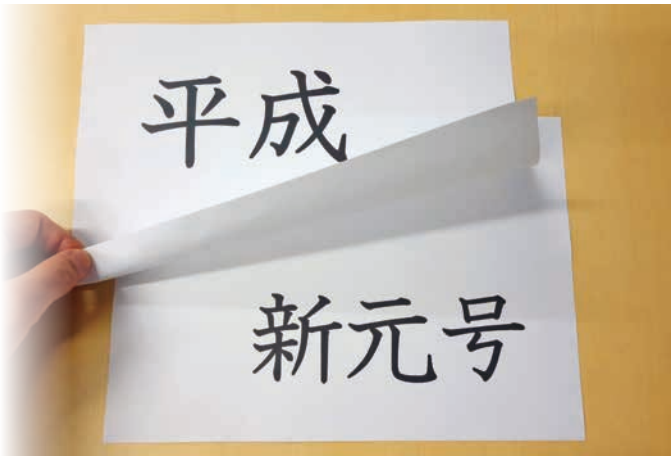
改元と情報システム



上原哲太郎 | 立命館大学情報理工学部

平成 30 年（2019 年）4 月 30 日に天皇陛下が退位され，翌 5 月 1 日に皇太子殿下が新しい天皇に即位される．これに伴い改元が行われ，新しい元号による年が始まることになる．改元は人々の生活や業務に少なからぬ影響を与えるであろうことから，新しい元号について改元前の公表が望まれていたが，さまざまな議論の末，政府は 4 月 1 日に新元号を公表することとした．本稿が『情報処理』に掲載される頃には，新しい元号が発表され，関係する人たちがその対応に追われているはずである．

改元に伴って改修が必要になる情報システムとはどのようなものだろうか．言うまでもなく，情報システムを構築するのに必要な OS やプログラミング言語処理系，ライブラリ，プログラミングフレームワークの多くは海外を起源とするものであり，標準では日付や時刻の処理に和暦を利用できない．よって日付や時刻を扱うソフトウェアを素直に記述すると西暦で扱うこととなり，改元の影響を受けない．問題となるのは，あえて和暦を用いた時刻や日付の内部表現もしくは入出力処理を行っている情報システムである．ではどの程度の数や規模の情報システムが和暦による内部表現もしくは入出力処理を行っているだろうか．これについてはきちんとした調査は恐らく行われておらず，はっきりしたことは分からない．ただ，和暦は官公庁や金融機関，交通機関などで業務上広く用いられており，印刷物等で和暦表記を多く目にすることから，これらの機関内で利用されている業務用の情報システム内では改元への対応を強いられていることは想像に難くない．

本稿は，情報システムの改修においてどのような作業が必要になるのか，筆者が掴んでいる限りでまとめたものである．必ずしも網羅的に調査した結果ではないことをご留意いただいた上でお読みいただきたい．

情報システムにおける時刻の内部表現

情報システム内にあるデータの内部表現が改元によって受ける影響を見積もるため，ソフトウェアにおける時刻や日付の内部表現について考えてみよう．多くのアプリケーションは，時刻の内部表現において OS が使用する形式を踏襲していると考えられる．たとえば Linux などの UNIX では，時刻は time_t 型の整数値として，世界協定時（UTC）1970 年 1 月 1 日 0 時 0 分からの秒数で表現する．これは POSIX.1 で定められた time システムコールが返す時刻情報の形式であり，しばしば UNIX Time または UNIX 時間と呼ばれる．これを tm 構造体と呼ばれる，年月日時分秒および曜日などが格納された構造体に変換して用いることがあるが，ここでの年の表現は西暦（厳密には西暦から 1900 を引いたもの）であり和暦は使えない（図 -1）．よって UNIX 上で動作するソ

フトウェアでは内部情報としての時刻情報も time_t 型か tm 構造体を用いて表現されていることが多く，この場合は内部状態は改元の影響を受けない．

Windows 系の OS については，時刻は内部的には SYSTEMTIME 構造体もしくは FILETIME 構造体で表現されることが多い（図 -2）．SYSTEMTIME 構造体は GetLocalTime 関数等が返す時刻情報の形式であり，年月日時分秒，さらに曜日とミリ秒単位の時間を格納しているが，このうち年の表現は西暦である．FILETIME 構造体は 1601 年 1 月 1 日 0 時 0 分からの経過時間（100 ナノ秒単位）を表す 64 ビット整数値であり，ファイルのタイムスタンプの表現等に用いられる．SYSTEMTIME 構造体と FILETIME 構造体の間を変換する関数も用意されているため，これらの構造体がアプリケーションソフトウェア内で時刻の内部表現として使われることは多い．この場合も，内部表現は改元の影響を受けない．

