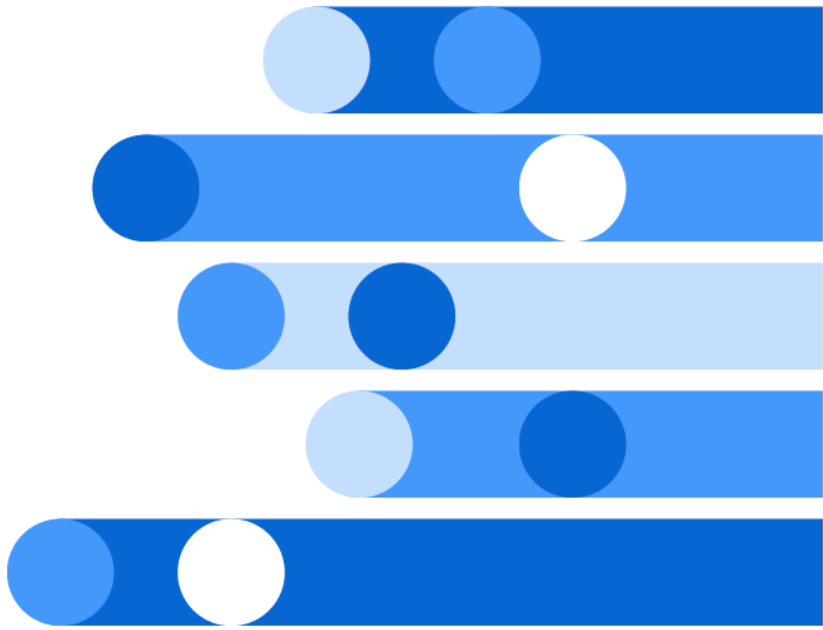
SAS® DS2 プログラマガイド

2020.1 - 2024.04*



* このドキュメントは、ソフトウェアの追加バージョンに適用される場合があります。このドキュメントを SAS Help Center で開き、バナーのバージョンをクリックすると、使用できるすべてのバージョンが表示されます。

SAS® ドキュメント
2024 年 4 月 17 日

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2020. *SAS® DS2 プログラマガイド*. Cary, NC: SAS Institute Inc.

SAS® DS2 プログラマガイド

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

April 2024

v_001-P1:ds2pg

目次

1 部

概要

1

DS2 言語について

DS2 言語について

DS2 は、高度なデータ操作に適した SAS 独自のプログラミング言語です。DS2 は SAS Viya プラットフォームに含まれており、SAS DATA ステップと交差します。また、ANSI SQL データ型を含む追加のデータ型、追加のプログラミング構造要素、およびユーザー定義のメソッドとパッケージも含まれています。

いくつかの DS2 言語要素は、埋め込まれた FedSQL 構文を受け入れ、ランタイムで生成されたクエリは、DS2 とサポートされているデータベース間でインタラクティブにデータを交換できます。これにより、入力テーブルの SQL 前処理が可能になり、2 つの言語の能力が効果的に組み合わされます。

SAS Viya プラットフォームでは、DS2 プログラムは次の環境で実行できます。

- SAS Cloud Analytic Services (CAS) Server
- SAS Compute Server
- SAS Event Stream Processing (ESP) Server
- SAS Federation Server
- SAS Micro Analytic Service

SAS Federation Server は、マルチユーザーの標準ベースのフェデレーションデータアクセスを提供する、個別にライセンスされたサーバーです。SAS Micro Analytic Service は、インストリームデータ分析とディシジョンの自動化をサポートする、メモリに常駐する高性能のプログラム実行プラットフォームです。マイクロサービスとして配置することも、ESP、CAS、Compute サーバーなどのサーバープロセス内で実行することもできます。このサービスは、SAS Event Stream Processing ソリューションに加えて、SAS Intelligent Decision および SAS Model Manager で使用できます。

DS2 プロシジャを使用すると、SAS Studio から DS2 言語ステートメントをサブミットできます。PROC DS2 の詳細については、Base SAS プロシジャガイドを参照してください。

注: DS2 言語は多くのデータソースで使用できるため、データ要素を説明するために行、列、およびテーブルという用語が使用されます。これを SAS DATA ステップの用語と比較すると、行はオブザベーションに対応し、列は変数に対応し、テーブルはデータセットに対応します。

DS2 プログラムの実行

DS2 プログラムは、次のいずれかの方法でサブミットできます。

- DS2 プロシジャを使用して SAS Studio を介する方法。DS2 プロシジャは、SAS Compute Server または CAS サーバーで DS2 コードを実行するために使用できます。1 つの PROC DS2 ステップに、複数の DS2 プログラムを含めることができます。

  詳細については、*"DS2 プロシジャ" (Base SAS プロシジャガイド)*を参照してください。

- SAS Studio で、CAS サーバーで runDS2 アクションを使用します。runDS2 アクションは、CAS プロシジャと組み合わせて使用されます。

  注: Python または Lua を使用している場合を除き、PROC DS2 を使用して DS2 コードを CAS サーバーにサブミットすることをお勧めします。

  詳細については、*"DS2 Action Set" (SAS Viya Platform: System Programming Guide)*および 19 章, "CAS の DS2: 概念" (297 ページ)を参照してください。

- SAS Federation Server で SAS Federation Server 用の SAS LIBNAME エンジンを使用します。詳細については、*SAS LIBNAME Engine for SAS Federation Server: ユーザーガイド*を参照してください。

- SAS Event Stream Processing、SAS Intelligent Decisions、および SAS Model Manager で、SAS Micro Analytic Service を使用します。詳細については、*SAS Micro Analytic Service: プログラミングと管理ガイド*の About Using SAS Micro Analytic Service および DS2 Programming for SAS Micro Analytic Service を参照してください。

サポートされているデータソース

DS2 は、次のデータソースにアクセスできます。

- Amazon Redshift
- CAS テーブル
- UNIX 動作環境用の DB2
- Google BigQuery
- Greenplum
- Hadoop (Hive)
- Impala
- JDBC に準拠しているデータベース(PostgreSQL など)
- メモリデータストア(MDS)
- Microsoft SQL Server
- MongoDB *
- MySQL (MySQL 8 サーバーを含む)
- Netezza
- ODBC に準拠しているデータベース(Microsoft SQL Server など)
- Oracle
- PostgreSQL
- Salesforce
- SAP (読み取り専用)
- SAP HANA
- SAP IQ
- SAS データセット
- SAS Scalable Performance Data Engine (SPD Engine)データセット
- UNIX 動作環境の SAS Scalable Performance Data Server (SPD Server)テーブル
- SingleStore
- Snowflake
- Spark
- UNIX 動作環境用の Teradata
- Vertica
- Yellowbrick

注: 次のデータソースはサポートされていません。

- Informix
- OLEDB
- SAP ASE

DS2 は、SAS プログラミングランタイム環境または CAS サーバー上の MDS および SPD Server を除くすべてのデータソースで 動作できます。

*MongoDB データソースには、テーブルを作成するための特別な要件があります。データに応じて、ルートテーブル、親テーブル、または子テーブルを作成することをお勧めします。DS2 ではルートテーブルのみを作成できます。親テーブルと子テーブルを作成するには、FedSQL を使用する必要があります。*SAS/ACCESS for Relational Databases: リファレンス*の MongoDB に関する情報を参照してくださ。

DS2 プロシジャと SAS Federation Server は、異なるデータソースをサポートします。それぞれがサポートするデータソースについては、*Base SAS プロシジャガイド*および *SAS Federation Server: 管理者ガイド*を参照してください。

データソースへのアクセスについては、必要に応じて *SAS/ACCESS for Relational Databases: リファレンス*および *SAS/ACCESS for Nonrelational Databases: リファレンス*を参照してください。

対象とするユーザー

このドキュメントの情報は、これらの役割を実行する次のユーザーを対象としています。

- クライアントアプリケーションを作成する**アプリケーション開発者**。テーブルの作成、テーブルのバルクロード、テーブルの操作、データのクエリを行うアプリケーションを作成します。
- クライアント/サーバー環境を設計および実装する**データベース管理者**。データベースを設計し、データソースのメタデータを設定することでデータを管理します。つまり、データベース管理者はデータモデルを構築します。
- 数値精度の向上、CPU バインドプロセスの並列計算、DATA ステッププロセスへの直接入力としての明示的なパススルー SQL クエリの使用、ビッグデータ環境での in-databas 処理など、DS2 言語の高度な機能を利用したいか、または利用する必要がある **SAS プログラマ**。

DS2 を使用する場合

通常、DS2 プログラムは、次のアクションを実行するアプリケーション用に作成されています。

- サポートされている新しいデータ型を使用した結果の精度を必要とする
- 変数スコープ、ユーザー定義のメソッドとパッケージ、スレッド処理のサポートなど、DS2 言語の新しいプログラミング要素を活用する
- DS2 プログラム内から SAS FedSQL を実行する必要がある
- SAS セッションの外部で実行する必要がある(たとえば、SAS Federation Server 上)

DATA ステッププログラムから DS2 プログラムへの変換

DSTODS2 プロシジャを使用して、DATA ステップコードを DS2 に変換することができます。すべての DATA ステップコードの変換がサポートされているわけではなく、手動での変換が必要になる場合があります。変換できないコード行はコメントに入れられます。詳細については、"DSTODS2 プロシジャ" (*Base SAS プロシジャガイド*)を参照してください。

DATA ステップコードを DS2 に変換し、プログラムが構文的に完全であることを確認したら、DS2 プロシジャを使用して、SAS Viya プラットフォームでプログラムを実行できます。

DS2 言語の構文規則

文字体裁の規則

DS2 言語構文のドキュメントで使用される書体には、特別な意味があります。

UPPERCASE BOLD
ステートメントや関数の名前などの DS2 キーワードを識別します(例: PUT)。

UPPERCASE ROMAN (大文字のローマン体)
リテラルの引数と値を識別します(例: FROM)。

italic
指定する引数または値を識別します。斜体の項目は、次のいずれかのユーザー指定の値を表すことができます。

- 引数に割り当てられた非リテラル値(例: ALTER=*alter-password*)。
- 非リテラル引数(例: KEEP=(*column-list*)。

斜体の項目を複数使用できる場合、その項目は *item* [, ...*item*]として表されます。

monospace
SAS コードの例を識別します。

構文規則

このドキュメントでは、バッカスナウア記法(BNF)を使用しています。具体的には、Jim Melton が *SQL:1999 Understanding Relational Language Components* で使用しているのと同じ構文表記法です。

従来の SAS 構文と DS2 言語リファレンスドキュメントで使用されている構文の主な違いは、オプションの構文引数の表示方法にあります。従来の SAS 構文では、オプションの構文を示すために山かっこ(< >)が使用されます。DS2 言語の構文では、角かっこ([ ])がオプションの構文を示すために使用され、山かっこは非終端の構成要素を示すために使用されます。

DS2 言語の構文では、次の記号が使用されます。

::=
この記号は、"で構成される"または"として定義されている"と解釈できます。

<>
山かっこは、非終端の構成要素(つまり、下位レベルの構文文法にさらに分解できる構文の構成要素を示します。

[ ]
角かっこはオプション引数を示します。角かっこで囲まれていない引数はすべて必須引数です。角かっこは、その前にリテラルであることを示すバックスラッシュ(\)が付いていない限り、入力しないでください。

{ }
中かっこは、必要な複数単語の引数を区別する方法を提供します。中かっこは、その前にリテラルであることを示すバックスラッシュ(\)が付いていない限り、入力しないでください。

|
縦棒は、グループの中から 1 つの値を選択できることを示しています。棒で区切られた値は相互に排他的です。

...
省略記号は、省略記号に続く引数または引数のグループが何度でも繰り返されることを示しています。省略記号と次の引数が角かっこで囲まれている場合はオプションです。

\
バックスラッシュは、次の文字がリテラルであることを示します。

次の例は、このセクションで説明されている構文規則を示しています。これらの例には、完全な構文ではなく、選択された構文要素が含まれています。

```
1 SET 2 <table-reference> [... [<table-reference>] [INDSNAME=variable];
  [ 3 BY [DESCENDING] 4 column [ 5 ...[DESCENDING]column];
  6 <table-reference>::=
  {table (table-options)} 7 | 8 \{sql-text 8 \}
```

1 SET はステートメントの名前であるため、大文字の太字で示されています。

2 <table-reference>は非終端引数であり、さらに下位レベルの構文文法に分解されるため、山かっこで囲まれています。少なくとも 1 つの<table-reference>を指定する必要があります。

3 BY と DESCENDING はリテラル引数であるため、大文字のローマン体で表記されます。DESCENDING はオプションの引数であるため、角かっこで囲まれています。

4 *column* は、指定できる引数であるため、斜体で示しています。

5 *column* の 2 番目のインスタンスを囲む角かっこと省略記号は、引数がカンマで区切られている限り、この引数を何度でも繰り返すことができることを示しています。

6 <table-reference>::=非終端引数の構文は次のように読み取られます: <table-reference>はテーブル名とテーブルオプション、または埋め込み SQL テキストで構成されます。

7 縦棒(|)は、*table* [*table-options*]または *sql-text* のいずれかを指定できますが、両方は指定できないことを示します。

8 *sql-text* を囲む中かっこの前のバックスラッシュ(\)は、それらの中かっこがリテラルであり、入力する必要があることを示します。

2 部

DS2 の概念

2

DS2 プログラミングセマンティクス

基本的な DS2 プログラムの構文

DS2 プログラムは、宣言のリストとそれに続くメソッドステートメントのリストで構成されます。簡単な宣言リストの例を次に示します。

```
declare int x;
dcl double d;
```

簡単なメソッドステートメントリストの例を次に示します。

```
method init();
  put 'in init';
end;

method run();
  put 'in run';
end;

method term();
  put 'in term';
end;
```

2 つのリストを組み合わせると、単純な DS2 プログラムが作成されます。

```
declare int x;
declare double d;

method init();
end;

method run();
end;

method term();
end;
```

通常、DS2 プログラムはより複雑ですが、この単純なプログラムにはいくつかの構文要素が含まれています。

キーワード:

- DECLARE/DCL
- DOUBLE
- METHOD
- END

識別子:

- x
- d
- INIT
- RUN
- TERM

字句区切り記号:

- (
-)
- ;

このプログラムは、DECLARE ステートメントまたは METHOD ステートメントで識別子を宣言する方法を示しています。また、DS2 プログラムの高レベル構造が、一連の変数宣言とそれに続く一連の METHOD ステートメントで構成されていることも示しています。次のセクションでは、これらの用語が DS2 で何を意味するかを説明します。

基本的な DS2 プログラムのセマンティクス

変数宣言ステートメント

変数宣言はメモリ空間を割り当て、変数名と呼ばれる識別子でそのメモリを識別します。宣言は、明示的または暗黙的に、変数にメモリを割り当て、そのメモリ位置に保存できるデータの型を指定します。DS2 では、DECLARE ステートメントを使用して変数を宣言します。DECLARE ステートメントは、次のアクションを実行します。

- 識別子をメモリ位置に割り当てます。その識別子が変数名になります。
- 変数が保持できるデータの型を指定します。
- 指定された量のメモリを変数に割り当てます。

1 つの DECLARE ステートメントで複数の変数を宣言できます。詳細については、“DECLARE ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

メソッド

メソッドは基本的なプログラム実行ユニットです。DS2 では、メソッド構造を使用して、関連するプログラムステートメントを 1 つの構文的に識別可能な場所にグループ化します。メソッド内のステートメントのグループは、簡単に複数回呼び出したり、実行したりできます。

DS2 メソッドは、C などの言語の関数や Java のメソッドに似ています。従来の DATA ステッププログラミングでは、LINK および RETURN ステートメントは、DS2 データプログラムのメソッド定義と同様に使用されていました。ただし、LINK と RETURN はローカルスコープの変数を作成できないため、再利用可能なコードモジュールの配布にはあまり役に立ちません。FCMP プロシジャを使用すると、ローカルスコープで再利用可能な関数を作成できます。これは、DS2 メソッドと同等です。

メソッドはスコープブロックを定義します。したがって、メソッド本体のパラメーターと変数宣言は、メソッドに対してローカルです。

> ヒント　DS2 メソッドは、最大 1000 個の引数をサポートできます。1000 を超える引数を持つ DS2 メソッドは、コンパイルエラーを生成する可能性があります。

型シグネチャ(または単に**シグネチャ**)は、メソッドのパラメーター型の順序付きリストとして定義されます。2 つのメソッド定義の名前が同じで、型シグネチャが異なる場合、そのメソッドは**オーバーロード**されます。2 つのメソッド定義が同じ名前と同じ型シグネチャを持つ場合、エラーが発生します。

注: セッションエンコーディングに複数バイトが必要な場合、CHAR および NCHAR データ型のみに基づくメソッドをオーバーロードすることはできません。セッションエンコーディングで文字ごとに複数のバイトが必要な場合(UTF-8 など)、CHAR と NCHAR は同じ型であり、どちらも NCHAR を使用します。したがって、2 つのメソッド定義は同じものと見なされます。

DS2 には、*システム*メソッドと*ユーザー定義*メソッドの 2 種類のメソッドがあります。

DS2 プログラムには、次の 3 つのシステムメソッドの少なくとも 1 つが含まれている必要があります。

```
method init();
  end;
method run();
  end;
method term();
  end;
```

これらのメソッドは、SAS DATA ステップの暗黙的なループの概念よりも構造化されたフレームワークを提供することを目的としています。Base SAS では、DATA ステッププログラム全体が暗黙的なループに含まれていました。DS2 では、暗黙的なループは RUN メソッドによって表され、INIT メソッドと TERM メソッドがそれぞれ初期化コードとファイナライズコードを提供します。

DS2 プログラムを実行した場合の結果を次に示します。

1 INIT メソッドが実行されます。初期化が行われます。

2 保持されていないプログラムデータベクトル内の変数は、適切な欠損値に設定されます。詳細については、"RETAIN ステートメント" (*SAS DS2 言語リファレンス*)を参照してください。

3 RUN メソッドが実行されます。

4 実行制御は、RUN メソッドの入力ステートメントのステータスに依存します。現在、DS2 の唯一の入力ステートメントは SET ステートメントです。RUN メソッドがこれらの条件のいずれかを満たす場合、処理はステップ 5 に進みます。それ以外の場合、RUN メソッドを再度実行できるように、処理はステップ 2 に進みます。

   - 入力ステートメントなし
   - 実行が完了した入力ステートメント

5 TERM メソッドが実行され、すべての最終ステートメントが実行されます。

INIT、RUN、および TERM メソッドは、パラメーターおよび戻り値なしで定義する必要があります。INIT、RUN、または TERM メソッドにパラメーターを指定すると、エラーが発生します。

DS2 プログラムで OUTPUT ステートメントを指定しない場合、DS2 コンパイラは、RUN メソッドの最後に実行されるパラメーターなしのステートメントを提供します。RUN メソッドを記述していない場合は、DS2 コンパイラによって暗黙的に作成されます。したがって、最小の DS2 プログラム形式は、次のようにデータプログラム内の単一の宣言です。

```
data;
dcl double x;
enddata;
```

このコードは、次の 2 つのコードセットと機能的に同等です。

```
data;
dcl double x;
method run();
end;
enddata;
```

```
data;
dcl double x;
method run();
  output;
end;
enddata;
```

各コードセットは、出力ウィンドウに単一の初期化されていない x の値を生成します。

DS2 プログラムから INIT、RUN、または TERM メソッドを直接呼び出そうとすると、エラーが発生します。

ユーザー定義メソッドは、1 回以上実行させるステートメントを METHOD ステートメントと END ステートメントで囲むことで作成できます。ユーザー定義メソッドの詳細については、“METHOD ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

注: PROC DS2 を使用する場合、DS2 プログラムは RUN ステートメントで区切られます。RUN ステートメントの後に追加の DS2 コードが見つかった場合、このコードは、前の RUN ステートメントの前の DS2 プログラムとは別の新しい DS2 プログラムを構成します。

DS2 識別子のスコープ

プログラミングブロック

プログラミングブロックは、順序付けられたキーワードのペアで始まり、終了するコードのセクションです。次のキーワードは、プログラミングブロックを作成します。

- DATA...ENDDATA
- PACKAGE...ENDPACKAGE
- THREAD...ENDTHREAD
- DO...END
- METHOD...END

このドキュメントでは、これらの用語はプログラミングブロックに使用されます。

- データプログラミングブロックまたは**データプログラム**は、DATA...ENDDATA ステートメントで囲まれたコードを指します。
- パッケージプログラミングブロックまたは**パッケージ**は、PACKAGE... ENDPACKAGE ステートメントによって囲まれた変数およびメソッドの保存されたライブラリを指します。パッケージの変数とメソッドは、DS2 プログラム、スレッド、またはその他のパッケージで使用できます。
- スレッドプログラミングブロックまたは**スレッドプログラム**は、THREAD... ENDTHREAD ステートメントで囲まれたストアドプログラムを指します。スレッドプログラムは、DS2 プログラムまたはパッケージの SET FROM ステートメントによって呼び出すことができます。
- DO プログラミングブロックまたは **DO ループ**は、DO ステートメントと END ステートメントで囲まれたプログラミングステートメントのサブブロックを指します。
- メソッドプログラミングブロックまたは**メソッドブロック**は、METHOD ステートメントと END ステートメントで囲まれたプログラミングステートメントのサブブロックを指します。

一部のブロックはネストできます。この例では、DATA および ENDDATA ステートメントによって定義された 1 つのデータプログラムと、3 つのメソッドステートメントによって定義された 3 つのネストされたメソッドブロックがあります。

```
data _null_;
  declare int x;

  method init();
    declare double d;
  end;

  method run();
  end;

  method term();
  end;
enddata;
```

最も外側のプログラミングブロックで宣言された変数は、**グローバル**変数、または**グローバルスコープ**を持つ変数と呼ばれます。ネストされたブロックで宣言された変数は、**ローカル**変数、またはそのブロックに対してローカルなスコープを持つ変数と呼ばれます。DS2 は、宣言されていない変数にもグローバルスコープを割り当てます。前の例では、X はグローバル変数であり、D はネストされた INIT メソッドに対してローカルな変数です。

注: DS2 プログラムには、複数のサブプログラムとそれに続くオプションのデータプログラムを含めることができます。次の制限が適用されます。

- データプログラムは 1 つしか存在できず、データプログラムは最後のサブプログラムである必要があります。
- ENDPACKAGE、ENDTHREAD、または ENDDATA ステートメントは、DS2 プログラムの最後のサブプログラムではオプションですが、適切なプログラミング形式にすることをお勧めします。これらのステートメントは、他のすべてのサブプログラムでは必須です。

変数ルックアップ

変数が参照されると、DS2 は参照のブロックで始まる変数の宣言を常に検索します。次に、そこで見つからない場合は、外側のブロックまたはプログラムを連続して検索します。この例では、INIT メソッドでの X への参照は、X のグローバル宣言を参照しています。

```
declare int x;

method init();
end;
```

メソッドはブロックであるため、宣言自体を含めることができます。この例では、INIT メソッドでの X への参照は X のローカル宣言を参照していますが、RUN メソッドでの X への参照は X のグローバル宣言を参照しています。

```
declare int x;

method init();
  declare int x;
end;

method run();
end;
```

スコープの定義

スコープは、識別子の属性と見なすことができます。識別子は、メソッド名、関数、データ名、ラベル、プログラム変数など、多くのプログラムエンティティを 参照できます。このセクションではプログラム変数を例として使用しますが、すべての識別子はスコーピングルールに従います。

スコープは、変数にアクセスできるプログラム内の場所を表します。グローバル変数にはグローバルスコープがあり、プログラムのどこからでもアクセスできます。ローカル変数にはローカルスコープがあり、変数が宣言されたプログラムまたはブロック内からのみアクセスできます。

DS2 では、プログラムの実行が変数のスコープ内で行われている場合にのみ、変数にアクセスできます。つまり、変数のスコープ内のステートメントがアクティブに実行されている場合にのみ、変数の値にアクセスできます。

```
data;
  declare int x;    /* global x in global scope */

  method init();
    x = 5;         /* global x assigned 5 */
  end;

  method run();
  end;

  method term();
  end;

enddata;
```

特定のスコープ内の各変数には一意の名前を付ける必要がありますが、異なるスコープ内の変数に同じ名前を付けることができます。スコープがネストされている場合、外側のスコープの変数が内側のスコープの変数と同じ名前を持つ場合、外側のスコープ内の変数は内側のスコープ内の変数によって隠されます。たとえば、次のプログラムでは、2 つの異なる変数が同じ名前 X を共有しています。グローバル変数 X にはグローバルスコープがあり、ローカル変数 X にはローカルスコープがあります。メソッド INIT のローカルスコープ内では、ローカル変数 X はグローバル変数 X を隠します。したがって、割り当てステートメントはローカル変数 X に 5 を割り当てます。

```
data;
   declare int x;    /* global x in global scope */

   method init();
     declare int x;  /* local x in local scope */

     x = 5;         /* local x assigned 5 */
   end;

   method run();
   end;

   method term();
   end;

enddata;
```

変数有効期間

変数の有効期間は、変数が存在する期間です。グローバル変数は、プログラムの期間中存在します。ローカル変数は、変数が宣言されたブロックの期間中存在します。グローバル変数の値は、そのグローバル変数が現在のプログラムブロックの RETAIN ステートメント内に出現しない限り、RUN メソッドに入る前に欠損値または null 値に設定されます。

次の例では、変数 X は INIT メソッドの実行中にのみ存在し、変数 Y はデータプログラムの実行中に存在します。

```
data;
  declare double y;

  method init();
    declare double x;
  end;

enddata;
```

変数の有効期間中、次の例のように、同名のローカルで宣言された変数によって重要性が低下する可能性があります。

```
declare int x;

method init();
  declare int x;
end;

method run();
end;
```

グローバル変数 X にはプログラム全体の有効期間がありますが、この例では、INIT メソッドで X がローカルに宣言されているため、INIT メソッドから直接アクセスすることはできません。

3

DS2 変数

DS2 変数の概要

DS2 プログラム変数の特性は、名前、スコープ、およびデータ型を持つことです。

名前または識別子は、変数に与えられる 1 つ以上のトークンまたはシンボルです。名前については、6 章, “DS2 識別子” (57 ページ)で説明されています。

変数は、変数が宣言されている場所に応じて、グローバルスコープまたはローカルスコープを持つことができます。詳細については、“変数スコープ” (34 ページ)を参照してください。

変数のデータ型は、宣言方法に応じて明示的または暗黙的に割り当てられます。詳細については、“変数宣言” (24 ページ)を参照してください。

注: "データ型"という用語には、精度、文字セットエンコーディング、長さなどのデータ型属性が含まれます。データ型の詳細については、5 章, “DS2 データ型” (47 ページ)を参照してください。

注: このセクションのコード例の DS2 ステートメントは、PROC DS2 を使用して SAS に送信されます。

変数宣言

変数を宣言する方法

変数とそのデータ型を宣言するには、次の 3 つの方法があります。

- DECLARE ステートメントを使用した明示的な宣言

  DECLARE ステートメントは、データ型を変数リストまたは配列内の各変数に関連付けます。メソッド外で DECLARE ステートメントを使用すると、グローバル変数が作成されます。メソッド内で DECLARE ステートメントを使用すると、ローカル変数が作成されます。メソッド内では、DECLARE ステートメントはメソッドステートメントの前に置く必要があります。それ以外の場合は、エラーが発生します。

  明示的に宣言された変数には、**Variable** 属性が割り当てられます。

  3 つのグローバル値を明示的に宣言し、それらに値を割り当てるコード例を次に示します。

```
proc ds2;
data input1 (overwrite=yes);
  dcl double x y z;
  method init();
    x=1; y=2; z=3;
    output;
  end;
enddata;
run;
quit;
```

  詳細については、“変数スコープ” (34 ページ)および“DECLARE ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

- SET または MERGE ステートメントを使用した暗黙的な宣言

SET または MERGE ステートメントは、指定された各テーブルの列情報を読み取ります。各テーブルの各列に対して、SET または MERGE ステートメントは、列と同じデータ型を持つグローバル変数を DS2 プログラムに作成します。

暗黙的に宣言された変数には、**Label** 属性が割り当てられます。

SET ステートメントを使用して変数を暗黙的に宣言するコード例を次に示します。

```
proc ds2;
data results (overwrite=yes);
  method run();
    set input1;
  end;
 enddata;
 run;
 quit;
```

PROC COMPARE からの次の出力は、テーブル Input1 の列に variable 属性があり、テーブル Results の列に label 属性があることを示しています。

```
proc compare data=input1 compare=results error note criterion=1e-4;
run;
```

図 3.1 PROC COMPARE 出力

```
Listing of Common Variables with Differing Attributes

    Variable  Dataset       Type  Length  Label

    x         WORK.INPUT1   Num        8
              WORK.RESULTS  Num        8  x
    y         WORK.INPUT1   Num        8
              WORK.RESULTS  Num        8  y
    z         WORK.INPUT1   Num        8
              WORK.RESULTS  Num        8  z
```

詳細については、“SET ステートメントを使用したデータの読み取り” (250 ページ)、“MERGE ステートメント” (*SAS DS2 言語リファレンス*)、および“SET ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

- プログラミングブロックで宣言されていない変数を使用した暗黙的な宣言

注: すべての変数を明示的に宣言することを強くお勧めします。そうすることで、データソース間のデータ型の不一致を回避できます。

変数を宣言せずに使用すると、DS2 は変数の属性を次のように割り当てます。

- 割り当てステートメントの左側にある宣言されていない変数のデータ型は、割り当てステートメントの右側にある値のデータ型によって決定されます。詳細は次のとおりです。
  - 割り当てステートメントの右側の値のデータ型が数値(DOUBLE、BIGINT、DECIMAL など)、ANSI NULL、または SAS 数値欠損値である場合、DOUBLE 型が左側の変数に割り当てられます。

- 割り当てステートメントの右側の値のデータ型が文字(CHAR、VARCHAR、NCHAR など)または SAS 文字欠損値である場合、固定長文字が左側の変数に割り当てられます。
- 割り当てステートメントの右側の値のデータ型がバイナリ(BINARY、VARBINARY)の場合、左側の変数には固定長バイナリが割り当てられます。
- 割り当てステートメントの右側の値のデータ型が DATE、TIME、または TIMESTAMP の場合、左側の変数には同じ型が割り当てられます。

□ 文字変数の長さは、右側の値が SAS 文字欠損値でない限り、右側の値の長さです。SAS 文字欠損値の長さは 200 文字です。

割り当てステートメントの右側が式の場合、左側の変数の長さは右側の式の結果の長さになります。この例では、変数 b の長さは 9 です。これは、x と y の文字列の長さを加算した結果です。

```
x='abc';
y='rstuvw';
b=x || y;
```

□ 割り当てステートメントの右側にある宣言されていない変数のデータ型は DOUBLE です。

□ 暗黙的に宣言された文字型には、変数の文字セットのセッションエンコーディングが割り当てられます。

デフォルトでは、宣言されていない変数が作成されると、SAS ログに警告が送信されます。警告は、宣言されていない変数に割り当てられているデータ型、長さ、および必要に応じて精度を示します。

> ヒント　DS2SCOND システムオプションまたは PROC DS2 SCOND オプションを使用して、警告のかわりにエラーまたは注をログに発行できます。オプションを使用すると、宣言されていない変数についてレポートしないように指定することもできます。

割り当てステートメントの詳細については、“DS2 による宣言されていない変数の処理方法の制御” (27 ページ)を参照してください。

次の例は、一部の変数を明示的に定義し、一部の変数を暗黙的に定義する方法を示しています。FIRSTNAME と LASTNAME は、データ型が CHAR(20)の変数として明示的に宣言されています。PNUM は割り当てによって暗黙的に宣言されます。ログに書き込まれた警告からわかるように、DS2 は割り当ての右側の値(CHAR(11))に基づいて PNUM にデータ型を割り当てます。

```
proc ds2;
data phonenums;
  dcl char(20) firstName lastName;
  method init();
    firstName='Sam';
    lastName='Alesski';
    pnum = '19192223454';
  end;
enddata;
run;
quit;
```

```
WARNING: Line 57: No DECLARE for assigned-to variable pnum; creating it as a
  global variable of type char(11).
```

割り当てと式を使用する場合は、型変換と欠損値に注意する必要があります。次の例では、等値オペランドが DOUBLE オペランドと VARCHAR オペランドで評価されます。DOUBLE オペランドは VARCHAR オペランドよりも優先されるため、VARCHAR オペランドは DOUBLE に変換されます。文字列'on1'は有効な数値ではないため、変換の結果、右側のオペランドの SAS 欠損値が数値になります。変数 **in2** はプログラムによって初期化されていないため、**in2** も SAS 欠損値に設定されます。SAS モードでは、DOUBLE データ型の missing = missing は真と評価されます。

```
proc ds2;
data _null_;
  dcl varchar(100) "results";
  dcl varchar(100) "in1";

  method run();
    "in1"='onlystring2';

    /* Mistype in2 instead of in1 */
    if ("in2"='onl') then do;
      "results" = 'Rule is fired AGAIN';
    end;
    put 'here are the results';
    put "results";
    end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
here are the results
 Rule is fired AGAIN
 WARNING: Line 106: No DECLARE for referenced variable in2; creating it as a global variable of type double.
```

DS2 による宣言されていない変数の処理方法の制御

DS2 が宣言されていない変数を処理する方法を制御するには、DS2SCOND システムオプションまたは DS2 プロシジャの SCOND オプションを使用します。

表 3.1　変数宣言を制御するための設定

DS2SCOND/ SCOND 設定	変数宣言への影響
WARNING	割り当てによる宣言が発生します。警告メッセージが SAS ログに書き込まれます。これがデフォルトの動作です。

DS2SCOND/ SCOND 設定	変数宣言への影響
NONE	割り当てによる宣言が発生します。SAS ログにメッセージは書き込まれません。
NOTE	割り当てによる宣言が発生します。SAS ログに注が書き込まれます。
ERROR	割り当てによる宣言は発生しません。SAS ログにエラーメッセージが書き込まれます。これは、変数宣言の strict モードとも呼ばれます。

詳細については、“DS2SCOND= システムオプション” (*SAS DS2 言語リファレンス*)および“DS2 プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。

DS2 変数リスト

変数リストの概要

DS2 には、変数のリストを操作する方法が 2 つあります。

- DS2 は、省略構文を使用した、類似した名前を持つか、または類似した型の複数のグローバル変数の選択をサポートします。この省略構文は、その構文で変数名のリストを指定する必要があるステートメント(DROP や KEEP など)で役立ちます。詳細については、“変数名を指定するための省略構文”を参照してください。
- DS2 は、変数をグループ化するためのデータ構造をサポートしています。DS2 は、同種データ型のグローバル変数をグループ化するための配列をサポートしています。DS2 は、異種データ型のグローバル変数をグループ化するための varlist をサポートしています。これらのデータ構造により、入力として変数のグループを必要とする DS2 メソッド、パッケージ、および関数に変数グループ化をサブミットできます。データ構造を使用すると、グループ化の要素のデータ値を設定、取得、および渡すことができます。詳細については、11 章, “DS2 配列” (121 ページ)および 12 章, “DS2 varlist” (145 ページ)を参照してください。

変数名を指定するための省略構文

注: 省略された変数選択構文は、グローバル変数から選択します。省略形は、DOT 演算子でアクセスされるローカル変数、メソッドパラメーター変数、またはパッケージ変数を参照しません。

多くの DS2 ステートメントとテーブルオプションは、複数の変数名を指定する省略構文をサポートしています。たとえば、KEEP、DROP、VARARRAY、VARLIST ステートメントでは、省略形を使用して、保持、削除、または参照する変数の名前を指定できます。省略構文の例を次に示します。

```
keep name address city state zip phone;
 drop rep1-rep5;
 vararray int grades[*] assignment: quiz: exam:;
```

DS2 は、次の省略リストをサポートしています。

- 名前変数リスト
- 番号付き範囲変数リスト
- 名前範囲変数リスト
- 名前接頭辞変数リスト
- 型変数リスト
- 特殊名変数リスト

1 つの変数リストの中に、異なる形式の省略変数リストを混在させることができます。例えば、次のような変数リストが有効です。

```
u x1-x3 u:
```

この変数リストでは:

- u は、u という名前の単一の変数を参照する名前リストです。
- x1-x3 は、x1、x2、および x3 という名前の変数を参照する番号付き範囲リストです。
- u:は、名前が文字列"u"で始まるすべての 変数を参照する名前接頭辞リストです。したがって、 u:は、u、unit、user、uniform を指します。

上述の変数リストは u x1 x2 x3 u unit user uniform に展開されます。変数リストの展開では、1 つの変数を複数回参照できることに注意してください。

詳細については、“省略変数リストの種類” (30 ページ)を参照してください。

varlist は、KEEP、DROP、VARARRAY、VARLIST ステートメントでは指定できません。

省略変数リストの種類

表 3.2　省略変数リストの種類のまとめ

種類	定義	例
名前リスト	グローバル変数名のリスト。	drop location pressure temperature;
番号付き範囲リスト	同じ名前の接頭辞が付いているが、最後の文字の数値が異なるグローバル変数で構成されるリスト。値は連続している必要があります。	keep x1-x5; これは、範囲として指定されていない次の番号付きリストと同等です。 keep x1 x2 x3 x4 x5;
名前範囲リスト	変数定義の順序に依存する名前範囲リスト。	keep sales_jan--sales_mar; すべての列をプログラムデータベクトルの順序どおりに指定します(keep sales_jan から sales_mar まで)。
名前接頭辞リスト	指定した文字列で始まるすべてのグローバル変数を参照するリスト。	keep sales:; sales_jan、sales_feb、sales_mar など、名前が"sales"で始まるすべての変数に展開されます。
型リスト	指定された DS2 データ型のすべてのグローバル変数を含むリスト。複数の型を指定できます。詳細については、“型変数リスト” (31 ページ)を参照してください。	keep double; DOUBLE 型のすべてのグローバル変数を DATA ブロックに含めるように展開します。 varlist a [smallint int]; SMALLINT および INT 型のすべてのグローバル変数を DATA ブロックに含めるように展開します。
特殊名リスト	読み取り専用のグローバル変数の特定のグループを参照するために展開されるリスト。_ALL_ 変数がサポートされています。_ALL_は、DS2 プログラム内のすべてのグローバル変数を参照します。	varlist x [_all_]; SET ステートメントで指定されたテーブルの変数を含む、DS2 プログラムのすべてのグローバル変数を varlist に含め、グループ化に名前 x を割り当てます。

種類	定義	例
	詳細については、“事前定義された DS2 変数” (36 ページ)を参照してください。	

注: 名前範囲リストでは、2 つのハイフン(- -)を使用して変数間の範囲を指定し、番号付き範囲リストでは単一のハイフン(-)を使用して範囲を指定することに注意してください。

型変数リスト

型変数リストは展開され、指定された型のすべてのグローバル変数を参照します。型変数リストの構文は次のとおりです。

data-type

型変数リストでサポートされている型は次のとおりです。

- BIGINT
- BINARY (BINARY および VARBINARY に一致)
- CHAR (CHAR、VARCHAR、および CHARACTER に一致)
- CHARACTER (CHAR、VARCHAR、および CHARACTER に一致)
- DATE
- DECIMAL
- DOUBLE (DOUBLE および FLOAT に一致)
- FLOAT (DOUBLE および FLOAT に一致)
- INTEGER
- NCHAR (NCHAR および NVARCHAR に一致)
- NUMERIC
- REAL
- SMALLINT
- TIME
- TIMESTAMP
- TINYINT

型変数リストを宣言するコードの例を次に示します。

```
proc ds2;
 data _null_;
 dcl double x y z;
 vararray double a[*] double;
```

```
 ...
 enddata;
 run;
quit;
```

DOUBLE 型のすべてのグローバル変数(x、y、および z)は、変数配列 a に含まれます。

VARARRAY ステートメントと VARLIST ステートメントで型リストを指定する方法の例を次に示します。

```
vararray decimal(8,4) va[*] decimal;
varlist vl[numeric];
```

例: 省略変数リストの混在

このプログラム例では、KEEP ステートメントで型変数リスト(double)と名前変数リストを指定します。

```
proc ds2;
data OnlyDoubleAndC / overwrite=yes;
  dcl double x y;
  dcl char(10) a b c;
  keep double c;
  method init();
    x = 10; y = 20;
    a = 'apple'; b = 'banana'; c = 'cherry';
    output;
  end;
enddata;
run;
quit;

proc print data=OnlyDoubleAndC;
run;
```

SAS ログに書き込まれる出力は次のとおりです。

```
Obs    x    y     c
1     10   20    cherry
```

出力には、DOUBLE として定義された変数(x および y)が含まれます。さらに、名前変数 c が含まれます。

関数の引数として省略変数リストを指定する

OF 演算子を使用すると、変数リストを一部の関数の引数として指定できます。OF 演算子は、省略変数リストの指定を変数名のカンマ区切りリストに変換し、これが関数の引数になります。

注: OF 演算子の使用にはルールと制限があります。詳細については、“OF 演算子” (91 ページ)を参照してください。

名前接頭辞変数リストを引数として SUM 関数にサブミットする例を次に示します。

```
proc ds2;
data X (overwrite=yes);
  dcl double x A1 A2 A4 y;
  method run();
    A1 = 1.0;
    A4 = 3.0;
    y = 1;
    lab:
    if (y = 2) then
      X = SUM(OF A: );
      A2 = 2.0;
    if (y = 1) then
    do;
      y = 2;
    goto lab;
    end;
    put X=;
  end;
enddata;
run;
quit;
```

SAS ログに書き込まれる出力は次のとおりです。

```
x=6
```

この例では、OF 演算子は、名前が文字 A で始まるすべてのグローバル変数を計算に含めるよう SUM 関数に指示します。

省略変数リストの展開

省略変数リストの指定は、省略形に一致するグローバルスカラー変数を参照するように展開されます。ローカル変数が変数リスト展開に含まれることはありません。

次の例では、変数 **x1** はグローバル変数であり、変数 **x2** はメソッド INIT に対してローカルです。したがって、ローカル変数 **x2** はテーブルに書き込まれません。

```
proc ds2;
data example (overwrite=yes);
  declare double x1;
  keep x:;

  method init();
    declare double x2;
  end;
enddata;
run;
quit;
```

変数リストの展開では、宣言されていない変数が考慮されます。次の例では、KEEP ステートメントの_ALL_が展開され、グローバル変数 **x1**、**x2**、および **x3** を参照します。DS2 は、DECLARE ステートメントで指定されていないため、変数 **x3** をグローバル変数として作成します。したがって、変数 **x1**、**x2**、および **x3** はテーブル **example** に書き込まれます。

```
proc ds2;
data example (overwrite=yes);
  declare double x1;
  keep _all_;
  declare double x2;

  method init();
    x3=17;
  end;
enddata;
run;
quit;
```

変数スコープ

グローバル変数

グローバルスコープを持つ変数であるグローバル変数は、次の 3 つの方法のいずれかで宣言されます。DS2 プログラムの最も外側のプログラミングブロックで DECLARE ステートメントを使用する方法、プログラミングブロック内で SET ステートメントを使用して暗黙的に宣言する方法、またはプログラミングブロック内で宣言されていない変数を使用して暗黙的に宣言する方法です。グローバルスコープを持つ変数は、プログラムのどこからでもアクセスでき、プログラムの期間中存在します。グローバル変数は、任意のプログラムブロックの THIS 式で使用できます。DS2 変数の宣言については、“変数宣言” (24 ページ)を参照してください。THIS 式の使用については、“THIS 式” (97 ページ)を参照してください。

ローカル変数

ローカルスコープを持つ変数であるローカル変数は、DECLARE ステートメントを使用して、メソッドやパッケージなどの内側のプログラミングブロック内で宣言されます。ローカルスコープを持つ変数は、変数が宣言されている内側のプログラミングブロック内でのみ認識されます。詳細については、“プログラミングブロック” (17 ページ)を参照してください。

DS2 出力のグローバル変数とローカル変数

デフォルトでは、グローバル変数のみが出力に含まれます。プログラムループとインデックスに使用されるローカル変数は、出力から明示的に削除する必要はありません。ローカル変数は、RUN メソッドの暗黙的なループの反復など、メソッド呼び出しの開始時に常に作成され、各呼び出しの最後に破棄されます。したがって、RUN メソッドでローカル変数をアキュムレータ変数として使用することはお勧めしません。

すべてのグローバル変数は、プログラムデータベクトル(PDV)で名前が付けられます。PDV は、DS2 が行を書き込むときに出力テーブルに書き込まれる値のセットです。PDV の詳細については、“Processing a DATA Step: A Walk-Through” (*SAS Programmer’s Guide: Essentials*)を参照してください。

グローバル変数とローカル変数の例

次のプログラムは、グローバル変数(A、B、および TOTAL)とローカル変数(C)の両方を示しています。

```
proc ds2;
data;
  dcl int a;              /* 1 */
  method init();
    dcl int c;            /* 2  */
    a = 1;                /* 3 */
    b = 2;                /* 4 */
    c = a + b;
    this.total = a + b + c; /* 5 */
  end;
enddata;
run;
quit;
```

1 A は、最も外側の DS2 プログラムで宣言されているため、グローバル変数です。

2 C は、メソッドブロック METHOD INIT()内で宣言されているため、ローカル変数です。

3 A はグローバル変数であるため、メソッドブロック METHOD INIT()内で参照できます。

4 B は METHOD INIT()で宣言されていないため、デフォルトでグローバル変数になります。DS2 は B に DOUBLE のデータ型を割り当てます。B は PDV と出力テーブルに表示されます。

5 THIS.TOTAL は同時に、変数 TOTAL をデータ型 DOUBLE のグローバル変数として宣言し、A、B、および C の値に基づいて値を割り当てます。

事前定義された DS2 変数

事前定義されたメソッド変数

事前定義された変数は、メソッドブロック内で自動的に宣言され、メソッドが完了すると破棄される変数です。事前定義された変数の値は、RUN メソッドのある反復から次の反復まで保持されます。これらの変数はプログラムデータベクトルに追加されますが、作成中のテーブルには書き込まれません。事前定義された変数は一時的なものであり、データとともに保存されることはありません。事前定義された変数はメソッドブロックで使用でき、自分で宣言した変数と同じように使用できます。

