連載

今からでも遅くない！ RPGⅢユーザーに贈る

RPGⅣの魅力と可能性

第６回　RPGⅣの関数を使おう！

中村 潤
株式会社アイ・ラーニング
IT 研修本部　IBM 製品研修部
ラーニング・アドバイザー

手続き型言語・非手続き型言語

本誌の発行は５月の若葉の頃ですが、４月からは街やオフィスで新入社員と思われる、フレッシュな人たちをたくさん見かけます。アイ・ラーニングでも多くの企業から新入社員をお預かりし、４月から新入社員研修を実施しています。カリキュラムは情報システム入門やアルゴリズムなど、IT技術者としてスタートする方々に必須の内容となっています。

先日、私は現場のマネージャーとして新入社員研修の受講生と一緒に「情報システム入門」の講義を聴いていました。その中でプログラミング言語の説明があり、RPGもCOBOLやJava、Cなどと一緒に紹介されていたのですが、COBOLは手続き型言語で、RPGは非手続き型言語、という解説がなされていたのです（図表１）。

本連載では、RPG Ⅳのフリーフォームの便利さを繰り返しお話ししてきましたが、私どもの教育現場ではまだ、RPGは非手続き型言語という認識なのです。それは、前号でも紹介したようにRPGの内部論理を使ったコードがまだ多く残っているからで、READ、WRITEのロジックを書かずともプログラムを作れるのがRPGだからです。

とはいえ、非手続き型言語が悪いというわけではけっしてありません。むしろ、RPG内部論理を使った非手続き型言語のスタイルが、IBM iにおけるプログラム開発の効率を大きく向上させたのです。しかしその一方で、他の言語との共通性は希薄になっていきました。RPG以外の言語は共通点が多く、JavaからVBA、VBAからCなどへの移行が簡単ではないにしても少しの努力で可能ですが、JavaやCからRPGへの難易度は高く、RPGプログラマーの養成は時間とコストがかかるというイメージが強くなったようです。

Java／Cプログラマーも理解できる

しかし、RPG Ⅳでフリーフォームコーディングが可能になり、RPGはJavaなどと同様のスタイルでコーディングできるようになっています。

実際に、RPGでForやDowhileを使ったロジックをJavaやCプログラマーに見てもらうと、何をやっているかわからないという答えは、まず返ってきません。専門学校や大学でJavaやCを勉強した若いプログラマーにも楽に理解してもら

図表１　プログラミング言語の分類

● 手続き型言語	Fortran、COBOL、PL/I、C、Visual Basic（VB）など
・オブジェクト指向型	Smalltalk、C++、Java、C#、Visual Basic .NET など
・スクリプト型	Perl、PHP、Ruby、Python、JavaScript など
● 非手続き型言	RPGなど

えるはずです。JavaやCのプログラマーの中には、READやWRITEといったファイルの読み書き命令が不得手の方がいますが、SQLをご存じの方は多いので、ファイルの読み書きをSQLを使ったロジックに変えるのも一つの手段です。このSQLについては次号で触れてみましょう。

RPG Ⅳを利用するメリット

また、本連載のRPG Ⅳについての話は、RPG Ⅲプログラマーに対して発信しているつもりですが、RPG Ⅲよりも一世代前の、RPG Ⅱをお使いの方もまだおられるようです。

私も20年以上前にS/36でRPG Ⅱのプログラムを書いていました。RPG Ⅱはプログラム記述ファイルを扱いますので、ファイルレイアウトを入力仕様書（仕様書コード＝I）で記述し、RPG内部論理でファイルの読み書きを行っていました。

RPG ⅡからRPG Ⅲへの移行は自動的に、というわけにはいきません。プログラム記述ファイルを外部記述ファイルに変更する必要がありますし、プロシージャ（これはRPG Ⅳでのプロシージャではなく、制御言語としてのプロシージャで、メインフレームのJCLに該当します）をCLプログラムに移行する必要もあります。かなりの作業量になると思いますが、システムが動き続ける限りは誰かがシステムを保守しなければなりません。RPG ⅢとRPG Ⅱも見た目は同じようですが、RPG Ⅱは使っているファイルがデータベース・ファイルであってもプログラム記述で書く必要があるので、RPG Ⅲプログラマーよりも人材確保が難しいでしょう。