このように OS の標準的データ構造では，時刻は特定の時点（Epoch と呼ばれる）からの経過時間で表現するか，西暦で表現された年月日時分秒で表現している．言語処理系が標準で持つライブラリ（たとえば Java の java.util.Calendar クラスや JavaScript の Date クラス）も多くがこのいずれかの方式で時刻を表現しており，いずれにせよ和暦は利用できない．データベースにおける時刻型（多くの DBMS が持つ timestamp 型）も同様である．

まとめると，ほとんどの OS，言語処理系の標準ライブラリ，データベースなどにおいて，時刻を表現するデータ型は和暦をサポートしない．これらが提供するデータ型を活用してアプリケーションソフトウェアを作成する限りは，内部表現として和暦を利用する必然性はないと思われる．にもかかわらず，仮に和暦をあえて表現した内部表現を持つことがあ

```
struct tm {
  int tm_sec;      // 秒 [0-61] 最大 2 秒までのうるう秒を考慮
  int tm_min;      // 分 [0-59]
  int tm_hour;     // 時 [0-23]
  int tm_mday;     // 日 [1-31]
  int tm_mon;      // 月 [0-11] 0 から始まることに注意
  int tm_year;     // 年 [ 西暦 1900 年からの経過年数 ]
  int tm_wday;     // 曜日 [0: 日 1: 月 ... 6: 土 ]
  int tm_yday;     // 年内の通し日数 [0-365] 0 から始まることに注意
  int tm_isdst;    // 夏時間を採用していれば正の値，そうでなければ 0
};

time_t time(time_t *tsec); // UTC で 1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数
struct tm *localtime(const time_t *tsec);  //UNIX 時間を現地時間で tm 構造体に変換
char *ctime(const time_t *tsec); // UNIX 時間を文字列表現に変換

struct timeval  { // BSD 起源の時刻表現構造体
       time_t  tv_sec;    // UNIX 時間
       suseconds_t  tv_usec; // 1 秒未満部分（マイクロ秒単位）
};
struct timespec  { // POSIX で定義された時刻表現構造体
       time_t  tv_sec;    // UNIX 時間
       long  tv_nsec; // 1 秒未満部分（ナノ秒単位）
};
```

図 -1
Linux / POSIX での
時刻表現と主な関数

るとすれば，入出力データをそのまま内部表現に利用する場合であろう．たとえば，生年月日の入力において

生年月日を入力（元号は○で囲んで下さい）	
1. 明治 2. 大正 3. 昭和 4. 平成	年　　月　　日

このような帳票があった場合，入力されたデータを西暦に変換せず，そのまま

```
typedef struct _BIRTHDAY {
    int gengo;   // 元号 ( 明治 =1, 大正 =2, 昭和 =3,
                 // 平成 =4)
    int year;    // 年
    int month;   // 月
    int day;     // 日
} BIRTHDAY;
```

という内部表現で保存し利用するようプログラムされていれば改元に伴う改修が必要である．このような場所をプログラム中から探し出し，新元号向けの処理を書き加えることは，結局ソフトウェアの解析から始めることになり，かつての 2000 年問題と同様の困難が伴うと思われる．

標準ライブラリ等における新元号対応

たとえ入出力において和暦での表現が必要になるとしても，情報システムの内部表現としては OS や言語処理系標準の時刻データ型に変換した方が何かと都合が良い．特に，時刻の前後関係の判定や 2 つの時刻間の経過時間を得ようとすれば，標準的な時刻データ型に変換した内部表現を持つことで，これらの処理を行うためのライブラリの恩恵が得られる．一方でその場合には，時刻の内部表現と和暦表現との間で変換の必要が生じる．仮に改元が必ず元日に行われるのであれば西暦と和暦の相互変換は容易だが，実際にはそうではないので変換は煩雑になる．

```
typedef struct _SYSTEMTIME {
    WORD wYear;          // 年（西暦）
    WORD wMonth;         // 月
    WORD wDayOfWeek;     // 曜日
    WORD wDay;           // 日
    WORD wHour;          // 時
    WORD wMinute;        // 分
    WORD wSecond;        // 秒
    WORD wMilliseconds;  // ミリ秒
} SYSTEMTIME;


SYSTEMTIME systime;
GetLocalTime(&systime); // ローカル時間
GetSystemTime (&systime); // 世界協定時 (UTC)

//FILETIME 構造体は 1601 年 1 月 1 日からの
//100 ナノ秒間隔の経過時間を表す 64 ビット整数値
typedef struct _FILETIME {
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
} FILETIME;
FileTimeToSystemTime();   //FILETIME 構造体から SYSTEMTIME 構造体へ変換
SystemTimeToFileTime();   //SYSTEMTIME 構造体から FILETIME 構造体へ変換
```