2 つの事前定義された変数が作成されます。

_N
: 初期値は INIT メソッドによって 0 に設定されます。RUN メソッドが実行されるたびに、値は 1 ずつ増加します。_N のデータ型は BIGINT です。これは読み取り専用の変数です。_N に値を割り当てることはできません。

_N_
: 初期値は INIT メソッドによって 1 に設定されます。RUN メソッドが実行されるたびに、値は 1 ずつ増加します。つまり、RUN メソッドが初めて実行されるとき、_N_の値は 1 です。この値は、プログラムがメソッドをループした回数を示します。この変数は行をカウントするために使用できますが、反復ごとに 1 つのレコードが読み取られる場合にのみ、適切な行カウンターになります。_N_のデータ型は BIGINT です。_N_に数値を割り当てることはできますが、RUN メソッドの次の反復では、値はプログラムによって割り当てられた値に戻ります。

事前定義された変数は出力変数ではありませんが、PUT ステートメントに_ALL_引数を指定すると、事前定義された変数の値を SAS ログに出力できます。

次の単純な DS2 プログラムは、PUT _ALL_ステートメントを使用して、_N および _N_変数の値を SAS ログに出力する方法を示しています。

```
proc ds2;
data inp /overwrite=yes;
  dcl double a;
  method init();
    a = 1; output;
  end;
enddata; run;

data;
  method init();
   put 'init' _n=; put _ALL_;
  end;
  method run();
   set inp;
   put 'run' _n=; put _ALL_;
```

```
  end;
  method term();
    put 'term' _n=; put _ALL_;
  end;
enddata;
run;
quit;
```

例のコード 3.1　_N および_N_変数値を含む SAS ログ

```
  .
  .
  .
NOTE: Execution succeeded. One row affected.
init _n=0
a=. _n_=1
run _n=1
a=1 _n_=1
term _n=2
a=1 _n_=2
```

DS2 スレッド変数

次の自動変数は、DS2 スレッド間で問題をサブセット化するために使用されます。これらの自動変数は、PUT ステートメントを使用してデバッグするときにコンテキストを提供するのにも役立ちます。

- _HOSTNAME_
- _LOCALNTHREADS_
- _LOCALTHREADID_
- _NTHREADS_
- _THREADID_

詳細については、“DS2 スレッドで役立つ自動変数” (245 ページ)を参照してください。

4

DS2 定数

定数の定義

定数は、固定値を示す数値、文字列、2 進数、日付、時間、タイムスタンプのいずれかです。DS2 定数の例を次に示します。

```
107
'Trends in Business'
date '2008-01-01'
b'10011001'
x'FFE3546F'
```

数値定数

数値定数は、整数定数または小数定数のいずれかである負または正の数値です。

整数定数

小数部分のない数値、つまり整数です。整数定数値は正確な数値として保存されます。N または n 接尾辞のない整数定数は、BIGINT、INTEGER、SMALLINT、TINYINT のいずれかのデータ型の値です。N または n 接尾辞が付いた整数定数は、DECIMAL または NUMERIC データ型の値です。整数定数の形式は次のとおりです。ここで、*integer* は 0 から 9 までの 1 つ以上の数字のシーケンスです。

integer

*integer***N**|**n**

次の値は、有効な整数定数です。

```
0
124
124N
+124n
-124n
```

小数定数

小数部分を持つ数値です。小数定数値は、正確な数値または近似数値として保存されます。N または n 接尾辞のない小数定数は、近似数値として保存される REAL または DOUBLE データ型の値です。N または n 接尾辞が付いた小数定数は、データ型の最大精度までの正確な数値として保存される DECIMAL または NUMERIC データ型の値です。近似分数定数は、標準表記または指数(E)表記をサポートします。正確な小数定数は、標準表記のみをサポートします。小数定数の形式は次のとおりです。ここで、*integer*、*fraction*、および *exponent* は、0 から 9 までの 1 つ以上の数字のシーケンスです。

*integer***.**

*integer***.N**|**n**

*integer***.***fraction*

.*fraction*

*integer***.***fraction***E**|**e**[+|-]*exponent*

*integer***E**|**e**[+|-]*exponent*

.*fraction***E**|**e**[+|-]*exponent*

*integer***.***fraction***N**|**n**

.*fraction***N**|**n**

次の値は、有効な小数定数です。

```
0.
124.0
+124.0
-57.33
.5
1E100
99.99e4
-.33e-2
-57.33n
12345678901234567890123456789O.12N
```

文字定数

文字定数の概要

文字定数は、単一引用符で囲まれた一連の文字であり、次の形式を使用して記述できます。

'*character-string*'

n'*character-string*'

各国文字を含む文字列の場合は、NCHAR 定数 **n**'*character-string*'を使用してください。各国文字は、複数バイトのストレージを使用できます。

文字定数と NCHAR 定数には、複数行にわたる文字定数の改行を含めることができます。

注: 二重引用符で囲まれた一連の文字は文字定数ではなく、識別子です。詳細については、“区切り識別子” (58 ページ)を参照してください。

有効な文字定数は次のとおりです。

```
'PHONE'
n'STÄDTE'
'Phone
Number'  /* constant that spans more than one line */
```

注: 文字列に単一引用符(')またはアポストロフィ(')を含めるには、追加の引用符を使用します。次に例を示します。

```
'Isn''t life beautiful'
```

長い文字定数

長い文字列定数(NCHAR、NVARCHAR、および VARCHAR データ型)は、割り当て時に二重角かっこ([[ ]])で囲むことができます。これは、LUA プログラミング言語が文字列を区切る方法に似ています。

長い文字定数は、[[、[=[、[==[など(つまり、0 個以上の等号が間にある 2 つの左角かっこ)で始まります。長い文字定数は、一致する]]、]=]、]==]など(つまり、同じ数の等号が間にある 2 つの右角かっこ)で終わります。終了トークンは、開始トークンの

パターンと一致する必要があります。たとえば、[=[は]]または]==]では閉じません。一致する]=]トークンでのみ閉じます。次に例を示します。

```
proc ds2;
data;
  dcl nvarchar(1000) cc;
  method init();
    cc = [====[
      proc ds2;
      data;
      dcl nchar(42) cc;
      method init();
      cc = [=[
      this is a test
      ]=];
      put '|' cc= '|';
      end;
      enddata;
    ]====];
    put '|' cc= '|';
  end;
enddata;
run;
quit;
```

アウトプット 4.1　定数角かっこの例からの出力

cc
proc ds2; data; dcl nchar(42) cc; method init(); cc = [=[this is a test]=]; put '|' cc= '|'; end; enddata;

注: 長い文字定数には改行が含まれる場合がありますが、改行は必須ではありません。これは、プログラムの動作に影響を与えずに n'xyzzy'を[==[xyzzy]==]に置き換えることができることを意味します。

注意
二重角かっこ内のコメントは破棄されます。

注意
角かっこで囲まれた長い文字定数の中に"run;"または"quit;"を入れないでください。 角かっこで囲まれた長い文字定数の中に"run;"または"quit;"を入れると、コンパイルエラーが発生し、プログラムが失敗します。

DS2 定数の文字エンコーディング

すべての DS2 文字データには、文字エンコーディングが関連付けられています。文字定数についても同様です。DS2 文字定数値は、DS2 セッションエンコーディング

または DS2 国別エンコーディングのいずれかでエンコードされます。各国文字変数には、DS2 セッションエンコーディングに関係なく、文字エンコーディングとして常に UTF-8 が割り当てられます。

DS2 セッションエンコーディングを使用してエンコードされる文字定数を指定する場合、DS2 プログラムテキストで定数の接頭辞を指定しないでください。

```
'Hello World'
'你好世界'
```

DS2 各国文字エンコーディングを使用してエンコードされる文字定数を指定するには、DS2 プログラムテキストで定数の前に n を付けます。

```
n'Hello World'
n'你好世界'
```

DS2 プログラムテキストのエンコーディングは、DS2 プログラムの処理中に文字定数に割り当てられたエンコーディングとは異なる場合があります。たとえば、DS2 プログラムテキストの文字エンコーディングが UTF-8 で、DS2 セッションエンコーディングが latin-1 の場合、次の PUT ステートメントの文字定数は、プログラムの処理中にトランスコードに失敗します。

```
put '你好世界';
```

文字定数は各国文字定数として指定されていないため、文字定数には DS2 セッションエンコーディングの latin-1 が割り当てられます。latin-1 文字セットは、漢字'你好世界'をサポートしていません。したがって、UTF-8 (DS2 プログラムテキストエンコーディング)から latin-1 (DS2 セッションエンコーディング)へのトランスコードは失敗します。トランスコードの失敗は、DS2 セッションエンコーディングを漢字をサポートするエンコーディング(EUC-CN や UTF-8 など)に変更するか、DS2 プログラムテキストで n を前に付けて DS2 各国文字エンコーディングを使用してエンコードする文字定数を指定することで解消できます。

文字定数でのマクロ変数の参照

文字定数でマクロ変数を参照するには、SAS マクロ関数%TSLIT を使用します。これにより、定数値を単一引用符で囲む必要がなくなります。たとえば、次のステートメントでは%TSLIT 関数を使用してマクロ変数&profit を参照します。

```
put %tslit(PROFIT: &profit);
```

%TSLIT マクロ関数は、デフォルトの自動呼び出しマクロライブラリに格納されています。構文は次のとおりです。

%TSLIT(*literal-text*);

%TSLIT マクロ関数は、リテラルテキスト引数でマクロ変数を参照し、結果のテキスト値を単一引用符で囲みます。これにより、DS2 文字定数値が作成されます。

2 進定数

2 進定数は、次の形式を使用して、ビット文字列または 16 進文字列として記述できます。

b'*bit-string*'

x'*hexadecimal-string*'

bit-string は、0 と 1 の 2 進数のシーケンスです。

hexadecimal-string は、0 - 9 および A - F の 16 進文字の文字列です。

bit-string と *hexadecimal-string* の両方を単一引用符で囲む必要があります。

DS2 は、DS2 プログラムで使用できるように、ビット定数と 16 進定数の両方を同等の 2 進数に変換します。2 進定数と 16 進定数は、バイト境界までゼロで埋められます。

2 進定数の例を次に示します。

```
b'11100011'
x'FF143E99'
```

16 進定数を文字変数に割り当てる場合、割り当てを有効にするには、16 進定数がセッションエンコーディングと同じエンコーディングである必要があります。テキスト値は、文字エンコーディングを使用して列のデータ値にエンコードされます。特定のテキスト値を格納するために使用されるバイトは、エンコーディングによって異なります。たとえば、UTF-8 および Latin-1 文字エンコーディングでの文字列 ágê の 16 進表現を考えてみましょう。

表 4.1　文字列 ágê の UTF-8 および Latin-1 エンコーディング

文字エンコーディング	16 進数のテキストデータ
UTF-8	C3 A1 67 C3 AA
Latin-1	E1 67 EA

UTF-8 セッションで Latin-1 エンコーディングの 16 進値を指定すると、エラーが発生するか、単に正しくない結果になる可能性があります。UTF-8 セッションに適切な 16 進値をサブミットする DS2 コード例のログ出力を次に示します。

```
1        proc ds2;
2          data _null_;
3            method init();
4            dcl varchar(100) s1;
5            s1 = x'C3A167C3AA';
6            put s1=;
7            end;
8          enddata;
9        run;
s1=ágê
NOTE: Execution succeeded. No rows affected.
quit;
```

注: 入力値から空白スペースを削除する必要があります。

DS2 には、数値定数の 16 進表記がありません。

日付と時間の定数

日付と時間の定数は、次の形式を使用して、日付、時間、またはタイムスタンプの文字列として記述できます。

date '*yyyy-mm-dd*'

time '*hh*:*mm*:*ss*'

time '*hh*:*mm*:*ss*.*fraction*'

timestamp '*yyyy-mm-dd hh*:*mm*:*ss*'

timestamp '*yyyy-mm-dd hh*:*mm*:*ss*.*fraction*'

yyyy は 4 桁の年です。

mm は 2 桁の月です。

dd は 2 桁の日です。

hh は 2 桁の時間です。

mm は 2 桁の分です。

ss は 2 桁の秒です。

fraction は、秒の端数を表す一連の数値です。

日付と時間の定数はすべて、単一引用符で囲む必要があります。年、月、日はハイフンで区切る必要があります。時、分、秒はコロンで区切る必要があります。秒と秒の端数はピリオドで区切る必要があります。タイムスタンプ定数では、日付と時間をスペースで区切ります。

日付定数の長さは、年、月、日を区切るハイフンを含めて 10 文字にする必要があります。

端数を除いた時間定数の長さは、時、分、および秒を区切るコロンを含めて 8 文字にする必要があります。端数を含む時間定数の長さは、秒の端数によって変化する可能性があります。

端数を除いたタイムスタンプ定数の長さは、日付と時間の構成要素を区切るハイフンとコロンを含めて 19 文字にする必要があります。端数を含むタイムスタンプの長さは、秒の端数によって変化する可能性があります。

日付と時間の定数の例を次に示します。

```
date '2008-01-01'
time '11:59:59'
timestamp '2007-09-30 02:33:31.59'
```

定数リスト

定数リストは、単純な配列割り当ておよび IN 式で使用できる、カンマまたは空白で区切られた、かっこで囲まれた一連の定数値です。定数リストには、ネストされた定数リストを含めることもできます。定数リストの例は(2, '33', –5, (1, '3'))です。この例では、(1, '3')はネストされた定数リストです。

各定数値またはネストされたリストには、反復子を接頭辞として付けることができます。反復子とは、数字の後にアスタリスク(*)を付けたものです。反復子は、アスタリスクに続く定数または定数リストを繰り返す回数を示します。たとえば、定数リスト(2, '33', 3*–5, 2*(1, '3'))では、リスト項目 3* –5'の結果は–5, –5, —5、およびリスト項目 2*(1,'3')の結果は 1, '3', 1, '3'になります。反復子"5*"を使用することと、リスト要素を 5 回繰り返すことに違いはありません。

注: IN 式で値が重複すると処理が遅くなる可能性があるため、反復子が IN 式で役立つ可能性はほとんどありません。

5

DS2 データ型

データ型とは

データ型は、列が格納するデータの型を指定するすべての列の属性です。データ型は、データの一部を文字列、整数、浮動小数点数、日付または時間として識別する特性です。データ型は、列の値に割り当てるメモリの量も決定します。

次の表に、DS2 でサポートされている一連のデータ型を示します。各データソースのテーブルストレージにすべてのデータ 型を使用できるわけではないことに注意してください。

表 5.1　DS2 データ型

データ型	説明
BIGINT	19 桁の精度で大きな符号付きの正確な整数を格納します。整数の範囲は-9,223,372,036,854,775,807 から 9,223,372,036,854,775,807 までです。整数データ型は小数値を保存しません。小数部分は破棄されます。
BINARY(*n*)	固定長のバイナリデータを格納します。ここで、*n* は格納する最大バイト数です。値の実際のサイズに関係なく、各値を格納するには最大バイト数が必要です。

データ型	説明
CHAR(*n*)	固定長の文字列を格納します。ここで、*n* は格納する最大文字数です。値の実際のサイズに関係なく、各値を格納するには最大文字数が必要です。**char(10)**が指定され、文字列の長さが 5 文字のみの場合、値の右側にスペースが 埋め込まれます。
DATE	カレンダー日付を格納します。日付リテラルは、*yyyy-mm-dd* の形式で指定されます。これは、4 桁の年(0001 - 9999)、2 桁の月(01 - 12)、および 2 桁の日(01 - 31)です。たとえば、1975 年 9 月 24 日の日付は **1975-09-24** と指定されます。 DS2 は、日付に関する ANSI SQL:1999 標準に準拠しています。ただし、すべてのデータソースが 日付の全範囲をサポートしているわけではありません。たとえば、0001-01-01 から 1582-12-31 までの日付は、SAS データセットまたは SPD データセットの有効な日付ではありません。
DECIMAL|NUMERIC(*p*,*s*)	ユーザー指定の精度とスケールで、符号付きの正確な固定小数点 10 進数を格納します。精度とスケールによって小数点の位置が決まります。精度は、小数点の左右に格納できる最大桁数で、範囲は 1 - 52 です。スケールは、小数点以下に格納できる最大桁数です。スケールは精度以下にする必要があります。たとえば、**decimal(9,2)**は、1234567.89 のように、2 桁の固定小数点小数部分を含む 9 桁までの 10 進数を格納します。
DOUBLE	符号付き近似倍精度浮動小数点数を格納します。大きな数値を許容し、小数点以下の桁数の精度を必要とする計算を可能にします。
FLOAT	符号付き近似倍精度浮動小数点数を格納します。FLOAT として定義されたデータは、DOUBLE と同じように扱われます。
INTEGER	10 桁の精度で通常サイズの符号付きの正確な整数を格納します。整数の範囲は-2,147,483,647 から 2,147,483,647 までです。整数データ型は小数値を保存しません。小数部分は破棄されます。 **注:** ゼロによる整数除算は、すべてのオペレーティングシステムで同じ結果になるわけではありません。ゼロによる整数除算は避けることをお勧めします。
NCHAR(*n*)	CHAR のような固定長の文字列を格納しますが、Unicode 各国語文字セットを使用します。*n* は、格納するマルチバイト文字の最大数です。プラットフォームに応じて、

データ型	説明
	Unicode 文字は 1 文字あたり 2 バイトまたは 4 バイトを使用し、すべての国際文字をサポートします。
NVARCHAR(*n*)	VARCHAR のような可変長の文字列を格納しますが、Unicode 各国語文字セットを使用します。*n* は、格納するマルチバイト文字の最大数です。プラットフォームに応じて、Unicode 文字は 1 文字あたり 2 バイトまたは 4 バイトを使用し、すべての国際文字をサポートできます。
REAL	符号付き近似単精度浮動小数点数を格納します。
SMALLINT	5 桁の精度で小さな符号付きの正確な整数を格納します。整数の範囲は-32,767 から 32,767 までです。整数データ型は小数値を保存しません。小数部分は破棄されます。
TIME(*p*)	時間値を格納します。時間リテラルは、*hh:mm:ss[.nnnnnnnnn]*の形式で指定します。これは、2 桁の時間 00 から 23、2 桁の分 00 から 59、2 桁の秒、およびオプションの小数値です。たとえば、午前 6:30 は **06:30:00** と指定されます。データソースでサポートされている場合、*p* パラメーターは秒の精度を指定します。*p* は、最大 9 桁の長さのオプションの小数値です。
TIMESTAMP(*p*)	日付と時間の両方の値を格納します。タイムスタンプリテラルは、*yyyy-mm-dd hh:mm:ss[.nnnnnnnnn]*の形式で指定します。これは、4 桁の年 0001 から 9999、2 桁の月 01 から 12、2 桁の日 01 から 31、2 桁の時間 00 から 23、2 桁の分 00 から 59、2 桁の秒、およびオプションの小数値です。たとえば、1975 年 9 月 24 日午前 6:30 という日付と時間は **1975-09-24 06:30:00** と指定されます。データソースでサポートされている場合、*p* パラメーターは秒の精度を指定します。*p* は、最大 9 桁の長さのオプションの小数値です。
TINYINT	3 の精度で非常に小さな符号付きの正確な整数を格納します。整数の範囲は-127 から 127 までです。整数データ型は小数値を保存しません。小数部分は破棄されます。
VARBINARY(*n*)	可変長のバイナリデータを格納します。ここで、*n* は格納する最大バイト数です。各値を格納するために最大バイト数は必要ありません。**varbinary(10)**が指定され、バイナリ文字列が 5 バイトのみを使用する場合、列には 5 バイトのみが格納されます。
VARCHAR(*n*)	可変長の文字列を格納します。ここで、*n* は格納する最大文字数です。各値を格納するために最大文字数は必要ありません。**varchar(10)**が指定され、文字列の長さが 5 文字のみの場合、列には 5 文字のみが格納されます。

次のプログラムは、DS2 で使用可能な多くのデータ型の宣言の例です。プログラムを正常に実行するには、これらのデータ型をサポートするデータソースに接続する必要があることに注意してください。

```
data ds2data (overwrite=yes);
   dcl int          v01;
   dcl bigint       v02;
   dcl smallint     v03;
   dcl tinyint      v04;
   dcl double       v05;
   dcl char(8)      v06;
   dcl varchar(8)   v07;
   dcl nchar(8)     v08;
   dcl nvarchar(8)  v09;
   dcl date         v10;
   dcl time         v11;
   dcl timestamp    v12;
   dcl binary(8)    v13;
   dcl varbinary(32) v14;
   dcl decimal(5,2)  v15;


   method init();
    v01 = 1;
    v02 = 1;
    v03 = 1;
    v04 = 1;
    v05 = 1.0;
    v06 = 'aa';
    v07 = 'aa';
    v08 = 'aa';
    v09 = 'aa';
    v10 = date'2001-01-01';
    v11 = time'01:01:01';
    v12 = timestamp'2001-01-01 01:01:01';
    v13 = x'0123456789ABCDEF';
    v14 = x'0123456789ABCDEF';
    v15 = 32.23n;
    output;
   end;
enddata;
run;
```

PROC FEDSQL DESCRIBE TABLE ステートメントを使用して、ds2data テーブルの内容の要約を作成できます。

```
CREATE TABLE DS2DATA (
    "v01"    INTEGER,
    "v02"    BIGINT,
    "v03"    INTEGER,
    "v04"    INTEGER,
    "v05"    DOUBLE PRECISION,
    "v06"    WCHAR(8),
    "v07"    WVARCHAR(8),
    "v08"    WCHAR(8),
    "v09"    WVARCHAR(8),
    "v10"    DATE,
    "v11"    TIME(0),
    "v12"    TIMESTAMP(4),
    "v13"    BINARY(8),
    "v14"    VARBINARY(32),
    "v15"    NUMERIC
);
```

データ型の特性

数値データ型

数値データ型には、数値が格納されます(数量や通貨の値など)。数値データ型の選択は、格納される数値の種類と数値の使用方法によって異なります。

数値データ型を選択する際に考慮すべき特性は次のとおりです。

- 正確な数値データ型を使用するか、近似数値データ型を使用するか。BIGINT、DECIMAL、INTEGER、NUMERIC、TINYINT などの正確な数値データ型は、値を正確に表します。REAL、DOUBLE、FLOAT などの近似数値データ型は、多数の数値に対して指定された正確な値を格納しません。多くのアプリケーションでは、正確な数値の動作が必要でない限り、指定された値と格納された近似値のわずかな違いは目立ちません。
- 整数または小数のどちらを格納するか。整数データ型の場合、小数値が挿入されると、小数部分が破棄されることに注意してください。たとえば、2.7 を挿入すると、格納される値は 2 になります。
- 小数点付きの数値を格納する方法。浮動小数点形式は、小数点が固定されていない数値です。浮動小数点データ型は、より長い 10 進値や、広範囲の数値をすばやく計算するために使用されます。浮動小数点データ型の場合、大きすぎる整数を入力するとエラーが発生します。浮動小数点表記を使用して値を入力します。固定小数点とは、小数点以下が 1 列に並ぶ数値の格納方法です。固定小数点データ型 DECIMAL は、通貨などの小数部分の正確な精度に使用されます。
- データ型の精度。データ型の精度は、数値に含めることができる合計桁数を指定します。桁数が多いほど、値の精度が高くなります。値が範囲外の場合、エラーが発生します。たとえば、BIGINT データ型の場合、9,223,372,036,854,775,807 より大きい値に対してエラーが発生します。

エラーや不正確な結果を避けるために、特に整数データ型の数値に対して演算を実行するときは、結果を考慮する必要があります。INTEGER データ型の精度は 10 桁で、範囲は–2,147,483,647 から 2,147,483,647 までです。2 つの大きな整数を乗算し(たとえば、33432*79879)、結果が 2,147,483,647 より大きい場合、オーバーフローエラーが発生します。データ型の精度または範囲よりも大きい結果が予想される場合は、結果を DOUBLE や BIGINT などのより大きなデータ型として割り当てるか、式を倍精度浮動小数点数として評価するように強制するために、整数ではなく、倍精度定数(たとえば、33432.0*79879.0)として入力できます。

結果を考慮する必要がある場合の具体例は、DS2 で除算を実行する場合です。次の例では、80/100 は整数除算を実行し、結果は 0 です。80.0/100 の浮動小数点除算は 0.8 になります。

```
data;
  dcl double x y;
  method run();
    /* this gives an integer value result of 0 */
    x=80/100;
    put x=;

    /* this gives the result of .8 */
    y=80.0/100;
    put y=;
  end;
enddata;
run;
```

文字データ型

文字データ型の概要

DS2 には、文字列(テキスト)データを格納する複数の文字データ型が用意されています。文字データ型には、英字、0 - 9 の数字、およびその他の特殊文字を含めることができます。

注: 文字列に数値のみが含まれている場合、DS2 は自動的にそれを数値型に変換し、その数値を計算に使用します。

各文字データ型には、最大文字数を指定するためのパラメーターがあります。

DS2 およびほとんどの SQL データベースでは、すべての文字データ型の文字長は文字列の文字数です。固定長データ型(CHAR および NCHAR)の場合、文字長は文字列内の正確な文字数です。可変長データ型(NVARCHAR および VARCHAR)の場合、文字長は文字列で許可される最大文字数です。

SAS データセットまたは CAS テーブルを作成するときの DATA ステップおよび PROC SQL の場合、CHAR 型の文字長はバイト単位で測定されます。

違いを理解するために、長さが 10 で UTF-8 エンコーディングの固定長文字列を想定してください。DS2 は、10 の長さを 10 文字として解釈します。DS2 列には、10 個の 4 バイト文字を含む任意の 10 個の UTF-8 文字を格納できます。これには 40 バイトのストレージが必要です。DATA ステップと PROC SQL は、10 の長さを 10 バイトとして解釈します。DATA ステップと PROC SQL によって 10 個の 4 バイト文字が書き込まれると、切り捨てエラーが発生します。これは、その 10 文字が、データ用に SAS データセットまたは CAS テーブルに割り当てられた 10 バイトのストレージを超えるためです。

文字データ型とエンコーディング

すべての DS2 文字データには、文字エンコーディングが関連付けられています。DS2 は、同じプログラム内の複数の文字エンコーディングでのデータ処理をサポートしていますが、各文字変数には、DS2 プログラムの実行中に固定される 1 つの関連付けられた文字エンコーディングがあります。SET ステートメントまたは割り当てステートメントのいずれかによって文字変数に割り当てられたデータは、そのデータが別の文字エンコーディングである場合、変数の文字エンコーディングにトランスコードされます。

文字変数の文字エンコーディングは、DECLARE ステートメントの CHARACTER SET オプションで指定できます。次のステートメントを考えてみましょう。

```
declare varchar(1024) character set utf8 s1;
declare varchar(1024) character set  latin1 s2;
declare varchar(1024) s3;
declare nvarchar(1024) s4;
```

文字変数 s1 は、文字エンコーディング UTF-8 で宣言されています。文字変数 s2 は、文字エンコーディング latin-1 で宣言されています。文字変数 s3 は、明示的な文字エンコーディングなしで宣言されています。したがって、DS2 は DS2 セッションエンコーディングを変数 s3 の文字エンコーディングとして割り当てます。PROC DS2 を使用して実行する場合、DS2 セッションエンコーディングは SAS セッションエンコーディングです。DS2 セッションエンコーディングが EUC-CN の場合、文字変数 s3 には文字エンコーディング EUC-CN が割り当てられます。各国文字変数は、DECLARE ステートメントで CHARACTER SET オプションをサポートしない特殊なケースです。各国文字変数には、DS2 セッションエンコーディングに関係なく、文字エンコーディングとして常に UTF-8 が割り当てられます。したがって、各国文字変数 s4 には、文字エンコーディングとして UTF-8 が割り当てられます。

NCHAR[*n*]宣言は CHARACTER[*n*] CHARACTER SET UTF-8 と同等であり、NVARCHAR[*n*]宣言は CHARACTER VARYING[*n*] CHARACTER SET UTF-8 と同等です。

SET または MERGE ステートメントによって暗黙的に宣言された文字変数には、データソース内の対応する列のエンコーディングが割り当てられます。SET または MERGE ステートメント以外のステートメントによって暗黙的に宣言された文字変数には、変数の文字エンコーディングに DS2 セッションエンコーディングが割り当てられます。

注: 出力データソースが SAS データセットなどの複数のエンコーディングをサポートしていない場合、さまざまなエンコーディングがテーブルの共通エンコーディングに抽出されます。

デフォルトのエンコーディングは、オペレーティングシステムとロケールによって異なります。文字セットエンコーディング値の完全なリストについては、*SAS 各国語サポート(NLS): リファレンスガイド*の"Character Sets for Encoding in NLS"を参照してください。

文字データ型の制限と考慮事項

文字変数に関する次の制限と考慮事項に注意してください。

- CHAR、VARCHAR、NCHAR、NVARCHAR のデータ型の列には、読み取りまたは生成される列に 10485760 文字という長さの制限があります。文字はマルチバイト文字(UTF-8 など)にすることができます。
- CHAR データ型と VARCHAR データ型の両方の比較演算では、末尾の空白は無視されます。末尾の空白は、CHAR と VARCHAR の両方について、メソッドに渡される引数などとして、割り当て、連結、出力中に無視されません。コピー時に末尾の空白は無視されません。VARCHAR(10)値'abc'が VARCHAR(20)にコピーされる場合、値は末尾の空白を含めてコピーされます。
- SAS データセットの読み取りまたは更新を行う場合、ENCODING=システムオプションはサポートされていません。CHARACTER SET 構文で複数のエンコーディング値が指定されている場合、DS2 は特定の ENCODING=値を最小公分母エンコーディングに変換します。使用される ENCODING=値は、データセットで実際に作成される値とは異なる場合があります。この競合により、トランスコーディングエラーが発生する可能性があります。
- DS2 は、データキャストに関して ANSI SQL 規格に準拠しています。たとえば、CHAR データ型の変数を VARCHAR データ型の新しい変数に割り当てる場合、CHAR 変数の内容が収まる場合は VARCHAR 変数に転送されます。次に例を示します。

  ```
  dcl char(8) u;
  dcl varchar(6) v;
  method run();
  u='eighteen';
  v=u;
  ```

  割り当て後、**v** の結果の値は'eighte'で、長さは 6 です。CHAR 変数の内容が収まらない場合は、空白文字であるかどうかに関係なく切り捨てられます。
  VARCHAR 変数に値を割り当てる際に、変数に格納されている CHAR 値から末尾の空白文字をコピーしない場合は、TRIM 関数を使用して、割り当て前に値から末尾の空白文字を削除します。

日付と時間のデータ型

DS2 は、日時を格納するという特定の目的のために、いくつかのデータ型をサポートしています。日付と時間の範囲は、データソース固有です。

CAS のバイナリデータ型

CAS で実行される DS2 プログラムは、VARBINARY データ型の列の作成、CAS テーブルからの VARBINARY 列の読み取り、VARBINARY 列の CAS テーブルへの書き込みを行うことができます。

CAS で実行される DS2 プログラムは、BINARY データ型の列を作成できます。ただし、CAS テーブルから BINARY データ列を読み取ったり、BINARY 列を CAS テーブルに書き込んだりすることはできません。DS2 プログラムが、CAS テーブルから BINARY 列を読み取ったり、BINARY 列を CAS テーブルに書き込んだりすると、エラーが発生します。このエラーを回避するには、CAS テーブルに対して読み取りまたは書き込みを行う前に、BINARY 列を削除します。

列のデータ型を定義する

データ型を定義するときは、DS2 でサポートされているデータ型のデータ型キーワードを使用します。

データを保存するには、そのデータソースのデータストレージでデータ型を使用できる必要があることに注意してください。DS2 はいくつかのデータ型をサポートしていますが、特定のテーブルに定義できるデータ型はデータソースによって異なります。これは、各データソースが必ずしもすべての DS2 データ型をサポートしているわけではないためです。さらに、データソースは標準 SQL データ型のバリエーションをサポートします。つまり、指定した特定のデータ型が別のデータ型にマッピングされる可能性があり、基になるデータソースで別の属性を持つ可能性もあります。これは、データソースが特定のデータ型をネイティブにサポートしていないが、同様のデータ型のデータ値をデータ損失なしで変換できる場合に発生します。たとえば、INTEGER データ型をサポートするために、SAS データセットはデータ型定義を DOUBLE である SAS 数値にマッピングします。

各データ型のデータソースの実装の詳細については、“データ型リファレンス” (*SAS DS2 言語リファレンス*)を参照してください。

さらに、データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。CT_PRESERVE=接続引数の詳細については、*SAS Federation Server: Administrator's Guide* を参照してください。

DOUBLE および REAL データ型を使用するエラーメッセージ

エラーメッセージに DOUBLE および REAL データ型の数値が含まれている場合、これらの値は BEST*w*.出力形式を使用してログに書き込まれます。BEST*w*.出力形式は、数値を書き込むためのデフォルトの出力形式です。BEST*w*.を使用すると、SAS は、使用可能なフィールド幅に従って、値に関する最も多くの情報を提供する出力形式を選択します。

SAS は BEST*w*.出力形式を使用するため、DOUBLE および REAL データ型の値は、プログラムで使用する値とまったく同じではない可能性があります。BEST*w*.は値を丸め、SAS が指定された幅内で小数点以下の部分に少なくとも 1 つの有効数字を表示できる場合、BEST*w*.は結果を 10 進数として生成します。それ以外の場合は、指数表記で結果が生成されます。SAS では、表現に使用する出力形式にかかわらず、常に完全な値を保存します。

6

DS2 識別子

識別子の概要

識別子は、テーブル名や列名などのデータソースオブジェクトだけでなく、変数、メソッド名、パッケージ名、配列などのプログラミング言語エンティティに名前を付ける 1 つ以上のトークンまたはシンボルです。

DS2 言語は、通常の識別子と区切り識別子の両方について ANSI SQL:1999 標準をサポートしています。

通常の識別子は、ほとんどのプログラミング言語で見られるタイプの識別子です。**大文字と小文字が区別されない**ため、識別子 Name は NAME および name と同じです。通常の識別子では、特定の文字のみが許可されます。

区切り識別子は、二重引用符を必要とする識別子、つまりテーブル名とスキーマ名に対してのみ大文字と小文字が区別されます。その他の区切り識別子は、大文字と小文字が区別されません。区切り識別子には任意の文字を使用でき、二重引用符で囲む必要があります。変数名"Name"、"NAME" Name、name、NaMe はすべて同じ変数を表します。

ANSI SQL:1999 識別子をサポートすることにより、DS2 言語は、ANSI SQL:1999 識別子もサポートするデータソースと互換性があります。

注: SAS および SPD データセットの識別子は、32 文字に制限されています。

通常の識別子

通常の識別子に名前を付けるときは、次の規則を使用します。

- 通常の識別子の長さは、1 文字から 255 文字までです。
- 通常の識別子の最初の文字は、文字またはアンダースコアにする必要があります。後続の文字には、文字、数字、またはアンダースコアを使用できます。たとえば、識別子が引用符で囲まれていない限り、識別子にスラッシュやかっこを使用することはできません。通常の識別子を 2 つのアンダースコアで始めることはできないことに注意してください。
- 通常の識別子は大文字と小文字を区別しません。

次の通常の識別子が有効です。

```
firstName
lastName
_phonenum
phone_num1
```

通常の識別子の文字は、大文字として内部的に格納されるため、大文字と小文字を区別せずに文字を書くことができます。たとえば、引用符で囲まれていない入力列名である phone_num1、Phone_Num1、また PHONE_NUM1 は、出力では大文字の PHONE_NUM1 として表示されます。特殊文字やスペースを含まず、二重引用符で囲まれた識別子は、引用符が存在しないかのように通常の識別子として扱われます。

注: 各データソースには独自の命名規則があり、そのすべてが DS2 言語でサポートされています。プログラムに特定のデータソースに固有の識別子が含まれている場合は、そのデータソースの命名規則に従う必要があります。詳細については、*SAS/ACCESS for Relational Databases: リファレンス*のデータソースの命名規則に関するトピックを参照してください。

区切り識別子

区切り識別子に名前を付けるときは、次の規則に従います。

- 区切り識別子を二重引用符で開始および終了します。
- 区切り識別子の長さは、1 文字から 253 文字までです。(最大 255 文字のうち 2 文字は引用符用に予約されています。)
- 区切り識別子は、先頭と末尾の二重引用符の間の任意の文字列(スペースや特殊文字を含む)で構成されます。

- 大文字と小文字が区別される区切り識別子は、テーブル名とスキーマ名のみです。その他の区切り識別子は、大文字と小文字が区別されません。
- 区切り識別子の特殊文字は、識別子に必要な開始および終了の二重引用符に加えて、対応する二重引用符で囲む必要があります。
- 区切り識別子にスラッシュ(/)を使用しないでください。これにより、一部の動作環境でエラーが発生する可能性があります。たとえば、"/"は UNIX システムではサブディレクトリを意味します。

二重引用符で囲まれた文字列は、文字定数ではなく識別子として解釈されます。文字定数は、単一引用符でのみ囲むことができます。

次の区切り識別子が有効です。

```
"x&y&z"
"Ü1"
"(area)phone_num"
"a**B"
```

区切り識別子は、予約語になる可能性のある用語に使用できます。たとえば、日付宣言以外で"date"という用語を使用するには、区切り識別子"date"として使用します。次に例を示します。

```
/* In DATA step, there are no reserved words for variables. */
/* So, this doesn't cause an error.  However, how do you    */
/*  use such a variable in DS2?                             */
data a;
  date = mdy(8,14,2012);
run;

/* This program gives an error because the reserved word DATE */
/*  is used when a variable is expected.   */
proc ds2;
data b(overwrite=yes);
  method run();
    set a;
    if month(date) = 8 then
      put 'August';
    else
      put 'Not August';
  end;
enddata;
run; quit;

/* One solution is to quote the reserved word */
proc ds2;
data b(overwrite=yes);
  method run();
    set a;
    if month("date") = 8 then
      put 'August';
    else
      put 'Not August';
  end;
enddata;
run; quit;
```

```
/* Another solution is to rename the variable on input and output */
/*  to avoid the reserved word in DS2 code.                       */
proc ds2;
data b(overwrite=yes rename=(sas_date="date"));
  method run();
    set a(rename=("date"=sas_date));
    if month(sas_date) = 8 then
      put 'August';
    else
      put 'Not August';
  end;
enddata;
run; quit;
```

大文字と小文字が区別されないデータソースにサブミットされる DS2 プログラムで参照される、区切られた列名で作成されたテーブルに対して警告が発行されます。大文字と小文字を区別しないデータソースでは、引用符が削除され、列名が区切られていないものとして扱われます。

区切り識別子は、SAS DATA ステップ言語の名前リテラルに似ていますが、同じではありません。

注: 各データソースには独自の命名規則があり、そのすべてが DS2 言語でサポートされています。プログラムに特定のデータソースに固有の識別子が含まれている場合は、そのデータソースの命名規則に従う必要があります。詳細については、*SAS/ACCESS for Relational Databases: リファレンス*のデータソースの命名規則に関するトピックを参照してください。

注意
LIBNAME ステートメントで PRESERVE_TAB_NAMES=NO オプションを使用すると、予期しない結果が生じる可能性があります。

非ラテン文字のサポート

DS2 言語では、通常の識別子としてラテン文字のみがサポートされています。非ラテン文字を使用するには、識別子を二重引用符で区切る必要があります。

7

DS2 による null および SAS 欠損値の処理方法

存在しないデータの DS2 モード

DS2 は、存在しないデータを処理する 2 つのモードをサポートしています。

- Oracle や DB2 などの ANSI SQL 互換データベースなど、存在しないデータを処理する ANSI モード。ANSI モードは、すべてのデータ型の存在しないデータを ANSI SQL null 値として扱います。ANSI システムでは、NULL は、データ値がないことを示すためだけに使用される特別なマークによって表されます。使用可能な情報をエンコードしません。
- Base SAS のように存在しないデータを処理する SAS モード。SAS モードは、固定文字列および DOUBLE 列の存在しないデータを欠損値として扱います。空白スペースは、欠損文字データを表すために使用されます。デフォルトでは、欠損数値データはピリオド(.)で表されます。ただし、特殊文字._ (ピリオドの後にアンダースコアが続く)および A から Z を使用して、数値データを表すこともできます。SAS モードでは、存在しないデータには既知の値があります。

null 値と SAS 欠損値の処理には大きな違いがあります。目的の結果が得られるようにモードが設定されていない場合、データは失われます。(詳細については、“ANSI から SAS への変換と SAS から ANSI への変換について”を参照してください。)したがって、DS2 が SAS 欠損値を処理する方法を理解することが重要です。

要点は次のとおりです。

- 存在しないデータの処理モードは、データソースへの接続方法によって異なります。
  - DS2 プロシジャ、HPDS2 プロシジャ、および DS2 CAS アクションでサブミットされる DS2 コードは、デフォルトで SAS モードを使用してデータを処理します。
  - SAS Federation Server にサブミットされる DS2 コードは、ANSI モードでデータを処理します。
- 欠損値を持つことができるのは固定文字列と DOUBLE 列だけです。DOUBLE または CHAR(*n*)以外の DS2 データ型の変数に欠損値が割り当てられている場合、それらの値は自動的に null 値に変換されます。その他の DS2 データ型は、常に ANSI null 値として扱われます。
- MODE=オプションは、PROC DS2 の ANSIMODE プロシジャオプション、DS2 CAS アクションの nullBehavior=パラメーター、および DS2_OPTIONS ステートメントの SAS オプションに取って代わります。MODE=オプションは、混乱を減らし、存在しない値の処理をインターフェイス間で一貫させるために提供されました。
- MODE=プロシジャオプションとアクションパラメーターは、DS2 プログラム全体の処理プリファレンスを指定します。DS2_OPTIONS ステートメントを使用すると、DS2 プログラム内の特定のプログラムブロック(および指定されたプログラムブロックから動作を継承するすべてのプログラムブロック)の処理プリファレンスを指定できます。DS2 プログラムの残りの部分は、アクティブなプロシジャまたはアクション設定を使用して処理されます。
- ほとんどの場合、存在しないデータを処理するためにモードを変更する必要はありません。
- モードを変更する場合の例を次に示します。
  - クライアントアプリケーションが SAS データセットを処理し、存在しないデータのモードが ANSI モードの場合
  - SAS データセットの処理が完了し、クライアントアプリケーションが ANSI モードに戻る準備ができたとき
  - クライアントアプリケーションが ANSI テーブルを処理し、存在しないデータのモードが SAS モードの場合
  - ANSI テーブルの処理が完了し、クライアントアプリケーションが SAS モードに戻る準備ができたとき
- 場合によっては、モードによって異なる結果が返されます。

null 値の処理と SAS 欠損値の処理の違い

SAS 欠損値の処理は、ANSI null 値の処理とは異なり、次の状況で重要な意味を持ちます。

- データをフィルタリングする場合(たとえば、WHERE 句、HAVING 句、サブセット化 IF ステートメント、または外部結合 ON 句)。SAS モードは、各 null 値を既知の値である SAS 欠損値(.)として解釈します。ANSI モードは、null 値を不明な値として解釈します。
- ANSI モードで外部結合をサブミットすると、内部処理によって中間結果テーブルに null が生成される場合があります。DS2 は、中間結果テーブルの SAS モードで SAS 欠損値を生成する場合があります。したがって、中間結果テーブルの場合、ANSI モードでは null は不明な値として解釈され、SAS モードでは欠損値は既知の値として解釈されます。
- 空白スペースだけで 構成される文字値を比較すると、SAS モードはその値を MISSING と解釈します。ANSI モードでは、完全に空白スペースで 構成される文字値は、空白文字で構成される文字列として扱われます(特別な意味はありません)。
- 次の場合、DS2 は null を SAS 欠損値として解釈します。
  - SAS モードで、浮動小数点数または固定長文字値を含む計算または割り当てで null が使用されている場合。可変長文字列の欠損値は、ANSI null 値のように扱われます。
  - DOUBLE または CHAR データ型を想定している SAS 出力形式または関数に null が渡された場合。

注: SAS Federation Server LIBNAME エンジンを使用してデータにアクセスしている場合、DS2_SASMISSING 環境変数が TRUE に設定されていると、FedSQL CALL 呼び出しで ANSI null 値が SAS 欠損値に変換されます。

null 値と SAS 欠損値の属性と動作の違いを次に示します。

表 7.1 ANSI SQL null 値と SAS 欠損値の属性と動作の違い

属性または動作	ANSI SQL null 値	SAS 欠損値
内部表現	メタデータ	浮動小数点または文字
論理演算子による評価	解決値が True、False、および null である 3 値ロジックを使用して比較される不明な値です。たとえば、WHERE col1 = null は null を返します。 **注:** ANSI モードでは、null=null は null です。	比較するとブール値に解決される既知の値です。たとえば、WHERE col1 =.A は、同じ欠損値が見つかった場合に True を返します。 欠損値を論理演算子で使用すると、値は False (ゼロ)になります。
照合順序	最小値として表示	最小値として表示
スコープ	すべてのデータ型の列は null 値を使用して存在しないデータを表します。	欠損値を保持できるのは固定文字列と DOUBLE 列のみです。

ANSI から SAS への変換と SAS から ANSI への変換について

ANSI モードから SAS モードへ、またはその逆のデータ変換を行うと、データが失われる可能性があります。したがって、存在しないデータを処理するデフォルトモードを変更する前に、目的の結果を把握しておくことが重要です。