しかし、RPG ⅡでもRPG Ⅲでも、蓄積したデータをIBM iの強固なデータベース上で使用し、RPG Ⅳという同じRPGを使ってシステムを継続できることが、これまでシステムを構築し、運用してきた方々にとって一番安心できる形ではないでしょうか。他社のプラットホームで、RPGプログラムをJavaで作り直すほうが、はるかに不安が大きいと思います。これまでの繰り返しになりますが、新しい世代のプログラマーにはフリースタイルのRPG Ⅳのほうが習得期間も短くて済み、今までのRPGプログラマーにとってもJavaやCに比べればはるかにわかりやすいスタイルのプログラムであるはずです。

EXCPT命令とRPG ⅣのUPDATE命令

プログラミング言語に限らず、新しい環境に移行する際に、今まで使っていたものが使えなくなるのが一番困ります。IBM iの最大の特徴は、今まで使ってきたシステムにいっさい手を加えることなく新しいシステムに移行できることです。

しかしRPG ⅡからRPG Ⅲになったとき、RPG ⅢのUPDAT命令やWRITE命令が、今までのEXCPT命令と違うところがありました。RPG ⅢでもEXCPT命令が使えるのでそちらを使えばよいのですが、EXCPTと違ってUPDAT命令やWRITE命令は、出力仕様書の記入は不要です。これ自体は大変便利なのですが、EXCPT命令が優れているのは、出力仕様書に記入したフィールドしかデータベースに反映されないことで、バグが発生しにくくなります**（リスト1）**。

ところがUPDAT命令やWRITE命令は、プログラム中で何も値を変更していないフィールドであっても、バッファ上のデータが書き込まれます。プログラム中で意図していないフィールドを変更するようなバグがあった場合、そのまま変更や書き出しをしてしまう可能性があるのです。その理由で、UPDATやWRITEを使わずにEXCPT命令と出力仕様書を使ったコーディング方法は、今でも重宝されているのです。コンパイルリストを見ると、出力ファイルの各フィールドに出力されていないものは出力なし、NOTOUT、などの表示が見られます**（リスト2）**。

しかしデータベース・ファイルにEXCPT命令を使うのは、無駄に出力仕様書のステップ数を増やす原因になります。EXCPTを使うかUPDATやWRITEを使うかは、結局好みの問題に帰せられるのですが、RPG ⅣのフリーフォームではUPDATE（RPG ⅣではUPDATがUPDATE

リスト1

```
C                   MOVE      *DATE          TKNYUK
C                   EXCEPT    @KOSHIN
C                   READ      TOKMSR
C                   ENDDO
C                   SETON                                          LR
OTOKMSR    E            @KOSHIN
O                       TKNYUK
```

のフルスペルになります）命令で、出力先のフィールドを限定できます（リスト3）。

リスト3では、UPDATE命令でパラメータに%FIELDS関数を使い、更新対象のフィールドのみを指定します。%FIELDS関数を省略すると、データベース・ファイル上のすべてのフィールドが更新対象となります。RPG Ⅳのフリーフォームでこのような使い方ができるのは、やはりEXCPTを使っていた頃の名残を意図的に復活させた感があります。しかしWRITE命令でも同じように%FIELDS関数が使えればなおよいのですが、今のところ、なぜかWRITE命令で%FIELDS関数を使うことはできません。

RPG Ⅳで関数を使うメリット

%FIELDS関数に触れましたが、これまでお話ししたように、RPG Ⅳでは多くの関数が使えるようになりました。日付の計算など、RPG ⅡやRPG Ⅲでは自前でロジックを組まないとできなかったことが関数一つでできるようになりました。また、フリーフォームでは使えなくなった演算命令がいくつかありますが、基本的には関数で対応しています。

