい月数。
day    : 加算したい日数。
hour   : 加算したい時数。
minute : 加算したい分数。
second : 加算したい秒数。
```

出力

```
返り値  : 加算後の日時。
```

備考

・演算時には閏年や夏時間を考慮する。
・演算は、年→月→日→時→分→秒の順に行う。

例えば、カレンダーを表示したい場合など、「 今月は何日まであるか 」というのを計算したいことはよくあります。月によって日数が変りますし、閏年のことも考慮すると自前で計算するのはちょっと面倒なのですが、このメソッドを使うと比較的簡単です。

今月は何日まであるかを計算

```
    NSCalendarDate *now = [ NSCalendarDate calendarDate ]; // 今の日時
    NSCalendarDate *firstDay; // 今月の1日
    NSCalendarDate *lastDay;  // 今月の最後の日

    firstDay = [ NSCalendarDate dateWithYear : [ now yearOfCommonEra ]
                                       month : [ now monthOfYear     ]
                                         day : 1
                                        hour : 0
                                      minute : 0
                                      second : 0
                                    timeZone : nil ]; // 今月の1日を求める

    lastDay = [ firstDay dateByAddingYears :  0
                                    months : +1 // 1ヶ月進める
                                      days : -1 // 1日戻す
                                     hours :  0
                                   minutes :  0
                                   seconds :  0 ];

    NSLog( @"%d (%@)", [ lastDay dayOfMonth ], lastDay );
```

calendarDate メソッドで今の日時を得ます。そして、dateWithYear : month : day : hour : minute : second : timeZone : を使って、今月の 1 日のインスタンスを生成します。今月の最後の日を

求めるには、月を 1 つ足して来月の 1 日にしてから、日を 1 つ引きます。

NSCalendarDate は、タイムゾーンを考慮していると書きました。実際にそれを試してみましょう。**タイムゾーンを扱うクラスは NSTimeZone** といい、いくつかの便利な生成用のクラスメソッドが用意されています。

NSTimeZone : タイムゾーン名からインスタンス生成

書式
```
+ (NSTimeZone *) timeZoneWithName : (NSString *) sName
```
入力
sName : タイムゾーン名。 ex. "JST"
出力
返り値 : タイムゾーンのインスタンス。

NSTimeZone : 世界標準時のずれからインスタンス生成

書式
```
+ (NSTimeZone *) timeZoneForSecondsFromGMT : (int) sec
```
入力
sec : 世界標準時とのずれ。進んでいるときは正の値。単位は秒。日本は60x60x9 = 32400。
出力
返り値 : タイムゾーンのインスタンス。

NSTimeZone : システム環境設定のタイムゾーン

書式
```
+ (NSTimeZone *) localTimeZone
```
出力
返り値 : システム環境設定のタイムゾーンのインスタンス。

以下のコードでは現在の時刻を取得した後、タイムゾーンのみを変更しています。これで世界時計を簡単に作れます。

```
NSCalendarDate *cd1 = [ NSCalendarDate calendarDate ];
NSCalendarDate *cd2 = [ NSCalendarDate calendarDate ];

[ cd1 setTimeZone : [ NSTimeZone timeZoneForSecondsFromGMT : 0 ] ]; // GMT
[ cd2 setTimeZone : [ NSTimeZone timeZoneWithName : @"JST"     ] ]; // JST