ANSI モードでは、DS2 が SAS データセットから SAS 数値欠損値を読み取るときに、SAS 欠損値を ANSI null 値に変換します。その後、ANSI null 値が SAS データセットに書き込まれると、DS2 は ANSI null 値を SAS 数値欠損値(.)に変換します。入力データセットからの SAS 数値欠損値が.A などの特殊数値欠損値であった場合、ANSI null との間の変換中に.A が失われ、SAS 数値欠損値(.)が出力データセットに書き込まれます。次の例では、列 **x** のデータ型は DOUBLE です。この例は、ANSI モードで実行された DS2 プログラムによって、SAS 特殊数値欠損値(4 行目の.A)が SAS 数値欠損値(.)に変換される方法を示しています。

```
/* assume input data set indata contains */
x
100
.
.A

/* Run the following program using ANSI mode */
proc ds2 mode=ansi;
data outdata;
  method run();
    set indata;
  end;
enddata;
run;
quit;

/* the output dataset outdata contains */
 x
 100
 .
 .
```

SAS モードでは、DS2 が ANSI データソースからデータ型 CHAR(*n*)の ANSI null 値を読み取るときに、ANSI null 値を SAS 文字欠損値(完全に空白スペースで 構成される文字列)に変換します。 DS2 が ANSI データソースの CHAR(n)列に存在しない値を書き込むと、その値は ANSI null 値として書き込まれます。ただし、処理中、DS2 は存在しない値を SAS 文字欠損値として扱います。次の例では、列 x のデータ型は CHAR(5)です。この例は、SAS モードで実行される DS2 プログラムによって、ANSI null 値(2 行目)が空白スペース(' ')で構成される文字列として扱われることを示しています。

```
/* assume input data set indata contains */
 x
```

```
'abc  '
 null

/* run the following program using SAS mode */
proc ds2;
data db.outdata;
  method run();
    set db.indata;
  end;
enddata;
run;
quit;

/* the output dataset outdata contains */
 x
'abc  '
'  '
```

存在しないデータの読み取りおよび書き込み時の ANSI モード動作の概要

次の表は、DS2 が ANSI モードでデータソースから null 値または SAS 欠損値を読み取るときに、クライアントアプリケーションに返される値を示しています。

表 7.2　ANSI モードでの存在しないデータ値の読み取り

列のデータ型	存在しないデータ値	アプリケーションに返される値
DOUBLE	.、._、または.A-.Z[1]	null
DOUBLE	null	null
CHAR	'␣'[2]	'␣'[2]
CHAR	null	null

1 値._ (ピリオドの後にアンダースコアが続く)または.A–.Z は、特殊数値欠損値を表します。SAS が特殊欠損値を出力する場合、文字またはアンダースコアのみを出力します。

2 値'␣'は単一引用符の間の空白スペースで、ANSI モードでは空白スペースであり、存在しないデータではありません。

次の表は、存在しないデータ値が ANSI モードでデータソースに書き込まれたときに格納される値を示しています。

表 7.3 ANSI モードでの存在しないデータの格納

列のデータ型	存在しないデータ値	SAS データセットに格納される値	ANSI SQL null サポートデータソースに格納される値
DOUBLE	null	.	null
CHAR	null	'␣'[1]	null

1 値'␣'は単一引用符の間の空白スペースで、(ANSI モードでは)存在しないデータではなく、空白スペースであることを示しています。

ANSI モードでは、空白スペースは常に空白スペースとして 解釈され、存在しないデータとして解釈されることはありません。また、ANSI モードでは、入力時または割り当て時に SAS 欠損値が ANSI null 値に変換されます。

存在しないデータの読み取りおよび書き込み時の SAS モード動作の概要

次の表は、存在しないデータ値が SAS モードでどのように読み取られるかを示しています。

表 7.4 SAS モードでの存在しないデータ値の読み取り

列のデータ型	存在しないデータ値	アプリケーションに返される値
DOUBLE	.、._、または.A-.Z[1]	.、._、または.A-.Z[1]
DOUBLE	null	.
CHAR	'␣'[2]	'␣'[2]
CHAR	null	'␣'[2, 3]

1 値._ (ピリオドの後にアンダースコアが続く)または.A–.Z は、特殊数値欠損値を表します。SAS が特殊欠損値を出力する場合、文字またはアンダースコアのみを 出力します。

2 値'␣'は単一引用符の間の空白スペースで、SAS モードでは存在しない文字データを表します。

3 SET ステートメントで固定幅文字列の null が検出されると、その文字列は文字列の長さ分の空白文字で埋められます。

次の表は、欠損しているデータ値が SAS モードでデータ送信先にどのように書き込まれるかを示しています。

表 7.5 SAS モードでの存在しないデータ値の書き込み

列のデータ型	存在しないデータ値	SAS データセットに格納される値	ANSI SQL null サポートデータソースに格納される値
DOUBLE[2]	.、._、または.A	.、._、または.A-.Z	null
CHAR[2]	'␣'[1]	'␣'[1]	null

1 値'␣'は単一引用符の間の空白スペースで、SAS モードでは存在しないデータです。
2 出力が SAS データセットに書き込まれる場合、他のデータ型は DOUBLE または CHAR データ型に強制されます。null 値は、その型変換中に欠損値(.)になります。

SAS モードでは、DOUBLE または CHAR 型の ANSI null 値は、入力時または割り当て時に SAS 欠損値に変換されます。FLOAT や VARCHAR などの他のデータ型の場合、null 値は入力時または割り当て時に保持されます。

注: 特殊数値欠損値に対する演算は、通常の SAS 欠損値、つまり単一のピリオドを生成します。たとえば、Y=.A の場合、X=Y+1 は X=.を生成します。ただし、他の演算とは異なり、直接割り当てでは特殊数値欠損値が保持されます。たとえば、Y=.A の場合、割り当てに応じて次の出力が生成されます。

```
X=Y+Y produces X=.
X=Y   produces X=A.
X=Y+1 produces X=.
```

null 値のテスト

DS2 は、null 値をテストする NULL 関数を提供します。NULL 関数には 1 つの引数があり、式にすることができます。式が null の場合、関数は **1** を返します。式が null でない場合、関数は **0** を返します。

この例は、null 値のテストを示しています。

```
if null(numCopies) then put 'Number of copies is unknown.'
  else put 'Number of copies is' numCopies;
```

詳細については、“NULL 関数” (*SAS DS2 言語リファレンス*)を参照してください。

欠損値のテスト

DS2 には、null または欠損値をテストするための MISSING 関数が用意されています。MISSING 関数には 1 つの引数があり、数値式または文字式にすることができます。式が null または欠損の場合、関数は **1** を返します。式が null でも欠損でもない場合、関数は **0** を返します。

NMISS 関数は、null および SAS 欠損数値の数を返します。NMISS 関数が数値を必要とし、複数の数値を使用できるのに対し、MISSING は数値か文字のいずれかの値を 1 つだけ使用できます。

詳細については、“MISSING 関数” (*SAS DS2 言語リファレンス*)および“NMISS 関数” (*SAS DS2 言語リファレンス*)を参照してください。

8

DS2 型変換

型変換の定義

バイナリデータ型
: BINARY および VARBINARY データ型を指します。

文字データ型
: CHAR、VARCHAR、NCHAR、NVARCHAR データ型を指します。

強制型変換可能なデータ型
: 文字データ型だけでなく、複数のデータ型に変換できるデータ型。

日付/時間データ型
: DATE、TIME、および TIMESTAMP データ型を指します。

強制型変換できないデータ型
: 文字データ型にのみ変換できるデータ型。

数値データ型
: DECIMAL、DOUBLE(または FLOAT)、REAL、BIGINT、INT、NUMERIC、SMALLINT、TINYINT データ型を指します。

標準文字変換
: 式が文字データ型でない場合は、CHAR データ型に変換されます。

標準数値変換
式が強制可能な非数値データ型の場合、DOUBLE データ型に変換されます。

型変換の概要

DS2 が式を解決するには、式のオペランドは、同じ一般的なデータ型、数値、文字、バイナリ、日付/時間である必要があります。DS2 は、必要な場合、式のオペランドと演算子に応じて、オペランドのデータ型を別のデータ型に変換します。このプロセスは**型変換**と呼ばれます。たとえば、連結演算子(||)は文字データ型を操作します。文字列'First'と数値整数 1 の連結では、オペランド 1 の INTEGER データ型は、連結が行われる前に CHAR データ型に変換されます。

オペランドのデータ型が同じ一般的なデータ型内で変換される場合、オペランドのデータ型はプロモートされます。SMALLINT および TINYINT のデータ型のオペランドは INTEGER にプロモートされ、REAL 型のオペランドは DOUBLE にプロモートされます。メソッドおよび関数式の引数を含む、SMALLINT、TINYINT、および REAL のすべての操作に対して型プロモーションが実行されます。

数値および文字データ型は強制型変換できます。BINARY、VARBINARY、および日付/時間データ型の DATE、TIME、および TIMESTAMP は強制型変換できず、いずれかの文字データ型にのみ変換できます。

DS2 が式を評価するとき、オペランドのデータ型が正確に一致する場合、型の変換やプロモーションは必要なく、式は解決されます。それ以外の場合、各オペランドは、演算子に応じて、標準の数値変換または標準の文字変換を行う必要があります。

数値式または文字式の結果は、データ型の優先順位に基づきます。両方のオペランドが同じ一般的なデータ型内で異なる型を持つ場合、式のデータ型は、優先順位の高いオペランドのデータ型になります。ここで、1 が最高の優先順位です。たとえば、数値データ型の場合、DOUBLE のデータ型が最も優先されます。式に INTEGER 型のオペランドと DOUBLE 型のオペランドがある場合、式のデータ型は DOUBLE です。優先順位のリストは、該当する場合は、さまざまなタイプの式の後続のトピックにあります。

すべての型変換を示す表については、付録 1, “式オペランドの DS2 型変換” (319 ページ)を参照してください。

単項式の型変換

+1 や-444 などの単項式では、標準の数値変換がオペランドに適用されます。次の表は、単項式のデータ型を示しています。

表 8.1　単項式のデータ型変換

式のタイプ	式のデータ型
単項プラス	オペランドと同じ、または変換されたオペランドの DOUBLE
単項マイナス	オペランドと同じ または変換されたオペランドの DOUBLE
単項否定	INTEGER

論理式の型変換

a & b や(a ~= start) | (f = finish)などの論理式では、標準の数値変換が各オペランドに適用されます。次の表は、式のデータ型を決定するために使用される優先順位を示しています。1 が最高の優先順位で、3 が最低です。式のデータ型は、優先順位の高いオペランドのデータ型です。

表 8.2　論理式のデータ型変換

優先順位	いずれかのオペランドのデータ型	式のデータ型
1	DOUBLE	DOUBLE
2	BIGINT	BIGINT
3	他のすべての数値データ型	INTEGER

算術式の型変換

a<>b や a + (b * c)などの算術式では、標準の数値変換が各オペランドに適用されます。

次の表は、加算、減算、乗算、除算演算子の算術式のデータ型を決定するために使用される優先順位を示しています。1 が最高の優先順位で、3 が最低です。式のデータ型は、優先順位の高いオペランドのデータ型です。

表 8.3　加算、減算、乗算、除算式の型変換

優先順位	いずれかのオペランドのデータ型	式のデータ型
1	DOUBLE	DOUBLE
2	DECIMAL、NUMERIC	DECIMAL、NUMERIC
3	BIGINT	BIGINT
4	他のすべての数値データ型	INTEGER

次の表は、最小、最大、およびべき乗演算子を使用する算術式のデータ型を示しています。

表 8.4　最小、最大、およびべき乗演算子式のデータ型変換

演算子	演算子のデータ型	式のデータ型
最小または最大	DOUBLE	DOUBLE
最小または最大	DECIMAL、NUMERIC	DECIMAL、NUMERIC
最小または最大	BIGINT	BIGINT
**	すべての数値データ型	DOUBLE

関係式の型変換

x <= y や i > 4 などの関係式では、適用される標準変換はオペランドのデータ型によって異なります。次の表に示すように、式のデータ型は常に INTEGER です。

表 8.5　IN 式以外の関係式のデータ型変換

データ型解決の順序	いずれかのオペランドのデータ型	標準変換	式のデータ型
1	任意の数値データ型	数値	INTEGER
2	CHAR、NCHAR	文字	INTEGER

データ型解決の順序	いずれかのオペランドのデータ型	標準変換	式のデータ型
3	DATE、TIME、TIMESTAMP	なし、データ型が一致する必要がある	INTEGER
4	NVARCHAR、VARCHAR	なし、エラーが返される	適用できません

表 8.6　IN 式のデータ型変換

優先順位	いずれかのオペランドのデータ型	式のデータ型
1	非数値	DOUBLE
2	DOUBLE	DOUBLE
3	DECIMAL、NUMERIC	DECIMAL
4	BIGINT 以外のすべての型	BIGINT

連結式の型変換

a || b や x !! y などの連結式では、標準の文字変換が各オペランドに適用されます。次の表は、式のデータ型を決定するために使用される優先順位を示しています。1 が最高の優先順位で、2 が最低です。式のデータ型は、優先順位の高いオペランドのデータ型です。

表 8.7　連結式のデータ型変換

優先順位	いずれかのオペランドのデータ型	式のデータ型
1	どちらかが NCHAR 型の場合	NCHAR
2	CHAR	CHAR

型変換とあいまいなメソッド呼び出し

呼び出されたメソッドと同じ名前のメソッドが複数見つかった場合、DS2 はメソッドのオーバーロードを解決して、呼び出すメソッドを決定しようとします。ただし、DS2 が解決できない特定の状況があります。

この例では、BIGINT パラメーターを持つメソッドと CHAR パラメーターを持つメソッドの 2 つの **ambig** メソッドが作成されます。メソッドが INIT メソッドで呼び出されると、プログラムは INTEGER 引数を渡します。DS2 は amig(bigint a)と ambig(char(1) a)のどちらを呼び出すべきか判断できないため、これによりあいまいなメソッド呼び出しエラーが生成されます。

```
data _null_;
  dcl integer inp;
  dcl char(100) res;
  method ambig(bigint a) returns char(10);
    return 'bigint';
  end;
  method ambig(char(1) a) returns char(10);
    return 'char(1)';
  end;

  method init();
    inp = 2;
    res = ambig(inp);
    put inp= res=;
  end;
  enddata;
run;
```

エラーを回避するには、INTEGER パラメーターを持つメソッド **ambig** を追加します。

```
data _null_;
  dcl integer inp;
  dcl char(100) res;
  method ambig(bigint a) returns char(10);
    return 'bigint';
  end;
  method ambig(char(1) a) returns char(10);
    return 'char(1)';
  end;

  /* Define ambig with a int parameter */
  /* fixes the problem by routing "integer" to "bigint" */

  method ambig(int a) returns char(10);
    dcl bigint l;
    l = a;
    return ambig(l);
  end;
```

```
    method init();
      inp = 2;
      /* call ambig(int) is no longer ambiguous */
      res = ambig(inp);
      put inp= res=;
    end;
    enddata;
  run;
```

DOUBLE 引数を指定してメソッド **ambig** を呼び出そうとすると、DOUBLE が既存のメソッド宣言の 1 つと正確に一致しないため、あいまいなメソッド呼び出しエラーが発生することに注意してください。かわりに、あいまいさを回避するために、新しいメソッド定義(これは DOUBLE パラメーターを受け取るもの)を作成する必要があります。

9

DS2 式

式とは

式は、データ値を生成するプログラミング要素です。式には、一連の命令を形成し、値に解決されるオペランドとオプションの演算子が含まれます。

オペランドは、単一の定数または変数にすることも、式にすることもできます。演算子は、オペランドの計算、比較、または連結を要求する記号です。

DS2 式の例を次に示します。

```
3
"col1"
a = b * c
```

```
s || 1 || z
a >= b**(c - 8)
system.put(a*5,hex.)
x'ff00effc'
```

式のタイプ

式の概要

式の基本的なタイプは、一次式です。演算子を使用して式(一次または複合)を結合する式は、複合式です。式は、関数式やメソッド式などの DS2 コンストラクトを呼び出すことができます。式は、“SYSTEM 式”や THIS 式など、関数または変数のスコープに影響を与える可能性があります。 IN 式は、式の結果がリストに含まれているかどうかに基づいて真理値を返します。

式が単純か複合か、コンストラクトを呼び出すか、スコープがグローバルかどうかに関係なく、すべての式には値とデータ型があります。式のデータ型は、変数や関数などの式が取る可能性のある値を制限します。DS2 は複合式に遭遇すると、ルールに従って、式の各部分を評価する順序を決定します。

一次式

最も単純な形式では、一次式は、識別子、数値、文字列、2 進数および 16 進数の定数、リテラル値、日付と時間の値、null 値です。

```
a
"var_a"
5.33
'Company'
date '2007-08-24'
```

次の表は、基本的な一次式、そのデータ型、および例を示しています。

表 9.1　一次式のデータ型と例

式のタイプ	簡単な説明	データ型	例
2 進定数	2 進定数と 16 進定数	VARBINARY	x'FE' b'01000011'
文字定数	文字列	CHAR、VARCHAR	'New Report' a='Stock'; b='Report';

式のタイプ	簡単な説明	データ型	例
			[1]
各国文字定数	各国文字列	NCHAR、NVARCHAR	n'New Report' a=n'Stock'; b=n'Report'; [1]
SYSTEMTHIS	System、THIS	式の解決された型	system.put(x,5.) this.s
日付/時間定数	日付と時間の値	DATE、TIME、TIMESTAMP[2]	date'2023-01-01' time'20:44:59' timestamp'2023-02-07 07:00:00.7569'
識別子	さまざまな言語要素の名前を提供する	宣言された型またはデフォルトの型。デフォルトは DOUBLE。	a "part1"
整数定数	整数	INTEGER、BIGINT	123
新規	パッケージメソッドをインスタンス化する	式の解決された型	a=_new_ package-name(); [1]
小数定数	浮動小数点数	DOUBLE、FLOAT、REAL	5. 4.3
Null 定数	Null 式	なし	NULL
かっこ	評価の優先順位を上げるためにかっこで囲まれた演算子とオペランド	かっこで囲まれた式の解決された型。	(x) (a + b) - c [1]
欠損定数	SAS 欠損数値式	DOUBLE	. .A
数値定数	正確な数値	DECIMAL、NUMERIC	99n 12345678901234567890123456789O.12N

1 この例では、一次式は式または割り当てステートメントに含まれています。式の例が強調表示されています。
2 TO_DOUBLE 関数を複合式で使用すると、SAS の日付、時間、または日時の値をエンコードする DOUBLE 値を生成できます。

複合式

複合式は、次の式のように、式と演算子を組み合わせてより拡張的な式を作成します。

```
a + b * -c - 5
a = b = 5
x | y & a < c * d
x**y**z - 9 >= f
z >< c + e <> u**y * 10
x || c + 7
a in (1,2,3)
```

複合式の評価は、表 9.8 (102 ページ)に示されている演算子の優先順位と、一次式のデータ型に基づいています。計算を行う前に、オペランドのデータ型が同じ一般的なデータ型(数値、文字、バイナリ、日付/時間)である必要があります。データ型が同じ場合、処理は続行されます。同じでない場合、オペランドのデータ型は、演算子とオペランドのデータ型に基づいて変換されます。

演算子の優先順位と一次式のデータ型がどのように評価されるかを理解するために、式 a + b / -c - 5 を考えてみましょう。この式には 4 つの演算子があります。

- 二項+演算子
- 二項/演算子
- 単項-演算子
- 二項-演算子

演算子の優先順位によって、演算の評価順序が決まります。表 9.8 (102 ページ)を見ると、単項-演算子の優先順位は 2、二項/演算子の優先順位は 3、二項+演算子と二項-演算子はどちらも優先順位が 4 です。2 つの演算子の優先順位が同じ場合、結合規則によって評価の順序が決まります。二項+演算子と二項-演算子には、左から右への結合があります。したがって、前述の式は評価のために次のようにグループ化されます。

```
((a + (b / (-c))) – 5)
```

型変換も演算の評価に影響します。二項算術演算では、両方のオペランドを同じ型にする必要があります。オペランドが同じ型でない場合、DS2 コンパイラは、算術式の優先規則の型変換を使用して、一方のオペランドを他方のオペランドと同じ型に変換します。

表 9.2　算術演算のデータ型変換

優先順位	いずれかのオペランドのデータ型	式のデータ型
1	DOUBLE、REAL	DOUBLE
2	DECIMAL、NUMERIC	DECIMAL、NUMERIC
3	BIGINT	BIGINT

優先順位	いずれかのオペランドのデータ型	式のデータ型
4	他のすべての数値データ型	INTEGER

次のように仮定します。

- a は、値が 1.35 の DOUBLE 変数です。
- b は、値が 2 の INTEGER 変数です。
- c は、値が 3 の INTEGER 変数です。

式の評価方法は次のとおりです。

1 c は INTEGER 値 3 であるため、-c は INTEGER 値-3 に評価されます。

2 b は INTEGER 値 2 であり、-c は INTEGER 値-3 に評価されるため、式 b/-c は 2/-3 になり、INTEGER 値 0 に評価されます。

3 a は DOUBLE 値 1.35 であり、式 b/-c は INTEGER 値 0 に評価されるため、INTEGER 値 0 は DOUBLE 値 0.0 に変換されます。式 a + b/-c は 1.35 + 0.0 になり、これは DOUBLE 値 1.35 に評価されます。

4 式 a + b /-c が DOUBLE 値 1.35 に評価され、5 が INTEGER 値であるため、INTEGER 値 5 は DOUBLE 値 5.0 に変換されます。式 a + b/-c - 5 は 1.35 - 5.0 になり、これは DOUBLE 値-3.65 に評価されます。

算術演算の非数値オペランドは、DOUBLE に強制変換されます。CHAR + INT の式の型は DOUBLE です。これは、CHAR オペランドが DOUBLE に強制変換されるためです。DOUBLE + INT は、DOUBLE の優先順位が INTEGER よりも高いため、式のデータ型が DOUBLE になります。

データ型変換の詳細については、8 章, “DS2 型変換” (69 ページ)を参照してください。

別の例として、a = b = 5 という式を考えてみましょう。最初の等号(=)は割り当て演算子です。2 番目の等号は論理等値演算子です。b が 5 以外の値の場合、b=5 は 0 と評価されます。したがって、a には値 0 が割り当てられます。詳細については、“複数の等号を持つ式の使用” (*SAS DS2 言語リファレンス*)を参照してください。

DOT 演算子式

ドット演算子式は、パッケージの外部にあるコードからパッケージデータにアクセスします。

DOT 演算子式の構文は次のとおりです。

package-variable.[*package-attribute* | *package-method(arguments)*]

package-variable1.*package-variable2*....*package-variableN*.[*package-attribute* | *package-method(arguments)*]

. (dot)
パッケージ属性にアクセスするか、左側のオペランドによって参照されるパッケージ変数から、右側のオペランドとして指定されたパッケージメソッドを呼び出します。

package-attribute
左側のオペランドによって参照されるパッケージの属性を指定します。パッケージ属性は、パッケージ内で定義されるグローバル変数です。このグローバル変数は、パッケージのスカラー変数または配列変数にすることができます。ドット表記は、変数配列属性、標準一時配列属性、および動的一時配列属性へのアクセスをサポートしています。

package-variable
アクセスする属性またはメソッドを持つパッケージのインスタンス化を指定します。

注 これはスカラーパッケージ変数または配列パッケージ変数の要素にすることができます。詳細については、“DECLARE PACKAGE ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

package-method(arguments)
左側のオペランドによって参照されるパッケージ変数のメソッドを指定します。

ドット演算子の左側のオペランドは、スカラーパッケージ変数または配列パッケージ変数の要素である必要があります。左側のオペランドが null の場合(つまり、パッケージ変数がパッケージインスタンスを参照していない場合)、ドット演算により致命的な欄ライムエラーが発生します。

ドット演算子の右側のオペランドは、左側のオペランドによって参照されるパッケージの属性またはメソッドでなければなりません。ドット演算子の右側のオペランドがスカラーパッケージ変数または配列パッケージ変数の要素である属性である場合、ドット演算子の結果自体を後続のドット演算の左側のオペランドとして使用できます。パッケージ変数を左側のオペランドとして指定する式は、ネストされたドット表記式です。

ネストされたドット表記は、パッケージの階層内のパッケージデータへのアクセスをサポートします。ネストされたドット演算子の左側の各オペランドがパッケージ変数またはパッケージ配列変数の要素に解決される限り、無制限の数のネストされたドット演算がサポートされます。

詳細については、“DS2 パッケージ内のドット表記” (166 ページ)を参照してください。

関数式

関数式は、DS2 プログラム内の関数を呼び出します。関数を呼び出すには、次の構文を使用します。

function-name ([*argument* [, ...*argument*]])

関数には引数が必要な場合があります。関数式に引数が含まれている場合、引数のデータ型は、必要に応じて、関数の引数の順序とデータ型である関数**シグネチャ**の

データ型に変換されます。関数が引数を取るか取らないかに関係なく、関数呼び出しにはかっこが必要です。たとえば、TIME 関数は引数を取りません。

```
t = time();
```

関数式は、関数によって返される値に解決されます。前述の関数式では、time()は現在時刻に解決されます。

メソッドと関数は似ています。関数はグローバルスコープを持ちます。メソッドはプログラミングブロックであり、ローカルスコープを持ちます。

関数の名前がメソッド名と同じ場合、DS2 はそのメソッドを呼び出します。メソッドと同じ名前の関数は、SYSTEM 式を使用してのみ呼び出すことができます。詳細については、“SYSTEM 式” (96 ページ)を参照してください。

DS2 関数のリストについては、“DS2 関数” (*SAS DS2 言語リファレンス*)を参照してください。

IF 式

IF 式の概要

条件付き IF 式は、条件式の評価が真(ゼロ以外の値)か偽(ゼロ)かに基づいて、2 つの値を選択するために使用されます。

IF 式を呼び出すには、次の構文を使用します。

IF *expression-1***THEN** *expression-2***ELSE** *expression-3*

expression-1 がゼロ以外の値の場合、IF 式の結果は *expression-2* の値になります。それ以外の場合、IF 式の結果は *expression-3* の値になります。次に例を示します。

```
 m=(if missing(u) then 0 else u);
```

IF 式は、他の式を使用できる場所であればどこでも使用できます。IF 式の優先順位は、算術演算子および論理演算子よりも低くなります。したがって、このような混合式ではかっこが必要です。

```
r = 25.5 + (if sum < 15 then -a else b*2);
```

かっこがないと、プラス(+)演算子が最初に評価され、サブ式 25.5 + if から解析エラーが発生します。

ネストされた IF 式

IF 式をネストして、多方向の決定のために多くの値から選択することができます。

IF *condition-expression-1***THEN***result-expression-1*

ELSE IF *condition-expression-2***THEN** *result-expression-2*

...

ELSE IF*condition-expression-n***THEN** *result-expression-n*

ELSE*result-expression-default*

条件式は順番に評価されます。ネストされた IF 式チェーンの結果は、真(ゼロ以外の値)と評価される最初の *condition-expression* に関連付けられた *result-expression* です。すべての *condition-expressions* が偽(ゼロ)と評価された場合、IF 式の結果は *result-expression-default* になります。次に例を示します。

```
grade = if score >= 90 then 'A'
  else if score >= 80 then 'B'
  else if score >= 70 then 'C'
  else if score >= 60 then 'D'
  else if score >= 0  then 'F'
  else NULL;
```

注: 相互に排他的な条件が長く連続する場合は、IF-THEN/ELSE ステートメントを連続で使用するかわりに SELECT ステートメントを使用します。ネストされた IF-THEN/ELSE ステートメントが多数あると、内部エラーが発生する可能性があります。対照的に、SELECT ステートメントは 1 回だけ評価されるため、エラーを発生させずにパフォーマンスを向上させることができます。

たとえば、次のプログラムのように、プログラムに多数の ELSE IF コンストラクトが含まれているとします。

```
if (e1) then ...
else if (e2) then ...
...
else if (en) then ...
else ...
```

かわりに、この SELECT ステートメントを使用します。

```
select;
when (e1): ...
when (e2): ...
...
when (en): ...
otherwise: ...
```

IF 式のデータ型

IF 式のデータ型は、最初の結果式である *expression-2* の型を調べることによって決定されます。

IF *expression-1***THEN** *expression-2***ELSE** *expression-3*

expression-2 が数値データ型でない場合、IF 式には *expression-2* の型が割り当てられます。

expression-2 が数値データ型の場合、IF 式には、*expression-2* と *expression-3* の広い方の数値データ型が割り当てられます。たとえば、*expression-2* が SMALLINT で、*expression-3* が DOUBLE の場合、IF 式には DOUBLE 型が割り当てられます。*expression-2* が数値データ型で、*expression-3* が数値データ型でない場合、IF 式には *expression-2* の型が割り当てられます。

ネストされた IF 式チェーンの最初の結果式が数値データ型の場合、すべての結果式が調べられ、ネストされた IF 式チェーンの型として割り当てる最も広い数値データ型が検索されます。次の例では、**t** は TINYINT、**b** は BIGNINT、**d** は DECIMAL(10,5) です。ELSE の 0 には BIGINT 型が割り当てられます。したがって、ネストされた IF 式チェーンには、TINYINT、BIGINT、および DECIMAL(10,5)の中で最も広い数値型である decimal(10,5)型が割り当てられます。

```
r = if n < 0 then t
  else if n = 0 then b
  else if n > 0 then d
  else 0;
```

注: ELSE 値に 0 ではなく 0.0 が使用された場合、ELSE の結果には、BIGINT 型ではなく DOUBLE 型が割り当てられます。DOUBLE 型の ELSE 式の場合、結果式の最も広い数値型は DOUBLE 型になります。したがって、ネストされた IF 式チェーンには、decimal(10,5)型ではなく、DOUBLE 型が割り当てられます。

IF 式の遅延評価

IF 式は、結果式に遅延評価を使用します。

IF *expression-1***THEN** *expression-2***ELSE** *expression-3*

たとえば、このコードを使用してゼロ除算をチェックできます。

```
a = if c ne 0 then b/c else null;
```

式 *expression-1* は常に評価されますが、*expression-2* または *expression-3* のいずれかのみが評価されます。IF 式の結果として選択されなかった式は評価されません。したがって、*expression-1* がゼロ以外の値(真)の場合、*expression-2* のみが評価されます。*expression-1* がゼロ(偽)の場合、*expression-3* のみが評価されます。

遅延評価は、ネストされた IF 式チェーンの結果式にも適用されます。

```
IF condition-expression-1 THEN result-expression-1
  ELSE IF condition-expression-2 THEN result-expression-2
  ...
  ELSE IF condition-expression-n THEN result-expression-2
  ELSE result-expression-default
```

選択された結果式は、評価される唯一の結果式です。遅延評価は条件式にも適用されます。条件式は、条件が真(ゼロ以外)と評価されるか、すべての条件が評価されるまで、順番に評価されます。IF 式に *n* 個の条件式があり、*i* 番目の条件が最初の非ゼロ条件である場合、最初の 1 から *i* までの条件のみが評価されます。*i*+1 から *n* までの条件は評価されません。

IN 式

IN 式は、式が定数リストに含まれているかどうかを判別します。“定数リスト” (46 ページ)を参照してください。

IN 式の構文は次のとおりです。

search-expression[*not-operator*]**IN** *constant-list*

検索式が定数リスト内のいずれかの定数と等しい場合、検索式は定数リストと一致します。NOT 演算子(またはそのエイリアス~および^)が IN 演算子の前に指定されている場合、IN 式の結果は、検索式が一致するかどうかの論理否定になります(論理 NOT 式の規則 (104 ページ)を参照)。

IN 式の結果の型は常に INTEGER であり、その値は 1 (TRUE)、0 (FALSE)、または null (不明)のいずれかです。次の表は、検索一致がどのように決定されるかを示しています。

表 9.3　IN 式の結果基準

TRUE	検索式は、定数リスト内の 1 つ以上の定数と等しいかを比較します。
FALSE	検索式は、定数リスト内のすべての定数と等しくないかを比較します。
UNKNOWN	前述のいずれも当てはまらず、少なくとも 1 つの同等性のテストが不明でした。通常、これは検索式が null の値に解決される場合に発生します。

検索式が定数リスト内の定数と等しいかどうかを判断する場合、DS2 は通常の等式(=)と同じ規則を使用します。比較されるいずれかの項目の値が不明(null 値で表される)の場合、比較の結果は不明です。

SAS モードに関する特別な考慮事項:

- データ型が CHAR または DOUBLE で、検索式が SAS 欠損値と等しい場合、定数リストの欠損値と一致します。
- 定数リストに null 値が指定されている場合、SAS 欠損値に変換されます。
- 定数リスト内の 1 つ以上の定数の型と一致するように検索式を DOUBLE に変換する必要があり、検索式の値が null である場合、null 値を DOUBLE に変換すると、SAS 欠損値が発生します。この場合、定数リストに欠損値が含まれていると、IN 式の結果は TRUE になります。

ANSI モードでは、定数リストに欠損値を指定しても意味がありません。ANSI モードでは、定数リストの欠損値値が null に変換され、不明な値として扱われるためです。

DS2 が null と SAS 欠損値を処理する方法の詳細については、“存在しないデータの DS2 モード” (61 ページ)を参照してください。

次の表は、SAS モードと ANSI モードのどちらが有効かによって、さまざまなケースでの IN 式の処理を示しています。これらの例では、次の変数宣言を想定しています。

```
dcl char(10) my_char;
dcl double my_double;
dcl integer my_int;
```

表 9.4 IN 式の例

モード	入力値	IN 式	結果	説明
両方	my_char = '3.14'	my_char in (3.14, 'def', 'ghi')	1	定数リストの 3.14 は文字列'3.14'に変換されます。
SAS	my_double = .;	my_double in (., 2, 3)	1	検索式の欠損値は、定数リストの欠損値と一致します。
SAS	my_double = .;	my_double in (1, 2, 3)	0	検索式の欠損値が非欠損値と一致しません。
ANSI	my_double = null;	my_double in (1, 2, 3)	null	検索式が不明の場合、定数リストに関係なく IN 式の値が不明です。
ANSI	my_int = -1;	my_int in (1, 2, 'not a number')	null	文字列'not a number'を BIGINT 型に変換しようとすると失敗し、null 値になります。一致するものがなく、不明な比較が少なくとも 1 つあるため、IN 式全体の値は不明です。
SAS	my_int = null;	my_int in (3, 3.14, .)	1	小数値を表すには、定数リストを DOUBLE にプロモートする必要があります。したがって、検索引数も DOUBLE に変換する必要があります。結果は欠損値であり、リスト内の定数と一致します。

LIKE 式

LIKE 式の概要

LIKE 式は、文字列がパターンマッチングの指定と一致するかどうかを判別します。

LIKE 式の構文は次のとおりです。

expression[NOT]**LIKE** *pattern-matching-expression*[ESCAPE *character-expression*]

式は、任意の文字列またはバイナリ文字列のデータ型にすることができます。

expression が *pattern-matching-expression* で指定されたパターンと一致する場合、値 1 (真)が返されます。それ以外の場合は、値 0 (偽)が返されます。

NOT LIKE は、LIKE の逆の値を返します。たとえば、x like y が真の場合、x not like y は偽です。

ESCAPE 引数は、通常パターンマッチングに使用されるパーセント(%)文字とアンダースコア(_)文字のリテラルインスタンスを検索するために使用されます。

検索パターン

パターンは、3 つのクラスの文字で構成されます。

表 9.5　パターンマッチング文字

文字	説明
アンダースコア(_)	任意の 1 文字に一致
パーセント記号(%)	0 個以上の文字の任意のシーケンスに一致 **注:** 末尾の空白の影響に注意してください。値を一致させるには、TRIM 関数を使用して末尾の空白を削除する必要がある場合があります。
その他の文字	その文字に一致

リテラル%および_の検索

%および_文字は LIKE 式のコンテキストで特別な意味を持つため、入力文字列でこれらの文字リテラルを検索するには、ESCAPE 引数を使用する必要があります。

これらの例では、値 **app**、**a_%**、**a__**、**bbaa1**、**ba_1** を使用しています。

- 条件 like 'a_%'は、**app**、**a_%**、および **a__**に一致します。検索パターンのアンダースコア(_)は、任意の 1 文字(アンダースコアを含む)と一致し、検索パターンのパーセント(%)は、'%'と'_'を含む 0 個以上の文字と一致します。
- 条件 like 'a_^%' escape '^'は、**a_%**のみに一致します。エスケープ文字(^)は、パターンがリテラル'%'を検索することを指定します。
- 条件 like 'a_%' escape '_'は、どの値とも一致しません。エスケープ文字(_)は、パターンが'a'の後に続くリテラル'%'を検索することを指定します。これは、これらの値のいずれにも適用されません。

大文字と小文字が混在する文字列の検索

DS2 LIKE 式では大文字と小文字が区別されます。大文字と小文字が混在する文字列を検索するには、次の例に示すように UPCASE 関数を使用します。

```
upcase(str) like 'SM%'
```

LIKE 式の例

次の表は、Smith, Smooth, Smothers, Smart, Smuggle という文字列を検索したときに一致する結果の例を示しています。

表 9.6 LIKE 式の例

LIKE 式の例	マッチング結果
str like 'Sm%'	Smith, Smooth, Smothers, Smart, Smuggle
str like '%th'	Smith, Smooth
str LIKE 'S__gg%'	Smuggle
str like 'S_o'	(一致なし)
str like 'S_o%'	Smooth, Smothers
str like 'S%th'	Smith, Smooth
str not like 'Z'	Smith, Smooth, Smothers, Smart, Smuggle

メソッド式

メソッド式は、METHOD ステートメントによって定義されたメソッドを呼び出します。

メソッドを呼び出すには、次の構文を使用します。

method-name ([*expression*[, ... *expression*]])

メソッドは、メソッド名とシグネチャに基づいて呼び出されます。DS2 は、最初にメソッド名を識別します。メソッド名が関数名と同じ場合、DS2 はそのメソッドを呼び出します。メソッドと同じ名前の関数は、SYSTEM 式を使用して呼び出すことができます。詳細については、“SYSTEM 式” (96 ページ)を参照してください。

DS2 ではオーバーロードされたメソッドが許可されているため、DS2 は、メソッドシグネチャの引数の数と引数のデータ型(メソッドの引数の順序とデータ型)が最も一致する引数を持つメソッドを呼び出します。最適な一致とは、メソッドパラメーターの数が引数の数と等しく、指定された引数リストに対して正確なパラメーターの型がこれほど多く一致するメソッドシグネチャが他にないようなものです。最適な一致が見つからない場合、エラーが発生します。

> ヒント　DS2 メソッドは、最大 1000 個の引数をサポートできます。1000 を超える引数を持つ DS2 メソッドは、コンパイルエラーを生成する可能性があります。

実行するメソッドが識別されると、DS2 は必要に応じて、引数のデータ型を対応するメソッドパラメーターのデータ型に変換します。その後、メソッドが実行されます。

メソッド式は、メソッドによって返される値に解決されます。

次の例では、メソッド CONCAT および ADD が定義され、INIT()メソッドで呼び出されます。INIT()メソッドで強調表示されている式はメソッド式です。

```
method concat(char(100) x, char(100) y) returns char(200);
  return trim(x)|| y;
end;

method add(double x, double y) returns double;
  return x + y;
end;

method init();
  dcl char(200) r;
  r = concat('abc', 'def');
  d = add(100,101);
end;
```

次の例では、D メソッドがオーバーロードされたメソッドです。DS2 は、メソッド式の引数をメソッドシグネチャと比較して、最適な一致を見つける必要があります。

```
method d(double x, double y) returns double;
  return x + y;
end;

method d(int x, int y) returns int;
  return x + y;
end;

method init();
  dcl double r;
  dcl int i;
  r = d(1.2345, 5.6789);
  i = d(99, 100);
end;
```

最初のメソッドは、シグネチャが DOUBLE データ型の値を必要とする D メソッドを呼び出します。2 番目のメソッドは、シグネチャが INTEGER データ型の値を必要とする D メソッドを呼び出します。

この最後の例は、DS2 がメソッド式の値のデータ型が INTEGER か DOUBLE かを判断できないことを示しています。あいまいであるため、DS2 はエラーを発行します。

```
method d(int x, double y) returns double;
  return x + y;
end;

method d(double x, int y) returns double;
  return x + y;
end;

method run();
  d = d(100, 102);
end;
```

詳細については、“METHOD ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

OF 演算子

概要と基本構文

OF 演算子を使用すると、変数リストと変数配列を関数呼び出しの引数として指定できます。変数グループを引数として関数にサブミットすることは、複合式における OF 演算子の唯一の目的です。

OF 演算子の構文は次のとおりです。

OF*variable-array*[*] | *variable-list*

variable-array[*]は、同種データ型のグローバル変数をグループ化したものです。添字演算子のアスタリスクは必須であり、変数配列内のすべての 要素が関数に渡されることを指定します。

variable-list は、省略形に一致する 1 つ以上のグローバル変数を選択する省略形の仕様です。DS2 は、いくつかのタイプの省略変数リストをサポートしています。詳細については、“省略変数リストの種類” (30 ページ)を参照してください。

OF 演算子は、変数配列または変数リスト内の各要素を変数名のカンマ区切りリストに変換し、これが関数呼び出しの引数になります。次に例を示します。

- OF 式 OF x1-x5 は引数リスト x1, x2, x3, x4, x5 に変換されます。
- VARARRAY double a[6] u v w x y z; として定義されたか、OF 演算子によって OF a[*] として参照された配列は、引数リスト: u, v, w, x, y, z に変換されます。

OF 演算子は、関数呼び出しで他の引数とインターリーブできます。たとえば、関数呼び出し

```
MAX(v1, v2, OF x1-x5, OF a[*], v3);
```

は次の変数のリストに変換されます。

```
MAX(v1, v2, x1, x2, x3, x4, x5, u, v, w, x, y, z, v3);
```

関数呼び出しの引数には、任意のタイプの式を使用できます。個々の引数は、OF 演算、添字演算、またはより複雑な式にすることができます。たとえば、次の関数呼び出しは有効です。

```
MAX(OF a[*], a[1], y[3], a[1]+y[3]*100)
```

この関数呼び出しの引数は、次を含むように展開されます。

- 変数配列 a のすべての要素。
- 配列 a の要素 1 の値。式の中の添字 1 は、グループ内の変数へのインデックスです。
- y という名前の配列の要素 3 の値。添字の 3 は、グループ内の変数へのインデックスです。
- 算術式 a[1]+y[3]*100 の結果。

OF 演算子には制限があります。

- DS2 関数の呼び出しでのみ使用できます。OF 演算子は、ユーザー定義メソッドまたはパッケージメソッドの呼び出しでは使用できません。
- さまざまな数のパラメーターを受け取る関数で使用できます。
- パラメーターの数が OF リストの要素の数と一致する関数で使用できます。

  注: 関数に渡されるパラメーターには、OF リストなし、1 つの OF リスト、いくつかの OF リスト、他の式を含むいくつかの OF リスト、などを含めることができます。すべての OF リストが展開された後の引数の総数は、関数に固定数の引数がある場合に関数が取る引数の数と一致する必要があります。

- WHERE 句で指定された関数では使用できません。
- DIF 関数、LAG 関数とは併用できません。
- OF *variable-array* [*]構文は、変数配列のみをサポートします。OF 演算子は、一時配列または varlist では使用できません。
- OF *variable-array*[*]構文は配列インデックスとして使用できません。
- 添字演算を OF 演算子のオペランドにすることはできません。添字演算の前に OF 演算子がある場合、DS2 は次のエラーを返します。Parse encountered constant when expecting '*'. これはサポートされていないものの例です。

  ```
  OF a[1]
  OF y[3]
  ```

  この例では、y は varlist です。変数情報カテゴリの関数のみが、varlist の要素を引数として受け取ることができます。

OF 演算子の拡張

2023.07 以降、OF 演算子の構文が拡張され、より柔軟になりました。

- OF 演算子は、引数リスト内で 1 回指定できます。この構文は引き続きサポートされていますが、引数リスト内の各変数配列または変数リストの前に OF 演算子

を付ける必要はなくなりました。引数リスト内のすべての変数配列と変数リストを展開するには、単一の OF 演算子で十分です。

- OF を 1 回指定し、それが引数リストの最初の引数である場合、引数リスト内のほとんどのグローバルスカラー変数はカンマではなくスペースで区切ることができます。ローカル変数、配列変数、または複雑な式である関数の引数は、カンマを使用して OF 演算とは区別する必要があります。

表 9.7 に、元の構文形式と拡張された構文形式の違いを示します。両方の構文形式がサポートされています。

表 9.7　有効な OF 式の例

元の構文	拡張された構文
x= SUM (OF x - - z, OF a - - c);	x= SUM (OF x - - z a - - c);
vsetmissing(z, of a[*], y)	vsetmissing(of z a[*] y)
vsetmissing(z, of s:, y)	vsetmissing(of z s: y)
vsetmissing(OF x, OF y, OF z, OF a[*])	vsetmissing(OF x y z a[*])
vsetmissing(z, of a[*], y, of t:, u)	vsetmissing(of z a[*] y t: u)

OF 演算子に対するその他の拡張は次のとおりです。

- 型変数リストは OF 演算子でサポートされています。これは、現在サポートされているものの例です。

  (OF double)

- OF *variable-array*[*]構文は、型変数リスト指定を持つ変数配列をサポートするようになりました。たとえば、次の使用法がサポートされるようになりました。

  vararray int x[*] int;

- OF *variable-array*[*]構文は、型変数リスト指定を含む別の配列の範囲の DIM に基づいた上限を持つ変数配列をサポートします。たとえば、次の使用法がサポートされるようになりました。

  ```
  vararray int x[*] int;
  vararray int  y[dim(x)];
  ```

- 宣言されていない変数があり、次のいずれかの場合、OF 演算子の使用はサポートされなくなりました。
  - OF 演算子の変数リストオペランドは型変数リストです。
  - OF 演算子の vararray[*]オペランドは型変数リストで定義されます
  - OF 演算子の vararray[*]オペランドは、型変数リストで定義された別の vararray の DIM に基づく範囲を使用して宣言されます。