その一つが%LOOKUP関数で、もうおわかりのように、これはLOOKUP命令に代わるものです。面白いのはRPG Ⅳになって、READ命令やCHAIN命令の結果の標識は省略できるようになり、%EOF関数や%FOUND関数で読み取り結果を判断できるようになりましたが、LOOKUP命令も配列の検索結果を%FOUND関数で判断することができるようになったにもかかわらず、結果の標識は省略できないのです。ですから結果の標識を指定しているのに、検索の結果は%FOUND関数で判断しているという、ちょっと不思議なロジックになりますが、LOOKUPに関しては%LOOKUP関数を使うのがスマートでしょう（リスト4）。

%LOOKUP関数でも検索の結果を%FOUND関数で知ることができますが、検索が不成功ですと指標の値は0にセットされますので、判断が非常に容易です。また%LOKUP関数はあくまで正確に一致するものを検索しますが、LOOKUP命令の結果の標識で、検索する値よりも大きい値、小さい値を検索する場合と同様に、%LOOKUPLT、%LOOKUPLE、%LOOKPGT、%LOOKUPGEがあります（リスト5）。

リスト2

```
     31=O*(NOTOUT)              TKADR1              65A CHAR       20
     32=O*(NOTOUT)              TKADR2              85A CHAR       20
     33=O*(NOTOUT)              TKTIKU              87A CHAR        2
     34=O*(NOTOUT)              TKPOST              95A CHAR        8
     35=O*(NOTOUT)              TKTELE             108A CHAR       13
     36=O*(NOTOUT)              TKGURI             113P PACK      9,0
     37=O*(NOTOUT)              TKNURI             118P PACK      9,0
     38=O*(NOTOUT)              TKZURI             123P PACK      9,0
     39=O*(NOTOUT)              TKUZAN             128P PACK      9,0
     40=O*(NOTOUT)              TKGEND             133P PACK      9,0
     41=O                       TKNYUK             138P PACK      8,0
     42=O*(NOTOUT)              TKSIME             139A CHAR        1
        * * * * * *   ソ ー ス の 終 わ り   * * * * * *
```

リスト3

```
/free
chain tkbang tokmsp;
tkuzan =- tkguri;
tknyuk = udate;
update tokmsr %fields(tkguri:tknyukknyuk);
/end-free
```

RPG ⅡもRPG Ⅲも標識を使ったので、データ名が記号化されてしまい、一読して意味がわからないことが多かったのですが、関数ならば必要なパラメータを渡して求める値が返ってくる、という形になりますので、非常にわかりやすいロジックになります。このようにRPG Ⅳは従来の命令が関数化されたことによって、RPGプログラマーにも、JavaやCのプログラマーにも、どちらにもわかりやすい言語になっています。

少しずつRPG Ⅳへ移行する

やはり近年のプログラミングでは、関数が使えない言語は敬遠されても仕方がないように思えます。またサブ・プロシージャやサービスプログラムを使えば独自の関数を作ることもできます。せっかくRPGがこういう機能を備えているのですから、使わない手はありません。これも繰り返し申し上げていることですが、一度にすべてRPG Ⅳに移行する必要はありませんから、少しずつこういう機能を試しながら移行していくやり方もいいと思います。RPG Ⅳの便利な関数やフリーフォームの柔軟性をぜひ実際に使ってみて実感していただければと思います。

リスト4

```
/FREE
arr(1) = 'FUKUOKA';
arr(2) = 'KYOTO';
arr(3) = 'NAGOYA';
arr(4) = 'OSAKA';
arr(5) = 'SAPPORO';
arr(6) = 'TOKYO';
n = %LOOKUP('OSAKA':arr);
//n=4
n = %LOOKUP('SHINAGAWA':arr);
//n=0
(not found)
/END-FREE
```

リスト5

```
/FREE
arr(1) = 'FUKUOKA';
arr(2) = 'KYOTO';
arr(3) = 'NAGOYA';
arr(4) = 'OSAKA';
arr(5) = 'SAPPORO';
arr(6) = 'TOKYO';
n = %LOOKUPLE('OSAKA);
//n=4
n = %LOOKUPLE('MORIOKA':arr);
//n=3
n = %LOOKUPGT('SHIZUOKA':arr);
//n=6
/END-FREE
```