図 -2
Windows での時刻表現と主な関数

Windowsの場合，「時刻と言語」内の設定によって日付の表記に和暦が利用できるが，これはAPI等の挙動にも影響を及ぼす．単純にWindows APIや標準ライブラリの機能を使って時刻を表す文字列を作っているプログラムでは設定だけで和暦を出力でき，改元対応もOSの更新だけで済む．マイクロソフトが提供するプログラミングフレームワーク.NET Frameworkでは，OSの設定によらず和暦を明示してデータ入出力を容易にするためのクラスやメソッドが用意されている．Javaも Java6以降はロケール設定が日本であれば和暦が使えるCalendarクラスが用意されているので，和暦表記での入出力が容易である．つまり，これらの機能を用いて作られたプログラムでは，OSやライブラリを最新版に更新するだけで改元への対応が終わる可能性がある．

ただし，実際の改元対応はそう容易ではない．たとえば.NET Frameworkの場合，時刻を扱える標準クラスとしてCalendarがあるが，これを継承したJapaneseCalendarというクラスが和暦を扱うためのメソッドを提供している．具体的にはJapaneseCalendar.GetEra(DateTime)とすれば，引数に与えたDateTimeオブジェクトの日付が和暦でいえばどの元号にあたるかを得ることができる．しかしこのメソッドの返り値は，明治＝1，大正＝2，昭和＝3，平成＝4といった数字なので，この値が新元号を意味する5であるときに，対応する処理がなされていないことがある．Windows 10が2018年春にバージョン1803に更新された際，改元対応のため2019年5月以降の日付に対するGetEraの値が5になるような設定が行われたが，これに伴い和暦を扱うプログラムの一部が正常に動かなくなるトラブルが発生したため，2018年9月21日にその設定を削除するアップデートが配布された．

このように，OSやライブラリの更新だけで改元に対応するためにはプログラムがよほど「行儀良く」書かれている必要がある．上記の例で言えば，元号が加わることを見越して，たとえばGetEraの返り値に対応する元号の文字列をCultureInfoクラスを通じて得ておく（そういう機能がある）と，改元にも対応したプログラムにできるのであるが，コードが煩雑になるためそうなっていないプログラムは少なくないだろうと予想している．

ちなみに，上記のWindowsのAPIや.NET Frameworkの挙動は図-3のようなレジストリによって制御されている．このレジストリは明治以降の各元号とその省略形，英語表記と省略形，改元の日付が書かれているので，一般のプログラムでもこのレジストリの値を参照することで和暦に関する処理が容易になるだろう．

アプリケーションにおける新元号対応

WordやExcelといったアプリケーションにおいても和暦に関する機能があるので，改元の際には対応が必要になると思われる．特にExcelは日付の入力に関するサポート機能が充実しており，たとえばセルに「昭和64年1月8日」と入力すれば自動的に「平成1年1月8日」に修正されるとともに，セルの値は1989/1/8になる（内部表現としては1900年1月1日を1とするシリアル値32516になる）．このような挙動も改元とともに修正されると思われるが，マイクロソフトは迅速に対応を行うためか，新元号が発表される前に改修を開始してい

```
[HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Control¥Nls¥Calendars¥Japanese¥Eras]
"1868 01 01"="明治_明_Meiji_M"
"1912 07 30"="大正_大_Taisho_T"
"1926 12 25"="昭和_昭_Showa_S"
"1989 01 08"="平成_平_Heisei_H"
```

図-3
和暦を表すレジストリ

るようである．ところがそれが不具合を誘発した事件が発生した．2019 年 1 月 2 日，Excel 2010 に対する更新 KB4461627 が Windows Update 経由で配布開始されたが，これが不具合を誘発し Excel が起動しなくなる不具合が発生した．実は 2018 年 11 月にも同様に Access 2010 が更新直後に起動しなくなる不具合を起こしている．いずれも改元に備えた更新が原因と見られており，このようなソフトウェア改修の難しさも示すこととなった．

なお，正常にアプリケーションの改元対応が済んだ後も運用上は問題が発生する可能性がある．たとえば改元対応の改修後は，Excel などにおいて和暦での表記が平成 31 年 5 月以降は新元号に切り替わる．これは，スプレッドシート上での日付表記が切り替わることを意味するため，たとえばシリアル値でなく文字列で日付を検索したりソートしたりしていた場合に問題になる．Excel の場合はシリアル値を平成 32 年などと強制的に表記する方法もないため，運用を変えるなどの工夫が必要である．