  OF 演算子を成功させるには、プログラム内のすべての変数を型変数リストで宣言する必要があります。宣言されていない変数が OF 演算で参照されない場合でも、宣言されていない変数が存在すると、型変数リストで OF 演算を使用できなくなります。

許可されていない変数リスト要求のログ出力を次に示します。

```
20        proc ds2;
21          data _null_;
22            method init();
23              dcl double total;
24              x = 10.0; /* undeclared variable */
25              y = 20.0; /* undeclared variable */
26              total = sum(OF double);
27            end;
28          enddata;
29        run;
29      !     quit;
ERROR: Compilation error.
ERROR: Line 26: OF operator with undeclared variables and type
variable list specification is not supported.
NOTE: PROC DS2 has set option NOEXEC and will continue to prepare
statements.
```

許可されていない変数配列要求のログ出力を次に示します。

```
1       proc ds2;
2        data _null_;
3        vararray double x[*] double;
4
5        method init();
6        a = 1.0;
7        b = 2.0;
8
9        VSETMISSING(OF x[*]);
10         end;
11         enddata;
12         run;
ERROR: Compilation error.
ERROR: Line 9: OF operator with undeclared variables and type variable list
specification is not supported.
```

DS2 と DATA ステップの演算子の違い

DS2 OF 演算子の機能は、DATA ステップ演算子の機能と似ています。ただし、DS2 OF 演算子と DATA ステップの OF 演算子で異なる結果が出る場合があります。これは、変数リストが各言語でどのように処理されるかによるものです。DATA ステップは、関数呼び出しの時点までの変数のみを取得します。DS2 は、プログラム全体で一致するすべての変数を取得します。DATA ステッププログラムの SUM 関数が変数 A1 と A4 のみを使用する例を次に示します。DS2 プログラムの SUM 関数は、変数 A1、A4、および A2 を使用します。

```
/* DATA step program */
/* sums A1 and A4 with a result of 4 */
data _null_;
A1 = 1.0;
A4 = 3.0;
y = 1;
lab:
```

```
if (y = 2) then
X = SUM(OF A: );
A2 = 2.0;
if (y = 1) then do;
y = 2;
goto lab;
end;
put X=;
;
run;

/* DS2 program */
/* sums A1, A4, and A2 with a result of 6*/
proc ds2;
data X (overwrite=yes);
dcl double x A1 A2 A4 y;
method run();
A1 = 1.0;
A4 = 3.0;
y = 1;
lab:
if (y = 2) then
X = SUM(OF A: );
A2 = 2.0;
if (y = 1) then do;
y = 2;
goto lab;
end;
put X=;
end;
enddata;
run;
quit;
```

パッケージメソッド式

パッケージメソッド式は、パッケージで定義されているメソッドをインスタンス化します。

パッケージメソッド式を呼び出すには、次の構文を使用します。

package-variable.method(*method-arguments*)

パッケージ変数はスカラー変数または配列パッケージ変数の要素にすることができます。配列パッケージ変数は、複数のパッケージインスタンスをグループ化できる一時配列であり、各パッケージインスタンスは配列要素としてアクセスされます。詳細については、DECLARE PACKAGE ステートメントを参照してください。配列パッケージ変数のパッケージインスタンスは、次のようにパッケージメソッド式で参照されます。

package-array\[*index*\].*method*(*method-arguments*)

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\)は、構文に角かっこが必要であることを示します。配列の境界は、角かっこ([ ])で囲む必要があります。

パッケージメソッド式は、メソッド式と同様の方法で実行されます。つまり、DS2 がパッケージとメソッドが存在すると判断した後、メソッドシグネチャの最適な一致が判断され、必要に応じて引数のデータ型が変換され、メソッドが実行されます。

> ヒント　DS2 メソッドは、最大 1000 個の引数をサポートできます。1000 を超える引数を持つ DS2 メソッドは、コンパイルエラーを生成する可能性があります。

次の例では、強調表示された式は、スカラーパッケージ変数をパッケージ変数として使用するパッケージメソッド式です。

```
declare package myadd a1() a2();
a1.sale(3,4);
a1.add(1,2);
a2.bonus(5,12);
a2.add(10,20);
```

DECLARE PACKAGE ステートメントは、a1 および a2 という名前の 2 つのパッケージインスタンスを宣言し、インスタンス化します。最初の 2 つのパッケージメソッド式は、MYADD パッケージからインスタンス化された A1 パッケージの SALE メソッドと ADD メソッドを呼び出します。最後の 2 つのパッケージメソッド式は、A2 パッケージの BONUS メソッドと ADD メソッドを呼び出します。

配列パッケージ変数の使用例については、"パッケージ配列の宣言と使用"を参照してください。

注: DS2 パッケージメソッド式を FedSQL SELECT ステートメントの関数として呼び出すことができます。詳細については、“式での DS2 パッケージの使用” (*SAS FedSQL 言語リファレンス*)を参照してください。

パッケージの詳細については、“PACKAGE ステートメント” (*SAS DS2 言語リファレンス*)および 13 章, “DS2 パッケージ” (157 ページ)を参照してください。

SYSTEM 式

メソッドと関数が同じ名前の場合、メソッド呼び出しが関数呼び出しよりも優先されます。関数は、SYSTEM 式を使用してのみ呼び出すことができます。

SYSTEM 式を呼び出すには、次の構文を使用します。

SYSTEM.*function-expression*

SYSTEM 式は、関数式の先頭にドット表記 system.を追加します。たとえば、SUM が関数の名前であると同時にメソッドの名前でもある場合、SUM 関数は、SYSTEM 式 system.sum(a,b,c)を使用してのみ呼び出すことができます。

THIS 式

THIS 式は、DS2 プログラム内の任意の場所からグローバルスカラー変数を同時に宣言して使用する代替方法を提供します。THIS 式は、標準の変数ルックアップを回避するために使用されます。THIS 式では、DS2 はグローバルスコープで識別子のスカラー変数宣言を検索します。そのような宣言がない場合、DS2 はグローバルスコープで識別子を DOUBLE 型で宣言します。グローバル変数は、DS2 プログラム内のすべてのプログラミングブロックで参照できます。

THIS 式を呼び出すには、次の構文を使用します。

THIS.*variable-name*

THIS 式は、変数の先頭にドット表記 THIS.を追加します。

注: DS2 は、THIS.*variable-name* を *variable-name* としてのみ保存します。グローバルスカラー変数と同じ名前のローカル変数があり、DS2 が変数に関する診断メッセージを発行する場合、どの変数が問題であるかを区別できません。たとえば、DS2 が **x** が宣言されていないという警告メッセージを発行した場合、そのメッセージがグローバル変数 **THIS.x** を参照しているのか、ローカル変数 **x** を参照しているのかわかりません。

次の例では、変数 s は、THIS 式を使用してグローバル変数になります。

```
method init();
 this.s = sum(a,b,c,d,e);
end;
method run();
  t = put(this.s 5.4);
end;
```

THIS 式は、同じ名前のローカル変数によって隠されているグローバル変数にアクセスする方法を提供します。次に例を示します。

```
proc ds2;
data;
  declare double x;    /* declare global x */

  method run();
    declare double x;  /* declare local x */

    /* Two variables exist with same name, "x". */
    /* Identifier "x" refers to local x in      */
    /* scope of run method.  Global x is hidden */
    /* by local x.                              */

    this.x = 1.0;    /* assign 1.0 to global x   */
    x = 0.0;         /* assign 0.0 to local x    */

    output;          /* global x with 1.0 output */
  end;
enddata;
```

```
run;
quit;
```

SELECT 式

SELECT 式の概要

SELECT 式は、他の式の値に基づいて複数の式から選択するために使用されます。

SELECT 式を呼び出すには、次の構文を使用します。

SELECT [(*select-expression*)]

WHEN (*when-expression*) [...WHEN (*when-expression*)]*result-expression*

[...WHEN (*when-expression*) [...WHEN (*when-expression*)] *result-expression*]

OTHERWISE [*default-result-expression*]

END

SELECT 式は、一致する式が見つかるまで、各 WHEN 式を順番に評価します。次に、関連付けられた *result-expression* が SELECT 式の結果として評価されます。

SELECT 式は、他の式を使用できる場所であればどこでも使用できます。次に例を示します。

```
r = 25.5 + select (t) when (1) -a when (3) b*2 end;
```

選択式を含む SELECT 式

選択式が存在する場合は、それが評価されます。次に、WHEN 式が順番に評価されます。SELECT 式の結果は、選択式と同じ値に評価される最初の WHEN 式の結果式です。すべての WHEN 式が選択式とは異なる値に評価される場合、SELECT 式の結果は、存在する場合のデフォルトの結果式です。それ以外の場合は、欠損値または null 値です。次に例を示します。

```
s = select (t)
  when (1) x*10
  when (3) x
  when (5) x*100
  when (0) 0
  otherwise .
end;
```

t が 5 の場合、SELECT 式は'x*100'に評価されます。

選択式のない SELECT 式

選択式が存在しない場合は、WHEN 式が順番に評価されます。SELECT 式の結果は、真(ゼロ以外の値)と評価される最初の WHEN 式の結果式です。すべての WHEN 式が偽(ゼロ)と評価された場合、選択式の結果は、存在する場合のデフォルトの結果式です。それ以外の場合は、欠損値または null 値です。次に例を示します。

```
grade = select
  when (score >= 90) 'A'
  when (score >= 80) 'B'
  when (score >= 70) 'C'
  when (score >= 60) 'D'
  when (score >= 0 ) 'F'
end;
```

score が 76 の場合、真と評価される最初の *when-expression* は **score>=70** です。*select-expression* は'C'に評価されます。

オプションのそれ以外の式

それ以外のデフォルトの結果式が提供されない場合、DS2 は、WHEN 式が選択されていないときに選択するデフォルトの結果値を提供します。SAS モードで SELECT 式の型が DOUBLE または CHAR の場合、デフォルトの結果値は欠損値(.)です。どちらのモードでも、他のすべてのデータ型の場合、デフォルトの結果値は NULL です。

複数の When 式を使用した結果式

複数の WHEN 式を 1 つの結果式に関連付けることができます。WHEN 式が連続してリストされ、その後に 1 つの結果式が続きます。結果式に関連付けられた WHEN 式のいずれかが最初に一致する WHEN 式である場合、SELECT 式の結果は結果式です。

たとえば、次の SELECT 式は、変数 **t** の値が'A'、'a'、'P'、'p'のいずれかである場合、'airplane'と評価されます。

```
s = select (t)
  when ('A')
  when ('a')
  when ('P')
  when ('p') 'airplane'
  when ('C')
  when ('c') 'car'
  when ('T')
  when ('t') 'train'
  otherwise  'walk'
end;
```

SELECT 式のデータ型

SELECT 式のデータ型は最初の結果式の型を調べることによって決定されます。最初の結果式が数値データ型でない場合、SELECT 式には最初の結果式の型が割り当てられます。

最初の結果式が数値データ型の場合、すべての結果式が調べられ、SELECT 式の型として割り当てる最も広い数値データ型が検索されます。

次の例では、**t** は TINYINT、**b** は BIGINT、**d** は DECIMAL(10,5)、**s** は CHAR(10)です。選択式には、TINYINT、BIGINT、DECIMAL(10,5)、CHAR(10)の中で最も広い数値型である DECIMAL(10,5)型が割り当てられます。

```
r = select (t)
  when (1) t
  when (2) b
  when (3) d
  otherwise s
end;
```

注: **s** が最初の結果式に割り当てられていた場合、SELECT 式の型は CHAR(10)になります。最初の結果式が非数値型の場合、非数値型が SELECT 式の型として割り当てられます。

SELECT 式の遅延評価

SELECT 式は、WHEN 式に遅延評価を使用します。WHEN 式は、一致する WHEN 式が見つかるまで、またはすべての式が評価されるまで順番に評価されます。SELECT 式に *n* 個の WHEN 式があり、*i* 番目の WHEN 式が選択されている場合、最初の 1 から *i* までの WHEN 式のみが評価されます。*i*+1 から n までの when 式は評価されません。

遅延評価は、SELECT 式の結果式にも適用されます。選択された結果式は、評価される唯一の結果式です。

次の例では、n[i]が 0 に等しい場合、最初の 2 つの WHEN 式(n[i] < 0 と n[i] = 0)のみが評価され、2 番目の結果式(y*10-r2)のみが評価されます。

```
m(select
   when (n[i] < 0) y*100-r1
   when (n[i] = 0) y*10-r2
   when (n[i] > 0) y*10
 end);
```

添字演算子

添字演算子を使用すると、DS2 配列または varlist の要素にアクセスできます。

添字演算の構文は、配列または varlist 識別子と、それに続く配列または varlist 内の各次元のインデックス式で構成されます。構文の形式は次のとおりです。

identifier \[*index-expression*[,...*index-expression*] \]

注: 構文規則の角かっこは、オプションの構文を示します。角かっこの前のエスケープ文字(\)は、構文に角かっこが必要であることを示します。添字演算子のインデックスは、角かっこ([ ])で囲む必要があります。

index-expression のデータ型は INTEGER です。指定された *index-expression* が整数でない場合、INTEGER に強制変換されます。

デフォルトでは、インデックスは 1 ベースです。varlist の下限は 1 です。DECLARE または VARARRAY ステートメントで下限が指定されていない場合、配列の下限は 1 です。ただし、配列には別の下限を指定できます。たとえば、次の配列の下限は-10、上限は 5 です。したがって、有効なインデックスの範囲は-10 から 5 です。

```
declare double ta[-10:5];
vararray double va[-10:5];
```

添字演算子の例は次のとおりです。

```
a[i]
s[j * 2, k-3]
this.c[2, vwind, a[i]]
```

各添字演算は、配列または varlist の 1 つの要素にアクセスします。多次元配列の場合、添字演算では、配列の範囲ごとに *index-expression* をカンマで区切って指定する必要があります。たとえば、次のログ出力を参照してください。

```
80   proc ds2;
81   data _null_;
82     method init();
83       dcl double x[2,2];
84       dcl int i j;
85       x := (100 200 300 400);
86       do i = 1 to dim(x, 1);
87         do j = 1 to dim(x, 1);
88           put x[i,j]=;
89         end;
90       end;
91     end;
92   enddata;
93   run;
93 !      quit;
x[1,1]=100
x[1,2]=200
x[2,1]=300
x[2,2]=400
```

インデックス式が配列または varlist の範囲制限を超えている場合、DS2 はエラーを発行します。

配列を使用した添字演算は、配列のデータ型のスカラー変数を受け入れる任意の式で使用できます。varlist を使用した添字演算は、varlist の要素を受け入れる任意の式で使用できます。現在、これは V*変数情報関数に限定されています。

式の演算子

複合式での演算子の優先順位

演算子は、加算、比較、論理否定など、オペランドに対して実行される演算の種類を表します。式に複数の演算子とオペランドが含まれている場合、DS2 は演算子の優先順位を使用して式を解決します。演算は、優先順位の高い順に実行されます。

最高の優先順位は 1 で、最低の優先順位は 9 です。優先レベル内では、べき乗、最小、および最大演算子を除いて、演算子は左から右に関連付けられます。べき乗、最小、および最大演算子は、右から左に関連付けられます。

表 9.8 (102 ページ)の優先順位を使用すると、式 5+a**b*3 では、a**b が最初に計算され、次に 3 が乗算され、その結果が 5 に加算されます。

> ヒント　DS2 では、x < y < z は x < y および y < z ように評価されます。

次の表に、演算子とその優先順位を示します。

表 9.8　DS2 複合式における演算子とその優先順位

優先順位	演算子名	記号	結合
1	かっこ	( )	左から右へ
1	ドット(.)	.	左から右へ
1	添字	*identifier*[*index*]	左から右へ
1		SELECT 式	左から右へ
2		単項+および–	右から左へ
2	論理 NOT	NOT または^または~	左から右へ
2	べき乗、最大、最小	**、<>、><	左から右へ

優先順位	演算子名	記号	結合
3	乗算、除算	*、/	左から右へ
4	プラス、マイナス	+、–	左から右へ
5	連結	|| または!!	左から右へ
5	削除	..	左から右へ
6		IN、LIKE	左から右へ
7	等値または非等値	=または eq、^=または ~=	右から左へ
7	より大(または小)か等しい、より大(または小)	>=、<=、>、<[1]	左から右へ
8	論理 AND	AND または&	左から右へ
9	論理 OR	OR、|、または!	左から右へ
10		IF 式	右から左へ
なし	変数割り当て	=	なし
なし	配列割り当て	:=	なし
なし	名前のない Varlist	[*list-of-variable-names*]	なし
なし		_NEW_	なし
なし		OF	なし
なし		メソッド式	なし

1 DS2 では、x < y < z は x < y および y < z ように評価されます。

演算子別の式の値

次の表は、演算子に基づく式の解決値を示しています。

表 9.9　演算子別の式の値

演算子	値
論理式 [1]	
NOT または^または~	オペランドがゼロ以外の場合、結果は 0 になります。オペランドがゼロまたは欠損している場合、結果は 1 になります。オペランドが null の場合、結果は null になります。
OR または|または!	■ 2 つのオペランドの論理 OR ■ 一方のオペランドがゼロまたは欠損しており、もう一方のオペランドが null の場合は null ■ 両方のオペランドが null の場合は null ■ いずれかのオペランドがゼロ以外の場合は(もう一方のオペランドが null または欠損している場合でも) 1 ■ 両方のオペランドがゼロまたは欠損している場合は 0
AND または&	■ 2 つのオペランドの論理 AND ■ 一方のオペランドがゼロ以外で、もう一方のオペランドが null の場合は null ■ 両方のオペランドが null の場合は null ■ 両方のオペランドがゼロ以外の場合は 1 ■ いずれかのオペランドがゼロまたは欠損している場合は(もう一方のオペランドが null であっても) 0
算術式	
単項プラス	式オペランドと同じ値を持つ
単項マイナス	オペランドの算術否定
+	オペランドの算術和
–	オペランドの算術差
*	オペランドの算術積
/	オペランドの算術商
**	左のオペランドを右のオペランドでべき乗したもの
><	左右のオペランドの最小値
<>	左右のオペランドの最大値

演算子	値
任意の算術演算子	いずれかまたは両方の演算子が null の場合は null
関係式	
<	左のオペランドが右のオペランドより小さい場合は 1、それ以外の場合は 0
>	左のオペランドが右のオペランドより大きい場合は 1、それ以外の場合は 0
<=	左のオペランドが右のオペランド以下の場合は 1、それ以外の場合は 0
>=	左のオペランドが右のオペランド以上の場合は 1、それ以外の場合は 0
=または eq	左のオペランドが右のオペランドと等しい場合は 1、それ以外の場合は 0
^=	左のオペランドが右のオペランドと等しくない場合は 1、それ以外の場合は 0
任意の論理演算子	いずれかまたは両方の演算子が null の場合は null
連結式	
|| または!![2]	左のオペランドが右のオペランドとマージされて新しい値を形成する
..	連結する前に各引数を削除する
IF 式	
IF	論理式がゼロでも null でもない場合は THEN 式の値、それ以外の場合は ELSE 式の値。詳細については、“IF 式” (83 ページ)を参照してください。
IN 式	
IN	定数リストに比較式が含まれる場合は 1。詳細については、“IN 式” (85 ページ)を参照してください。
LIKE 式	
LIKE	式が指定されたパターンに一致する場合は 1、それ以外の場合は 0 (ゼロ)。詳細については、“LIKE 式” (87 ページ)を参照してください。

演算子	値
SELECT 式	
SELECT	SELECT 式と一致する最初の WHEN 式の結果式の値。詳細については、“SELECT 式” (98 ページ)を参照してください。
その他の式	
.	パッケージ属性にアクセスするか、パッケージの外部のコードからパッケージメソッドを呼び出します。詳細については、“DOT 演算子式” (81 ページ)を参照してください。
=	値または式を変数に割り当てます。
:=	配列に値を割り当てます。
identifier[*index-expression*]	配列または varlist の要素にアクセスします。
[*list-of-variable-names*]	名前のない varlist を作成します。
NEW	パッケージのインスタンスを構築します。詳細については、“_NEW_ 演算子” (*SAS DS2 言語リファレンス*)を参照してください。
OF	変数リストまたは変数配列を関数の引数のリストに変換します。
メソッド式	システムメソッドまたはユーザー定義メソッドを呼び出します。詳細については、“DS2 メソッドについて” (*SAS DS2 言語リファレンス*)を参照してください。

1 DS2 は、TRUE、FALSE、UNKNOWN の 3 つの真理値をサポートしています。ゼロ以外のオペランドは TRUE にマッピングされます。ゼロまたは欠損オペランドは FALSE にマッピングされます。null オペランドは UNKNOWN にマッピングされます。

2 ||連結演算子はスペースを削除しません。TRIM 関数を使用して、末尾のスペースを削除できます。ただし、..演算子は、TRIM とそれに続く連結と同じ機能を実行します。TRIM 関数 (TRIM(a) || TRIM(b))を使用するよりも、..演算子(a .. b)を使用する方が高速です。

DS2 での短絡評価

場合によっては、両方のオペランドを評価せずに論理 AND または OR 式の値を決定できます。たとえば、オペランド A が偽の場合、式(A AND B)の値は、オペランド B の値に関係なく偽になります。最初のオペランドのみに基づいて論理式全体の値を決定できる場合、プログラミング言語は評価を"短絡"し、2 番目のオペランドを評価するために必要な計算処理を回避できます。

オプション LOGICALEXPR=OPTIMIZED を設定することにより、DS2 で短絡評価を使用できます。LOGICALEXPR=オプションは、DS2_OPTIONS ステートメントの引数、DS2 プロシジャオプション、ならびに DS2 runDS2 および runModel アクションのパラメーターとして使用できます。

DS2 は、論理 AND および OR 式のオペランドを左から右に評価します。LOGICALEXPR=OPTIMIZED が有効な場合、DS2 は論理演算子を次のように評価します。

- AND の場合、最初のオペランドがゼロまたは欠損している場合、2 番目のオペランドは評価されず、式全体が false になります。最初のオペランドがゼロ以外または null の場合、DS2 は 2 番目のオペランドを評価する必要があります。
- OR の場合、最初のオペランドがゼロ以外の場合、2 番目のオペランドは評価されず、式全体が TRUE になります。最初のオペランドがゼロ、欠損、または null の場合、DS2 は 2 番目のオペランドを評価する必要があります。

LOGICALEXPR=OPTIMIZED を使用すると、数学的に無効な計算の実行を回避できます。次のプログラムを考えてみましょう。プログラムでは、式 1/sqrt(a)は、a が非負(平方根による)で、sqrt(a)が非ゼロ(除算による)である場合にのみ計算できます。LOGICALEXPR=OPTIMIZED が設定されている場合、AND 式の 2 番目のオペランドが評価されないため、プログラムはエラーなしで完了します。

```
proc ds2 errorstop;
  ds2_options logicalexpr=optimized;
  data _null_;
    declare double a;
    method init();
      a=0;
      if (a>0 AND 1/sqrt(a) < .5) then
        put 'criteria satisfied';
      else
        put 'unacceptable value';
      put 'execution continues ...';
    end;
  enddata;
run; quit;
```

次のような行がログに書き込まれます。

```
unacceptable value
execution continues ...
NOTE: Execution succeeded.No rows affected.
```

LOGICALEXPR=OPTIMIZED を使用して、計算コストの高い計算の実行を回避することもできます。次のコードスニペットを考えてみましょう。ユーザー定義パッケージ credit_eval の score メソッドには、複雑なロジックが含まれています。このビジネスケースでは、顧客がすでにローンの債務不履行に陥っている場合、そのメソッドの呼び出しは避ける必要があります。

```
dcl package credit_eval ce;
reject_loan_request = (has_defaulted OR ce.score() < 500);
...
```

AND 演算子と OR 演算子、およびそれらの代替演算子"&"、"|"、"!"には短絡評価が用意されています。

10

DS2 の日付と時間

DS2 の日付、時間、およびタイムスタンプ

DS2 の日付、時間、およびタイムスタンプの概要

DS2 は、他のデータソースで使用されている SQL スタイルの日付と時間の規則をサポートしています。データソースが SAS データセットでない場合、DS2 は DATE、TIME、および TIMESTAMP のデータ型を持つ日付と時間を処理できます。

DATE、TIME、および TIMESTAMP のデータ型の日付と時間の値は、SAS の日付、時間、または日時の値に変換できます。数値列が SAS データセットから読み取られ、数値列に SAS の日付、時間、または日時の出力形式が関連付けられている場合、その列は DS2 型の DATE、TIME、または TIMESTAMP データ型に変換されます。

DS2 は、任意の日付または時間の値を SAS 日付、時間、および日時の値に変換し、認識可能な日付または時間の値に戻す日付と時間関数を提供します。詳細については、“日付、時間、および日時関数” (114 ページ)を参照してください。

ANSI SQL でサポートされている日付と時間の間隔は、DS2 ではサポートされていません。

日付、時間、およびタイムスタンプ変数の宣言

次の例のように、DECLARE ステートメントで DATE、TIME、または TIMESTAMP データ型を使用して、日付、時間、またはタイムスタンプ変数を宣言します。

```
dcl date dt;
dcl time tm;
dcl timestamp tmstmp;
```

注: 時間変数またはタイムスタンプ変数を宣言するときに精度を使用する場合、時間またはタイムスタンプの値は、DATA ステートメントによって生成されるまで、指定された精度に丸められません。内部的には、時間またはタイムスタンプの定数値は単に時間またはタイムスタンプ変数にコピーされます。

注: SAS 以外のデータソースで TIME 値と TIMESTAMP 値を操作していて、精度を指定しない場合、デフォルトの精度は DS2 のデフォルトの精度である TIME は 6、TIMESTAMP は 6 になります。

DS2 の日付と時間のデータ型の詳細については、5 章, “DS2 データ型” (47 ページ)を参照してください。

DS2 の日付、時間、およびタイムスタンプの値

日付、時間、またはタイムスタンプの変数を宣言すると、その変数の値は、次の構文を持つ DS2 の日付、時間、またはタイムスタンプの定数のみにすることができます。

DATE'*yyyy-mm-dd*'

TIME'*hh*:*nn*:*ss*[.*fraction*]'

TIMESTAMP'*yyyy-mm-dd hh*:*nn*:*ss*[.*fraction*]'

ここでは、次のとおりです。

- *yyyy* は 4 桁の年です
- *mm* は 01 - 12 の 2 桁の月です
- *dd* は 01 - 31 の 2 桁の日です
- *hh* は 2 桁の軍用時間です
- *mm* は 2 桁の分です

- *ss* は 2 桁の秒です
- *fraction* は 1 から 9 桁(0 から 9)で、オプションであり、秒の端数を表します

DATE、TIME、または TIMESTAMP キーワードの後の値の文字列部分は、単一引用符で囲む必要があります。

日付定数にはハイフンが必要で、日付文字列の長さは 10 にする必要があります。

時間定数にはコロンが必要です。秒の端数が存在しない場合、時間文字列は 8 文字の長さでピリオドを除く必要があります。端数なしでピリオドが存在する場合、DS2 はエラーを発行します。秒の端数が存在する場合、端数の長さは最大 9 桁、時間文字列の長さは最大 18 文字(ピリオドを含む)です。

タイムスタンプ定数では、日付のハイフンと時間のコロンが必要です。秒の端数が存在しない場合、タイムスタンプ文字列は 19 文字の長さでピリオドを除く必要があります。秒の端数が存在する場合、端数の長さは最大 9 桁、タイムスタンプ文字列の長さは最大 29 文字です。

DS2 の日付、時間、およびタイムスタンプ定数の例を次に示します。

```
date '2017-01-31'
time '20:44:59'
timestamp '2017-02-07 07:00:00.7569'
```

DS2 の日付と時間の操作

DATE、TIME、および TIMESTAMP 値に対して実行できる操作は、次のステートメントのように関係演算子<、>、<=、>=、=、^=、IN を使用する操作のみです。

```
if tm in(time'10:22:31', time'12:55:01') then
  if tm < time'13:30:00' then put 'Early afternoon';
  else put 'Time not available';
```

DS2 は、DATE、TIME、および TIMESTAMP のデータ型を持つ値の日付と時間の間隔を計算しません。

SAS の日付、時間、および日時の値

SAS 日付値は、1960 年 1 月 1 日から指定日付までの日数です。1960 年 1 月 1 日より前の日付は負の数、それより後の日付は正の数です。たとえば、1960 年 1 月 1 日の SAS 日付値は 0、1959 年 1 月 1 日の場合は-365、2008 年 1 月 1 日の場合は 17532 です。

SAS 時間値は、現在日の午前 0 時からの秒数です。SAS 時間値は 0 - 86400 です。

SAS 日時値は、1960 年 1 月 1 日から特定の日付の特定の時、分、秒までの秒数です。

数値列が SAS データセットから読み取られ、数値列に SAS の日付、時間、または日時の出力形式が関連付けられている場合、その列は DS2 型の DATE、TIME、または TIMESTAMP 列に変換されます。SAS データセットの数値列に出力形式がないか、SAS の日付、時間、日時出力形式以外の出力形式である場合、列は DOUBLE 型として処理されます。

DOUBLE が DS2 型の DATE、TIME、または TIMESTAMP に変換されると、日付と時間に関するすべての計算が SAS 日付値、SAS 時間値、または SAS 日時値として実行されます。詳細については、“日付、時間、および日時関数” (114 ページ)を参照してください。計算が完了したら、SAS の日付、時間、および日時の値を認識可能な日付と時間の出力形式にフォーマットできる他の関数があります。

SAS の日付、時間、および日時の値を DS2 の日付、時間、またはタイムスタンプの値に変換する

SAS の日付、時間、および日時の値は、TO_DATE、TO_TIME、および TO_TIMESTAMP 関数を使用して、DS2 の日付、時間、およびタイムスタンプの値に変換できます。これらの関数の引数は、SAS の日付、時間、または日時の値を表し、DOUBLE 型を持つ任意の値または式です。次に、PUT ステートメントまたは DECLARE ステートメントの出力形式を使用して、日付、時間、またはタイムスタンプの値をフォーマットできます。

次に例を示します。

```
data _null_;
  dcl date ds2d having format YYMMDD10.;
  dcl time ds2t having format TIME18.9;
  dcl timestamp ds2dt having format DATETIME28.9;
  dcl double d t ts;
  method init();
    d = 20854;
    ds2d = to_date(d);
    ds2t= to_time(d);
    ds2dt= to_timestamp(d);
    put ds2d ds2t ds2dt;
  end;
enddata;
```

次の行が SAS ログに書き込まれます。

```
2017-02-04
 5:47:34.000000000
01JAN1960:05:47:34.000000000
```

詳細については、“TO_DATE 関数” (*SAS DS2 言語リファレンス*)、“TO_TIME 関数” (*SAS DS2 言語リファレンス*)、および“TO_TIMESTAMP 関数” (*SAS DS2 言語リファレンス*)を参照してください。

DS2 の日付、時間、およびタイムスタンプの値を SAS の日付、時間、または日時の値に変換する

DS2 の DATE、TIME、および TIMESTAMP 値は、TO_DOUBLE 関数を使用して SAS の日付、時間、および日時の値に変換できます。

- TO_DOUBLE 関数は、DS2 DATE データ型の値を、SAS 日付値である DS2 DOUBLE データ型の値に変換します。
- TO_DOUBLE 関数は、DS2 TIME データ型の値を、SAS 時間値である DS2 DOUBLE データ型の値に変換します。
- TO_DOUBLE 関数は、DS2 TIMESTAMP データ型の値を、SAS 日時値である DS2 DOUBLE データ型の値に変換します。

その後、SAS の日付、時間、または日時出力形式を使用して、DS2 DOUBLE 値を目的の表現で表示できます。SAS の日付、時間、または日時の値の型は、出力形式の型と一致する必要があることに注意してください。

次の DS2 プログラムは、DS2 の日付、時間、およびタイムスタンプの値を SAS の日付、時刻、および日時の値に変換する方法を示しています。

```
data _null_;
 method run();
   dcl date ds2Date;
   dcl time ds2Time;
   dcl timestamp ds2TimeStamp;
   dcl double sasDate having format date.;
   dcl double sasTime having format time.;
   dcl double sasDateTime having format datetime.;
   dcl varchar(100) fmtDate fmtTime fmtDateTime;

   ds2Date = date'2017-06-13';
   ds2Time = time'10:54:34.012';
   ds2TimeStamp = timestamp'2017-06-13 10:54:34.012';

   put ds2Date=;
   put ds2Time=;
   put ds2TimeStamp=;

   sasDate = to_double(ds2Date);
   sasTime = to_double(ds2Time);
   sasDateTime = to_double(ds2TimeStamp);

   put sasDate= sasDate:best16.7;
   put sasTime= sasTime:best16.7;
   put sasDateTime= sasDateTime:best16.7;
```

```
    fmtDate = put(sasDate, monname.);
    fmtTime = put(sasTime, hour.);
    fmtDateTime = put(sasDateTime, dateampm.);

    put fmtDate=;
    put fmtTime=;
    put fmtDateTime=;
  end;
enddata;
```

次の出力が SAS ログに書き込まれます。

```
ds2Date=2017-06-13
ds2Time=10:54:34.012000000
ds2TimeStamp=2017-06-13 10:54:34.012000000
sasDate=13JUN17 20983
sasTime=10:54:34 39274.012
sasDateTime=13JUN17:10:54:34 1812970474.012
fmtDate=    June
fmtTime=11
fmtDateTime=13JUN17:10:54:34 AM
```

この例では、DS2 の DATE、TIME、および TIMESTAMP 列の値が SAS 日付値に変換され、出力値が DS2 DOUBLE 型の列に格納されます。DOUBLE 列には、日付、時間、および日時の出力形式が定義されています。

DS2 の DATE、TIME、および TIMESTAMP 変数に、SAS の日付、時間、および日時の値を割り当てることはできません。それらのデータ型は異なります。SAS の日付または時間の値を DS2 の DATE、TIME、または TIMESTAMP 変数に割り当てようとすると、DS2 は無効なデータ型変換エラーを発行します。

詳細については、“PUT 関数” (*SAS DS2 言語リファレンス*)を参照してください。出力形式の完全なリストについては、“日付、時間、および日時の出力形式” (116 ページ)を参照してください。

日付、時間、および日時関数

日付と時間の計算を実行するために、DS2 の日付と時刻関数は次のことを行います。

- 日付と時間を SAS の日付、時間、または日時値として取得または変換する
- SAS の日付、時間、または日時の値を認識可能な日付または時間にフォーマットする
- SAS 日時値から日付または時間を抽出する

次の表に、日付と時間の関数とその機能を示します。これらの関数の具体的な情報については、“DS2 関数” (*SAS DS2 言語リファレンス*)を参照してください。

表 10.1 日付と時間を取得、あるいは SAS の日付、時間、または日時値に変換する関数

関数	説明
DATE	SAS 日付値として現在の日付を返す
TODAY	SAS 日付値として現在の日付を返す
DATEJUL	ユリウス暦の日付を SAS 日付値に変換する
DHMS	日、時、分、秒の値から SAS 日時値を返す
HMS	時、分および秒の値から SAS 時間値を返す
MDY	月、日および年の値から SAS 日付値を返す
TIME	現在時刻を SAS 時間値として返す
YYQ	年と四半期の値から SAS 日付値を返す

表 10.2 SAS 日付または日時値を認識可能な日付または時間としてフォーマットする関数

関数	説明
DAY	SAS 日付値として月の日を返す
HOUR	SAS 時間または日時値から時間を返す
JULDATE	SAS 日付値からユリウス暦の日付を返す
JULDATE7	SAS 日付値から 7 桁のユリウス暦の日付を返す
MINUTE	SAS 時間または日時値から分を返す
MONTH	SAS 日付値から月を表す数字を返す
QTR	SAS 日付値から年の四半期を返す
SECOND	SAS 時間または日時値から秒を返す
WEEKDAY	SAS 日付値から、曜日に対応する整数を返す
YEAR	SAS 日付値から年を返す

表 10.3　SAS 日時値から日付と時間を抽出する関数

関数	説明
DATEPART	SAS 日時値から日付を抽出し、その日付を SAS 日付値として返す
TIMEPART	SAS 日時値から時間を抽出し、その時間を SAS 日時値として返す

日付、時間、および日時の出力形式

DS2 出力形式は、認識可能な日付と時間として SAS の日付、時間、および日時の値を書き込みます。PUT 関数を使用して、SAS の日付、時間、または日時の値をフォーマットします。PUT 関数で出力形式を指定する構文は次のとおりです。

PUT(*SAS-value*, *format*.);

PUT 関数の第 1 引数は、SAS 日付値、SAS 日時値、または SAS 時間値です。第 2 引数は出力形式です。

PUT 関数がどのように実行されるかを示す単純な DS2 プログラムは次のとおりです。

```
proc ds2;
data _null_;
  declare double d;
  declare varchar(15) x;
  method run();
    d=SAS-date-value;
    x=put (d,format-name.);
    put x=;
  end;
enddata;
run;
quit;
```

入力変数は DOUBLE です。出力変数は文字変数です。PUT 関数は文字変数を操作します。

表 10.4 (117 ページ)は、さまざまな DS2 の出力形式の種類を使用して、2017 年 6 月 13 日(2017-06-13 10:54:34.012)を表す SAS の日付、日時、および時間の値をフォーマットした結果を示しています。SAS の日付、時間、および日時の値がどのように取得されたかについては、前述の“DS2 の日付、時間、およびタイムスタンプの値を SAS の日付、時間、または日時の値に変換する” (113 ページ)を参照してください。

注: SAS の日付、時間、および日時出力形式を使用する場合、SAS 入力値はその出力形式の種類に適している必要があります。つまり、SAS 日付値は日付出力形式で指

定します。SAS 日時値は日時出力形式で指定します。SAS 時間値は時間出力形式で指定します。SAS 日付値は、四半期および年の出力形式で使用されます。適切な値を指定しないと、DS2 は不適切な応答を返します。

表 10.4　DS2 の日付と時間の出力形式の例

出力形式の種類	出力形式名	SAS 入力値	出力形式指定	結果
日付	DATEw.	20983	デフォルト	13JUN17
		20983	DATE9.	13JUN2017
	DAYw.	20983	デフォルト	13
	DDMMYYw.	20983	デフォルト	13/06/17
		20983	DDMMYY10.	13/06/2017
	DDMMYYxw.	20983	DDMMYYB.	13 06 17
		20983	DDMMYYB10.	13 06 2017
		20983	DDMMYYC.	13:06:17
		20983	DDMMYYC10.	13:06:2017
		20983	DDMMYYD.	13-06-17
		20983	DDMMYYD10.	13-06-2017
		20983	DDMMYYN6.	130617
		20983	DDMMYYN8.	13062017
		20983	DDMMYYP.	13.06.17
		20983	DDMMYYP10.	13.06.2017
		20983	DDMMYYS.	13/06/17
		20983	DDMMYYS10.	13/06/2017
	DOWNAMEw.	20983	デフォルト	Tuesday
	JULIANw.	20983	デフォルト	17164
	MMDDYYw.	20983	デフォルト	06/13/17
	MMDDYYw.	20983	MMDDYY10.	06/13/2017
	MMDDYYxw.	20983	MMDDYYP.	06.13.17

出力形式の種類	出力形式名	SAS 入力値	出力形式指定	結果
		20983	MMDDYYP10.	06.13.2017
		20983	MMDDYYS.	06/13/17
		20983	MMDDYYS10.	06/13/2017
	MMYYw.	20983	デフォルト	06M2017
	MMYYxw.	20983	MMYYC.	06:2017
		20983	MMYYD.	06-2017
		20983	MMYYN.	062017
		20983	MMYYP.	06.2017
		20983	MMYYS.	06/2017
	MONNAMEw.	20983	デフォルト	June
	MONTHw.	20983	デフォルト	6
	MONYYw.	20983	デフォルト	JUN17
	WEEKDATEw.	20983	デフォルト	Tuesday, June 13, 2017[1]
	WEEKDATXw.	20983	デフォルト	Tuesday, 13 June 2017[1]
	WEEKDAY2.	20983	デフォルト	3
四半期	QTRw.	20983	デフォルト	2
	QTRRw.	20983	デフォルト	II
日時	DATEAMPMw.d	1812970474.012	デフォルト	13JUN17:10:54:34 AM[1]
	DATETIMEw.	1812970474.012	デフォルト	13JUN17:10:54:34[1]
	DTDATEw.	1812970474.012	デフォルト	13JUN17
	DTMONYYw.	1812970474.012	デフォルト	JUN17
	DTWKDATXw.	1812970474.012	デフォルト	Tuesday, 13 June 2017[1]

出力形式の種類	出力形式名	SAS 入力値	出力形式指定	結果
	DTYEARw.	1812970474.012	デフォルト	2017
	DTYYQCw.	1812970474.012	デフォルト	17:2
時間	HOURw.d	39274.012	デフォルト	11
	TIMEw.d	39274.012	デフォルト	10:54:34
	TIMEAMPMw.d	39274.012	デフォルト	10:54:34 AM
	TODw.d	39274.012	デフォルト	10:54:34
年	YEARw.	20983	デフォルト	2017
	YYMMw.	20983	デフォルト	2017M06
	YYMMxw.	20983	YYMMC.	2017:06
		20983	YYMMD.	2017-06
		20983	YYMMN.	201706
		20983	YYMMP.	2017.06
		20983	YYMMS.	2017/06
		20983	デフォルト	17-06-13
		20983	YYMMDDB.	17 06 13
		20983	YYMMDDC.	17:06:13
		20983	YYMMDDD.	17-06-13
		20983	YYMMDDN.	20170613
		20983	YYMMDDP.	17.06.13
		20983	YYMMDDS.	17/06/13
	YYMONw.	20983	デフォルト	2017JUN
年/四半期	YYQw.	20983	デフォルト	2017Q2
	YYQxw.	20983	YYQC.	2017:2
		20983	YYQD.	2017-2

出力形式の種類	出力形式名	SAS 入力値	出力形式指定	結果
		20983	YYQP.	2017.2
		20983	YYQS.	2017/2
		20983	YYQN.	20172
	YYQRw.	20983	デフォルト	2017QII
	YYQRxw.	20983	YYQRC.	2017:II
		20999	YYQRD.	2017-II
		20983	YYQRP.	2017.II
		20983	YYQRS.	2017/II
		20983	YYQRN.	2017II
	YYQZw.	20983	デフォルト	1702

1 この例では、出力変数は VARCHAR(15)ではなく VARCHAR(29)として定義されています。

11

DS2 配列

DS2 配列の概要

DS2 では、配列は同種データの名前付き集計コレクションです。DS2 には、変数配列と一時配列の 2 種類の配列があります。一時配列には、固定範囲または動的範囲を設定できます。

変数配列と標準一時配列のサイズは固定です。固定サイズの配列は、ランタイム条件に依存しない同様のデータをグループ化するための優れた方法を提供します。ただし、配列に必要な要素数がランタイム条件に依存する場合、固定配列は、最悪のケースのランタイム条件で必要な要素数を考慮して静的にサイズ設定する必要があります。通常、最悪のケースは平均的なケースではありません。そのため、実際に必要なサイズを大幅に超えるサイズで配列が宣言されることが多く、メモリの使用率が低下します。

動的範囲制限ありの配列は、実行中に必要に応じて配列のサイズを拡大または縮小できるようにすることで、メモリを節約できます。DS2 は、一時配列に対してのみ動的範囲をサポートします。

次の表は、変数配列、標準一時配列、および動的範囲を持つ一時配列の違いをまとめたものです。

変数配列	標準一時配列	動的一時配列
PDV の変数への参照の集合	一時要素の集合	一時要素の集合
VARARRAY ステートメントで作成される	DECLARE ステートメントで作成される	DECLARE ステートメントで作成される
符号付き整数値でインデックス付けされるか、DS2 が配列内の変数をカウントして添え字を決定する。これは、VARARRAY ステートメントにリストする必要がある。	符号付き整数値によるインデックス	下限 1 で作成され、0 要素で初期化される
DS2 プログラムの実行中、配列内の要素の数は静的	DS2 プログラムの実行中、配列内の要素の数は静的	配列は RESIZEARRAY ステートメントを使用して実行時にサイズ調整される
範囲の数は>=1 にすることができる	範囲の数は>=1 にすることができる	1 次元
グローバルスコープで宣言する必要がある	ローカルまたはグローバルスコープで宣言できる	ローカルまたはグローバルスコープで宣言できる

変数配列	標準一時配列	動的一時配列
他の言語で見られるポインターまたは参照の配列に似ている	他の言語で見られる配列に似ている	他の言語で見られる配列に似ている
DATA ステップで、ARRAY ステートメントで作成される	DATA ステップで、_TEMPORARY_引数を指定した ARRAY ステートメントで作成される	対応する DATA ステップなし

配列の割り当ては、すべての配列タイプで同じです。

変数配列

変数配列の概要

変数配列は、入力データで類似した名前または目的を持つ一連の変数の処理を簡素化する方法です。変数配列の要素は、PDV 内の変数を参照します。変数配列は、DS2 プログラムの期間中のみ存在します。ただし、参照された変数の内容は、1 つ以上の結果セットに保持される場合があります。

VARARRAY ステートメントを使用して、名前、データ型、および配列範囲の数とサイズを指定します。たとえば、次の VARARRAY ステートメントは、PDV 内の 3 つの double 変数(a1、a2、a3)を参照する 3 要素の変数配列を指定します。

```
vararray double a[3];
```

前の例の VARARRAY ステートメントは、未作成の double 変数(a1、a2、a3)のいずれかを作成します。変数配列要素 a[1]は変数 a1 を参照し、a[2]は変数 a2 を参照し、a[3]は変数 a3 を参照します。

変数配列が作成されると、変数配列要素は、PDV 内の参照変数に保存されているデータの読み取りまたは変更に使用できる 2 番目の識別子セットとして機能します。

VARARRAY ステートメントで HAVING 句を使用して、ラベル、出力形式、および入力形式属性を変数配列に関連付けることができます。

注: 文字変数配列に含まれるすべての文字変数は、同じ長さとエンコーディングである必要があります。

詳細については、“VARARRAY ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

変数配列宣言

変数配列宣言はスカラー宣言に似ています。データ型と名前に加えて、配列範囲の数とサイズも指定できます。複数の範囲(または次元)は、カンマ区切り文字を使用して指定されます。