NSLog( @"cd1 = %@", cd1 ); // ex. "cd1 = 2002-06-27 06:56:19 +0000"
NSLog( @"cd2 = %@", cd2 ); // ex. "cd2 = 2002-06-27 15:56:19 +0900"
```

■ カレンダーを作る

日時を扱えるようになりましたので、カレンダーを作成してみましょう。今月のカレンダーの画像を生成するクラス CalendarImage を作ります。生成するのは以下のようなイメージです。

● 修飾属性付き文字列の生成

今回作成するカレンダーは、ちょっと変った方法で画像を生成します。カレンダーは、数字や文字の集まりですので文字列で表現することもできます。日曜日を赤にしたりとか、今日の日にはアンダーラインを引きたいということも考えると、修飾属性付き文字列が必要になります。Cocoa には修飾属性付き文字列を扱うクラス NSAttributedString／NSMutableAttributedString があり、これを使うと上のようなカレンダーを表現できます。

面白いことに、この**修飾属性付き文字列は HTML から生成することが出来る**ようになっています。ですので、以下の手順を取ることにします。

1. カレンダーを表現するHTMLをNSMutableStringで作成する。
2. HTMLからNSAttributedStringを生成する。
3. NSAttributedStringからイメージを作成する。

まず、修飾属性付き文字列の生成の方法のところだけを抜き出して予め説明しておきます。以下のコードでは、カレンダーの先頭に表示する「 2002 7 」という年と月の部分だけを作っています。

```
NSCalendarDate *curTime = [ NSCalendarDate calendarDate ]; // 今の日時
int iYear  = [ curTime yearOfCommonEra ];
int iMon   = [ curTime monthOfYear     ];

NSString           *sHtml;
NSAttributedString *as;

// HTMLを生成
sHtml = [ NSString stringWithFormat :
                   @"<FONT FACE=Helvetica><B>%d</B> %d</FONT>", iYear, iMon ];

// 修飾付き文字列生成
as = [ [ [ NSAttributedString alloc ]
             initWithHTML : [ sHtml dataUsingEncoding : NSASCIIStringEncoding ]
      documentAttributes : nil ]
      autorelease ];
```

ポイントは、NSAttributedString の **initWithHTML : documentAttributes :** メソッドです。このメソッドは HTML を入力として修飾付き文字列を生成します。

NSAttributedString : HTMLから生成

書式

```
- (id) initWithHTML : (NSData        *) data
  documentAttributes : (NSDictionary **) docAtt
```

入力

data : HTMLのバイナリ。
docAttr : 生成した文字列の書式などの情報を受け取る辞書。nilだと受け取らない。

出力

返り値 : 修飾情報付き文字列のインスタンス。

● コーディング

では、実際にコーディングしていきましょう。まずは、ヘッダーから。**thisMonth** というメソッドで今月のカレンダーを生成します。

CalendarImage.h

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

@interface CalendarImage : NSObject {
}
  + (NSImage *) thisMonth; // 今月のカレンダーを生成
@end
```

つづいてソースの方です。

CalendarImage.m

```
#import "CalendarImage.h"

@implementation CalendarImage

+ (NSImage *) thisMonth {

  NSCalendarDate *now = [ NSCalendarDate calendarDate ]; // 今の日時
  NSCalendarDate *topDate; // 今月の1日
  NSCalendarDate *endDate; // 今月の末日

  NSMutableString    *sHtml;   // HTMLを格納
  NSAttributedString *asCal;   // 修飾付き文字列
  NSImage            *imgCal;  // カレンダーイメージ
```

今月のカレンダーですので、calendarDate メソッドで現在の日時をまず取得します。それ以外は、宣言ですが、今月の 1 日と末日のインスタンスや生成する HTML、修飾情報付き文字列、イメージを使います。

CalendarImage.m > thisMonth ...続き

```
  int iYear  = [ now yearOfCommonEra ]; // 今日の年
  int iMonth = [ now monthOfYear     ]; // 今日の月

  topDate = [ NSCalendarDate dateWithYear : iYear
                                    month : iMonth
                                      day : 1
                                     hour : 0
                                   minute : 0
                                   second : 0
                                 timeZone : nil ]; // 今月の1日を取得

  endDate = [ topDate dateByAddingYears :  0
                                 months : +1
                                   days : -1
                                  hours :  0
                                minutes :  0
                                seconds :  0 ]; // 今月の末日を取得
```