合字の問題

かつて我が国においてビジネスで盛んに使われた NEC の国産パソコン PC-9801 には，限られた表示領域の中で和暦を表示するために，JIS 標準の漢字コードにはない合字を拡張していた．各元号を図 -4 のように 1 文字で表示するものであり，NEC 機種依存文字と呼ばれていた．Unicode の策定にあたっては従来の各社の機種依存文字も取り入れられた結果，元号の合字は U＋337B から U＋337E に割り当てられている．この合字は現在でも利用実績があるとの調査結果があったことから，Unicode コンソーシアムは 2019 年 3 月 5 日発表の Unicode 12.0 において，改元に先立って新元号の合字が割り当てられるべき文字コードを U＋32FF に予約し，5 月 7 日発表予定の Unicode 12.1 で実際に割り当てを行うとしている．

平成 31 年	㍻ 31 年
昭和 64 年	㍼ 64 年
大正 15 年	㍽ 15 年
明治 45 年	㍾ 45 年
（a）通常の表記	（b）合字

図 -4　元号の合字

実際の情報システムでは，新しい合字への対応は単に文字コードの割り当てだけでは完了しない．プログラムの改修だけではなく，フォントの更新が必要になる．また，合字とそうでない表記の年号，たとえば「平成」と「㍻」を同一視する処理を行っていたり，自動変換したりする処理を行っているようなアプリケーションは少し大きなプログラム改修が必要になる．さらに，平成までの合字は連続した文字コードが割り当てられていたが，新元号については離れたコードポイントであるため，その対応が必要になる場合もあるだろう．

そもそも，元号の合字は Unicode では利用可能であるが JIS X 0208 には含まれていないため，プレーンテキストの生成ができないなどの不都合が起きやすい．合字が Unicode に取り入れられたのは後方互換性のためであると割り切って，今後のアプリケーションではできるだけ利用しないようにすることが必要なのではないだろうか．

なお，すでに図 -3 で示した和暦対応レジストリに関連して，担当した Shawn Steele 氏が 2018 年 8 月 7 日にマイクロソフトの開発者ブログで公開したエントリの中で，誤ってレジストリのうち元号の略称を「平」などの頭文字の代わりに「㍻」などの合字にしたものを公表するトラブルがあった．これを参照してテストのために利用した開発者の中には，合字が JIS X 0208 に変換できないなどの影響を受けた例があったようである．このことからも，合字の使用は今後避けるべきなのではないかと考えている．

「元年」問題と印刷帳票の問題

前述の合字の問題とも絡むが，特に官公庁の情報システムにおいては，改元に伴う対応で最も工数がかさむのは印刷帳票の確認である．自治体を含む公的機関の業務では帳票や証明証などが大量に印刷されるが，これらの多くに和暦による日付が含まれており，これが問題なく印字されるかの確認は大変な作業である．特に，ここで合字が使われていたりした場合はフォントの対応が終わるまで確認作業にも入れないため，限られた時間での対応が強いられることになる．

また，和暦では1年を「元年」と表記する慣習があるが，これを表示や印字において厳密に対応しようとするとプログラム上も例外処理になるだけでなく，印刷レイアウトも採用しているフォントによっては崩れやすくなるため対応が難しくなってくるだろう．

運用の問題

ほかにも，今回の改元が年度の途中で行われることによる業務上のトラブルも予想される．たとえば，平成31年4月1日から始まる1年間を「平成31年度」として運用している機関は多いだろうが，改元後もその表記を使い続けるかは難しいところである．仮に5月1日以降は新元号による年度表記に切り替えるとすると，同一の年度に対して表記が2つ発生することになり，これに情報システムを対応させようとするとかなり難しい例外処理になりかねない．

このような表記の揺らぎはしばらくの間，さまざまな場所でトラブルを起こすことが想像される．特に和暦表記を用いたデータを複数のシステム間で交換しているような場合には，複数の異なる表記が同一の日付を表す状況がシステム改修が落ち着くまでの間続くと思われるため，今のうちにデータ交換のフォーマットを見直して西暦で正規化するなどの処理を行うべきだろう．

新しい時代に向けて

2018年夏に降ってわいたサマータイム問題とは異なり，改元はいつかは来ることが予想された事態であるので，情報システムにおける対応は十分な時間があったはずである．それでもやはり新元号発表以降の1カ月の準備期間でできることは限られるので，改元後も平成表記の和暦を私たちはあちこちで目にすることになると思われる．それを何とかうまく切り抜けて，新しい時代を気持ちよく迎えたいと心から願っている．

(2019年3月8日受付)

上原哲太郎（正会員） t-uehara@fc.ritsumei.ac.jp

立命館大学情報理工学部教授．京都大学博士（工学）．サマータイム導入反対，うるう秒廃止，ネ申 Excel 根絶，パスワード付き zip ファイル添付メール反対の立場を取る．専門：サイバーセキュリティ，ディジタル・フォレンジック，情報システム管理，自治体情報システム，情報倫理教育など．