配列範囲には 2 つの形式があります。

符号付き整数ペア
: 符号付き整数ペアの形式は、配列の各次元の下限と上限[*l*:*h*]を指定します。ここで、*l* は指定された範囲の最小インデックスを表し、*h* は指定された範囲の最大インデックスを表します。下限の指定 *l* はオプションです。次元の下限が指定されていない場合、下限はデフォルトで 1 になります。

  上限 *h* が 1 より小さい場合、エラーが返されます。1 つの整数のみで配列範囲を指定すると、その整数が上限として解釈されます。デフォルトの最下限は 1 です。

  vararray double a[5];は、1 から 5 までのインデックスが付けられた 5 要素を持つ double 型の配列 a を宣言します。vararray char b[5,10];は、1 番目の次元に 5 要素、2 番目の次元に 10 要素で、合計 50 の要素を持つ 2 次元文字配列 b を宣言します。vararray int c[3] x y z;は、3 要素を持つ配列 c を宣言します。配列には、下限 1 と上限 3 でインデックスが付けられます。

  配列の上限は、以前に宣言された配列の次元の要素数に基づいてサイズを変更することもできます。上限には DIM 関数呼び出しを使用します。DIM 関数は、配列の上限を指定するために使用できる唯一の関数です。DIM 関数は、次元の下限を指定するためには使用できません。

* (アスタリスク)
: *形式は、下限が 1 で、上限が変数リストに含まれる変数の数である 1 次元配列を指定します。

詳細については、“DS2 変数リスト” (28 ページ)を参照してください。変数配列を宣言する方法と複数の範囲を指定する方法の詳細については、“VARARRAY ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

VARARRAY ステートメントでの変数の定義

VARARRAY ステートメントは、変数リスト内の未定義の変数を定義します。一部の変数リストタイプは、既存の変数のみを参照するため、新しい変数の定義にはなりません。次の表は、新しい変数を定義できる変数リストタイプと、既存の変数のみを参照する変数リストタイプを示しています。

変数リスト	例	変数展開	説明
名前	x y z	即時	新しい変数を定義できる

変数リスト	例	変数展開	説明
番号付き範囲	x1–x5	即時	新しい変数を定義できる
名前範囲	sales_jan--sales_mar	遅延	既存の変数のみを参照する
名前接頭辞	sales:	遅延	既存の変数のみを参照する
型	smallint	遅延	既存の変数のみを参照する
特殊名	_all_	遅延	既存の変数のみを参照する

DS2 プログラム内のすべての変数を検査しなくても、名前および番号付き範囲の変数リストを展開できます。したがって、これらのタイプの変数リストは、プログラム内で VARARRAY ステートメントが検出されるとすぐに展開されます。

他のタイプの変数リストでは、DS2 プログラムで定義されているすべての 変数を検査する必要があります。これらの変数リストの展開は、DS2 プログラム内のすべてのステートメントが検査され、すべての 変数が定義されるまで遅延します。

この遅延により、予期しないエラー状態が発生する可能性があります。たとえば、次のプログラムを考えてみましょう。

注: このセクションのコード例の DS2 ステートメントは、PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data;
  vararray int x[1] x:; /* 1 */

  method run();
    x1 = 5.0;         /* 2 */
  end;
enddata;
run;
quit;
```

1 接頭辞変数リスト x:は既存の変数のみを参照するため、VARARRAY ステートメントは変数を作成しません。接頭辞変数リスト x:の展開は、プログラム内のすべてのステートメントが検査されるまで遅延します。

2 割り当てステートメントでは、変数 x1 は未定義です。したがって、割り当てステートメントは、右辺の値の型(DOUBLE)を変数 x1 に割り当てます。

すべてのプログラムステートメントが検査された後、接頭辞変数リスト x:は、接頭辞 x を持つ唯一の既存の変数である x1 に展開されます。DOUBLE 型の変数 x1 は INTEGER 型の変数配列 x と互換性がないため、DS2 コンパイラはコンパイルエラーを発行します。

エラー状態を取り除く 1 つの方法は、VARARRAY ステートメント vararray int x[1] x:; を vararray int x[1] x1;または vararray int x[1];に変更することです。改訂されたステートメントは、変数 x1 を INTEGER 型として定義します。

DIM 変数配列範囲を使用した遅延変数定義

変数配列の次元の上限が別の配列の次元に基づいている場合、プログラム内のすべてのステートメントが検査されるまで、配列の変数の定義を遅延させることができます。次に例を示します。

```
proc ds2;
data;
  vararray double x[*] x:;  /* 1 */
  vararray int out[dim(x)]; /* 2 */

  method init();
  out1 = 0.0;             /* 3 */
  end;

  method run();
    set in;
  end;
enddata;
run;
quit;
```

1 接頭辞変数リスト x:の展開は、プログラム内のすべてのステートメントが検査されるまで遅延します。したがって、変数配列 x のサイズは、すべてのステートメントが検査されるまでわかりません。

2 out 変数配列には、デフォルトの変数リスト out1-out*n* があります。ここで、*n* は、変数配列に指定された要素の数です。out の要素数の決定は、配列 x のサイズが判明するまで遅延します(これは、すべてのステートメントが検査されたときに発生します)。

3 割り当てステートメントでは、変数 out1 は未定義です。したがって、割り当てステートメントは、右辺の値の型(DOUBLE)を変数 out1 に割り当てます。

前述の DS2 プログラムのすべてのステートメントが処理されると、次のことが起こります。テーブル in には 3 つの double 変数 x1 x2 x3 があるとします。

- 接頭辞変数リスト x:は x1 x2 x3 に展開されます。
- 変数配列 x のサイズは 3 に決定されます。
- 変数配列 out のサイズは 3 (dim(x))に決定されます。
- デフォルトの変数リスト out1-out3 は out1 out2 out3 に展開されます。
- 変数 out2 および out3 は、以前に定義されていなかったため、INTEGER 型として定義されています。out1 は、割り当てステートメント out1 = 0.0;によって DOUBLE 型として定義されていることに注意してください。

DOUBLE 型の変数 out1 は INTEGER 型の変数配列 out と互換性がないため、DS2 コンパイラはコンパイルエラーを発行します。

エラー状態を取り除く 1 つの方法は、配列の割り当て out1 = 0.0;を out[1]=0.0;に変更することです。この変更により、out 配列参照を使用して out1 データ値が更新され、

割り当てステートメントが DOUBLE 型を変数 out1 に割り当てないようになります。

変数配列を関数の引数として指定する

OF 演算子を使用すると、変数配列を一部の関数の引数として指定できます。OF 演算子は、変数配列を変数名のカンマ区切りリストに変換し、これが関数の引数になります。

注: OF 演算子の使用にはルールと制限があります。詳細については、“OF 演算子” (91 ページ)を参照してください。

変数配列を引数として SUM 関数にサブミットする例を次に示します。

```
proc ds2;
data _null_;
  vararray double a[4];
  method init();
    a:=(1,2,3,4);
  end;

  method run();
    dcl double x y z;
      y=99.0;
      z=100.0;
      x=sum(z, of a[*], y);
      put x=;

    x= sum(of a:, z);
    put x=;
  end;
enddata;
run;
quit;
```

例のコード 11.1　変数配列の使用例のログ出力

```
x=209
x=110
```

一時配列

一時配列の概要

一時配列の要素は、PDV に配置されていないため、どの結果テーブルにも表示されないという点で一時的です。一時データ要素の値は、次の反復の開始時に欠損にリセットされるのではなく、反復間で自動的に保持されます。一時配列は、DS2 プログラムの期間中のみ存在します。

DECLARE ステートメントを使用して、一時配列を指定します。DS2 一時配列には、固定範囲または動的範囲を設定できます。標準配列には固定範囲があります。標準一時配列の範囲はコンパイル時に設定され、実行時に変更することはできません。標準配列の要素数も実行時に固定されます。動的範囲を持つ配列の要素数は実行時に設定されるため、ランタイム条件に応じて配列の要素数を調整できます。

いずれかのタイプの配列の DECLARE ステートメントで HAVING 句を使用して、ラベル、出力形式、および入力形式属性をいずれかのタイプの一時配列に関連付けることができます。

標準一時配列宣言

標準一時配列宣言は、変数配列宣言に似ています。データ型と名前に加えて、配列範囲の数とサイズを指定します。複数の範囲(または次元)は、カンマ区切り文字を使用して指定されます。

PDV の外部に 3 つの一時的な double 値を保存する 3 要素の一時配列を指定する DECLARE ステートメントの例を次に示します。

```
declare double a[3];
```

符号付き整数ペアの形式は、配列の各次元の下限と上限[*l*:*h*]を指定します。ここで、*l* は指定された範囲の最小インデックスを表し、*h* は指定された範囲の最大インデックスを表します。下限の指定 *l* はオプションです。次元の下限が指定されていない場合、下限はデフォルトで 1 になります。

上限 *h* が下限より小さい場合、エラーが返されます。1 つの整数のみで配列範囲を指定すると、その整数が上限として解釈されます。デフォルトの最下限は 1 です。

配列の上限は、以前に宣言された配列の次元の要素数に基づいてサイズを変更することもできます。上限には DIM 関数呼び出しを使用します。DIM 関数は、配列の上限を指定するために使用できる唯一の関数です。DIM 関数は、次元の下限を指定するためには使用できません。

動的一時配列宣言

DECLARE ステートメントで動的範囲を持つ一時配列を宣言するには、配列の名前とデータ型を指定し、配列範囲としてアスタリスク([*])を指定します。この例では、CHAR(n)型の動的一時配列 d を宣言しています。

```
declare char(10) d[*];
```

アスタリスクは、1 次元、下限 1、初期上限 0 の配列を定義し、その結果、0 要素を含むように初期化された 1 次元配列になります。配列の上限は、実行時に RESIZEARRAY ステートメントを使用して変更できます。動的範囲制限ありの一時配列の次元数と下限は変更できません。動的範囲制限ありの配列は常に、下限が 1 の 1 次元配列です。

動的範囲を持つ一時配列の要素を設定する方法の例を次に示します。

```
RESIZEARRAY d[100];
```

RESIZEARRAY ステートメントは、動的一時配列 d の要素数を 100 に設定します。動的配列 d には、CHAR(10)型の要素が 100 個あります。

動的範囲を持つ一時配列の要素数は、整数定数または整数式として指定できます。たとえば、次の式は有効です。

```
RESIZEARRAY d[top+100];
```

この式の使用例については、“動的範囲を持つ一時配列の使用例” (131 ページ)を参照してください。

動的一時配列の操作

RESIZEARRAY ステートメントは、配列の末尾に要素を追加および削除します。RESIZEARRAY ステートメントを使用して配列のサイズを複数回変更すると、古い要素が削除されると配列の値が削除され、新しい要素が追加されると配列の値が初期化されます。動的範囲制限ありの一時配列を作成し、そのサイズを複数回変更する DS2 プログラムの次のログ出力について考えてみましょう。

```
1        proc ds2;
2        data _null_;
3          method init();
4            dcl double a[*];
5
6            put 'Resizing array a from 0 elements to 10 elements';
7            resizearray a[10];
8            put a[*]=;
9
10           put 'Assigning values to 10 elements of array a';
11           a := (1 2 3 4 5 6 7 8 9 10);
12           put a[*]=;
13
14           put 'Resizing array a from 10 elements to 5 elements';
15           resizearray a[5];
16           put a[*]=;
17
18           put 'Resizing array a from 5 elements to 15 elements';
19           resizearray a[15];
20           put a[*]=;
21         end;
22       enddata;
23       run;
Resizing array a from 0 elements to 10 elements
a[1]=. a[2]=. a[3]=. a[4]=. a[5]=. a[6]=. a[7]=. a[8]=. a[9]=. a[10]=.
Assigning values to 10 elements of array a
a[1]=1 a[2]=2 a[3]=3 a[4]=4 a[5]=5 a[6]=6 a[7]=7 a[8]=8 a[9]=9 a[10]=10
Resizing array a from 10 elements to 5 elements
a[1]=1 a[2]=2 a[3]=3 a[4]=4 a[5]=5  /* 1 */
Resizing array a from 5 elements to 15 elements
a[1]=1 a[2]=2 a[3]=3 a[4]=4 a[5]=5 a[6]=. a[7]=. a[8]=. a[9]=. a[10]=. a[11]=. a[12]=. a[13]=. a[14]=. a[15]=. /* 2
*/
NOTE: Execution succeeded. No rows affected.
24       quit;
```

1 RESIZEARRAY ステートメントによって要素数が 10 から 5 配列要素に減らされると、最後の 5 つの配列値が削除されます。

2 RESIZEARRAY ステートメントによって追加された新しい要素には、新しい値が割り当てられるまで null 値または欠損値が含まれます。

グローバル配列の場合、RESIZEARRAY ステートメントは、グローバル配列を宣言したパッケージ、スレッド、またはデータブロック内のメソッドから配列の範囲のサイズを変更する場合に呼び出すことができます。ローカル配列の場合、RESIZEARRAY ステートメントは、ローカル配列を宣言したメソッドから配列の範囲のサイズを変更する場合に呼び出すことができます。RESIZEARRAY ステートメントの詳細については、“RESIZEARRAY ステートメント” (*SAS DS2 言語リファレンス*) を参照してください。

動的範囲を持つ一時配列の配列割り当ては、変数配列および標準一時配列の場合と同じです。動的範囲を持つ一時配列は、他の DS2 配列を使用できる任意のステートメント(Array 割り当てステートメントを含む)で、配列引数として、および PUT ステートメントで使用できます。

動的範囲を持つ一時配列の使用例

スタックは、動的配列のサイズ変更によってメリットが得られるデータ構造のタイプの例です。スタックを使用すると、コレクション内の要素の数を増減できますが、要素を同じ順序に保つことができます。

標準の DS2 一時配列を使用してスタックを実装する DS2 パッケージは、スタックによって格納できる可能性のある最大の要素のコレクションに合わせてサイズ設定された DS2 配列を静的に宣言する必要があります。たとえば、次の実装では、最悪のケースの 100,000 要素に対して DS2 配列のサイズを静的に設定します。実行中にスタックに最大 10 個の要素しか格納されなかったとしても、100,000 個の要素を格納するのに十分なスペースが静的に割り当てられます。

```
proc ds2;
package stack / overwrite=yes;
  /* Statically size array values for worst case of 100,000 values */
  dcl double values[100000];
  dcl int top;

/* Stack constructor method */
/* Initialize the stack as empty */
  method stack();
    top = 0;
  end;

 /* Add a value to the stack */
 /* Insert value to top of stack */
  method push(double value);
    top = top + 1;
    values[top] = value;
  end;

/* Remove and return value from the stack */
  method pop() returns double;
    dcl double value;
    if (top < 1) then do;
      value = null;
    end;
    else do;
      value = values[top];
      top = top - 1;
    end;
    return value;
  end;
endpackage;
run;
quit;
```

動的範囲を持つ一時配列を使用する DS2 スタックパッケージは、メモリをより効率的に使用できます。次の例では、実行中にスタックに最大 10 個の要素しか格納されていない場合、100 個の要素のスペースのみがスタックによって 割り当てられます。

```
proc ds2;
```

```
/* Declare array values without a specified size */
package stack / overwrite=yes;
 dcl double values[*];
 dcl int top;

/* Stack constructor method */
/* Initialize the stack as empty */
  method stack();
   top = 0;
 end;

/* Add a value to the stack. */
 method push(double value);

  /* Use the RESIZEARRAY statement to expand the array as needed.
   * Memory allocation is typically an expensive operation, so blocks
   * of 100 elements are allocated rather than allocating 1 element
   * at a time. */
   if (dim(values) <= top) then RESIZEARRAY values[top + 100];

/* Insert the value on top of stack */
   top = top + 1;
   values[top] = value;
 end;

 /* Remove and return value from top of stack */
 method pop() returns double;
   dcl double value;
   if (top < 1) then do;
     value = null;
   end;
   else do;
     value = values[top];
     top = top - 1;
   end;
   return value;
 end;
endpackage;

/* use methods from package stack in a data program */
data _null_;
 method init();
   dcl package stack s();
   dcl double value;

   s.push(1);
   s.push(2);
   s.push(3);

   value = s.pop(); put value=;
   value = s.pop(); put value=;

   s.push(4);
   s.push(5);

   value = s.pop(); put value=;
```

```
    value = s.pop(); put value=;
    value = s.pop(); put value=;
  /* array still has 100 elements after the pops */
   end;
  enddata;
  run;
   quit;
```

例のコード 11.2　スタックプログラムからの出力

```
value=3

value=2

value=5

value=4

value=1
```

一時配列の詳細については、“DECLARE ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

HAVING 句を使用した配列の宣言

変数配列または一時配列の宣言ステートメントには、HAVING 句を含めることができます。HAVING 句は、ラベル、出力形式、および入力形式属性を配列に関連付けます。配列が変数配列の場合、HAVING 句は、変数配列によって参照される変数にも関連付けられます。

変数配列によって参照される変数にいつ HAVING 句を適用するかについての決定は、変数参照を含む変数リストがいつ展開されるかによって異なります。通常、VARARRAY ステートメントが処理されると、名前および番号付き範囲変数リストが展開されます。したがって、HAVING 句は、その時点でこれらのリストによって参照されるすべての 変数に適用されます。プログラム内のすべてのステートメントが検査され、プログラム内のすべての 変数が定義された後、名前範囲、名前接頭辞、型、特殊名の変数リストが展開されます。したがって、これらの変数リストによって参照されるすべての変数の HAVING 句は、DS2 プログラム内のすべてのステートメントが検査された後に適用されます。

次のプログラムを考えてみましょう。

```
proc ds2;
data;
  declare double x1 x2 having format 5.0;  /* 1 */
  vararray double x[3] having format 5.2;  /* 2 */
  declare double x3 having format ROMAN5.; /* 3 */
enddata;
run;
quit;
```

1　DECLARE ステートメントが処理されます。変数 x1 と x2 が定義されています。HAVING 句 format 5.0 は変数 x1 および x2 に関連付けられています。

変数	x1	x2
出力形式	5.0	5.0

2 VARARRAY ステートメントが処理されます。デフォルトの変数リスト x1-x3 は x1 x2 x3 に展開されます。HAVING 句 format 5.2 は変数 x1 x2 x3 に関連付けられています。

変数	x1	x2	x3
出力形式	5.2	5.2	5.2

3 DECLARE ステートメントが処理されます。変数 x3 が定義されています。HAVING 句 format ROMAN5.は変数 x3 に関連付けられています。

変数	x1	x2	x3
出力形式	5.2	5.2	ROMAN5.

ここで、変数リストの型が、変数リストの処理が遅延する型に変更された場合に何が起こるかを考えてみましょう。

```
proc ds2;
data;
 declare double x1 x2 having format 5.0;    /* 1 */
 vararray double x[*] x: having format 5.2; /* 2, 4 */
 declare double x3 having format ROMAN5.;   /* 3 */
enddata;
run;
quit;
```

1 DECLARE ステートメントが処理されます。変数 x1 と x2 が定義されています。HAVING 句 format 5.0 は変数 x1 および x2 に関連付けられています。

変数	x1	x2
出力形式	5.0	5.0

2 VARARRAY ステートメントの処理が遅延するか、処理を開始します。接頭辞変数リスト x:は、プログラム内のすべてのステートメントが検査され、すべての変数が定義されるまで展開できません。

変数	x1	x2
出力形式	5.0	5.0

3 DECLARE ステートメントが処理されます。変数 x3 が定義されています。HAVING 句 format ROMAN5.は変数 x3 に関連付けられています。

変数	x1	x2	x3
出力形式	5.0	5.0	ROMAN5.

4 VARARRAY ステートメントは処理を完了します。接頭辞変数リスト x:は x1 x2 x3 に展開されます。HAVING 句 format 5.2 は変数 x1 x2 x3 に関連付けられています。

変数	x1	x2	x3
出力形式	5.2	5.2	5.2

配列割り当て

配列割り当ての概要

DS2 は、:=演算子による配列割り当てをサポートしています。配列または定数リストを割り当てるための構文は次のとおりです。

array:=*array*;

array:=(*constant list*);

配列割り当てでは、*array* は一時配列または変数配列のいずれかになります。

別の配列からの配列割り当て

ある配列を別の配列に割り当てる場合、2 つの配列のデータ型は互換性がある(同じまたは変換可能)必要があります。次元数と各次元の要素の総数は同じである必要はありません。

次のステートメントに示すように、配列 y から配列 x への割り当てを考えてみましょう。

x:=y;

割り当て中に、配列 y の各要素が配列 x の各要素に割り当てられます。たとえば、次のようになります。x[1]=y[1];...x[*n*]=y[*n*];

x[*i*] = y[*i*]を評価するための基本的なアルゴリズムは次のとおりです。最初に y[*i*]が調べられ、欠損(SAS モード)か null (ANSI モード)かが確認されます。

- y[*i*]が欠損または null の場合、x[*i*]には欠損または null が割り当てられます。欠損または null を配列要素に割り当てる決定は、配列のデータ型と、プログラムが SAS または ANSI モードで実行されているかどうかによって異なります。詳細については、7 章, “DS2 による null および SAS 欠損値の処理方法” (61 ページ)を参照してください。
- y[*i*]が欠損でも null でもない場合、x[*i*]と y[*i*]の型が調べられます。
  - y[*i*]の型が x[*i*]の型と異なる場合、y[*i*]は x[*i*]の型に変換されます。
    - y[*i*]の変換が成功した場合、変換の結果が x[*i*]に割り当てられます。
    - y[*i*]の変換が失敗した場合、プログラムが実行されているモードに応じて、欠損または null が x[*i*]に割り当てられます。
  - y[*i*]の型が x[*i*]の型と同じ場合、y[*i*]は x[*i*]に割り当てられます。

配列 x と配列 y の要素数が同じでない場合、配列 y から配列 x にできるだけ多くの要素が割り当てられ、配列 x の残りの要素には null または欠損が割り当てられます。配列 x の長さは割り当てによって変更されません。

次の例では、配列 **x** に 10 の要素があり、配列 **y** に 7 つの要素があり、配列 **y** が配列 **x** に割り当てられています。したがって、配列 **y** の 7 つの要素は配列 **x** の最初の 7 つの要素に割り当てられ、配列 **x** の最後の 3 つの要素には欠損が割り当てられます。

```
proc ds2;
data _null_;
  method init();
    declare double x[10];
    declare double y[7];

    x := (0 0 0 0 0 0 0 0 0 0);
    y := (1 2 3 4 5 6 7);
    x := y;
    put x[*]=;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
x[1]=1 x[2]=2 x[3]=3 x[4]=4 x[5]=5 x[6]=6 x[7]=7 x[8]=. x[9]=. x[10]=.
```

Double 欠損値の特殊なケース

y[*i*]が SAS 特殊欠損値(.Z など)を持つ DOUBLE である場合、DS2 は次のルールに従って割り当て中に SAS 欠損値を保持しようとします。

- x[*i*]が DOUBLE の場合、y[*i*]の SAS 欠損値が x[*i*]に割り当てられます。
- x[*i*]が文字列の場合、y[*i*]の欠損文字表現(たとえば、.Z の場合は Z)が x[*i*]に割り当てられます。

この特殊なケースの処理は、DS2 が SAS モードで、y[*i*]の型が DOUBLE の場合にのみ発生します。

null 値と欠損値の詳細については、7 章, “DS2 による null および SAS 欠損値の処理方法” (61 ページ)を参照してください。配列割り当ての例については、“例: 配列” (*SAS DS2 言語リファレンス*)を参照してください。

変数配列による配列割り当て

変数配列の要素は、PDV 内の変数を参照します。ARRAY 割り当てステートメント x := y では、y の要素の値が要素ごとに x の要素に割り当てられます。x または y が変数配列(vararray)の場合、割り当ては常に x[i]=y[i]です。

変数配列への配列割り当ては、変数配列内の要素(参照)を変更しません。かわりに、配列の要素によって参照される変数のデータが変更されます。同様に、変数配列からの配列割り当てでは、変数配列の要素によって参照される変数内のデータが割り当てに使用されます。

定数リストからの配列割り当て

定数リストから配列に割り当てるには、定数リスト内の定数が配列のデータ型と互換性がある(同じか変換可能である)必要があります。定数リストと配列は、同じ次元数または各次元の要素数が同じである必要はありません。

この Array 割り当てステートメントを想定します。

```
x := (c1 c2 c3 ... cn);
```

定数リストから配列 x への配列割り当て時に、次の展開形式に示すように、定数の各要素が配列 x の各要素に割り当てられます。

```
x[1] = c1;
x[2] = c2;
...
x[i] = ci;
...
x[n] = cn;
```

x[*i*] = c*i* を評価するための基本的なアルゴリズムは次のとおりです。最初に *ci* が調べられ、欠損か null かが確認されます。欠損または null を配列要素に割り当てる決定は、配列のデータ型と、プログラムが SAS と ANSI モードのどちらで実行されているかによって決まります。詳細については、7 章, “DS2 による null および SAS 欠損値の処理方法” (61 ページ)を参照してください。

- c*i* が欠損または null の場合、x[*i*]には欠損(SAS モード)または null (ANSI モード)が割り当てられます。
- c*i* が欠損でも null でもない場合、x[*i*]と c*i* の型が調べられます。
  - c*i* の型が x[*i*]の型と異なる場合、c*i* は x[*i*]の型に変換されます。
    - c*i* の変換が成功した場合、変換の結果が x[*i*]に割り当てられます。
    - c*i* の変換が失敗した場合、プログラムが実行されているモードに応じて、SAS 欠損または ANSI null が x[*i*]に割り当てられます。
  - c*i* の型が x[*i*]の型と同じ場合、c*i* は x[*i*]に割り当てられます。

定数リストと配列 x の要素数が同じでない場合、定数リストから配列 x にできるだけ多くの定数が割り当てられ、配列 x の残りの要素には null または欠損値が割り当てられます。配列 x の長さは割り当てによって変更されません。

次の例では、7 つの double 要素を持つ配列 **x** に、5 つの定数を持つ定数リストが割り当てられます。定数リスト内の 5 つの定数は、配列 **x** の最初の 5 つの要素に割り当てられます。プログラムが SAS モードで実行されている場合、SAS 数値欠損値が配列 **x** の最後の 2 つの要素に割り当てられます。SAS 数値欠損値も、型変換の結果として要素 4 に割り当てられます。定数.Z は SAS の特殊数値欠損値であり、その値は出力に保持されます。

```
proc ds2;
```

```
data _null_;
  method init();
    declare double x[7];
    x := (1 '2' 3.3 '' .Z);
    put x[*]=;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
x[1]=1 x[2]=2 x[3]=3.3 x[4]=. x[5]=Z x[6]=. x[7]=.
```

注: 定数リスト内の要素の型は、すべての要素の型が割り当てられた配列の型に変換可能である限り、異種であってもかまいません。

次に、SAS モードで実行されているプログラムの定数リストから配列を割り当てる別の例を示します。

```
declare char(2) a[2, 3];
  ...
a := (('aa' 'bb' 'cc')('dd' 'ee' ''));
```

前述の割り当てステートメントの後の配列 **a** の要素は、次のようになります。

'aa'	'bb'	'cc'
'dd'	'ee'	''

a は文字配列であるため、空の定数は SAS 文字欠損値として出力されます。

同じ要求が ANSI モードでどのように処理されるかについては、“存在しないデータの読み取りおよび書き込み時の ANSI モード動作の概要” (65 ページ)を参照してください。

配列引数

配列割り当ての概要

DS2 配列は、引数として DS2 メソッドに渡すことができます。DS2 配列は、常にメソッドへの参照によって渡されます。DS2 配列は値によって渡すことはできません。つまり、配列のコピーを引数としてメソッドに渡すことはできません。DS2 配列引数は、配列パラメーターと一致させるために、配列パラメーターで指定されたものと同じ型である必要があります。配列引数は、別の型への暗黙的な型変換をサポートしていません。DS2 配列は、範囲制限ありの配列パラメーター(a[8]など)または範囲制限なしの配列パラメーター(a[*]など)として渡されます。

配列パラメーターの定義

DS2 メソッドは、配列パラメーターを使用して定義できます。配列のタイプ(一時または変数)は、パラメーター定義で指定されます。次の表は、さまざまなタイプのパラメーターを定義するための構文を示しています。

パラメーターのタイプ	パラメーターの構文	例
スカラーパラメーター	*data-type parameter-name*	double x
一時配列パラメーター	*data-type parameter-name* [*bounds*]	double x[5]
変数配列パラメーター	**VARARRAY***data-type parameter-name* [*bounds*]	vararray double x[2,4]

配列引数のデータ型とタイプは、配列パラメーター定義で指定された型と種類と正確に一致する必要があります。DS2 は、配列引数を別の種類またはデータ型に変換しません。たとえば、パラメーターが倍精度の一時配列として定義されている場合、引数は倍精度の一時配列でなければなりません。倍精度の変数配列または整数の一時配列が、倍精度の一時配列パラメーターの引数として渡されると、エラーが発生します。

次の DS2 プログラムは、配列パラメーターを持つメソッドの定義と、配列引数を使用したメソッドの呼び出しを示しています。

```
proc ds2;
data _null_;
  declare  double x i;
  declare  double y[4];
  vararray double z[4];

  method m(double u, double v[4], vararray double w[4]);
    do i = 1 to 4;
      v[i] = u;
      w[i] = u;
    end;
  end;

  method init();
    m(x, y, z);     /* call method m */
    put y[*]=;
    put z[*]=;
  end;
 enddata;
run;
quit;
```

範囲制限ありの配列パラメーター

範囲制限ありの配列パラメーターは、配列引数の要素にアクセスするための明示的な範囲情報を提供します。次に例を示します。

```
method m(double a[4]);

method m(vararray double a[5:10,3:6]);
```

範囲制限ありの配列パラメーターは、配列引数の次元に関係なく、要素数が同じ任意の DS2 配列引数と一致します。たとえば、範囲制限ありの配列パラメーター a[2,4]は、配列 a1、a2、および a3 にそれぞれ 8 つの要素があるため、配列引数 a1[8]、a2[11:12,11:14]、および a3[2,2,2]と一致します。配列引数の次元が配列パラメーターと異なる場合、配列パラメーターの要素は配列引数の対応する要素にマッピングされます。このマッピングは、行優先順を使用して、配列内の要素の位置に基づいています。たとえば、配列パラメーター a1[2,1]は 8 要素配列の 5 番目の要素にアクセスするため、a1[5]、a2[12,11]、および a3[2,2,1]にマッピングされます。

範囲制限なしの配列パラメーター

範囲制限なしの配列パラメーターは、対応する配列引数の明示的な範囲情報を提供しません。次に例を示します。

```
method m(double a[*]);
```

配列範囲のアスタリスク(*)は、パラメーター a が範囲制限なしの配列パラメーターであることを指定します。

範囲制限なしの配列パラメーターは、配列引数の要素数や次元に関係なく、任意の DS2 配列引数と一致します。DS2 メソッドでは、対応する配列引数が多次元配列であっても、配列パラメーターは 1 次元配列として扱われます。範囲制限なしの配列パラメーターは、行優先順を使用して多次元配列にマッピングされます。2x3 配列 a[2, 3]を考えてみましょう。

```
1   2   3
4   5   6
```

配列 a が範囲制限なしの配列パラメーター b としてメソッドに渡される場合、b は 6 要素の 1 次元配列としてアクセスされます。

```
1   2   3   4   5   6
```

配列パラメーター b の要素にアクセスすると、配列 a の要素にアクセスすることに注意してください。これは、配列 a が参照によって DS2 メソッドに渡されるためです。次に例を示します。

```
method m(double b[*]);
  b[1] = 10;        /* assigns 10 to a[1, 1] */
  b[2] = 20;        /* assigns 20 to a[1, 2] */
  b[3] = 30;        /* assigns 30 to a[1, 3] */
```

```
  b[4] = 40;        /* assigns 40 to a[2, 1] */
  b[5] = 50;        /* assigns 50 to a[2, 2] */
  b[6] = 60;        /* assigns 60 to a[2, 3] */
end;

method init();
  declare double a[2, 3];
  m(a);
end;
```

a[i]の形式の配列式(a は範囲制限なしの配列パラメーター)では、インデックス i の範囲制限チェックが実行時に実行されます。配列パラメーターのインデックスが配列引数の範囲制限を超えている場合、DS2 はエラーを発行し、配列式は NULL または欠損と評価されます。

注: 範囲制限なしの配列パラメーターは、範囲制限ありの配列パラメーターの引数として使用できません。

配列の次元をクエリする方法

次の関数を使用して、配列に関する次元情報を取得できます。各関数の詳細については、"DS2 関数" (*SAS DS2 言語リファレンス*)を参照してください。

DIM(*a*)
　配列 *a* の最初の次元の要素数を返します

DIM(*a*, *n*)
　配列 *a* の次元 *n* の要素数を返します

LBOUND(*a*)
　配列 *a* の最初の次元の下限を返します

LBOUND(*a*, *n*)
　配列 *a* の次元 *n* の下限を返します

HBOUND(*a*)
　配列 *a* の最初の次元の上限を返します

HBOUND(*a*, *n*)
　配列 *a* の次元 *n* の上限を返します

NDIMS(*a*)
　配列 *a* の次元数を返します

どのクエリ関数でも、配列引数 **a** は一時配列または変数配列にすることができ、次元引数 **n** は整数値に評価される式にする必要があります。次の例は、これらのクエリ関数を示しています。

```
do i = 1 to dim(a1);
  put a1[i];
end;
```

```
numelems = 0;
do i = 1 to ndims(a2);
  numelems = numelems + dim(a2, i);
end;

do i = lbound(a2, 1) to hbound(a2, 1);
  do j = lbound(a2, 2) to hbound(a2, 2);
    do k = lbound(a2, 3) to hbound(a2, 3);
      sum = sum + a2[i,j,k];
    end;
  end;
end;
```

配列の次元外の次元値を使用して配列関数が呼び出された場合、ランタイムエラーが発生し、関数は NULL 整数値を返します。

配列の内容を書き込む方法

DS2 PUT ステートメントを使用して、配列の個々の要素または配列のすべての要素を書き込むことができます。

個々の配列要素を書き込む構文は次のとおりです。

PUT *array-name*[*element*]<=>;

配列のすべての要素を書き込む構文は次のとおりです。

PUT *array-name*[*]<=>;

PUT ステートメントは、一時配列と変数配列の要素を書き込むことができます。配列のすべての要素が *array-name*[*]構文で書き込まれる場合、配列のすべての要素は同じ出力形式で書き込まれます。つまり、*array-name*[*]で異なる配列要素に対して異なる出力形式を指定することはできません。

次の例は、配列の内容の PUT ステートメントの出力を示しています。

```
proc ds2;
data _null_;
  vararray varchar(10) x[10];
  declare  double     y[2,2,2];
  method init();
    x[1] = 'a';
    do i = 2 to dim(x);
      x[i] = x[i-1] || x[1];
    end;
    put 'X:' x[*];

    y := (10 20 30 40 50 60 70 80);
    put 'Y:' y[*]=;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
X: a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa
Y: y[1,1,1]=10 y[1,1,2]=20 y[1,2,1]=30 y[1,2,2]=40 y[2,1,1]=50 y[2,1,2]=60 y[2,2,1]=70 y[2,2,2]=80
```

詳細については、“PUT ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

12

DS2 varlist

DS2 varlist の概要

varlist は、異種変数をグループ化するためのデータ構造です。varlist は、DS2 ビルトインデータ型のグローバルなスカラー変数への一連の参照で構成されます。参照される変数は、次のいずれかにすることはできません。

- ローカル変数
- 配列または別の varlist
- ビルトインまたはユーザー定義のパッケージ型の変数

varlist は、ローカルメソッド、パッケージメソッド、および varlist をパラメーターとして受け取る DS2 ビルトイン関数に渡すことができます。パッケージは、ユーザー定義パッケージと DS2 事前定義パッケージのいずれかです。

DS2 は、名前付き varlist と名前のない varlist をサポートしています。名前付き varlist は、VARLIST ステートメントで作成されます。名前のない varlist は、使用時に作成されます。それ以外は、機能は同じです。

配列と同様に、varlist は DS2 プログラムの実行中だけ存在します。ただし、参照された変数の内容は 1 つ以上の結果セットに保存される場合があります。

varlist と配列の類似点

varlist には、変数配列と同様の機能があります。varlist には次の特徴があります。

- 名前付き varlist は、VARLIST ステートメントを使用して作成されます。参照する変数の名前は角かっこ([ ])で指定します。(変数配列は VARARRAY ステートメントで作成され、配列範囲は角かっこ内に指定されます。)
- グローバルスコープで宣言する必要がある
- 要素には添字演算子を使用してアクセスします。たとえば、式 a[2]は、a という名前の varlist または配列の 2 番目の要素を指定します。
- ドット表記を使用して(配列構文を使用して)アクセスできます。次に例を示します。

  ```
  pkgvar.a
  pkgvar.a[2]
  ```

  最初のコードは、a という名前の varlist を参照します。2 番目は、varlista の 2 番目の要素を参照します。詳細については、“DS2 パッケージ内のドット表記” (166 ページ)を参照してください。
- THIS 式で使用できます。次に例を示します。

  ```
  this.a
  ```

varlist と配列の違い

- varlist には、異種の型の変数が含まれています。
- varlist は常に 1 次元であり、下限は 1 です。上限は、VARLIST ステートメントの変数リストまたは[*list-of-variables*]演算で指定された変数名の数をカウントすることによって決定されます。配列の下限は柔軟で、配列は多次元にすることができます。
- varlist と配列は、データ構造でデータ値を設定および取得する方法が異なります。
  - varlist 要素のデータ値は、添字演算子と VSETVALUE 関数を使用して設定されます。配列要素のデータ値は、割り当てステートメントで添字演算子を使用して設定できます。たとえば、va が vararray で、vl が varlist の場合、次のステートメントは、変数配列 va と varlist vl の最初の要素をデータ値 100 に設定します。

```
va[1] = 100;
VSETVALUE(vl[1], 100);
```

- varlist 要素のデータ値は、VGETVALUE 関数で添字演算子を使用して取得する必要があります。配列要素のデータ値は、添字演算子のみを使用して取得できます。たとえば、va が vararray で vl が varlist の場合、次のステートメントは、vararray va と varlist vl の最初の要素を取得し、データ値を変数 y に割り当てます。

```
y = va[1];
VGETVALUE(vl[1], y);
```

詳細については、"VSETVALUE 関数" (*SAS DS2 言語リファレンス*)および "VGETVALUE 関数" (*SAS DS2 言語リファレンス*)を参照してください。

- 変数配列では、添字演算子を使用して取得した要素値を、同じステートメント内の後続の演算で使用できます。たとえば、次のように、取得した値を算術演算で使用し、その結果を引数としてメソッド呼び出しに渡すことができます。

```
compute(va[1] + 100);
```

varlist では、添字演算子と VGETVALUE 関数を使用して取得した要素値は、同じステートメント内の後続の演算では使用できません。取得した値に対するその他の演算は、後続のステートメントで実行する必要があります。次に例を示します。

```
vgetvalue(vl[1], var);
compute(var + 100);
```

- OF 演算子を使用して varlist を指定することはできません。OF 演算子で変数のリストを指定するには、省略変数リスト構文 (29 ページ)を使用する必要があります。.

例: 名前付き varlist の作成

名前付き varlist を作成する例を次に示します。varlist は変数情報関数 VNAME および VTYPE に引数として渡され、varlist 内の変数に関する name と type のメタデータを取得します。

```
proc ds2;
data _null_;
  vararray double x[5];
  dcl bigint xyz;
  dcl date xanadu;

  varlist t1 [x:];                /* 1 */

  method printNames(varlist v);       /* 2 */
    dcl varchar(100) name type;
    dcl bigint i;
    do i = 1 to dim(v);             /* 3 */
      name = vname(v[i]);
```

```
      type = vtype(v[i]);
      put name= type=;
    end;
  end;

  method init();
    printNames(t1);              /* 4 */
  end;
enddata;
run;
quit;
```

1 VARLIST ステートメントは、varlist t1 の作成を指定し、名前接頭辞 x:を使用して使用可能なグローバル変数から選択するように指定します。

2 varlist を入力として受け取る printNames メソッドが定義されています。varlist は、メソッドの本体で v と呼ばれます。このメソッドは、ローカル変数 name、type、および i も定義します。

3 メソッドの次に、DO ステートメントを DIM 関数とともに使用して、varlist v の変数を反復処理します。変数ごとに、VNAME 関数と VTYPE 関数を使用してメタデータが取得され、ローカル変数 name と type に割り当てられます。PUT ステートメントは、変数 name と type の値を SAS ログに出力します。

4 varlist t1 は、INIT()メソッドのメソッド引数として printNames メソッドに渡されます。

SAS ログに書き込まれる結果は次のとおりです。

```
name=x1 type=double
name=x2 type=double
name=x3 type=double
name=x4 type=double
name=x5 type=double
name=xyz type=bigint
name=xanadu type=date
```

詳細については、“VARLIST ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

名前のない varlist

名前のない varlist は、VARLIST ステートメントで varlist を明示的に宣言せずに、関数呼び出しまたはメソッド呼び出しの引数として渡されます。名前のない varlist は、[*list-of-variable-names*] 演算子を用いて、使用時に作成されます。

名前のない varlist を DS2 プログラムで使用する方法の例を次に示します。

```
proc ds2;
data _null_;
  dcl double a b x y z;

  method init();
```

```
    dcl package hash h([a b], [x y z]);
  end;
enddata;
run;
quit;
```

このプログラムは、コンストラクターメソッドで 2 つの varlist をハッシュパッケージに指定します。最初の varlist には、変数 a と b が含まれます。2 番目の varlist には、変数 x、y、z が含まれます。

前述のプログラムは、次のプログラムと同じ機能を持っています。

```
proc ds2;
data _null_;
  dcl double a b x y z;
  varlist keyvars [a b];
  varlist datavars [x y z];

  method init();
    dcl package hash h(keyvars, datavars);
  end;
enddata;
run;
quit;
```

このプログラムは、VARLIST ステートメントを使用して 2 つの varlist を定義します。変数 a と b を含むグループ化には、keyvars という名前が割り当てられています。変数 x、y、z を含むグループ化には、datavars という名前が割り当てられます。

どちらのプログラムも同じ varlist を作成します。ハッシュパッケージは、両方のプログラムでまったく同じデータ構造を受け取ります。違いは、2 番目のプログラムが varlist を変数として宣言し、varlist 変数をハッシュパッケージのコンストラクターメソッドに渡すことです。最初のプログラムでは、varlist 変数が宣言されていません。

名前のない変数リスト構文が式で使用されると、名前のない varlist の変数を参照する名前のないデータ構造がグローバルスコープで作成されます。名前のない varlist は、名前のない varlist が指定されたステートメント以外のステートメントからはアクセスできません。したがって、他の式で再利用することはできません。

名前のない varlist をパラメーターとして受け取る DS2 事前定義パッケージの他のメソッドの例は、事前定義 DS2 ハッシュパッケージの KEYS メソッドと、DS2 事前定義 SQLSTMT パッケージの BINDPARAMETERS メソッドです。詳細については、*“KEYS メソッド” (SAS DS2 言語リファレンス)*および*“BINDPARAMETERS メソッド” (SAS DS2 言語リファレンス)*を参照してください。

例: varlist のデータ値の設定

varlist の要素の値を設定するには、添字演算子を VSETVALUE 関数とともに使用します。

VSETVALUE 関数を使用し、名前のない varlist 内の要素のデータ値を設定する“例: 名前付き varlist の作成”の修正バージョンを次に示します。

```
proc ds2;
data _null_;
 vararray double x[5];
 dcl bigint xyz;
 dcl date xanadu;

 method addValues(varlist v);          /* 1 */
  dcl varchar(100) type;
  dcl integer i;

  do i = 1 to dim(v);
   type = vtype(v[i]);

   select(type);
    when('double') vsetvalue(v[i], 100 * i);
    when('bigint') vsetvalue(v[i], 100000000000000);
    when('date')   vsetvalue(v[i], date'2023-03-27');
   end;

  end;
 end;

 method printValues(varlist v);        /* 2 */
  dcl varchar(100) name type value;
  dcl integer i;

  do i = 1 to dim(v);
   name = vname(v[i]);
   type = vtype(v[i]);
   vgetvalue(v[i], value);

   put 'printValues ' name= type= value=;
  end;
 end;

 method init();          /* 3 */
  addValues([x:]);
  printValues([x:]);
 end;

enddata;
run;
quit;
```

1 varlist を入力として受け取る addValue メソッドが定義されています。

addValue メソッドは、次のタスクを実行します。

- ローカル変数の type と i を定義します。
- varlist の変数を反復処理して、各変数の型メタデータを取得します。
- 変数ごとに取得した型の値を使用して、変数の型に従って各変数にデータ値を割り当てます。VSETVALUE 関数は、変数のデータ値を設定します。

2 varlist を入力として受け取る printValues メソッドが定義されています。このメソッドは、ローカル変数 name、type、value、i を定義し、varlist を反復処理して、VGETNAME、VGETTYPE、および VGETVALUE 関数を使用して変数の値を取得します。ローカル変数は PUT ステートメントに渡されます。

3 INIT()メソッドでは、名前接頭辞の省略形 x:が[*list-of-variable-names*]演算子で指定され、メソッド引数として addValues メソッドと printValues メソッドに渡されます。[x:]演算子は、名前が文字 x で始まるすべてのグローバル変数を指定する名前のない varlist を作成し、addValues メソッドと printValues メソッドで処理します。

ログに書き込まれる結果は次のとおりです。