ここでは、今月の 1 日と末日を求めています。先程説明したコードと同じです。次からは、HTML の生成になります。

CalendarImage.m > thisMonth ...続き

```
  { ///// HTMLを作成 /////

    int iTopWeek;                          // 今月の1日の曜日
    int iToday  = [ now     dayOfMonth ]; // 今日の日付
    int iEndDay = [ endDate dayOfMonth ]; // 今月の末日
    int i, iWeek;

    sHtml = [ NSMutableString stringWithFormat :
                @"<pre><font size=4><b>%d %d</b></font><br>",
                iYear, iMonth ]; // 年月の部分を作成
```

ここで、カレンダーの先頭にある「 年月 」の部分を作っています。

CalendarImage.m > thisMonth ...続き

```
    iTopWeek = [ topDate dayOfWeek ];

    for ( i = 0 ; i < iTopWeek ; i++ ) // 一週目の空白を作成
       [ sHtml appendString : @"   " ];
```

カレンダーの一週目は、先月の部分の空白が必要になりますので、その空白を作成しています。今月の 1 日の曜日を iTopWeek に入れておいて、その直前までループさせています。

CalendarImage.m > thisMonth ...続き

```
    iWeek = iTopWeek;

    for ( i = 1 ; i <= iEndDay ; i++ ) { // 一ヶ月のループ

      if ( iWeek  == 0 ) { [ sHtml appendString : @"<font color=red>"  ]; }
      if ( iWeek  == 6 ) { [ sHtml appendString : @"<font color=blue>" ]; }
      if ( iToday == i ) { [ sHtml appendString : @"<u><font color=green>" ]; }

      [ sHtml appendString : [ NSString stringWithFormat : @"%2d", i ] ];

      if ( iToday == i ) { [ sHtml appendString : @"</font></u>" ]; }
      if ( iWeek  == 6 ) { [ sHtml appendString : @"</font>" ]; }
      if ( iWeek  == 0 ) { [ sHtml appendString : @"</font>" ]; }

      [ sHtml appendString : @" " ]; // 数字と数字の間の空白
```

ここからが一ヶ月分のループに入ります。iWeek には処理中の日の曜日が入ります。iWeek が日曜日ときは赤、土曜日なら青、今日なら緑のアンダーライン付きになるように数字の前後にタグを付けていきます。

CalendarImage.m > thisMonth ...続き

```
      iWeek++;
      if ( iWeek == 7 ) { // 土曜日すぎたら改行
          iWeek = 0;
          [ sHtml appendString : @"<br>" ];
      }

    } // 一ヶ月のループの終わり

    [ sHtml appendString : @"</pre>" ];

  }
```

一日分の処理が終わったら、現在の曜日である iWeek を進めます。土曜日の後で改行をする必要がありますので、iWeek が 7 になったら
タグを入れます。これを末日まで繰り返します。一ヶ月のループが終わったら、全体を囲っている<pre>タグを閉じます。

続いては、修飾情報付き文字列とイメージの生成です。

CalendarImage.m > thisMonth ...続き

```
  { ///// 修飾情報付き文字列 & イメージ生成 /////

    NSSize sizeCal; // カレンダーのサイズ

    asCal= [ [ [ NSAttributedString alloc ]
             initWithHTML : [ sHtml dataUsingEncoding : NSASCIIStringEncoding ]
        documentAttributes : nil ] autorelease ];
```

ここで HTML から修飾情報付き文字列を生成しています。先程説明したものと同じです。

CalendarImage.m > thisMonth ...続き

```
    sizeCal = [ asCal size ]; // 文字列のサイズ

    imgCal = [ [ [ NSImage alloc ] initWithSize : sizeCal ] autorelease ];
    [ imgCal lockFocus ];
    [ asCal drawAtPoint : NSMakePoint( 0, 0 ) ]; // 描画
    [ imgCal unlockFocus ];

  }

  return( imgCal );

}
```

修飾情報付き文字列は、グラフィックとしてどれくらいの大きさを持っているのかというのを **size** メソッドで得ることが出来ます。その大きさと同じ NSImage を作成して、その中に **drawAtPoint :** メソッドで文字列を描画すれば、カレンダーのイメージが作成されます。

比較的簡単なコードでカレンダーが実現できることがお分かりいただけたでしょうか。

以上