```
printValues  name=x1 type=double value=100
printValues  name=x2 type=double value=200
printValues  name=x3 type=double value=300
printValues  name=x4 type=double value=400
printValues  name=x5 type=double value=500
printValues  name=xyz type=bigint value=100000000000000
printValues  name=xanadu type=date value=2023-03-27
```

例: varlist のデータ値の取得

varlist のデータ値の取得の概要

varlist 内の要素のデータ値を取得するには、添字演算子を VGETVALUE 関数とともに使用します。

このセクションでは、添字演算子と VGETVALUE 関数の使用例を 2 つ示します。1 つは名前付き varlist を使用し、もう 1 つは名前のない varlist を使用します。

プログラム例では次のことを行います。

- 入力データセットを読み取る
- VGETVALUE 関数を使用して、入力データの各行の変数のデータ値を取得する
- データ値を処理して数値欠損値を発見および分析する
- 2 つの列を追加して出力データセットを作成する

出力データセットの追加の新しい列は次のとおりです。

- sum: 現在の行の非欠損数値の合計が含まれる
- mean: 現在の行の非欠損数値の平均が含まれる

入力データ

プログラムは、indata という名前の入力データセットを使用します。入力データセットは次のとおりです。

```
Obs    v1    v2    v3    v4    v5

1    100   100   100   100   100
2      1     2     3     4     5
3     18     4     9    16    24
```

どちらのプログラムも、次の入力データセットを変更せずに処理できます。

```
     first_   last_
Obs    name     name      score1   score2   score3   score4   score5   score6

1    Mary     Brown       97       95       92       88       99       76
2    James    Smith       88       76       89       90       73       92
3    Linda    Johnson     77       73       68       95       82       78
4    Robert   Williams    56       99       82       76       90       88
```

名前付き varlist の例

このプログラムは、名前付き varlist を使用して、入力テーブルの値を取得します。

```
proc ds2;
data outdata / overwrite=yes;

   varlist v [ _all_ ];              /* 1 */

   dcl double sum mean;

   method run();
      dcl double value nvalues;
      dcl int i rc;

      set indata;

      sum = 0;                       /* 2 */
      nvalues = 0;
      do i = 1 to dim(v);
       if (vname(v[i]) ne 'sum' and vname(v[i]) ne 'mean') then do;
          rc = vgetvalue(v[i], value);
          if (rc = 0 and not missing(value)) then do;
             nvalues = nvalues + 1;
             sum = sum + value;
```

```
        end;
      end;
    end;

      if (0 < nvalues) then do;     /* 3 */
      mean = sum / nvalues;
    end;
  end;
enddata;
run; quit;
```

1 VARLIST ステートメントは、このデータブロックのすべてのグローバル変数を参照する v という名前の varlist を作成します。_all_変数リストの指定には、SET ステートメントで指定されたデータセットから読み取られる変数と、このデータブロックのグローバルスコープで宣言された sum 変数と mean 変数が含まれます。

2 これらのステートメントは、VGETVALUE 関数を使用して各変数の値を取得し、現在の入力行からすべての 非欠損数値の合計を計算します。

3 これらのステートメントは、現在の入力行からすべての 非欠損数値の平均を計算します。

名前のない varlist の例

名前のない varlist は、varlist パラメーターが必要なメソッド呼び出し式への引数としてのみ有効です。この例では、名前付き varlist の例の DATA プログラムを変更して、varlist パラメーターが必要なメソッドを持つようにします。

```
proc ds2;
data outdata / overwrite=yes;

  dcl double sum mean;

  method compute(varlist v);
    dcl double value nvalues;
    dcl int i rc;

  /*
   * Compute the same of all the non-missing numeric values
   * from the current input row.
   */
    sum = 0;
    nvalues = 0;
    do i = 1 to dim(v);
      if (vname(v[i]) ne 'sum' and vname(v[i]) ne 'mean') then do;
        rc = vgetvalue(v[i], value);
        if (rc = 0 and not missing(value)) then do;
          nvalues = nvalues + 1;
          sum = sum + value;
        end;
      end;
    end;
```

```
      /*
       * Compute the mean of all the non-missing numeric values from
       * the current input row.
       */
      if (0 < nvalues) then do;
         mean = sum / nvalues;
      end;
   end;

   method run();
     set indata;
   /* Create a varlist that references all global
    * variables of this data block. The _all_ variable list
    * specification includes the variables input from the data set
    * specified on the SET statement as well as the sum and mean
    * variables declared in the global scope of this data block.
    */
     compute([_all_]);
   end;
enddata;
run;
quit;

proc print data=outdata; run;
```

OutData データセット

最初の indata データセットを処理した後に両方のプログラムが返す結果は次のとおりです。

Obs	sum	mean	V1	V2	V3	V4	V5
1	500	100.0	100	100	100	100	100
2	15	3.0	1	2	3	4	5
3	71	14.2	18	4	9	16	24

2 番目の indata データセットを処理した後に両方のプログラムが返す結果は次のとおりです。

Obs	sum	mean	FIRST_NAME	LAST_NAME	SCORE1	SCORE2	SCORE3	SCORE4	SCORE5	SCORE6
1	547	91.1667	Mary	Brown	97	95	92	88	99	76
2	508	84.6667	James	Smith	88	76	89	90	73	92
3	473	78.8333	Linda	Johnson	77	73	68	95	82	78
4	491	81.8333	Robert	Williams	56	99	82	76	90	88

演算子の優先順位

DS2 プログラムでの添字演算子とと[*list-of-variable-names*]演算子の優先順位については、“複合式での演算子の優先順位” (102 ページ)を参照してください。

13

DS2 パッケージ

DS2 パッケージの概要

DS2 パッケージは、DS2 プログラムで使用できるメソッドと変数のコレクションです。DS2 パッケージは一連の関連タスクをサポートし、再利用できるように設計されています。

パッケージには、次の 2 種類があります。

ユーザー定義パッケージ
: これらは、任意の目的で属性にデータを格納したり、メソッドにコードを格納したりするために使用できるパッケージです。

  詳細については、“ユーザー定義パッケージ” (177 ページ)を参照してください。

事前定義パッケージ
: これらは、DS2 で事前定義されているパッケージです。

  詳細については、“事前定義された DS2 パッケージ” (180 ページ)を参照してください。

パッケージは、DS2 プログラムの別のパッケージ、スレッド、またはデータブロックのコードで使用できます。これらのプログラミング要素については、“ブロックステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

> ヒント　DS2 パッケージメソッドを FedSQL SELECT または FedSQL CALL ステートメントの関数として呼び出すこともできます。詳細については、“式での DS2 パッケージの使用” (*SAS FedSQL 言語リファレンス*)および“パッケージメソッド式” (95 ページ)を参照してください。

注: DS2 スレッドによって作成されたパッケージインスタンスは、他の DS2 スレッドからはアクセスできず、パッケージインスタンスの属性データはスレッド間で共有されません。スレッドブロックに、パッケージをインスタンス化するグローバル DECLARE ステートメントがある場合、DS2 プログラムによって生成された各 DS2 スレッドは、独自のパッケージインスタンスをインスタンス化します。たとえば、DS2 プログラムが 10 個の DS2 スレッドを生成する場合、メモリには 10 個のパッケージインスタンスがありますが、各スレッドはインスタンス化した 1 つのパッケージインスタンスにしかアクセスできません。

データプログラムでのパッケージの定義と使用

パッケージは、パッケージのインスタンスを作成するために使用されるテンプレートです。パッケージは、パッケージのすべてのインスタンスに 共通する一連の属性とメソッドを定義します。パッケージを定義したら、パッケージ定義を使用して、パッケージのインスタンスを必要な数だけ作成できます。各パッケージインスタンスには、パッケージによって定義された属性の独自のコピーがあります。パッケージはオブジェクト指向プログラミング言語におけるクラスに似ており、パッケージインスタンスはオブジェクトに似ています。パッケージは、“PACKAGE ステートメント” (*SAS DS2 言語リファレンス*)を使用して作成されます。

パッケージの属性にアクセスしたりメソッドを呼び出したりする前に、パッケージのインスタンスを構築し、新しく構築されたパッケージインスタンスを参照する変数を作成する必要があります。パッケージ変数は、“DECLARE PACKAGE ステートメント” (*SAS DS2 言語リファレンス*)を使用して作成されます。

パッケージインスタンスの属性とメソッドには、ドット演算子を使用して、パッケージの外部にあるコードからアクセスします。詳細については、“DS2 パッケージ内のドット表記” (166 ページ)を参照してください。パッケージインスタンスは、DECLARE PACKAGE ステートメントまたは_NEW_演算子のいずれかを使用して作成されます。

パッケージの作成と使用に関する基本情報については、"入門ガイド"、"例による学習: サンプルプログラムの使用"を参照してください。

メソッドでのパッケージの使用

メソッドでパッケージを使用するには、いくつかの方法があります。このセクションでは、パッケージインスタンスをメソッドから返す方法、またはメソッドの引数として使用する方法について説明します。メソッドはパッケージ変数を返すことはできません。

注: このセクションの例では、DS2 プロシジャを使用して DS2 プログラムを SAS Compute Server にサブミットします。

メソッドからパッケージインスタンスを返す

RETURN ステートメントを使用して、メソッドからパッケージインスタンスを返すことができます。次に例を示します。

```
proc ds2;
package mypkg/overwrite=yes;
  method m(double x) returns double;
    return x+99;
  end;
endpackage;

data _null_;
  dcl package mypkg p;
  dcl double x;

  method r() returns package mypkg;
    return _new_ mypkg();
  end;

  method init();
    p = r();
    x = p.m(100);
    put x=;
  end;

  method term();
    x = p.m(200);
    put x=;
   end;

enddata;
run;
quit;
```

この例では、p という名前のパッケージ mypkg のパッケージ変数がデータブロックで宣言されています。ただし、DECLARE PACKAGE ステートメントではコンストラクターを呼び出すためのかっこが指定されていないため、パッケージはまだインスタンス化されていません。メソッド **r** は、_NEW_演算子を使用してパッケージ mypkg のインスタンスを作成し、そのインスタンスを呼び出しプロセスに返します。INIT メソッドはメソッド r を呼び出し、インスタンスをパッケージ変数 p に割り当てます。次に、INIT()メソッドはインスタンスの m メソッドを使用して、変数 x に値を割り当てます。パッケージ変数 **p** はデータブロックのグローバル部分で宣言されているため、グローバルスコープを持ち、ここで TERM メソッドに示されているように、他のメソッドで再び使用できます。

パッケージ変数が INIT()メソッドに対してローカルに宣言されていた場合、TERM メソッドでは使用できませんでした。

DELETE メソッドは、パッケージ変数によって参照されるパッケージインスタンスを破棄し、そのパッケージ変数と、破棄されたパッケージインスタンスを参照する他のすべてのパッケージ変数に NULL を割り当てます。次に例を示します。

```
method init();
  p = r();
  x = p.m(100);
  put x=;
  p.delete();
end;
```

p.delete()が実行されると、**p** によって参照されるパッケージインスタンスが破棄され、**p** に NULL 値が割り当てられます。

この効果は、**p** が INIT メソッドに対してローカルであることを宣言することによって、より簡単に実現できます。

パッケージを引数としてメソッドに渡す

DS2 では、メソッドからパッケージインスタンスを返すだけでなく、パッケージインスタンスをメソッドの引数としてメソッドに渡すことができます。

この例では、グローバル変数 **p2** によって参照されるパッケージインスタンスがメソッド **tp** に渡され、ローカル変数 **p2** によって参照されます。メソッド **tp** の実行中、グローバル変数 **p2** とローカル変数 **p2** の両方が同じパッケージインスタンスを参照します。メソッド **tp** では、新しいパッケージインスタンスが作成され、ローカル変数 **p2** によって参照されるパッケージインスタンス内のパッケージ変数 **p** に割り当てられます。

```
proc ds2;
package mypkg/overwrite=yes;
  method m(double x) returns double;
    return x+99;
   end;
 endpackage;

 package mypkg2/overwrite=yes;
  dcl package mypkg p;
 endpackage;

data _null_;
  dcl package mypkg2 p2();
  dcl double x;

  method tp(package mypkg2 p2);
     p2.p=_new_ mypkg();
  end;

  method init();
    dcl package mypkg p;
    tp(p2);
    p=p2.p;
    x = p.m(100);
    put x=;
   end;
enddata;
run;
quit;
```

パッケージ、スコープ、および有効期間

パッケージ変数のスコープとパッケージインスタンスの有効期間

パッケージ変数は、パッケージインスタンスへの 1 つ以上の参照をメモリに格納します。DECLARE PACKAGE ステートメントは、パッケージ変数を作成し、オプションでパッケージインスタンスを構築します。DECLARE PACKAGE ステートメントがパッケージインスタンスを構築する場合、パッケージインスタンスはパッケージ変数に割り当てられ、パッケージ変数を使用してインスタンスのデータメンバーにアクセスできます。DECLARE PACKAGE ステートメントがパッケージインスタンスを構築しない場合、パッケージ変数は null に初期化されます(何も参照しません)。null を参照するパッケージ変数を使用してデータメンバーにアクセスしたり、パッケージメソッドを呼び出したりしようとすると、致命的なエラーが発生します。割り当てステートメントで_NEW_演算子を使用して、パッケージインスタンスを既存のパッケージ変数に割り当てることができます。_NEW_演算子は、新しいパッケージインスタンスを構築します。

他の変数と同様に、プログラムの実行中にさまざまな値をパッケージインスタンスに割り当てることができます。

パッケージインスタンスの有効期間は、インスタンスが作成されるスコープによって異なります。インスタンスが作成されたスコープを実行が終了すると、パッケージインスタンスは自動的に削除されます。たとえば、メソッドで作成されたパッケージ変数は、メソッドのローカルスコープで作成されます。その結果、スカラー変数と配列変数はメソッドに対してローカルとなり、メソッドが完了すると削除されます。

複数のパッケージ変数が単一のパッケージインスタンスを参照できます。

パッケージインスタンスは、DS2 パッケージ変数がそれを参照している限り、メモリ内に残ります。パッケージインスタンスの有効期間は、インスタンス自体が作成されたスコープを超えて延長できます。たとえば、ローカルスコープのパッケージ変数を使用してメソッド内でインスタンス化されたパッケージインスタンスの有効期間は、グローバルスコープのパッケージ変数にも割り当てられている場合に延長されます。メソッドの実行が終了すると、ローカルパッケージ変数はスコープ外に渡されて破棄されますが、グローバルパッケージ変数は引き続きメソッドで作成されたインスタンスを参照します。パッケージインスタンスを参照するパッケージ変数がなくなると、パッケージインスタンスは自動的に破棄されます。

例: パッケージインスタンスの有効期間

この例は、パッケージインスタンスがどのように作成および破棄されるかを示しています。

```
proc ds2;
 package mypkg/overwrite=yes; /* 1 */
   dcl varchar(50) name;

   /* constructor - automatically invoked when a package instance is
    * created. */
   method mypkg(varchar(50) name);
     this.name = name;
     put 'creating' this.name;
   end;

   /* destructor - automatically invoked when a package instance is
    * destroyed. */
   method delete();
     put 'destroying' this.name;
   end;
endpackage;

 data _null_;
  dcl package mypkg var1('pkga'); /* 2 */

  method init();
    dcl package mypkg var2('pkgb'); /* 3 */
    dcl package mypkg var3('pkgc');

    var1 = var2;                  /* 4 */
  end;                          /* 5 */

  method term();
    var1 = _new_ mypkg('pkgd');    /* 6 */
  end;

 enddata;                        /* 7 */
run;
quit;
```

1 この例では、mypkg という名前のパッケージを作成します。パッケージ mypkg は、パッケージ属性 name、mypkg という名前のカスタムコンストラクターメソッド、および delete()という名前のカスタムコンストラクターメソッドを定義します。mypkg パッケージは、デフォルトのコンストラクターメソッドが実行された後に実行され、コンストラクターパラメーターで 指定された値を name 属性に割り当ててから、name 属性の値を"creating"という単語を前に付けてログに出力します。delete メソッドは、name 属性の値を SAS ログに出力し、自動デストラクターメソッドを呼び出す前に、"destroying"という単語を前に付けます。

2 4 つのパッケージ変数が定義され、インスタンス化されます。最初に、パッケージインスタンス **pkga** が作成され、パッケージ変数 **var1** によって参照されます。

3 INIT メソッドでローカル変数を作成した後の結果は次のとおりです。
   - パッケージインスタンス **pkgb** が作成され、パッケージ変数 **var2** によって参照されます。
   - パッケージインスタンス **pkgc** が作成され、パッケージ変数 **var3** によって参照されます。
4 パッケージ変数 **var2** がパッケージ変数 **var1** に割り当てられると、結果は次のようになります。
   - パッケージインスタンス **pkga** はパッケージ変数によって参照されていないため、破棄されます。
   - パッケージインスタンス **pkgb** は、パッケージ変数 **var1** およびパッケージ変数 **var2** によって参照されます。
   - パッケージインスタンス **pkgc** は、パッケージ変数 **var3** によって参照されます。
5 INIT メソッドが戻ると、メソッドのローカルパッケージ変数(**var2** および **var3**)が削除されます。
   - パッケージインスタンス **pkgb** は、パッケージ変数 **var1** によって参照されます。
   - パッケージインスタンス **pkgc** はパッケージ変数によって参照されていないため、破棄されます。
6 新しいパッケージインスタンス **pkgd** がパッケージ変数 **var1** に割り当てられると、結果は次のようになります。
   - パッケージインスタンス **pkgb** はパッケージ変数によって参照されていないため、破棄されます。
   - パッケージインスタンス **pkgd** は、パッケージ変数 **var1** によって参照されます。
7 プログラムが終了すると、プログラムのグローバル変数(**var1**)が削除されます。
   - パッケージインスタンス **pkgd** はパッケージ変数によって参照されていないため、破棄されます。

次の行が SAS ログに書き込まれます。

```
creating pkga
creating pkgb
creating pkgc
destroying pkga
destroying pkgc
creating pkgd
destroying pkgb
destroying pkgd
```

属性とメソッド

PRIVATE アクセス修飾子は、パッケージ内での内部使用を目的とした属性またはメソッドに使用できます。これにより、ユーザーが直接取得または設定することを意

図した属性のみを公開することで、プログラムの複雑さを管理できます。プライベート属性は、メソッドを呼び出した結果の状態情報を保存するのに役立ちます。ユーザーが直接触れた場合、それ以降の結果が無効になる可能性があります。**nextNumber** メソッドで属性 **min**、**max**、または **sum** を直接変更しないプライベート属性を使用する例を次に示します。

```
proc ds2;
package stats / overwrite=yes;
  /*-- put scratch space in private attributes --*/
  dcl private double min max sum ini;
  /*-- put shared logic in a private method --*/
  private method p_update( double v );
    if missing(v) then return;
     if missing(ini) then do;
       min=v;
        max=v;
        sum=v;
        ini = 1;
     end;
    else do;
      if v < min then min = v;
      if v > max then max = v;
      sum = sum + v;
     end;
    return;
  end;

  method nextNumber( double v );
      put 'in the double method';
    p_update( v );
  end;

  method nextNumber( char(20) c );
    dcl double v;
      v = c;
      put 'in the char method';
    p_update( v );
  end;

  method nextNumber( int i );
    dcl double v;
      v = i;
      put 'in the int method';
    p_update( v );
  end;

  method getStats( in_out double min, in_out double max, in_out double sum );
    min = this.min;
    max = this.max;
    sum = this.sum;
  end;
endpackage;

data;
dcl package stats st();
dcl double min max sum;
method run();
```

```
  st.getStats( min, max, sum );
  output;
  st.nextNumber( 5 );
  st.nextNumber( '4.0' );
  st.nextNumber( 2.0 );
  st.getStats( min, max, sum );
  output;
end;
enddata;
run;
quit;
```

パッケージのコンストラクターとデストラクター

コンストラクターとデストラクターは、パッケージインスタンスの構築と破棄の際に使用される特別なパッケージメソッドです。コンストラクターメソッドは、新しく構築されたパッケージインスタンスを初期化します。デストラクターメソッドは、クリーンアップを実行するか、パッケージインスタンスが破棄される前にパッケージインスタンスによって保持されているリソースを解放するか、またはその両方を行います。パッケージのコンストラクターメソッドはクラスと同じ名前を持ち、パッケージのデストラクターは DELETE メソッドです。コンストラクターとデストラクターには戻り値の型がなく、値を返しません。

DS2 は、パッケージのインスタンスが構築されるときに、パッケージのコンストラクターを自動的に呼び出します。パッケージインスタンスがパッケージ変数によって参照されなくなり、破棄されると、DS2 は自動的にパッケージのデストラクターを呼び出します。DECLARE PACKAGE ステートメントでパッケージ変数を作成しても、DS2 はパッケージのコンストラクターを呼び出さないことに注意してください。パッケージのコンストラクターは、パッケージインスタンスが、コンストラクター引数を指定した DECLARE PACKAGE ステートメントまたは_NEW_演算子を使用して構築された場合にのみ呼び出されます。

詳細については、このドキュメントの言語リファレンスセクションで、該当する DECLARE PACKAGE ステートメントと_NEW_演算子を参照してください。

DS2 パッケージ内のドット表記

ドット表記のサポートと使用

ドット演算子を使用してパッケージの外部にあるコードから、パッケージインスタンスの属性にアクセスし、そのメソッドを呼び出します。パッケージの属性には、パッケージのグローバルスカラー変数と配列変数が含まれます。

パッケージの属性には、パッケージインスタンスを参照するパッケージ変数を使用して、パッケージの外部にあるコードからアクセスします。表 13.1 で説明されている構文を参照してください。

表 13.1 パッケージの属性にアクセスするための基本的な構文

属性	構文
スカラー変数	*package-variable*.*scalar-attribute*
配列変数	*package-variable*.*array-attribute*
配列変数の要素	*package-variable*.*array-attribute*\[*index*\] [1]

1 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\)は、構文に角かっこが必要であることを示します。配列の境界は、角かっこ([ ])で囲む必要があります。

前述の例では、式のドット演算子は、*package-variable* によって参照されるパッケージインスタンスから指定された *attribute* にアクセスします。

パッケージのメソッドは、次の構文を使用して、パッケージの外部にあるコードによって呼び出されます。

package-variable.*method(arguments)*

この場合、ドット演算子を含む式は、*package-variable* によって参照されるパッケージインスタンスから指定された *method* を呼び出します。*package-variable* によって参照されるパッケージインスタンスは、暗黙的な THIS 引数をメソッド呼び出しに提供しますが、*arguments* はメソッド呼び出しに明示的な引数を提供します。

ネストされたドット表記を使用して、パッケージの階層内でネストされているパッケージの属性とメソッドにアクセスできます。パッケージ階層内でネストされたパッケージの属性またはメソッドにアクセスするには、次の形式でドット表記をネストした式を使用します。

package-variable1.*package-variable2* ... .*package-variableN*.[*attribute*|*method(arguments)*]

ドット演算子の右側のオペランドがパッケージ変数である属性であると、ドット演算の結果を後続のドット演算の左側のオペランドとして使用できます。ドット演算を連結すると、パッケージの階層内でネストされたパッケージの属性へのアクセスがサポートされます。

ネストされたドット演算では:

- 最初のオペランドはパッケージ変数である必要があり、最後のオペランドは、ネストされたドット演算のチェーンによって参照されるパッケージインスタンスの属性やメソッドである必要があります。
- 間にあるドット演算の結果は、後続のドット演算の左側のオペランドとして使用されます。そのため、間にある各ドット演算の右側のオペランドはパッケージ変数、つまり左側の演算子によって参照されるパッケージインスタンスの属性である必要があります。

ネストされたドット表記を使用する次の簡単なプログラム例について考えてみましょう。

```
proc ds2;
```

```
/* pkga defines a method that assigns a value to package attribute x */
 package pkga / inline;
  dcl varchar(1024) x;
  method pkga();
   x = 'apple';
  end;
 endpackage;

/* pkga is instantiated as an attribute of pkgb as pa */
 package pkgb / inline;
  dcl package pkga pa();
 endpackage;

/* pkgb is instantiated as an attribute of pkgc as pb */
 package pkgc / inline;
  dcl package pkgb pb();
 endpackage;

/* pkgc is instantiated in the INIT method as pc.
Then, x is called as an attribute of the package that is referenced
by package variable pc.pb.pa. */
 data _null_;
  method init();
   dcl package pkgc pc();
   put pc.pb.pa.x;
  end;
 enddata;
 run;
 quit;
```

pc.pb.pa.x は、3 つのドット演算で構成される複合式です。

表 13.2　複合式でのドット演算の説明

処理順序	ドット演算 [1]	説明
1	pc.pb	pb はパッケージ変数であり、pc の属性です。
2	pc.pb.pa	pa はパッケージ変数であり、pc.pb の属性です。
3	pc.pb.pa.x	x は、pc.pb.pa の属性です。

1 ドット式の網掛け部分は、ドット式の左側の演算子です。網掛けでない部分は、右側の演算子です。

ドット表記を使用すると、PUT ステートメントは次のようになります。

- INIT メソッドのローカル変数 pc で参照されるパッケージ pkgc のインスタンスを取得します。
- パッケージ属性 pb で参照されるパッケージ pkgb のインスタンスを、パッケージ pkgc インスタンスから取得します。
- パッケージ pkga のインスタンスを取得します。このインスタンスは、パッケージ pkgb のインスタンスのパッケージ属性 pa で参照されています。

- パッケージ属性 x の値を、パッケージ pkga のインスタンスから取得します。
- 値を SAS ログに書き込みます。

各ドット演算の左側のオペランドがパッケージ変数であるか、パッケージ変数に解決されるドット演算である限り、式で使用できるドット演算の数に制限はありません。DS2 式でのドット演算子の評価方法については、“式の演算子 ” (102 ページ)を参照してください。ドット表記で PUT ステートメントを使用する方法の詳細については、“DOT 演算子式” (81 ページ)を参照してください。

ドット表記は、ユーザー定義の DS2 パッケージのパッケージ属性とパッケージメソッド、および事前定義パッケージのパッケージメソッドへのアクセスをサポートします。

変数を指定できる場所にあるパッケージ属性にアクセスすために、ドット表記を使用できますが、次の場合は除きます。

- パッケージスカラー属性を DS2 メソッド呼び出しの IN_OUT パラメーターへの引数として指定する。
- ユーザー定義の DS2 メソッド呼び出しの配列パラメーターにパッケージからの配列属性を指定する。
- SUM ステートメントの左側の変数を指定する。
- Array 割り当てステートメント(:=) 内。

使用例については、“例: ドット表記を使用したスカラー属性およびメソッドへのアクセス”、“例: ドット表記を使用した配列属性へのアクセス”、“例: ドット表記を使用してパッケージ階層内のデータにアクセスする” (172 ページ)を参照してください。

例: ドット表記を使用したスカラー属性およびメソッドへのアクセス

この例では、CIRCLE という名前のパッケージを作成し、パッケージ変数を使用してパッケージを参照します。

```
proc ds2;
package circle / inline;          /* 1 */
  dcl double diameter;

  method radius() returns double;
    return diameter/2;
  end;

  method area() returns double;
    return 2 * 3.14159 * radius();
  end;
endpackage;

data;
  dcl package circle c();         /* 2 */
  dcl double area;                /* 3 */

method init();
```

```
  c.diameter = 10;              /* 4 */
  area = c.area();
 end;
enddata;
run;
quit;
```

1 circle という名前のパッケージは、PACKAGE ステートメントを使用して作成されます。パッケージ circle は、1 つの属性 diameter と、2 つのメソッド radius()および area を定義します。パッケージ circle 内では、グローバル変数 diameter とメソッド radius()が名前で参照されます。

2 データブロックでは、DECLARE PACKAGE ステートメントがパッケージ変数を宣言し、パッケージ circle のインスタンスを作成します。DECLARE PACKAGE ステートメントは、必要なコンストラクターパラメーター値がかっこ内に指定されている場合、パッケージ変数に加えてパッケージインスタンスを作成します。この場合、circle パッケージのコンストラクターにはパラメーター値は必要ありません。パッケージ変数 C に続く空のかっこは、コンストラクターをトリガーしてインスタンスを作成します。

3 DECLARE ステートメントは、パッケージインスタンスで参照できるように、パッケージ内のパッケージ属性 diameter のスカラー変数を定義します。

4 INIT()メソッドのコードは、ドット表記を使用して、パッケージ変数 c によって参照される circle パッケージインスタンスからデータメンバーにアクセスします。式 c.diameter は、パッケージインスタンスのパッケージ属性 diameter の値を設定します。式 c.area()は、パッケージインスタンスからメソッド area を呼び出します。

例: ドット表記を使用した配列属性へのアクセス

この例は、ドット表記を使用してパッケージの配列属性にアクセスする方法を示しています。

```
proc ds2;
package dynamic_record / inline;          /* 1 */
 dcl double sales[*];
 dcl double profit[*];

 method resize(int n);
  resizearray sales[n];
  resizearray profit[n];
 end;
endpackage;

data _null_;                              /* 2 */
 method compute_totals(package dynamic_record record);
  dcl double total_sales total_profit;
  dcl int i;

  total_sales = 0;
  do i = 1 to dim(record.sales);
   total_sales = total_sales + record.sales[i];
```

```
  end;

  total_profit = 0;
  do i = 1 to dim(record.profit);
    total_profit = total_profit + record.profit[i];
  end;

  put '           ----------    ----------';
  put '   TOTAL ' total_sales dollar12.2 ' ' total_profit dollar12.2;
 end;

 method init();                              /* 3 */
  dcl package dynamic_record record();
  dcl int quarter;

  /* Resize record to hold sales and profit for 4 quarters. */
  record.resize(4);

  record.sales[1] = 1001.50;
  record.sales[2] = 155.25;
  record.sales[3] = 405.00;
  record.sales[4] = 2000.50;

  record.profit[1] = 24.00;
  record.profit[2] = 75.00;
  record.profit[3] = 26.00;
  record.profit[4] = 100.00;

  put ' QUARTER       SALES       PROFIT';    /* 4 */
  do quarter=1 to 4;
     put quarter 8. ' '
        record.sales[quarter] dollar12.2 ' '
        record.profit[quarter] dollar12.2;
  end;

  compute_totals(record);
 end;
enddata;
run;
quit;
```

1 このコードは、2 つの動的一時配列 sales と profit を宣言する dynamic_record という名前のパッケージを定義します。このコードでは、resize という名前のメソッドも定義して、配列内の要素の数を簡単に変更できるようにしています。パッケージメソッドへのすべての 呼び出しには、暗黙的パラメーター(パッケージメソッドの呼び出しに使用されるパッケージのインスタンス)が含まれます。resize パッケージメソッドが呼び出されると、このメソッドは、整数パラメーター n で指定された要素数を持つように、dynamic_record パッケージインスタンスの sales 配列と profit 配列のサイズを変更します。

2 DATA プログラムは、dynamic_record パッケージインスタンスの sales 変数の合計と profit 変数の合計を計算して出力する compute_totals という名前のメソッドを定義します。パラメーター record は、dynamic_record パッケージインスタンスを参照します。このメソッドは、インスタンスの sales 配列の要素の値をローカル変数 total_sales に加算し、インスタンスの profits 配列の要素をローカル変数 total_profits に加算することによって、dynamic_record パッケージインスタンスの

総売上高と総利益を計算します。ドット表記は、パラメーター record によって参照される dynamic_record パッケージインスタンス内の sales 配列と profits 配列の要素にアクセスするために使用されます。

3 dynamic_record パッケージのインスタンスは、INIT メソッドでインスタンス化され、パッケージ変数 record に割り当てられます。この DS2 プログラムによって dynamic_record パッケージのインスタンスが構築されるのは、このときだけです。新しく構築された dynamic_record パッケージインスタンスの sales 配列と profit 配列の要素は 0 です。INIT メソッドは、パッケージ変数 record を指定して dynamic_record パッケージの resize メソッドを呼び出し、dynamic_record パッケージインスタンス内の配列のサイズを変更して、それぞれが 4 つの要素を持つようにします。次に、INIT メソッドは、パッケージ変数 record によって参照される dynamic_record パッケージインスタンス内の sales 配列と profit 配列の 4 つの要素のそれぞれに値を割り当てます

4 INIT メソッドは、配列の要素をループし、その値を出力します。最後に、INIT メソッドはパッケージ変数 record を指定して compute_totals メソッドを呼び出し、dynamic_record パッケージインスタンスの各配列の合計を出力します。

SAS ログに書き込まれる結果は次のとおりです。

```
QUARTER       SALES       PROFIT
      1    $1,001.50      $24.00
      2      $155.25      $75.00
      3      $405.00      $26.00
      4    $2,000.50      $100.00
          ----------    ----------
  TOTAL    $3,562.25      $225.00
```

例: ドット表記を使用してパッケージ階層内のデータにアクセスする

パッケージの階層に存在するスカラー属性とメソッドにドット表記を使用してアクセスする方法を示すプログラム例を次に示します。

```
proc ds2;
package ContactAddress / inline; /* 1 */
  declare varchar(100) street;
  declare varchar(100) city;
  declare varchar(100) state;
  declare varchar(100) zipcode;
endpackage;

package ContactInfo / inline;
  declare package ContactAddress address();
  declare varchar(100) phone;
endpackage;

package CustomerHistory / inline;
  declare int numberOfOrders;
  declare int numberOfReturns;
endpackage;
```

```
package Customer / inline;
  declare varchar(100) name;
  declare package ContactInfo work();
  declare package ContactInfo home();
  declare package CustomerHistory history();
  declare double discount;
endpackage;

data _null_;                              /* 2 */
  method createTestCustomer(package Customer person);
    person.name = 'Jane Doe';
    person.work.address.street = '100 SAS Campus Dr';
    person.work.address.city = 'Cary';
    person.work.address.state = 'NC';
    person.work.address.zipcode = '27513';
    person.work.phone = '(919) 677-8000';
    person.home.address.street = '123 Main St';
    person.home.address.city = 'Apex';
    person.home.address.state = 'NC';
    person.home.address.zipcode = '27502';
    person.home.phone = '(919) 555-5555';
    person.history.numberOfOrders = 4;
    person.history.numberOfReturns = 0;
  end;

  method computeDiscount(package Customer person);
    person.discount = .00;

    /* If customer is located in North Carolina, discount 5%. */
    if (person.home.address.state = 'NC') then
      person.discount = .05;

    /* If customer has at least 10 orders, discount 10%. */
    if (10 <= person.history.numberOfOrders) then
      person.discount = .10;
  end;

  method printDiscount(package Customer person);
    declare varchar(100) first_name;
    declare varchar(100) percentage;
    first_name = substr(person.name, 1, find(person.name, ' ')-1);
    percentage = person.discount * 100 || '%';
    put first_name 'receives a discount of' percentage;
  end;

  method init();                          /* 3 */
    declare package Customer person();
    createTestCustomer(person);
    computeDiscount(person);
    printDiscount(person);
  end;
enddata;
run;
quit;
```

1 プログラム例では、ContactAddress、ContactInfo、CustomerHistory、Customer の 4 つのパッケージが定義されています。パッケージ Customer は、2 つの ContactInfo パッケージ属性を持つものとして定義されます。1 つは顧客の職場の連絡先情報を格納するためのもので、2 つ目は顧客の自宅の連絡先情報を格納するためのものです。パッケージ Customer には、顧客の注文と返品の履歴を保存するための CustomerHistory パッケージ属性もあり、name と discount という 2 つの追加属性を定義します。パッケージ ContactInfo は、連絡先の住所情報を格納するための ContactAddress パッケージ属性を持つものとして定義されています。

2 データブロックは、次の 3 つのメソッドを作成します。

- メソッド createTestCustomer は、単一の引数(パッケージ Customer のインスタンス)を取り、Jane Doe という名前の顧客に関する情報を、パッケージ Customer をルートとするパッケージ階層内のパッケージの属性に割り当てます。ドット表記を使用して、パッケージ Customer から Jane Doe の情報が格納される属性までパッケージ階層を走査します。メソッド createTestCustomer および computeDiscount の呼び出し後にメモリに格納されるパッケージデータの図については、図 13.1 (175 ページ)を参照してください。
- メソッド computeDiscount は、パッケージ Customer とパッケージ変数 person をメソッド引数として指定し、IF 式を指定して、場所と注文履歴に基づいて顧客が価格割引を受ける資格があるかどうかを判断します。式 person.discount は、Customer パッケージのパッケージ属性 discount を参照します。式 person.home.address.state は、ContactAddress パッケージのパッケージ属性 State を参照します。式 person.history.numberOfOrders は、CustomerHistory パッケージのパッケージ属性 numberOfOrders を参照します。
- メソッド printDiscount は、パッケージ Customer とパッケージ変数 person をメソッド引数として指定し、式を使用して値を定義し、ローカル変数 first_name と percentage に割り当てます。変数 first_name の値は、式 person.name をスカラー引数として SUBSTR および FIND 関数に渡すことによって作成されます。変数 percentage の値は、式 person.discount を DS2 数式のスカラー引数として組み込み、結果の値にパーセント記号を追加することによって作成されます。

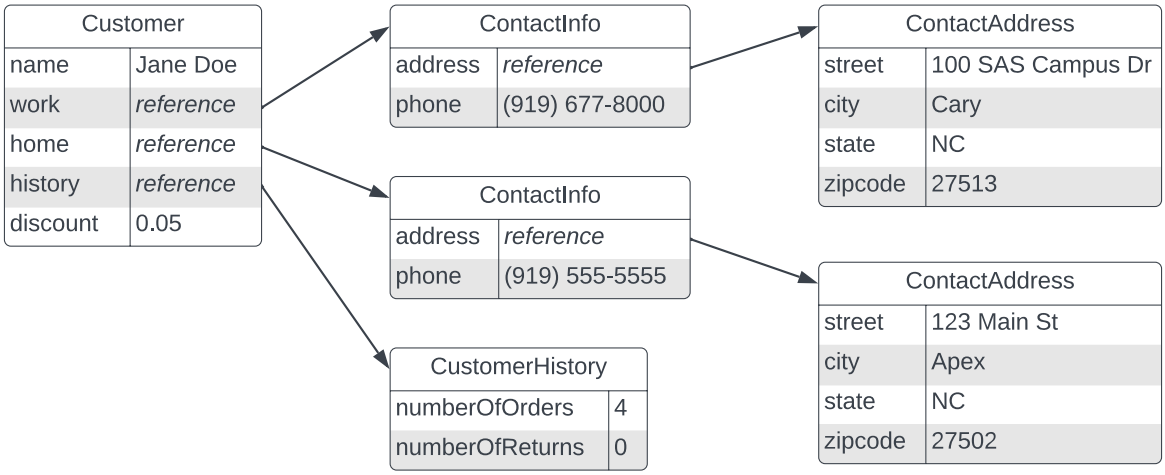
3 INIT()メソッドは、person という名前の単一の Customer パッケージをインスタンス化し、createTestCustomer(person)、computeDiscount(person)、および computeDiscount(person)メソッドを呼び出します。

プログラムからの出力を次に示します。

```
Jane receives a discount of 5%
```

この図は、パッケージ階層内のパッケージ間の関係を示しています。

図 13.1　パッケージ Customer によってインスタンス化されるパッケージ階層



配列パッケージ変数

配列パッケージ変数は、パッケージインスタンスを参照する配列です。配列パッケージ変数は、大規模なデータを扱う場合に便利です。配列パッケージ変数を使用すると、1 つの変数を通じてパッケージの多数のインスタンスを参照できます。配列パッケージ変数の各要素はパッケージのインスタンスを参照し、スカラーパッケージ変数と同等です。

次は、ドット演算式で配列パッケージ変数がどのように参照されるかの例です。

array-package-variable\[*index*\].[*attribute*|*method*(*arguments*)]

array-package-variable は配列の名前です。*Index* は、配列内で参照するパッケージインスタンスを含む要素の位置を示します。配列式のインデックスには、定数、変数、または別の式を指定できます。式内のドット演算子は、指定された *attribute* にアクセスするか、配列パッケージ変数によって参照されるパッケージインスタンスから指定された *method* を呼び出します。

ネストされたドット表記を使用する場合、ドット演算の左側のオペランドがパッケージ変数または配列パッケージ変数の要素であり、右側のオペランドが左側のオペランドのパッケージの属性またはメソッドである限り、スカラーパッケージ変数と配列パッケージ変数要素は交換可能です。

有効なネストされたドット演算の例を次に示します。

```
x[i].y[j].z[1]                    1
a.b[i+100].c[min(x, y, z) - 1].d.e.m()  2
```

1　変数 x は配列パッケージ変数です。属性 y は x[i]の配列パッケージ変数です。属性 z は x[i].y[j]の配列属性です。

2　変数 a、d、および e はパッケージ変数です。変数 b と c は配列パッケージ変数です。メソッド m は、a.b[i+100].c[min(x, y, z) - 1].d.e によって参照されるパッケージインスタンスで呼び出されます。

詳細については、"配列パッケージ変数の宣言と使用"を参照してください。

ドット表記を使用した PUT ステートメントの機能

PUT ステートメントでドット表記を使用する場合、PUT ステートメントを使用してローカルパッケージ変数を出力する場合との次の違いに注意する必要があります。

- ドット表記を使用してパッケージ属性を出力する場合、PUT ステートメントは、変数の DECLARE ステートメントの HAVING 句で変数に指定されている出力形式を無視します。ローカルパッケージ変数を出力する場合、PUT ステートメントは、デフォルトで、DECLARE ステートメントの HAVING 句で指定された出力形式を使用します。パッケージ属性に出力形式を適用するには、PUT ステートメントや PUT 関数で出力形式を指定する必要があります。

  この動作を示す例を次に示します。

```
proc ds2;
/* define package item */
 package item / inline;
 dcl varchar(1024) name having format $upcase.;
 dcl double price having format dollar10.2;
   method item(varchar(1024) name1, double price1);
   name = name1;
   price = price1;
   put 'Package attributes printed locally';
   put name price;
   end;
   endpackage;

/* Instantiate the package and declare variables representing package attributes */
data _null_;
method init();
   dcl package item e1('hammer', 10.00);
   put 'Default output of dot-notated package attributes';
   put e1.name e1.price;
   put 'Dot-notated package attributes specified with a format';
   put e1.name $upcase. e1.price dollar10.2;
   end;
enddata;
run;
quit;
```

  次の出力が SAS ログに書き込まれます。

```
Package attributes printed locally
HAMMER    $10.00
Default output of dot-notated package attributes
hammer 10
Dot-notated package attributes specified with a format
HAMMER   $10.00
```

- PUT ステートメントを使用してパッケージから配列属性を出力するとき、ドット表記を* (アスタリスク)表記と組み合わせることはサポートされません。
-

同種のデータ配列で使用できるのと同じ PUT ステートメント機能が、配列パッケージ変数でも使用できます。PUT ステートメントは、配列パッケージ変数の要素から属性の値を出力するためにサポートされています。たとえば、次のものがサポートされています。

put *array-package-variable*[i].*attribute*;

パッケージ属性の出力の詳細については、"PUT ステートメント" (*SAS DS2 言語リファレンス*)を参照してください。

ユーザー定義パッケージ

ユーザー定義パッケージの概要

作成したメソッドをユーザー定義パッケージに格納できます。これらのパッケージは、メソッドのライブラリと考えることができます。(PACKAGE ステートメントを使用して)パッケージにメソッドを格納したら、DECLARE PACKAGE ステートメントまたは_NEW_演算子を使用してパッケージのインスタンスを作成することにより、メソッドにアクセスできます。

コンストラクター引数を指定した DECLARE PACKAGE ステートメントは、パッケージ変数を作成し、プログラムのコンパイル時にパッケージインスタンスを構築します。この方法は、"コンパイル時のインスタンス化"と呼ばれることがよくあります。

```
declare package complex c();
```

または、コンパイル時にパッケージ変数を宣言し、プログラムの実行中に_NEW_演算子を使用してパッケージ インスタンスを作成することもできます。この方法は、"実行時のインスタンス化"と呼ばれることがよくあります。

```
declare package complex c;

 c = _new_ complex();
```

詳細については、"PACKAGE ステートメント" (*SAS DS2 言語リファレンス*)、"DECLARE PACKAGE ステートメント" (*SAS DS2 言語リファレンス*)、および"_NEW_ 演算子" (*SAS DS2 言語リファレンス*)を参照してください。

注: ユーザー定義パッケージを使用して、パッケージ名をオーバーロードすることにより、事前定義された DS2 パッケージを非表示にすることはできません。ビルトインパッケージ名へのパッケージ参照は、ユーザー定義パッケージではなくビルトインパッケージに解決されます。

MATH という非常に単純なユーザー定義パッケージの例を次に示します。2 つの数値を加算するメソッドが含まれています。

```
package math;
  method add(double x, double y) returns double;
    return x+y;
```

```
  end;
endpackage;
```

次の例では、前の例で作成した MATH パッケージの ADD メソッドを使用して、2 つの数値を加算します。最初に、MATH パッケージが宣言され、インスタンス化されます。次に、ADD メソッドが呼び出され、結果が SUM に割り当てられます。

```
data _null_;
  dcl double sum;
  method init();
    dcl package math f();
    sum = f.add(2,3);
      put 'sum= ' sum;
  end;
enddata;
```

単純なリストパッケージの実装

DS2 パッケージの構文を使用すると、複合構造を作成し、その構造にアクション(メソッド)を割り当てることができます。DS2 パッケージは、別のパッケージから属性またはメソッドを継承できませんが、他のパッケージへの参照を含めることができます。したがって、合成(has-a)関係をサポートできます。

単純だが具体的なリストの例を次に示します。これは、MYELEMENT という特定の要素を保持する MYLIST というリストパッケージから始まります。小さなデータプログラムは、MYLIST および MIELEMENT パッケージを実行します。

```
proc ds2;
/*-- Define a simple list element --*/
package myElement/overwrite = yes;
  dcl package myElement next;
  dcl int d;

  /*-- Custom constructor with parameter --*/
  method myElement( int d );
    this.d = d; /*-- Use “this” to differentiate d’s --*/
  end;

  method print();
    put d=;
  end;

  /*-- A default delete method is implicitly defined --*/
endpackage;


/*-- Define a package to hold the elements --*/
package myList/overwrite=yes;
  dcl package myElement front;
  dcl package myElement back;

  method add( package myElement element );
    if null( front ) then do; /*-- first element --*/
      front = element;
```

```
      back = element;
    end;
    else do; /*-- link in new element --*/
      back.next = element;
      back = element;
    end;
  end;

  method printFront();
    if ^null( front ) then front.print();
    else put 'Front is NULL';
  end;

  method printBack();
    if ^null( back ) then back.print();
    else put 'Back is NULL';
  end;

/*-- A custom destructor – called when references go to zero --*/
  method delete();
    dcl package myElement cur next;
    cur = front;

    /*-- Explicitly empty the list --*/
    do while ( ^null( cur ) );
      next = cur.get_next();
      cur.print();
      cur.delete();
      cur = next;
    end;
  end;
endpackage;
run;

data _null_;
  method run();
    dcl int i;
    dcl package myList ml(); /*-- Instantiated --*/
    dcl package myElement me; /*-- Reference only --*/
    do i = 1 to 10;
       me = _new_ myElement(i); /*-- Constructor with parms --*/
       ml.add(me);
    end;
    ml.printFront();
    ml.printBack();
  end;
enddata;
run;
quit;
```

MYELEMENT パッケージが、内部構造の最初の要素として、自身への参照である next を宣言する方法に注意してください。これは、リンクされた構造をメモリ内に構築するための基盤です。パッケージ自体を参照する機能です。次に、リストが保持する内容(この場合は **d** という単一の整数)を宣言します。2 つのメソッドが定義されています。最初の myElement(int d) はパッケージと同じ名前です。これは、パッケージのコンストラクターであることを 意味します。コンストラクターを定義する

必要はありませんが、定義すると、この場合の **d** の値などの初期化パラメーターを取ることができます。2 番目のメソッド PRINT は、単に **d** で PUT を呼び出して、要素の値を SAS ログに表示します。ユーザー作成の DELETE メソッドが含まれていない場合、デフォルトのデストラクターがコンパイラによって提供されます。

次に、MYLIST パッケージには、タイプ MYELEMENT の 2 つの要素である front と back が含まれています。それらと ADD メソッドを使用すると、単純なリスト構造が得られます。たとえば、MYLIST パッケージには PRINTFRONT および PRINTBACK メソッドが含まれており、これらのメソッドは MYELEMENT パッケージの PRINT メソッドを呼び出します。最後に、あるパッケージが他のパッケージへの参照を保持している場合に推奨されるカスタム DELETE メソッドがあります。他のパッケージへの参照は、参照の循環につながる可能性があります。実行が RUN メソッドを離れると、**ml** によって保持されている MYLIST のインスタンスで DELETE が暗黙的に呼び出され、リスト内の MYELEMENT のすべてのインスタンスで DELETE が明示的に呼び出されます。

事前定義された DS2 パッケージ

事前定義された DS2 パッケージの概要

SAS には、DS2 言語で使用する次の事前定義パッケージが用意されています。

データグリッド
: ネイティブデータグリッドオブジェクトの作成を有効にします。

  詳細については、“データグリッドパッケージの使用” (181 ページ)を参照してください。

FCMP
: DS2 言語内からの FCMP 関数およびサブルーチンの呼び出しをサポートします。

  詳細については、“FCMP パッケージの使用” (191 ページ)を参照してください。

  注: FCMP パッケージは、CAS サーバーではサポートされていません。

ハッシュおよびハッシュ反復子
: 一意のルックアップキーに基づいてデータを迅速かつ効率的に格納、検索、および取得できます。ハッシュパッケージのキーとデータは変数です。キーおよびデータ値には、定数値、テーブルからの値、または式で計算された値を直接割り当てることができます。

  詳細については、“ハッシュパッケージの使用” (194 ページ)および“ハッシュ反復子パッケージの使用” (209 ページ)を参照してください。

HTTP
: HTTP Web サービスにアクセスするための HTTP クライアントを構築します。

  詳細については、“HTTP パッケージの使用” (209 ページ)を参照してください。

JSON
JSON テキストの作成と解析を可能にします。

ロガー
SAS ログ機能への基本的なインターフェイス(開く、書き込み、およびレベルクエリ)を提供します。

詳細については、“ロガーパッケージの使用” (218 ページ)を参照してください。

行列
強力で柔軟な行列プログラミング機能を提供します。SAS/IML 機能の DS2 レベルの実装を提供します。

詳細については、“MATRIX パッケージの使用” (221 ページ)を参照してください。

PCRXFIND と PCRXREPLACE
指定された文字列内の部分文字列を検索したり、部分文字列を置き換えたりする方法を提供します。

SQLSTMT
FedSQL ステートメントを DBMS に渡して実行し、DBMS から返された結果セットにアクセスする方法を提供します。

詳細については、“SQLSTMT パッケージの使用” (232 ページ)を参照してください。

注: SQLSTMT パッケージは、CAS サーバーではサポートされていません。

TZ
ローカルおよび国際的な時刻と日付の値を処理する方法を提供します。

詳細については、“TZ パッケージの使用” (237 ページ)を参照してください。

データグリッドパッケージの使用

データグリッドパッケージの概要

DS2 パッケージのデータグリッドを使用すると、ネイティブデータグリッドオブジェクトを作成できます。パッケージデータグリッドには、オブジェクトをインスタンス化し、列と行を追加、管理、および操作できるようにする一連のメソッドが含まれています。データグリッドパッケージの使用方法については、“DS2 データグリッドパッケージのメソッド、演算子、およびステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

データグリッドについて

データグリッドは、値のテーブルです。すべての列は、10 進数、文字、整数(32 ビット)、長整数(64 ビット)、ブール型のいずれかの値を定義します。各行には、1 つのデータレコードが含まれます。

注: このデータグリッドパッケージのドキュメントでは、倍精度型の列にキーワード *decimal* が使用されています。

たとえば、データグリッドには、すべての顧客の保険契約のレコードを含めることができます。このテーブルは、表 13.3 に示されているテーブルのようになります。

表 13.3　保険契約テーブル

PolicyHolder	PolicyNumber	YearlyPremium
Smyth, Joe	453975R398	439.50
Smyth, Joe	987348P210	132.90
Dupree, Marcel	983092B228	334.00
Dupree, Marcel	274933P412	219.25

または、データグリッドを使用して、この保険契約情報を図 13.2 で表されるように格納することもできます。ここで、表 13.3 に表示されるデータは、データグリッドを使用してテーブルに再構築されます。特定の保険契約番号と保険料は、各保険契約者レコードのデータグリッドにまとめられます。

図 13.2　2 つのデータグリッドを使用した保険契約テーブル

PolicyHolder	Policies
Smyth, Joe	PolicyNumber / YearlyPremium 453975R398 / 439.50 987348P210 / 132.90
Dupree, Marcel	PolicyNumber / YearlyPremium 983092B228 / 334.00 274933P412 / 219.35

データグリッドの JSON 文字列について

データグリッドは、JavaScript Object Notation (JSON)文字列で表されます。データグリッドの JSON 文字列には、次の基本的な形式があります。

[{"metadata":[*column-definitions*]},{"data":[*column-data*]}]

列の定義は、カンマで区切られた名前と値のペアです。

{"*column1-name*":"*data-type*"},{"*column2-name*":"*data-type*"},...

データグリッドの各行のデータは、各値の間にカンマを入れて角かっこで指定します。

[*column1-data*,*column2-data*...]

たとえば、図 13.2 に示されているデータグリッドがシリアル化されている場合、保険契約テーブルは表 13.4 に示されているように表示されます。

表 13.4　シリアル化された保険契約テーブル

PolicyHolder	Policies
Smyth, Joe	[{"metadata":[{"POLICYNUMBER":"string"}, {"YEARLYPREMIUM":"decimal"}]}, {"data":[["453975R398",439.50], ["987348P210",132.90]]}]
Dupree, Marcel	[{"metadata":[{"POLICYNUMBER":"string"},{"YEARLYPREMIUM":"decimal"}]}, {"data":[["983092B228",324.00],[

データグリッド JSON 表現の詳細と例については、“データグリッドの JSON 表現について” (187 ページ)を参照してください。

データグリッドパッケージの宣言とインスタンス化

まず、データグリッドパッケージを宣言してインスタンス化する必要があります。データグリッドパッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

  ```
  declare package datagrid dg;
  dg = _new_ datagrid();
  ```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

  ```
  declare package datagrid dg();
  ```

詳細については、“DECLARE PACKAGE ステートメント: データグリッドパッケージ” (*SAS DS2 言語リファレンス*)、および“_NEW_ 演算子: データグリッドパッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

例: データグリッドパッケージの使用

この DS2 の例では、データグリッドパッケージの逆シリアル化メソッドを使用して、JSON 文字列をデータグリッドに変換し、グリッドから値を取得します。

```
proc ds2;
data _null_;
dcl package datagrid Joe_Smyth();
dcl double myDouble;
dcl varchar(64000) str;

method run();
  put '--------------------';
  str = '[{"metadata":[{"POLICYNUMBER":"string"},
        {"YEARLYPREMIUM":"decimal"}]},
        {"data":[["453975R398",439.50]
, ["987348P210",132.90]]}]';
  Joe_Smyth.deserialize(str);
  str = Joe_Smyth.getValue('PolicyNumber', 2);
  put 'Policy Number: ' str;
  put '--------------------';
end;
enddata;
run;
```

これにより、次の出力が生成されます。

Policy Number:	453975R398

比較演算子

次の演算子のいずれかを、filteredGet 関数や matchCount 関数などの値を比較する関数で使用できます。

演算子	説明
EQ、=、==	等しい
NE、!=、^=、<>	等しくない
GT、>	より大
LT、<	より小
LE、<=	より小か等しい

演算子	説明
GE、>=	より大か等しい

データグリッド、列名、およびインデックス値の指定について

すべてのデータグリッド引数は、DATAGRID 型である必要があります。

メソッドで列が必要な場合、その列は、*col*、*cmpCol*、*colIndex* などの引数名を使用してメソッドシグネチャに表示されます。列名を指定するには、文字列値を使用します。列インデックスを指定するには、数値を使用します。データグリッドの列には、1 から始まる番号が付けられます。

注: データグリッドパッケージは、列名に含まれる先頭および末尾のスペースを削除します。

行処理について

データグリッドの行には、1 から始まる番号が付けられます。

一部のメソッドは、定義された行範囲をオプションの引数として受け入れます。これらの引数は、 *rowStartIndex* および *rowEndIndex* としてメソッドシグネチャに表示されます。行は、*rowStartIndex* から始まり、*rowEndIndex* まで処理されます。

rowStartIndex の 0 (ゼロ)値は、最初から開始することを示します(これは 1 と同じです)。*rowEndIndex* の 0 (ゼロ)は、すべての行を使用することを示します。

ここでは、データグリッドの行処理を説明する例を示します。次のテーブルで考えてみましょう。

	Col_1	Col_2
1	A1	A2
2	B1	B2
3	C1	C2
4	D1	D2
5	E1	E2

	Col_1	Col_2
6	F1	F2
7	G1	G2
8	H1	H2

値*(rowStartIndex = 5*、*rowEndIndex = 7*)が与えられた場合、影響を受ける行は次のとおりです。

	Col_1	Col_2
5	E1	E2
6	F1	F2
7	G1	G2

データグリッドの null 値と欠損値の処理方法について

7 章, “DS2 による null および SAS 欠損値の処理方法” (61 ページ)で説明したように、CHAR 列の null 値に対して検索または比較操作を実行すると、DS2 は常に null 値を返します。これにより、filteredGet や columnCountMatch など、null 値または欠損値を検索または比較するデータグリッドメソッドを使用すると、予期しない結果が生じる可能性があります。これを回避するために、データグリッドパッケージの動作が少し異なります。

次の表は、null 値、欠損値、および文字列文字を相互に比較するときに使用されるデータグリッドロジックを示しています。各交点のセル値は、データグリッドが比較を有効と見なして要求された結果を返すか(**1**)、無効と見なすか(**0**)、または **null** 値を返すかを表します。

次の表は、等式演算に適用されるデータグリッドロジックを示しています。

=	**null 値**	**欠損値**	**文字列文字**
null 値	**1**	**null**	**null**
欠損値	**null**	**1**	**0**
文字列文字	**null**	**0**	**1**

不等式演算の場合、データグリッドロジックは等式演算の逆で動作します。次の表は、不等式演算に適用されるデータグリッドロジックを示しています。

^=	null 値	欠損値	文字列文字
null 値	**null**	**1**	**1**
欠損値	**1**	**0**	**1**
文字列文字	**1**	**1**	**0**

いずれの場合も、null が検索または比較操作に含まれる場合、null は null とのみ等しくなります。逆に、null は null 以外のすべてのものと等しくありません。

データグリッドの JSON 表現について

データグリッドパッケージの構文規則は、標準の DS2 構文規則とは異なることに注意してください。データグリッドパッケージの構文規則は次のとおりです。

< >
　山かっこはオプションのコンテンツを示します。

[ ]
　角かっこは、値の配列を定義するために必要な構文エントリを示します。

{ }
　中かっこは、オブジェクトを定義するために必要な構文エントリを示します。

斜体のテキスト
　斜体のテキストは、指定する引数または値を示します。

...
　省略記号は、省略記号に続く引数または引数のグループを何度でも繰り返すことができることを示します。省略記号とそれに続く引数が山かっこで囲まれている場合、それらはオプションです。

ここでは、データグリッドの JavaScript Object Notation (JSON)構造を示します。

[<{"metadata":[{"*column-1*":"*data-type*"} <...,{"*column-n*":"*data-type*"}>]} , >
　{"data":[< [*row-1-data*] <... , [*row-n-data*] >]}]

- 最初の部分はメタデータと呼ばれます。テーブル内の各列を名前と型で指定します。割り当てられる型は、ブール、10 進数、整数、文字列のいずれかです。

  注: メタデータが定義されている場合は、少なくとも 1 つの列を定義する必要があります。

- 次の部分には、メタデータ部分で指定された列の順序と型に対応するデータの行が含まれています。
- データグリッドに行データが含まれていても、メタデータが含まれていない可能性があります。これは、*ヘッドレス*データグリッドとして知られています。ヘッドレスデータグリッドに関する次の情報に注意してください。

- 最初の非 null データ行によって、文字列または数値を含む列のデータ型が決まります。
- 数値列型は次のように割り当てられます。
  - 列のすべての値が整数の場合、その列には long の数値型が割り当てられます。
  - 列の少なくとも 1 つの値が 10 進数の場合、その列には decimal の数値型が割り当てられます。

  次に例を示します。

  JSON 表現:

  ```
  [{"data":[[null,null,null,null,null],["Str_01",true,null,15,1],
  ["Str_02",false,null,77,2],["Str_03",false,null,93,3.03],
  ["Str_04",true,null,15,4.04],["Str_05",false,null,null,5.05]]}]
  ```

  結果のデータグリッド:

  ```
  row[0000]: string(mv) boolean(mv) decimal(mv) long(m1v) decimal(mv)
  row[0001]: string(Str_01) boolean(true) decimal(mv) long(15) decimal(1.00)
  row[0002]: string(Str_02) boolean(false) decimal(mv) long(77) decimal(2.00)
  row[0003]: string(Str_03) boolean(false) decimal(mv) long(93) decimal(3.03)
  row[0004]: string(Str_04) boolean(true) decimal(mv) long(15) decimal(4.04)
  row[0005]: string(Str_05) boolean(false) decimal(mv) long(mv) decimal(5.05)
  ```

列 5 の少なくとも 1 つのセルに 10 進数値が含まれているため、行 1 と 2 の整数値は 10 進数値に変換されます。

データグリッドの JSON 表現の例

- メタデータとデータを含むテーブルを次に示します。

```
[{
   "metadata" : [
        { "aStringColumn" : "string" },
        { "anIntColumn" : "integer" },
   { "aFloatColumn" : "decimal" },
   { "aBooleanColumn" : "boolean" }]
},{
"data" : [[ "one", 1, 1.11, true ],
        [ "two", 2, 2.22, false ],
        [ "three", 3, 3.33, true ],
        [ "four", 4, 4.44, false ],
        [ "five", 5, 5.55, true ],
   [ null, null, null, null ]]
}]
```

- データのないテーブルを次に示します。

```
[{
   "metadata" : [
        { "aStringColumn" : "string" },
        { "anIntColumn" : "integer" },
   { "aFloatColumn" : "decimal" },
   { "aBooleanColumn" : "boolean" }]
```

```
  },{
  "data" : []
}]
```

- メタデータのないテーブル(ヘッドレスデータグリッド)を次に示します。

```
[{
  "data" : [["Be1","Bd1","Bc1","Ba1"],
        ["Be2","Bd2","Bc2","Ba2"],
        ["Be3","Bd3","Bc3","Ba3"]]
}]
```

- null 値テーブルを次に示します。

```
null
```

データグリッドパッケージのエラー報告

無効なパラメーター

getInt、setValue、getString など、値を取得または設定するメソッドに無効なパラメーターが指定された場合、データグリッドパッケージはエラーをログに記録します。

SAS Micro Analytic Service でデータグリッドパッケージを使用している場合、環境変数を設定することで、これらのデータグリッドの問題を SAS Micro Analytic Service の実行エラーとしてプロモートできます。詳細については、“Promoting DS2 Execution Issues to a SAS Micro Analytic Execution Failure” (*SAS Micro Analytic Service: Programming and Administration Guide*)を参照してください。

エラーがログに記録される状況の例を次に示します。

表 13.5　データグリッドメソッドのエラーをログに記録する条件

オブジェクト	条件	例
列	指定されたインデックス値が無効。たとえば、0 より小さい。	インデックス値が 0 より小さいため、ここでエラーが発生します。 dgA.getValue(-1,2);
	指定されたインデックス値がデータグリッドの列数より大きい。	dgA には 10 列が含まれているため、ここでエラーが発生します。 dgA.getValue(11,2);
	指定された列名が存在しない。	dgA には Date という名前の列が含まれていないため、ここでエラーが発生します。 dgA.getValue('Date',2);

オブジェクト	条件	例
行	指定されたインデックス値が無効。たとえば、0 より小さい。	インデックス値が 0 より小さいため、ここでエラーが発生します。 dgA.getValue('Name',-1);
	指定されたインデックス値がデータグリッドの行数より大きい。	dgA には 10 行が含まれているため、ここでエラーが発生します。 dgA.getValue(1,12);
演算子	必要な演算子がない。	ColA と 200 の間の演算子がないため、ここでエラーが発生します。 dgA.filteredget('ColA',200,'ColB',5,10);
	無効またはサポートされていない演算子が指定されている。	ここでは、除算演算子(/)が無効なため、エラーが発生します。 dgA.filteredGet('ColA','/',200,'ColB',5,10);

範囲外のインデックス

デフォルトでは、データグリッドパッケージで範囲外のエラーが発生すると、ERROR ではなく NOTE としてログに記録されます。この状態の重要度を ERROR にアップグレードできます。これを行うには、SAS_MAS_DATAGRID_INDEX_OOB_IS_ERROR 環境変数を定義し、値を True に設定します。

環境変数の設定については、コンピューティング環境のドキュメントを参照してください。たとえば、SAS Compute Service、SAS Compute Server、SAS Cloud Analytic Services、または SAS Micro Analytic Service などです。

- SAS Viya Platform: Programming Run-Time Servers
- “SAS Cloud Analytic Services: リファレンス” (*SAS Viya プラットフォーム: SAS Cloud Analytic Services*)
- “Configuring Supplemental SAS Micro Analytic Service Environment Variables” (*SAS Micro Analytic Service: Programming and Administration Guide*)

FCMP パッケージの使用

FCMP パッケージの概要

注: FCMP パッケージは、CAS サーバーではサポートされていません。

DS2 言語は、FCMP パッケージを介して FCMP プロシジャで使用可能または作成された関数およびサブルーチンの呼び出しをサポートします。

PACKAGE ステートメントで LANGUAGE=FCMP および TABLE=オプションを使用して、FCMP パッケージを作成します。パッケージを作成したら、FCMP パッケージのインスタンスを宣言します。FCMP パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package fcmp banking;
banking = _new_ fcmp();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package fcmp banking();
```

詳細については、“DECLARE PACKAGE ステートメント: FCMP パッケージ” (*SAS DS2 言語リファレンス*)および“PACKAGE ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

FCMP パッケージの機能

これらは、DS2 言語で FCMP パッケージを使用する機能です。

- スカラー DOUBLE、CHAR、および NCHAR パラメーターを使用して FCMP 関数またはサブルーチンを呼び出し、両方のパラメーターを返します。

  DS2 言語は自動型変換を行うため、ほぼすべての型がサポートされます。例として、TINYINT から DOUBLE への変換があります。詳細については、“型変換の概要” (70 ページ)を参照してください。

- スカラー DOUBLE OUTARGS パラメーターを使用して FCMP 関数またはサブルーチンを呼び出します。FCMP プロシジャの DOUBLE OUTARGS パラメーターは、METHOD ステートメントでは IN_OUT パラメーターとして扱われます。IN_OUT パラメーターの詳細については、“METHOD ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

  注: DS2 は、OUTARGS パラメーターを介して、式ではなく DOUBLE 変数を渡す必要があります。

FCMP パッケージを使用する際の考慮事項と制限事項

- FCMP パッケージは VARARGS 関数呼び出しをサポートしていないため、FCMP プロシジャの VARARGS インターフェイスを使用できません。
- FCMP プロシージャと FCMP パッケージの間で情報が渡されるときに発生するエラーは、常に正しく報告されるとは限りません。たとえば、PACKAGE ステートメントで誤ったテーブル名を指定すると、ログファイルにエラーが書き込まれます。ただし、操作が失敗したことは示されません。
- FCMP パッケージはセッションエンコーディングを想定しており、現在、同じ関数呼び出し内の異なるパラメーター、または複数の関数呼び出し間での同じパラメーターに対して異なるエンコーディングを許可するメカニズムはありません。
- 接続文字列で FCMP ライブラリが配置されているカタログを定義している限り、任意の FCMP ライブラリにアクセスできます。
- 次の FCMP 関数は、DS2 プログラムから呼び出された場合、DS2 プログラムで設定されたシード値を認識しません。

  COMPCOST

  COMPGED

  RAND

  STREAMINIT

  DATA ステップでは、RAND からのすべての乱数はデフォルトで同じストリームから取得されますが、CALL STREAM ルーチンを使用して複数の独立したストリームを作成できます。

  PROC DS2 では、予期しない機能の結果として、FCMP パッケージ内で呼び出される RAND の乱数は、別のストリームから取得されます。再現可能な乱数を取得するには、RAND を呼び出す各 FCMP 関数内で CALL STREAMINIT ルーチンを使用します。独立したストリームを取得するには、RAND を呼び出す各 FCMP 関数内で CALL STREAM ルーチンも使用します。そうすると、DS2 からの結果は DATA ステップおよび将来のリリースと一致します。

  次に例を示します。

```
title1 'Generate 3 Uniform and 3 Normal Pseudorandom Variables' ;
title2 'With and Without FCMP' ;

%let seed = 1234; %put seed=&seed;
%let nobs = 10;   %put nobs=&nobs;

* To get reproducible streams in FCMP, call STREAMINIT in each FCMP function;
* To get independent streams in FCMP, call STREAM in each FCMP function;
proc fcmp outlib=work.funcs.rand;

  function rand_uniform( stream );
    call streaminit( &seed );
    call stream (stream );
    u = rand('uniform' );
```

```
    return (u);
  endsub;

  function rand_normal( stream, mean, std );
    call streaminit( &seed );
    call stream (stream );
    z = rand('normal', mean, std );
    return (z);
  endsub;

run;
quit;


title2 'PROC DS2 Step With and Without FCMP';

proc ds2;
  package pkg / overwrite=yes language='fcmp' table='work.funcs';
  run;
  data ds2(overwrite=yes);
    dcl package pkg randpkg();
    declare double obs u1 u2 u3 z1 z2 z3;
    method run();
      declare double junk;
      junk = streaminit( &seed );
      do obs = 1 to &nobs;
        junk = stream( 1 );
        u1 = rand( 'uniform' );
        u2 = randpkg.rand_uniform( 2 );
        u3 = randpkg.rand_uniform( 3 );
        junk = stream( 4 );
        z1 = rand( 'normal', 100, 20 );
        z2 = randpkg.rand_normal( 5, 100, 20 );
        z3 = randpkg.rand_normal( 6, 100, 20 );
        output;
      end;
    end;
  enddata;
run;
quit;

proc print data=ds2;
run;
```

次のテーブルが生成されます。

アウトプット 13.1　乱数

Generate 3 Uniform and 3 Normal Pseudorandom Variables
PROC DS2 Step With and Without FCMP

OBS	obs	u1	u2	u3	z1	z2	z3
1	1	0.28215	0.91417	0.70279	120.205	81.990	85.187
2	2	0.51896	0.62759	0.82619	52.955	83.050	96.879
3	3	0.34428	0.75150	0.10642	139.520	128.227	105.486
4	4	0.99869	0.26466	0.55477	123.377	112.573	92.993
5	5	0.29200	0.79337	0.34503	62.937	114.600	86.801
6	6	0.50000	0.72109	0.07985	116.885	118.569	71.318
7	7	0.02342	0.93284	0.24532	87.582	151.621	83.920
8	8	0.29646	0.15810	0.51957	71.924	90.825	101.612
9	9	0.30978	0.61928	0.24458	115.001	116.015	75.750
10	10	0.75230	0.45879	0.71742	135.418	74.332	115.295

ハッシュパッケージの使用

ハッシュパッケージの概要

ハッシュパッケージは、迅速なデータの格納と取得のための効率的で便利なメカニズムを提供します。ハッシュパッケージは、一意のルックアップキーに基づいてデータを格納および取得します。一意のルックアップキーの数とテーブルのサイズによっては、ハッシュパッケージのルックアップは、標準形式のルックアップや配列よりも大幅に高速になる場合があります。

DS2 ハッシュパッケージを使用する前に、ハッシュパッケージのインスタンスを定義および構築(インスタンス化)する必要があります。

ハッシュパッケージインスタンスを定義して作成すると、次のような多くのタスクを実行できます。

- データの格納と取得。
- データの置換と削除。
- ハッシュパッケージ内のデータを含むテーブルを生成する。

たとえば、一意の患者番号と体重に対応する数値検査結果を含む大きなテーブルと、患者番号を含む小さなテーブル(大きなテーブルのサブセット)があるとします。一意の患者番号をキーとして、体重値をデータとして使用して、大きなテーブルをハッシュパッケージにロードできます。次に、患者番号を使用して小さなテーブルを反復処理し、体重が特定の値を超えている現在の患者をハッシュパッケージ内で検索し、そのデータを別のテーブルに出力できます。

ハッシュパッケージインスタンスの定義と作成

ハッシュパッケージのインスタンスを作成するには、構築するハッシュインスタンスに関するキー、データ、およびオプションの初期化データを指定します。ハッシュパッケージインスタンスは、構築時に完全に定義することも、構築時と後続の一連のメソッド呼び出しで定義することもできます。

次の例では、ハッシュインスタンス **h1** と **h2** に同じインスタンス定義があります。ハッシュインスタンス **h1** は構築時に完全に定義されますが、**h2** は構築時および一連のメソッド呼び出しによって定義されます。

```
declare package hash h1([key], [data1 data2 data3],
  0, 'testdata', '', '', '', 'multidata');

declare package hash h2();
  method init();
    h2.keys([key]);
    h2.data([data1 data2 data3]);
    h2.dataset('testdata');
    h2.multidata();
    h2.defineDone();
  end;
```

詳細については、“コンストラクターを使用したハッシュインスタンスの定義” (195 ページ)および“メソッド呼び出しを使用したハッシュインスタンスの定義” (196 ページ)を参照してください。

コンストラクターを使用したハッシュインスタンスの定義

コンストラクターは、ハッシュパッケージをインスタンス化し、ハッシュパッケージデータを初期化するために使用できるメソッドです。

コンストラクターを使用してハッシュパッケージインスタンスを作成するには、3 つの異なる方法があります。

- 部分的に定義されたハッシュインスタンスを作成します。

DECLARE PACKAGE HASH *instance*(*hashexp*, {*datasource*' | '\{*sql-text*\}'}, '*ordered*', '*duplicate*', '*suminc*', '*multidata*');

キー変数とデータ変数は、メソッド呼び出しによって定義されます。初期化データを提供するオプションのパラメーターは、前述の DECLARE PACKAGE ステートメント、_NEW_演算子、メソッド呼び出し、またはこれらの組み合わせのいずれかで指定できます。DEFINEDONE メソッドを 1 回呼び出すだけで、定義が完了します。

注: *sql-text* は CAS サーバーではサポートされていません。

- 指定されたキー変数とデータ変数を使用して、完全に定義されたハッシュインスタンスを作成します。

  DECLARE PACKAGE HASH *instance*(\[*keys*\], \[*data*\]
  [, *hashexp*, {'*datasource*' |
  '\{*sql-text*\}'}, '*ordered*', '*duplicate*', '*suminc*', '*multidata*']);

  キー変数とデータ変数は DECLARE PACKAGE ステートメントで定義されます。これは、完全に定義されたインスタンスを作成する必要があることを示しています。後続のメソッド呼び出しで追加の初期化データを指定することはできません。

- 指定されたキー変数のみを使用して、完全に定義されたハッシュインスタンス(キーのみのハッシュインスタンス)を作成します。データ変数はありません。

  DECLARE PACKAGE HASH *instance*(\[*keys*\][, *hashexp*, {'*datasource*' | '\{*sql-text*\}'},
  '*ordered*', '*duplicate*', '*suminc*', '*multidata*']);

  キー変数とデータ変数は DECLARE PACKAGE ステートメントで定義されます。これは、完全に定義されたインスタンスを作成する必要があることを示しています。後続のメソッド呼び出しで追加の初期化データを指定することはできません。

オプションパラメーターの詳細については、“ハッシュパッケージの初期化データの提供” (198 ページ)を参照してください。メソッド呼び出しを使用したオプションパラメーターの定義の詳細については、“メソッド呼び出しを使用したハッシュインスタンスの定義” (196 ページ)を参照してください。

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

メソッド呼び出しを使用したハッシュインスタンスの定義

インスタンスの構築中にハッシュインスタンスが部分的に定義されている場合は、次のメソッドを呼び出すことでインスタンスをさらに定義できます。

KEYS
DEFINEKEY
DATA
DEFINEDATA
DATASET
DUPLICATE
HASHEXP
ORDERED
MULTIDATA
SUMINC
DEFINEDONE

これらのメソッドの詳細については、“DS2 ハッシュおよびハッシュ反復子パッケージの属性、メソッド、演算子、ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

注: DEFINEDONE メソッドの呼び出しによってハッシュインスタンスの指定が完了した後、前述のいずれかのメソッドを呼び出すと、エラーが発生します。

メソッド呼び出しを使用して定義されたハッシュインスタンス **h** の例を次に示します。

```
data _null_;
  declare package hash h(0, 'testdata');
  method init();
    h.keys([key]);
    h.data([data1 data2 data3]);
    h.ordered('descending');
    h.duplicate('error');
    h.defineDone();
  end;
enddata;
```

キー変数とデータ変数の定義

ハッシュパッケージは、一意のルックアップキーを使用してデータを保存および取得します。キーとデータは、ドット表記メソッド呼び出しを使用してハッシュパッケージを初期化するために使用する変数です。

キー変数とデータ変数は、3 つの方法のいずれかで定義できます。

- 変数メソッドの DEFINEDATA および DEFINEKEY を使用します。
- 変数リストメソッドの DATA および KEYS を使用します。
- DECLARE PACKAGE ステートメントで指定されたキーおよびデータ変数リストを使用します。

構築時にハッシュパッケージのインスタンスが完全に定義されていない場合、つまり構築時にキー変数とデータ変数が指定されていない場合は、DEFINEDONE メソッドを呼び出して、ハッシュインスタンスの初期化を完了する必要があります。

次に例を示します。

```
/* Keys and data defined using the implicit variable method */
declare package hash h();
h.definekey('k');
h.definedata('d');
h.definedone();

/* Keys and data defined using the variable list methods */
declare package hash h();
h.keys([k]);
h.data([d]);
h.definedone();

/* Keys and data defined using the variable list constructors */
```

```
declare package hash h([k],[d]);
```

キー変数は、DS2 ビルトイン型(文字、数値、または日時)にする必要があります。データ変数は、DS2 ビルトイン型、あるいはビルトインまたはユーザー定義のパッケージ型のいずれかです。

詳細については、“暗黙の変数と変数リストのメソッド” (199 ページ)を参照してください。

ハッシュパッケージの初期化データの提供

キーとデータに加えて、ハッシュパッケージを初期化するときに、次のオプションパラメーターを指定できます。

- ハッシュテーブルのサイズが 2^n である内部テーブルサイズ(*hashexp*)
- ロードするテーブルの名前(*datasource*)、またはロードするデータを選択するための FedSQL クエリ

  注: FedSQL クエリの使用は、CAS サーバーではサポートされていません。

- データがキーと値の順序で返されるかどうか、またはどのように返されるか(*ordered*)
- テーブルをロードするときに重複キーを無視するかどうか(*duplicate*)
- ハッシュパッケージキーの要約カウントを維持する変数の名前(*suminc*)
- キーごとに複数のデータアイテムを許可するかどうか(*multidata*)

初期化データは、DECLARE PACKAGE ステートメント、_NEW_演算子、メソッド呼び出し、またはこれらの方法の組み合わせで指定できます。

注: DECLARE PACKAGE ステートメントまたは_NEW_演算子でコンストラクターを使用してハッシュパッケージデータを初期化する場合、オプションパラメーターを *hashexp*、*datasource*、*ordered*、*duplicate*、*suminc*、*multidata* の順序で指定する必要があります。これらの位置コンストラクターパラメーターはすべて 、1 組のかっこで囲み、カンマで区切り、*hashexp* パラメーターを除いて、単一引用符で囲む必要があります。オプションパラメーターは位置指定であるため、指定する最後のパラメーターまで、各パラメーターのプレースホルダーを提供する必要があります。使用する必要があるプレースホルダーは、パラメーターによって異なります。プレースホルダーの詳細については、“DECLARE PACKAGE ステートメント: ハッシュパッケージ” (*SAS DS2 言語リファレンス*)または“_NEW_ 演算子: ハッシュパッケージ” (*SAS DS2 言語リファレンス*)を参照してください。次の例では、昇順を指定して重複を置き換えるには、*hashexp* パラメーターのプレースホルダーとして -1、*datasource* パラメーターのプレースホルダーとして空の単一引用符(' ')、*ordered* パラメーターに**'a'**、*duplicate* に**'replace'**を使用する必要があります。*duplicate* パラメーターは最後に指定されたものであるため、*suminc* および *multidata* パラメーターにプレースホルダーは必要ありません。

```
declare package hash variable-name(8,'', 'a', 'replace');
```

詳細については、“コンストラクターを使用したハッシュインスタンスの定義” (195 ページ)、“メソッド呼び出しを使用したハッシュインスタンスの定義” (196 ページ)、および“_NEW_演算子を使用してハッシュインスタンスを作成する” (199 ページ)を参照してください。

_NEW_演算子を使用してハッシュインスタンスを作成する

DECLARE PACKAGE ステートメントを使用してハッシュ変数とハッシュインスタンスを作成するかわりに、DECLARE PACKAGE ステートメントを使用してハッシュ変数を作成し、_NEW_演算子を使用してハッシュインスタンスを作成することができます。DECLARE PACKAGE ステートメントを使用して、ハッシュパッケージ変数を宣言します。次に、_NEW_演算子を使用してハッシュパッケージのインスタンスをインスタンス化し、新しくインスタンス化されたハッシュインスタンスを参照するようにハッシュ変数を設定します。このシナリオでは、DECLARE PACKAGE ステートメントを使用して初期化データを提供することはできません。ハッシュインスタンスが完全に定義された状態で構築されていない場合は、_NEW_演算子と後続のメソッド呼び出しを使用して、ハッシュインスタンスの初期化データを提供できます。

次の例の DECLARE PACKAGE ステートメントは、変数 MYHASH がハッシュパッケージ型であることをコンパイラに伝えます。この時点では、変数 MYHASH のみを宣言しています。ハッシュインスタンスを参照する可能性がありますが、現在は何も参照していないため、null パッケージ参照です。ハッシュ変数パッケージは一度だけ宣言する必要があります。_NEW_演算子は、ハッシュパッケージのインスタンスを作成し、それを変数 MYHASH に割り当てます。

```
declare package hash myhash();
myhash = _new_ hash(8, 'mytable', 'yes', 'replace', 'sumnum', 'y');
```

前述のステートメントは、次のコードと同等です。

```
declare package hash myhash(8, 'mytable', 'yes', 'replace', 'sumnum', 'y');
```

詳細については、“_NEW_ 演算子: ハッシュパッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

暗黙の変数と変数リストのメソッド

ハッシュインスタンスを定義するときは、一連のキー変数とデータ変数を指定します。ハッシュインスタンスが完全に定義された後、キー変数とデータ変数は、後続の操作中に暗黙的または明示的に読み書きできます。

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

キー変数とデータ変数をハッシュインスタンスに渡すには、暗黙の変数メソッドと変数リストメソッドの 2 つの方法があります。

暗黙変数メソッドは、DATA ステップのハッシュオブジェクトインターフェイスに似ています。暗黙変数メソッドを使用すると、一連の DEFINEKEY および

DEFINEDATA メソッド呼び出しと単一の DEFINEDONE メソッド呼び出しによって、キー変数とデータ変数が定義されます。次に、キー変数とデータ変数の定義のセットは、他のハッシュパッケージメソッドの実行中に暗黙的な引数として使用されます。

次の例では、2 つのキー変数 **k1** と **k2**、および 2 つのデータ変数 **d1** と **d2** を持つハッシュテーブルを定義します。FIND メソッドは、暗黙のキー変数の値を読み取り、ハッシュテーブルでキー値を検索します。キー値が見つかった場合、DS2 は対応するデータ値をハッシュインスタンスに定義された暗黙的なデータ変数に書き込みます。

```
declare package hash h();
  h.definekey('k1');
  h.definekey('k2');
  h.definedata('d1');
  h.definedata('d2');
  h.definedone();

  /* No explicit arguments specify what key values to find */
  /* or what to do with the data values if keys are found. */
  /* Implicitly uses key variables k1 and k2 and          */
  /* data variables d1 and d2.                            */
  h.find();
```

変数リストメソッドでは、ハッシュメソッドが呼び出されるときに、変数リストで対象の変数を明示的な引数として指定します。

この例では、変数リストメソッドを使用しており、暗黙変数メソッドを使用する前述の例と同じです。

```
declare package hash h();
rc=h.keys([k1 k2]);
rc=h.data([d1 d2]);
rc=h.definedone();
h.find([k1 k2], [d1 d2]);
```

変数リストメソッドは、暗黙のキー変数とデータ変数以外の変数を柔軟に使用できるようにします。次の FIND メソッドは、 **x** および **y** に格納されている値を検索します。値が見つかった場合、それらは **u** および **v** に書き込まれます。

```
h.find([x y], [u v]);
```

変数リストの詳細については、“DS2 変数リスト” (28 ページ)を参照してください。

すべてのハッシュ暗黙変数メソッドは、キーとデータの両方を持つハッシュインスタンスと、キーのみのハッシュインスタンスで機能します。

変数リストメソッドには、キーのみのハッシュインスタンスでのみ機能するものと、キーとデータの両方を持つハッシュインスタンスでのみ機能するものがあります。キーのみのハッシュインスタンスが、キーとデータを持つハッシュインスタンスに対してのみ機能する変数リストメソッドを呼び出すと、ランタイムエラーが発生します。逆の場合も同様です。キーのみのハッシュインスタンスで機能するメソッドの詳細については、“DS2 ハッシュおよびハッシュ反復子パッケージの属性、メソッド、演算子、ステートメント” (*SAS DS2 言語リファレンス*)の各メソッドを参照してください。

一意でないキーとデータのペア

デフォルトでは、ハッシュパッケージ内のすべてのキーは一意です。これは、キーごとに 1 セットのデータ変数が存在することを意味します。状況によっては、ハッシュパッケージに重複キーを含めること、つまり、複数のデータ変数のセットを 1 つのキーに関連付けることが必要になる場合があります。

たとえば、キーが患者 ID で、データが来院日であるとします。患者が複数回来院する場合、複数の来院日が患者 ID に関連付けられます。MULTIDATA パラメーターまたはメソッドを YES に設定してハッシュパッケージを作成すると、データ変数の複数のセットがキーに関連付けられます。

テーブルに重複キーが含まれている場合、デフォルトでは、最初のインスタンスはハッシュパッケージに保存され、後続のインスタンスは無視されます。最後のインスタンスをハッシュパッケージに格納するには、DUPLICATE パラメーターまたはメソッドを使用します。DUPLICATE パラメーターまたはメソッドは、重複キーがある場合にも SAS ログにエラーを書き込みます。

ただし、MULTIDATA パラメーターまたはメソッドを使用する場合、ハッシュパッケージではキーごとに複数の値を保存できます。ハッシュパッケージは、キーに関連付けられたリストに複数の値を保持します。このリストは、HAS_NEXT や FIND_NEXT などのいくつかのメソッドを使用して走査および操作できます。

複数のデータアイテムリストを走査するには、現在のリストアイテムを知っている必要があります。まず、特定のキーに対して FIND メソッドを呼び出します。FIND メソッドは、現在のリストアイテムを設定します。次に、キーに複数のデータ値があるかどうかを判断するために、HAS_NEXT メソッドを呼び出します。キーに別のデータ値があることを確認したら、FIND_NEXT メソッドを使用してその値を取得できます。FIND_NEXT メソッドは、現在のリストアイテムをリスト内の次のアイテムに設定し、そのアイテムに対応するデータ変数を設定します。

特定のキーのリストを順方向に移動するだけでなく、HAS_PREV および FIND_PREV メソッドを同様の方法で使用して、リストを逆方向にループできます。

注: 複数データアイテムリストのアイテムは 、挿入した順序で維持されます。

MULTIDATA および DUPLICATE パラメーターの詳細については、“DECLARE PACKAGE ステートメント: ハッシュパッケージ” (*SAS DS2 言語リファレンス*)または“_NEW_ 演算子: ハッシュパッケージ” (*SAS DS2 言語リファレンス*)を参照してください。 MULTIDATA および DUPLICATE メソッドの詳細については、“MULTIDATA メソッド” (*SAS DS2 言語リファレンス*)および“DUPLICATE メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

キー要約の維持

SUMINC パラメーターまたはメソッドを使用して、ハッシュパッケージキーの要約カウントを維持できます。SUMINC は、レコードが FIND、CHECK、または REF メソッドによって使用されるたびに、レコードに要約値を保存できるように、各レコ

ードに内部ストレージを割り当てるようにハッシュパッケージに指示します。SUMINC 値は、FIND、CHECK、または REF メソッドの後にハッシュパラメーターキーの要約カウントを維持するためにも使用されます。

SUMINC には、合計増分、つまり、キーへの各参照のキー要約に追加する量を保持する変数が指定されます。SUMINC 値は、0 より大きい、小さい、または等しい場合があります。SUMINC 値は、ADD メソッドの要約を初期化するためにも使用されます。ADD メソッドが発生するたびに、SUMINC 値のキーが初期化されます。

SUM メソッドは、キーに 1 つのデータアイテムのみ存在する場合、指定されたキーに対する要約値を取得します。

複数のアイテムが存在する場合、SUMDUP メソッドは、キー要約の現在の値を取得します。

キー要約は、DATASOURCE パラメーターまたは DATA メソッドと組み合わせて使用できます。DEFINEDONE メソッドまたは DECLARE PACKAGE ステートメントを使用してテーブルがハッシュパッケージに読み込まれると、すべてのキー要約が SUMINC 値に設定され、後続のすべての FIND、CHECK、または ADD メソッドによって、対応するキー要約が変更されます。

SUMINC パラメーターの詳細については、“DECLARE PACKAGE ステートメント: ハッシュパッケージ” (*SAS DS2 言語リファレンス*)を参照してください。SUMINC および SUMDUP メソッドの詳細については、“SUMINC メソッド” (*SAS DS2 言語リファレンス*)および“SUMDUP メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

データの格納と取得

ハッシュパッケージのキー変数とデータ変数を初期化したら、ADD メソッドを使用してハッシュパッケージにデータを格納できます。または、DATASOURCE パラメーターまたは DATASET メソッドを使用して、テーブルをハッシュパッケージにロードすることもできます。DATASOURCE パラメーターまたは DATASET メソッドを使用し、テーブルに同じキー値を持つ複数の行が含まれている場合、SAS はデフォルトでハッシュテーブルの最初の行を保持し、後続の行を無視します。最後のインスタンスをハッシュパッケージに格納するか、重複キーがある場合にログにエラーを送信するには、DUPLICATE パラメーターまたはメソッドを使用します。各キーの重複値を許可するには、MULTIDATA パラメーターまたはメソッドを使用します。

その後、FIND メソッドを使用して、ハッシュパッケージからデータを検索および取得できます。各キーに複数のデータアイテムが存在する場合は、FIND_NEXT および FIND_PREV メソッドを使用してデータを検索および取得します。

詳細については、“ADD メソッド: ハッシュパッケージ” (*SAS DS2 言語リファレンス*)、“FIND メソッド” (*SAS DS2 言語リファレンス*)、“FIND_NEXT メソッド” (*SAS DS2 言語リファレンス*)、“FIND_PREV メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

REF メソッドを使用して、FIND メソッドと ADD メソッドを統合できます。次の例では、このコードの量を減らすことができます。

```
rc = h.find();
  if (rc != 0) then
    rc = h.add();
```

これを単一のメソッド呼び出しにします。

```
rc = h.ref();
```

詳細については、“REF メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

注: また、ハッシュ反復子パッケージを使用して、ハッシュパッケージデータを一度に 1 データアイテムずつ正順および逆順で取得することもできます。ハッシュ反復子パッケージの詳細については、“ハッシュ反復子パッケージの使用” (209 ページ)を参照してください。

データの置換と削除

次のいずれかの方法を使用して、ハッシュパッケージ内のデータを削除または置換できます。

- 指定したキーからデータアイテムを 削除するには、REMOVE メソッドを使用します。
- すべてのデータアイテムを 削除するには、REMOVEALL メソッドを使用します。
- 複数のデータアイテムを 持つキーのデータアイテムを削除するには、REMOVEDUP メソッドを使用します。
- すべてのデータアイテムを 置き換えるには、REPLACE メソッドを使用します。
- 現在のデータアイテムのみを 置き換えるには、REPLACEDUP メソッドを使用します。

注: 関連付けられたハッシュ反復子がキーを指している場合、REMOVE メソッドはキーまたはデータをハッシュパッケージから削除しません。エラーメッセージがログに発行されます。

詳細については、“REMOVE メソッド” (*SAS DS2 言語リファレンス*)、“REMOVEALL メソッド” (*SAS DS2 言語リファレンス*)、“REMOVEDUP メソッド” (*SAS DS2 言語リファレンス*)、“REPLACE メソッド” (*SAS DS2 言語リファレンス*)、“REPLACEDUP メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

ハッシュパッケージデータをテーブルに保存する

OUTPUT メソッドを使用して、指定したハッシュパッケージ内のデータを含むテーブルを作成できます。

次の例では、最初のテーブルプログラムが **data1** テーブルを生成します。2 番目のテーブルプログラムは、1 つのキー **k** と 2 つのデータ **d1** および **d2** を含むハッシュパッケージ **h** を作成します。次に、**data1** テーブルの各行が読み取られ、キーとデータがハッシュパッケージ **h** に追加されます。最後に、ハッシュパッケージ **h** に格納されているデータ値が **out1** テーブルに書き込まれます。3 番目のテーブルプログラムは、**out1** テーブルの内容を書き込みます。

```
/* Generate and output 5 rows for table data1. */
data data1(overwrite=yes);
 declare double k d1 d2;
 method init();
  declare int i;
  do i = 1 to 5;
   k = i; d1 = i*10; d2 = i*2; output;
  end;
 end;
enddata;
run;

data _null_;
 declare double k d1 d2;
 declare package hash h(0, '', 'descending');

 /* Define key and data variables for hash h. */
 method init();
  h.defineKey('k');
  h.defineData('d1');
  h.defineData('d2');
  h.defineDone();
 end;

 /* Read rows from table data1.
  * Add key and data values from rows to hash h. */
 method run();
  set data1;
  h.add();
 end;

 /* Add additional key and data values to hash h.
  * Output hash h to table out1. */
 method term();
  k = 11; d1 = 110; d2 = 22; h.add();
  k = 12; d1 = 120; d2 = 24; h.add();
  k = 13; d1 = 130; d2 = 26; h.add();
  k = 14; d1 = 140; d2 = 28; h.add();
  h.output('out1');
 end;

enddata;
run;

/* Outputs rows from table out1. */
data;
 method run();
  set out1;
 end;
enddata;
run;
```

アウトプット 13.2　テーブル OUT1 からの出力行

d1	d2
140	28
130	26
120	24
110	22
50	10
40	8
30	6
20	4
10	2

ハッシュパッケージキーは、出力テーブルの一部として格納されないことに注意してください。出力テーブルにキーを含める場合は、DEFINEDATA メソッドでキーをデータとして定義する必要があります。前の例では、DEFINEDATA メソッドは次のように記述されます。

```
h.defineKey('k');
h.defineData('k');
h.defineData('d1');
h.defineData('d2');
```

前述の変更により、次の行が出力されます。

アウトプット 13.3　キーとデータを含むテーブル OUT1 からの出力行

k	d1	d2
14	140	28
13	130	26
12	120	24
11	110	22
5	50	10
4	40	8
3	30	6
2	20	4
1	10	2

FedSQL クエリとハッシュインスタンスを使用して実行時に動的に行を取得する

注: FedSQL クエリの使用は、CAS サーバーではサポートされていません。

ハッシュインスタンスを使用すると、テーブルから取得する行の決定を実行時まで遅らせることができます。実行時に、ハッシュインスタンスが作成され、選択された行がロードされます。行は、データソースに指定された FedSQL クエリに基づいて選択されます。ハッシュ反復子を使用して行をループし、行データにアクセスできます。

次の例では、SELECT * FROM test WHERE a=1 FedSQL クエリによって行が選択されています。このクエリは、それらをハッシュパッケージにロードする **execute** メソッドに渡されます。ハッシュ反復子は行をループし、選択された行を SAS ログと **result** テーブルに書き込みます。

```
data test;
  a=1;  b=2;output;
  a=11; b=22;output;
  a=1;  b=3;output;
  a=22; b=44;output;
run;

proc ds2;
package pkg /overwrite=yes;
  dcl double a b;
```

```
  method execute(char(200) sql);
    dcl package hash  h();
    dcl package hiter hi(h);
    h.keys([a]);
    h.data([a b]);
    h.multidata('yes');
    h.dataset(sql);
    h.definedone();

    put 'SELECTED ROWS:';
    rc = hi.first();
    do while(rc = 0);
      put a= b=;
      rc = hi.next();
    end;

    h.output('result');
  end;
endpackage;
run; quit;

proc ds2;
data _null_;
  method init();
    declare package pkg p1();
    p1.execute('{SELECT * FROM test WHERE a=1}');
  end;

enddata;
run; quit;


proc print data=test;
  title2 'TEST TABLE';
run; quit;

proc print data=result;
  title2 'RESULT TABLE';
run; quit;
```

次の行が SAS ログに書き込まれます。

```
SELECTED ROWS:
a=1 b=2
a=1 b=3
```

入力テーブルと出力テーブルは次のとおりです。

アウトプット 13.4　入力テーブル

TEST TABLE

Obs	a	b
1	1	2
2	11	22
3	1	3
4	22	44

アウトプット 13.5　出力テーブル

RESULT TABLE

Obs	a	b
1	1	2
2	1	3

ハッシュパッケージ属性の使用

ハッシュパッケージで使用できる属性は 2 つあります。NUM_ITEMS はハッシュパッケージ内のアイテムの数を返し、ITEM_SIZE はアイテムのサイズ(バイト単位)を返します。

次の例では、ハッシュパッケージ内のアイテムの数を取得します。

```
num_items = myhash.num_items;
```

次の例では、ハッシュパッケージ内のアイテムのサイズを 取得します。

```
item_size = myhash.item_size;
```

ITEM_SIZE および NUM_ITEMS 属性で、ハッシュパッケージが使用しているメモリ量を把握できます。ITEM_SIZE 属性は、ハッシュパッケージに必要な初期オーバーヘッドを反映したものではなく、必要な内部調整も考慮されていません。したがって、ITEM_SIZE を使用しても正確なメモリ使用量はわかりませんが、適切な概算値が得られます。詳細については、“ITEM_SIZE 属性” (*SAS DS2 言語リファレンス*)および“NUM_ITEMS 属性” (*SAS DS2 言語リファレンス*)を参照してください。

ハッシュ反復子パッケージの使用

ハッシュ反復子パッケージを使用して、一意のルックアップキーに基づいてデータを格納および検索します。ハッシュ反復子パッケージを使用すると、ハッシュパッケージデータを順方向または逆方向のキー順で取得できます。

DECLARE PACKAGE ステートメントを使用して、ハッシュ反復子パッケージを宣言します。新しいハッシュ反復子パッケージを宣言した後、ハッシュパッケージ名をパラメーターとして、_NEW_演算子を使用してパッケージをインスタンス化します。次に例を示します。

```
declare package hiter myiter;
myiter = _new_ hiter('h');
```

DECLARE PACKAGE ステートメントは、変数 MYITER がハッシュ反復子型であることをコンパイラに伝えます。この時点では、変数 MYITER のみを宣言しています。ハッシュ反復子インスタンスを参照する可能性がありますが、現在は何も参照していないため、null パッケージ参照です。ハッシュ反復子パッケージ変数は一度だけ宣言する必要があります。_NEW_演算子は、ハッシュ反復子パッケージのインスタンスを構築し、それを変数 MYITER に割り当てます。ハッシュパッケージ H は、コンストラクターパラメーターとして渡されます。

DECLARE PACKAGE と_NEW_演算子を使用してハッシュ反復子パッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package hiter myiter('h');
```

詳細については、“DECLARE PACKAGE ステートメント: ハッシュ反復子パッケージ” (*SAS DS2 言語リファレンス*)、および“_NEW_ 演算子: ハッシュ反復子パッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

注: ハッシュ反復子パッケージを作成する前に、ハッシュパッケージを宣言してインスタンス化する必要があります。

HTTP パッケージの使用

HTTP パッケージの概要

HTTP パッケージを使用して、HTTP Web サーバーにアクセスするための HTTP クライアントを構築します。

一般的なタスクは次のとおりです。

1 HTTP パッケージを宣言してインスタンス化します。

2 HTTP GET、HEAD、または POST メソッドを作成します。

追加の HTTP パッケージメソッドを使用して、ヘッダー情報を追加し、要求データを送信できます。

3 HTTP GET、HEAD、または POST メソッドを実行します。

4 Web サーバーから応答情報を取得します。

- 完全なエンティティとして、またはストリーミングによる応答本文
- 応答のコンテンツの種類
- 応答ヘッダー

HTTP パッケージを使用すると、次のタスクも実行できます。

- HTTP 応答からステータスコードを取得します。
- ソケットのタイムアウト値を設定します。
- クライアント HTTP インスタンスが接続する URL またはプロキシ URL を指定します。
- URL またはプロキシ URL で必要な場合は、ユーザー名とパスワードを指定します。
- 要求本文のエンコード時または応答本文のデコード時に使用する文字セットを指定します。
- 要求に OpenAuthorization (OAuth)アクセストークンを指定するか、SAS 環境でトークンを検索します。
- SAS ログ機能を使用して、HTTP クライアントとサーバー間の HTTP トラフィックをログに記録します。

HTTP パッケージの宣言とインスタンス化

まず、HTTP パッケージを宣言してインスタンス化する必要があります。HTTP パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package http httpclt;
httpclt = _new_ http();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package http httpclt();
```

詳細については、“DECLARE PACKAGE ステートメント: HTTP パッケージ” (*SAS DS2 言語リファレンス*)および“_NEW_ 演算子: HTTP パッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

ヒント Web サービスアプリケーションでは、HTTP トラフィックを同期的に処理するために必要な HTTP クライアントが 1 つだけの場合があります。または、アプリケーションで必要な場合は、複数の HTTP クライアントをインスタンス化して、データを非同期的に要求および処理することができます。

HTTP GET、HEAD、または POST メソッドの作成

1 CREATEGETMETHOD、CREATEHEADMETHOD、または CREATEPOSTMETHOD メソッドを使用して、GET、HEAD、および POST メソッドを作成します。

詳細については、“CREATEGETMETHOD メソッド” (*SAS DS2 言語リファレンス*)、“CREATEHEADMETHOD メソッド” (*SAS DS2 言語リファレンス*)、および“CREATEPOSTMETHOD メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

2 (オプション) HTTP GET メソッドにヘッダーを追加するには、ADDREQUESTHEADER メソッドを使用します。

詳細については、“ADDREQUESTHEADER メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

3 (オプション) 要求本文を HTTP メソッドに追加するには、SETREQUESTBODYASBINARY または SETREQUESTBODYASSTRING メソッドを使用します。

詳細については、“SETREQUESTBODYASBINARY メソッド” (*SAS DS2 言語リファレンス*)および“SETREQUESTBODYASSTRING メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

4 (オプション) HTTP メソッドで要求コンテンツの種類を指定するには、SETREQUESTCONTENTTYPE メソッドを使用します。

詳細については、“SETREQUESTCONTENTTYPE メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

HTTP GET、HEAD、および POST メソッドの実行

HTTP GET、HEAD、および POST メソッドを実行すると、要求が HTTP Web サーバーに送信されます。

注: HTTP GET、HEAD、および POST メソッドを実行する前に、HTTP GET、HEAD、または POST メソッドを作成する必要があります。詳細については、“HTTP GET、HEAD、または POST メソッドの作成” (211 ページ)を参照してください。

ほとんどの HTTP メソッドでは、EXECUTEMETHOD メソッドを使用して要求を Web サーバーに送信します。詳細については、“EXECUTEMETHOD メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

EXECUTEMETHOD メソッドは、応答本文のストリーミングをサポートしていません。応答本文をストリーミングする場合は、別の実行メソッド EXECUTEMETHODSTREAM が必要です。詳細については、“HTTP リソースの取得” (213 ページ)を参照してください。

別の要求を送信するには、GET、HEAD、または POST 要求の作成から始まり、EXECUTEMETHOD または EXECUTEMETHODSTREAM メソッドで終わるプロセスを繰り返します。

このプログラム例は、HTTP パッケージ(クライアント)をインスタンス化し、HTTP GET メソッドを作成し、GET メソッドを実行して HTTP Web サービスに要求を送信し、HTTP Web サービスからの応答から本文情報を文字列として取得します。

```
data _null_;
 method run();
   /* instantiate the package */
   declare package http h();
   declare varchar(1024) character set utf8 body;
   declare int rc;

   /* create a GET */
   h.createGetMethod('http://api.worldbank.org/countries/fr/');
   /* execute the GET */
   h.executeMethod();
   /* retrieve the response body as a string */
   h.getResponseBodyAsString(body, rc);
   put body;
 end;
```

次の行が SAS ログに書き込まれます。

```
<?xml version="1.0" encoding="utf-8"?>
<wb:countries page="1" pages="1" per_page="50" total="1"
xmlns:wb="http://www.worldbank.org">
 <wb:country id="FRA">

<wb:iso2Code>FR</wb:iso2Code>
   <wb:name>France</wb:name>
   <wb:region id="ECS">Europe
&amp;amp; Central Asia (all income levels)</wb:region>
   <wb:adminregion id="" />

<wb:incomeLevel id="OEC">High income: OECD</wb:incomeLevel>
   <wb:lendingType id="LNX">Not
classified</wb:lendingType>
   <wb:capitalCity>Paris</wb:capitalCity>

<wb:longitude>2.35097</wb:longitude>
   <wb:latitude>48.8566</wb:latitude>

</wb:country>
</wb:countries>
```

HTTP リソースの取得

次の HTTP パッケージメソッドを使用して、HTTP リソースからヘッダー、応答本文、および応答本文の種類を取得できます。応答本文を取得する場合は、1 つのエンティティとして 取得するか、応答本文をストリーミングできます。

タスク	HTTP パッケージメソッド
ヘッダーを取得する	“GETRESPONSEHEADERSASSTRING メソッド” (*SAS DS2 言語リファレンス*)
応答本文を 1 つのエンティティとして取得する [1]	“GETRESPONSEBODYASBINARY メソッド” (*SAS DS2 言語リファレンス*) “GETRESPONSEBODYASSTRING メソッド” (*SAS DS2 言語リファレンス*)
応答本文をストリーミングする [1]	“STREAMRESPONSEBODYASBINARY メソッド” (*SAS DS2 言語リファレンス*) “STREAMRESPONSEBODYASSTRING メソッド” (*SAS DS2 言語リファレンス*)
応答本文のコンテンツの種類を取得する	“GETRESPONSECONTENTTYPE メソッド” (*SAS DS2 言語リファレンス*)

1 応答本文を 1 つのエンティティとして取得する場合は、最初に GET メソッドを作成し、そのメソッドを EXECUTEMETHOD メソッドで実行する必要があります。応答本文をストリーミングする場合は、最初に GET メソッドを作成し、そのメソッドを EXECUTESTREAMMETHOD メソッドで実行する必要があります。

HTTP パッケージを使用する際の考慮事項

- HTTP パッケージは、GET、HEAD、および POST HTTP メソッドのみをサポートします。
- 各クライアントは要求を送信し、応答を同期的に処理します。アプリケーションは、複数のクライアントを作成して、HTTP リソースに対してアクションを非同期に実行できます。
- HTTP パッケージは、要求のデータと応答のデータを DS2 文字列値またはバイナリ値としてメモリに格納します。データをファイルとしてディスクに格納する場合は、HTTP プロシジャの使用を検討してください。詳細については、“HTTP プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。
- HTTP パッケージは、認証が必要なセキュア HTTP エンドポイントに要求を送信できます。HTTP エンドポイントがクライアント認証を必要とする場合、サポートされている認証メカニズムのリストでクライアントに応答します。HTTP パッ

ケージは現在、3 つの最も一般的な認証メカニズムのうちの 2 つ、基本とネゴシエートをサポートしています。基本認証自体は資格情報の機密性を提供しないため、トランスポート層セキュリティ(TLS)を介してデータが暗号化されている場合にのみ使用する必要があります。SAS が TLS を検証する方法の詳細については、*Encryption in SAS* を参照してください。ネゴシエート認証は Kerberos をサポートし、Windows では NT LAN Manager (NTLM)をサポートします。

オープン認証

Open Authorization (OAuth)は、インターネット上でのトークンベースの認証と権限のためのオープンスタンダードです。OAuth は特に HTTP で動作するように設計されており、基本的にリソース所有者の承認を得て、権限サーバーによってサードパーティクライアントにアクセストークンを 発行できるようにします。そして、サードパーティクライアントはアクセストークンを 使用して、リソースサーバーによってホストされている保護されたリソースにアクセスします。

HTTP パッケージの場合、SETAUTHTOKEN メソッドを使用してトークンを提供するか、ADDSASOAUTHTOKEN メソッドを使用して、SAS が OAuth アクセストークンの環境を検索できるようにすることができます。どちらの場合も、OAuth アクセストークンは Authorization ヘッダーフィールドの Bearer 値として要求ヘッダーに追加されます。

詳細については、“SETOAUTHTOKEN メソッド” (*SAS DS2 言語リファレンス*)および“ADDSASOAUTHTOKEN メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

HTTP トラフィックのログ記録

HTTP パッケージは、SAS ログ機能によるログをサポートしています。

App.TableServices.d2pkg.HTTP ロガーは、HTTP クライアントと Web サーバーとの間で送受信されるエラー、ヘッダー、およびデータをログに記録します。

詳細については、“HTTP パッケージロガー” (326 ページ)を参照してください。

JSON パッケージの使用

JSON パッケージの概要

Java Script Object Notation (JSON)は、人間が判読できるデータ交換用に設計された、テキストベースのオープンスタンダードデータ 形式です。JSON は JavaScript プログラミング言語のサブセットに基づいており、JavaScript 構文を使用してデータオブジェクトを記述します。

JSON パッケージは、JSON テキストを作成および解析するためのインターフェイスを提供します。JSON パッケージの Write メソッドは書き込み要求をメモリに蓄積し、テキストを取得できます。JSON パッケージパーサーを使用すると、テキストを読み取って解析できます。

JSON パッケージの宣言とインスタンス化

JSON パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package json myjsonpkg;
myjsonpkg = _new_ json();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package json myjsonpkg();
```

詳細については、“DECLARE PACKAGE ステートメント: JSON パッケージ” (*SAS DS2 言語リファレンス*)、および “_NEW_ 演算子: JSON パッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

JSON テキストの書き込み

JSON テキストを作成するには、CREATEWRITER メソッドを使用して JSON ライターインスタンスを作成します。

JSON 出力は、次の 2 種類のデータ構造コンテナーで構成されます。

JSON オブジェクトコンテナー({ })
: 左中かっこ({)で始まり、右中かっこ(})で終わります。オブジェクトコンテナーは、名前と値のペアとして書き込まれる名前と値のペアを収集します。値は、サポートされている JSON データ型、オブジェクト、または配列のいずれかです。各名前の後にはコロンと値が続きます。名前と値のペアはカンマで区切られます。

  WRITEOBJOPEN および WRITECLOSE メソッドを使用して、オブジェクトコンテナーを作成します。

JSON 配列コンテナー([])
: 左角かっこ([)で始まり、右角かっこ(])で終わります。配列コンテナーは、名前のない値のリストとして書き込まれる値のリストを収集します。値は、サポートされている JSON データ型、オブジェクト、または配列のいずれかです。値はカンマで区切ります。

  WRITEARRAYOPEN および WRITECLOSE メソッドを使用して、配列コンテナーを作成します。

トップレベルのコンテナーには、任意の数のコンテナーを含めることができます。同様に、コンテナーはコンテナーを任意の深さにネストできます。コンテナーをネストするときは、現在のコンテナーのデータ構造要件を守るように注意してください。

- オブジェクトには、名前と値のペアのリストが必要です。値自体はオブジェクトまたは配列にすることができます。
- 配列には、名前と値のペアのような構造上の要件はなく、単なる値、オブジェクト、または配列のリストです。

JSON パッケージメソッドを使用すると、文字値と数値、null 値、ブールの真偽値を書き込むことができます。

JSON テキスト出力の例を次に示します。

```
{"SASJSONExport":"1.0","SASTableData+CLASS":[{"Name":"Joyce","Sex":
"F","Age":11,"Height":51.3,"Weight":50.5},{"Name":"Thomas","Sex":"M",
"Age":11,"Height":57.5,"Weight":85}]}
```

WRITERGETTEXT メソッドを使用して、ライターによって生成された JSON テキストを取得できます。

テキストの書き込みが終了したら、DESTROYWRITER メソッドを呼び出してライターインスタンスを削除します。

次の例では、JSON テキストを作成し、SAS ログに書き込みます。

```
data _null_;
   method init();
      dcl package json j();
      dcl double dblVal;
      dcl int rc;
      dcl nvarchar(30) jsontxt;

      rc = j.createWriter();
         if rc=0 then rc = j.writeArrayOpen();
      if rc=0 then rc = j.writeString( '     Hello World!     ' );
      if rc=0 then rc = j.writeClose();
      j.writerGetText( rc, jsontxt );
      put rc= jsontxt=;
   end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
rc=0 jsontxt=[" Hello World! "]
```

JSON テキストの書き込みを可能にするメソッドの詳細については、“DS2 JSON パッケージのメソッド、演算子、およびステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

JSON テキストの解析

JSON テキストを解析または読み取るには、CREATEPARSER メソッドを使用して JSON ライターインスタンスを作成します。CREATEPARSER メソッド、SETPARSERINPUT メソッド、またはその両方を使用して、入力 JSON テキストをパーサーに提供できます。

GETNEXTTOKEN メソッドは、JSON テキストから次の検証 JSON 言語要素を返すために使用されます。GETNEXTOKEN メソッドは、トークンタイプ、解析フラグ、行番号、列番号も返すことができます。JSON パッケージの IS*メソッド(ISLEFTBRACE など)を使用すると、次のトークンタイプをクエリできます。

ブール真
ブール偽
浮動小数点数
整数
ラベル
左中かっこ({)
左角かっこ([ )
null
数値
部分
右中かっこ(})
右角かっこ( ])
文字列

テキストの解析が終了したら、DESTROYPARSER メソッドを呼び出してパーサーインスタンスを削除します。

次の例では、パーサーを作成し、CREATEPARSER メソッドを使用して入力 JSON テキストを提供します。

```
data _null_;
   method init();
      dcl package json j();
      dcl int rc tokenType parseFlags;
      dcl bigint lineNum colNum;
      dcl nvarchar(128) token abc t1;
      abc = 'xyz';
      t1 = '{"abc" : 1 }';
      rc = j.createParser( t1 );
          if (rc ne 0) then goto TestError;

      * obj open;
      j.getNextToken( rc, token, tokenType, parseFlags, lineNum, colNum );
      if ( rc ne 0 ) then goto TestError;

      * obj label;
      j.getNextToken( rc, token, tokenType, parseFlags, lineNum, colNum );
      if ( rc ne 0 ) then goto TestError;

      * obj value;
      j.getNextToken( rc, token, tokenType, parseFlags, lineNum, colNum );
      if ( rc  ne 0) then goto TestError;

      * obj close;
      j.getNextToken( rc, token, tokenType, parseFlags, lineNum, colNum );
      if ( rc ne 0 ) then goto TestError;

   Exit:
```

```
      rc = j.destroyParser();
      return;

   TestError:
      put 'Test ended abnormally.';
      goto Exit;

   end;
enddata;
run;
```

JSON テキストの解析を可能にするメソッドの詳細については、“DS2 JSON パッケージのメソッド、演算子、およびステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

JSON パッケージロガー

JSON パッケージは、SAS ログ機能によるログをサポートしています。

DS2 JSON パッケージは、内部 DS2 JSON パッケージの DEBUG および TRACE 情報を収集するためのロガー App.tk.DS2PKG.JSON を提供します。JSON パッケージを使用する DS2 プログラムに TRACE ステートメントを簡単に追加できない場合は、App.tk.DS2PKG.JSON ロガーレベルを DEBUG または TRACE に設定します。

詳細については、“JSON パッケージロガー” (326 ページ)を参照してください。

ロガーパッケージの使用

ロガーパッケージの概要

SAS ログ機能では、ロガーはメッセージカテゴリを識別する名前付きエンティティです。ロガーの属性は、レベルと、メッセージカテゴリのログイベントを処理する 1 つ以上のアペンダーで構成されます。レベルは、このメッセージカテゴリに対して処理されるしきい値、または最低イベントレベルを示します。

ロガーパッケージを使用して、SAS ログ機能とのインターフェイスを取ります。新しいロガーパッケージを宣言した後、指定されたログレベルでロガーにメッセージを送信できます。

注: SAS Enterprise Guide セッションからロガーパッケージを使用することはできません。

DS2 ロガーと SAS ログ機能の詳細については、付録 2, “DS2 ロガー” (323 ページ)および *SAS Viya ログ機能: 構成とリファレンス*を参照してください。

ロガーパッケージの宣言とインスタンス化

ロガーパッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package logger logpkg;
logpkg = _new_ logger();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package logger logpkg();
```

詳細については、*"DECLARE PACKAGE ステートメント: ロガーパッケージ" (SAS DS2 言語リファレンス)*、および *"_NEW_ 演算子: ロガーパッケージ" (SAS DS2 言語リファレンス)*を参照してください。

未フォーマットメッセージとフォーマット済みメッセージ

SAS ログ機能に直接書き込まれる生(未フォーマット)のメッセージを指定できます。

$形式マーカーを使用して、フォーマット済みメッセージを作成することもできます。メッセージ形式の各$s 形式マーカーは、指定した対応する引数の内容に置き換えられます。

詳細については、*"LOG メソッド: ロガーパッケージ" (SAS DS2 言語リファレンス)*を参照してください。

メッセージをテーブルにログ記録する

1 メッセージを格納する空のテーブルを作成します。

次に例を示します。

```
libname logs 'library-path';
data logs.edmlog;
 length seqno  8;
 length date   8; format date DATETIME19.;
 length msg    $ 256;
 stop;
run;
```

2 XML ログ構成ファイルを使用して、ロガーとアペンダーを作成します。

App.Program.EDM という名前の App.Program ロガーを作成する例を次に示します。ロガーが書き込むアペンダーは **EDMApender** です。アペンダーは、***library-path*** の SAS データセット名 **edmlog** に書き込むように構成されています。この XML ログ構成ファイルを **edm-l4s.xml** として保存します。

```
<?xml version="1.0"?>
<logging:configuration xmlns:logging="http://www.sas.com/xml/logging/1.0/">

  <appender name="EDMAppender" class="DBAppender">
    <param name="ConnectionString" value="
      DRIVER=base;CATALOG=base;
      schema=(name=base;primarypath='library-path')"/>
    <param name="MaxBufferedEvents" value="1000"/>
    <param name="TableName" value="edmlog"/>
    <param name="Column" value="sn"/>
    <param name="Column" value="d"/>
    <param name="Column" value="m"/>
  </appender>

  <appender name="null" class="ConsoleAppender">
    <filter class="DenyAllFilter"/>
  </appender>

  <logger name="App.Program.EDM">
    <appender-ref ref="EDMAppender"/>
  </logger>

  <root>
    <appender-ref ref="null"/>
  </root>
</logging:configuration>
```

3 ステップ 2 で作成した XML 構成ファイルの名前に LOGCONFIGLOC システムオプションを設定して、SAS を起動します。

次に例を示します。

```
options logconfigloc edm-l4s.xml
```

4 ステップ 2 の XML 構成ファイルで作成したロガーに関連付けられた DS2 ロガーパッケージインスタンスを作成します。

次に例を示します。

```
data _null_;
 dcl package logger l('App.Program.EDM');
 method init();
   dcl double i;
   i = l.islevelactive(4);  put i=;
   l.log('T', 'Hello World! Trace');
   l.log('D', 'Hello World! Debug');
   l.log('I', 'Hello World! Info');
   l.log('W', 'Hello World! Warning');
   l.log('E', 'Hello World! Error');
 end;
enddata;
run;
```

edmlog データセットの内容を出力できます。

```
libname logs 'library-path';
proc print data=logs.edmlog; run;
```

データセットの内容はこのようになります。

Obs	seqno	date	msg
1	285	25APR2013:17:11:42	Hello World! Trace
2	286	25APR2013:17:11:42	Hello World! Debug
3	287	25APR2013:17:11:42	Hello World! Info
4	288	25APR2013:17:11:42	Hello World! Warning
5	289	25APR2013:17:11:42	Hello World! Error

SAS ログ機能の詳細については、*SAS Viya ログ機能: 構成とリファレンス*を参照してください。

MATRIX パッケージの使用

MATRIX パッケージの概要

行列は、数値または文字値の 2 次元配列です。行列の次元は、行と列の数によって定義されます。*n* x *p* 行列の要素は、*n* 行 *p* 列に配置されます。

行列パッケージは、SAS/IML 機能の DS2 レベルの実装を提供します。行列パッケージメソッドを使用して、逆行列計算などの複雑なタスクを実行できます。算術演算、関係演算、および論理演算を実行できます。ある演算は要素ごとに、他の演算はデータ行列全体で実行することができます。

配列または外部データを使用して、行列パッケージにデータをロードできます。行列全体を一度に、または行ごとに配列に書き込むことで、データを生成できます。その後、配列を結果テーブルに書き込むことができます。

これらのメソッドの詳細については、"DS2 行列パッケージのメソッド、演算子、およびステートメント" (*SAS DS2 言語リファレンス*)を参照してください。

MATRIX パッケージの宣言とインスタンス化

行列パッケージは、DS2 DECLARE PACKAGE ステートメントを使用して行列パッケージを宣言およびインスタンス化することによって作成されます。次に例を示します。

```
declare package matrix m(2, 2);
```

このステートメントは、2x2 行列を作成し、そのインスタンスを変数 **m** に格納します。

注: 行列は、null 値または欠損値ではなく、ゼロで埋められます。

詳細については、"DECLARE PACKAGE ステートメント: 行列パッケージ" (*SAS DS2 言語リファレンス*)を参照してください。

行列データ入力

データの初期化または行列パッケージへのロードには、2 つの方法があります。

- ゼロ値で初期化する
  - declare package matrix *package-variable*(*rows*,*columns*);
  - *package-variable*=_new_ matrix(*rows*,*columns*);
- 配列値で初期化する
  - /* loads an array using the array name */

    package-variable=_new_ matrix(*array-name*, *rows*, *columns*);
  - /* loads an array using the IN method */

    package-variable=_new_ matrix(*rows*,*columns*);
    package-variable.in(*array-name*);
  - /* loads row-by-row using an input table*/
    /* a variable array, and the SET statement */

    package-variable=_new_ matrix(*rows*,*columns*);
    . . .
    set *table-name*;
    package-variable.in(*variable-array-name*, *i*);

これらの例では、ゼロ値で初期化される 2x2 行列を作成します。

```
declare package matrix m(2,2);

m=_new_ matrix(2, 2);
```

これらの例では、配列 **a** を行列 **m** にロードします。

```
/* simple array */
method init();
  dcl double a[3, 3];
  dcl package matrix m;

  a :=(1, 2, -1, 2, 1, 0, -1, 1, 2);
  m=_new_ matrix(a, 3, 3);
end;

/* variable array */
vararray double a[3,3];
dcl package matrix m;

method init();
  a := (1, 2, -1, 2, 1, 0, -1, 1, 2);
  m = _new_ matrix(a, 3, 3);
end;
```

これらの例では、IN メソッドを使用して配列 **va** を行列 **m** にロードします。

```
/* simple array */
```

```
dcl double va[3,3];
dcl package matrix m;
m = _new matrix(3,3);
m.in(va);

/* variable array */
vararray double va[3,3];
dcl package matrix m;

m.in(va); */
```

この例では、SET ステートメントを使用してテーブル **x** を変数配列 **a** に読み込みます。その変数配列は、行列 **m** をロードするために使用されます。

```
vararray double a[4];
dcl package matrix m;

/* Create an empty matrix to hold the input values */
method init();
  m = _new_ [this] matrix(4, 4);
  i = 1;
end;

/* Read and initialize each row of matrix from vararray a */
method run();
   set x;
   m.in(a, i);
   i + 1;
end;
```

詳細については、"_NEW_ 演算子: 行列パッケージ" (*SAS DS2 言語リファレンス*)および"IN メソッド" (*SAS DS2 言語リファレンス*)を参照してください。

注: _NEW_演算子はコードストリームの一部です。宣言では使用されません。

行列データ出力

行列全体を一度に、または行ごとに配列に書き込むことで、行列データを生成できます。

TOARRAY または TOVARARRAY メソッドを使用して、一度に行列を配列に書き込みます。この例では、配列 **a** が行列 **m** にロードされています。次に、行列 **m** の転置が計算され、配列 **c** に書き込まれます。

```
dcl double a[3,3];
dcl double b[3,3];

method run();
  dcl package matrix m r;
    dcl double i j;

  a := (1,2,3,4,5,6,7,8,9);
```

```
  m = _new_ matrix(a, 3, 3);
  r = m.trans();
  r.toarray(b);

  do i = 1 to 3;
    do j = 1 to 3;
      put b[i,j];
    end;
  end;
end;
```

詳細については、“TOARRAY メソッド” (*SAS DS2 言語リファレンス*)および“TOVARARRAY メソッド” (*SAS DS2 言語リファレンス*)を参照してください。

変数配列は、OUT メソッドと OUTPUT ステートメントを使用してデータを書き込むために使用できます。OUT メソッドは、行列の行データを変数配列に書き込みます。OUTPUT ステートメントは、結果テーブルにデータを書き込みます。行列は、変数配列を使用して一度に 1 行ずつ結果テーブルに書き込まれます。詳細については、“OUT メソッド” (*SAS DS2 言語リファレンス*)および“データのロードと書き込み” (*SAS DS2 言語リファレンス*)を参照してください。

行列演算

次の表は、一般的な DS2 ドット構文メソッド呼び出しを使用して行列に対して実行できる演算をまとめたものです。

表 13.6　行列演算

演算の種類	メソッド名	実行される演算	注記
二項算術	ADD	加算	■ 加算演算または減算演算で使用される行列の配列次元は、互換性がある必要はありません。ただし、最初の行列の列数が 2 番目の行列の行数と同じであるか、2 番目の行列が 1x1 行列である必要があります。 ■ 乗算演算で使用される行列の配列次元には互換性が必要です。最初の行列の列数は、2 番目の行列の行数と等しい必要があります。
	MULT	乗算	
	SUB	減算	
単項	ABS	絶対値	■ 単項演算は、1 つの行列に対して実行されます。 ■ 行列は正方または特異である必要があります。逆行列および行列式の演算の場合、行列は正方にする必要があります。
	COPY	行列のコピー	
	DET	正方行列の行列式	
	EXP	指数値	
	FLOOR	各行列値の整数部	

演算の種類	メソッド名	実行される演算	注記
	INVERSE	逆行列	
	LOG	自然対数	
	SQRT	平方根	
	TRANS	転置	
二項関係	EQ	等しい	■ 二項関係演算の結果は、最初の行列の[*i*, *j*]th 要素が 2 番目の行列の[*i*, *j*]th 要素とどのように比較されるかを示す行列になります。結果の値は、比較が偽の場合は 0、比較が真の場合は 1 です。 ■ 行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。
	GT	より大	
	GE	より大か等しい	
	LT	より小	
	LE	より小か等しい	
	NE	等しくない	
二項論理	AND OR	and 比較 or 比較	■ 二項論理演算の結果は、最初の行列の[*i*, *j*]th 要素が 2 番目の行列の[*i*, *j*]th 要素とどのように比較されるかを示す行列になります。結果の値は、比較が偽の場合は 0、比較が真の場合は 1 です。
ALL 関係	ALL_EQ	ALL 等しい	■ ALL 関係演算は、最初の行列の[*i*, *j*]th 要素が 2 番目の行列の[*i*, *j*]th 要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。結果が 1 になるためには、すべての[*i*, *j*]要素比較が真でなければなりません。それ以外の場合、結果は 0 になります。 ■ 行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。
	ALL_GT	ALL より大	
	ALL_GE	ALL より大か等しい	
	ALL_LT	ALL より小	
	ALL_LE	ALL より小か等しい	
	ALL_NE	ALL 等しくない	
ANY 関係	ANY_EQ	ANY 等しい	■ ANY 関係演算は、最初の行列の[*i*, *j*]th 要素が 2 番目の行列の[*i*, *j*]th 要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[*i*, *j*]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。
	ANY_GT	ANY より大	
	ANY_GE	ANY より大か等しい	
	ANY_LT	ANY より小	

演算の種類	メソッド名	実行される演算	注記
	ANY_LE	ANY より小か等しい	■ 行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。
	ANY_NE	ANY 等しくない	
ALL 論理	ALL_AND	ALL AND	■ ALL 論理演算は、最初の行列の[*i*, *j*]th 要素が 2 番目の行列の[*i*, *j*]th 要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。結果が 1 になるためには、すべての[*i*, *j*]要素比較が真でなければなりません。それ以外の場合、結果は 0 になります。 ■ 行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。
	ALL_OR	ALL OR	
ANY 論理	ANY_AND	ANY AND	■ ANY 関係演算は、最初の行列の[*i*, *j*]th 要素が 2 番目の行列の[*i*, *j*]th 要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[*i*, *j*]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。 ■ 行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。
	ANY_OR	ANY OR	
要素ごと	EDIV	要素ごとの除算	■ 要素ごとの演算では、同じ次元の行列、行の次元が一般行列の行の次元と一致するベクトル、列の次元が一般行列の列の次元と一致するベクトル、または 1x1 行列を使用して一般行列に演算を適用し、各*[i,j]*要素に対してスカラー演算を効率的に実行することができます。 ■ 要素ごとの演算は、2 つの引数行列に対する要素ごとの演算から結果行列を生成します。
	EMAX	要素ごとの最大値	
	EMIN	要素ごとの最小値	
	EMOD	要素の除算の要素ごとの剰余	
	EMULT	要素ごとの乗算	
	EPOW	要素ごとのべき乗	

行列パッケージを使用する際の考慮事項

- コンストラクターを使用して行列を作成する場合(たとえば、 m= _new_ matrix(2,2);または declare package matrix m(2, 2);)、行列は欠損値ではなくゼロで埋められます。

- null または欠損値を含む行列はあまり役に立ちません。行列に null または欠損値が含まれている場合、その行列に対する一部の演算は異常と見なされる可能性があります。たとえば、null または欠損値で行列を乗算すると、ランタイムエラーが発生します。ただし、null または欠損値を含む行列を加算または減算しても、エラーは発生しません。行列要素をゼロで除算しても、浮動小数点例外は発生しません。結果は欠損値です。
- ループを使用して行列データを入力または出力する場合、ループの処理中に行列に対する演算を避ける必要があります。その理由は、ループが完了するまで行列が部分的にしか埋められないためです。
- 行列のサイズを追跡するのは必ずしも容易ではありません。行列演算の次元が一貫していることを確認してください。
- 行列がスレッドプログラムで宣言されると、各スレッドプログラムは独自の個別の行列インスタンスを持ちます。DS2 行列パッケージは、並列行列演算を実行するためのノードまたはスレッド間のデータパーティション分割をサポートしていません。かわりに、各スレッドは、行列の独自のインスタンスに対して行列演算を実行します。

行列から DS2 配列への値の移動

次の例は、TOARRAY メソッドの使用方法を示しています。DS2 プログラムで使用するために、行列から DS2 配列に値を移動できる唯一の方法は、このメソッドを使用することです。

```
dcl double a[3, 3];
dcl double c[3, 3];
declare package matrix m;

a := (1, 2, 3, 4, 5, 6, 7, 8, 9);
m=_new_ matrix(a, 3, 3);
m.toarray(c);

do i=1 to 3;
  do j=1 to 3;
    put c[i, j];
  end;
end;
```

PCRXFIND パッケージの使用

PCRXFIND パッケージの概要

PCRXFIND パッケージを使用すると、特定のテキスト文字列が別のテキスト文字列内にあるかどうかを確認できます。このパッケージは、軽量のテキスト分析に使用され、テキスト一致から詳細を取得するための使いやすい関数が含まれています。

これらの詳細には、テキストブロック内の部分文字列の開始位置または終了位置の検索、または文字列からの一致の特定部分の取得が含まれます。

PCRXFIND パッケージの宣言とインスタンス化

DECLARE PACKAGE ステートメントを使用して、PCRXFIND パッケージを宣言します。新しい PCRXFIND パッケージを宣言した後、Perl 互換の正規表現を解析できます。この表現は、パッケージが宣言されたときに渡すことができます。パッケージ宣言中に表現を渡すことはオプションです。次に、ロードされた表現を使用して、一致の開始と終了のインデックス、および以前に解析された正規表現内の各かっこで囲まれたキャプチャグループに関連するインデックスを提供することにより、一致操作を実行できます。

PCRXFIND パッケージのインスタンスを構築する方法は 3 つあります。

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

  declare package pcrxfind *variable*(<*regular-expression*>);

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

  declare package pcrxfind *variable*;
  variable = _new_ pcrxfind(<*regular-expression*>);

- 表現を指定せずに DECLARE PACKAGE ステートメントを使用します。

  declare package pcrxfind *variable*;
  variable = _new_ pcrxfind();

詳細については、“DECLARE PACKAGE ステートメント: PCRXFIND パッケージ” (*SAS DS2 言語リファレンス*)、および “_NEW_ 演算子: PCRXFIND パッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

例

PCRXFIND パッケージを使用して住所を調べ、いくつかの情報を抽出する例を次に示します。指定された正規表現と一致しない場合、住所は無効であると報告されます。この例では、GETGROUP、GETGROUPLENGTH、および GETMATCH メソッドだけでなく、大文字と小文字を区別しない一致も示しています。

```
/* Generate a set of valid and invalid addresses */
data sampleaddr/overwrite=yes;
  dcl varchar(256) address;
  method init();
    address = '123 Pineforest ln Cary NC 27513'; output;
    address = 'Extra text before the match 5123 Oakwood rd Cary NC 27513';
    output;
    address = '24 Durham NC 27519'; output;
    address = '0 INVALID BLVD NC 27605'; output;
    address = 'APT. 731 1008 TRADE AVE RALEIGH NC 27605'; output;
    address = 'HWY 55 123 Thistlecrest ln Cary NC 27511'; output;
    address = 'This one is invalid due to length of the street...
      423 XYYYSDDDDDTHHHHHHZZZJJJXQTYTRRRRRRRASDAOOIIKKKK CT JASDGTJ
```

```
      UW 5290889';
    output;
  end;
enddata; run;

data _null_;
  /*
   * Instantiate the address validator with a regular expression that does
   * a case insensitive match on addresses. This will be used to extract
   * each component of the address or reject the address as invalid.
   */
  dcl package pcrxfind addrValidator
    ('/(apt\.?[ ]?[#]?\d+ |hwy \d+ )?(\d{1,5}) (\w+) (
    rd|ln|cir|st|blvd|ave|ct) (\w+) ([A-Z]{2}) (\d+)/i' );

  method run();
    dcl int valid havePrefix streetLength cityLength;
    dcl char(2) state;
    dcl varchar(256) city prefix zipText street streetType streetNum outaddr
      streetAddress;
    dcl double zip;
    set sampleaddr;

    /* Perform the match */
    valid = addrValidator.match( address );

    /* Confirm that a match was found before proceeding */
    if valid > 0 then do;
      /*-- Grab each component of the match --*/
      addrValidator.getGroup( state, 6 );
      addrValidator.getGroup( city, 5 );
      cityLength = addrValidator.getGroupLength( 5 );
      addrValidator.getGroup( streetType, 4 );
      addrValidator.getGroup( street, 3 );
      streetLength = addrValidator.getGroupLength( 3 );
      addrValidator.getGroup( streetNum, 2 );

      /* Assemble the street address */
      streetAddress = cat( streetNum, ' ', street, ' ', streetType );
      havePrefix = addrValidator.getGroup( prefix, 1 );
      /* Convert text zip to numeric */
      addrValidator.getGroup( zipText, 7 );
      zip = inputn( zipText, 5 );

      /*
       * Make sure that the city and street didn't match over 25 chars.
       * Filters potentially false results.
       */
      if (cityLength > 25) || (streetLength > 25) then do;
        addrValidator.getMatch( outaddr );
        put 'City or street likely invalid due to length: ' outaddr;
      end;
      else do;
        /* output the details for this address */
        put;
        put 'Valid address:';
```

```
            if havePrefix = 0 then put 'prefix: ' prefix;
            put streetAddress=;
            put city=;
            put state=;
            put zip=;
            put;
          end;
        end;
        /* If valid is -1, the expression was run and no match was found */
        else if valid = -1 then do;
          outaddr = trim( address );
          put 'Invalid address: ' outaddr;
        end;
      end;
    enddata;

    run;
```

次の行がログに出力されます。

```
Valid address:
streetAddress=123 Pineforest ln
city=Cary
state=NC
zip=27513

Invalid address:  Extra text before the match 5123 Oakwood rd Cary NC 27513
Invalid address:  24 Durham NC 27519
Invalid address:  0 INVALID BLVD NC 27605

Valid address:
prefix:  APT. 731
streetAddress=1008 TRADE AVE
city=RALEIGH
state=NC
zip=27605


Valid address:
prefix:  HWY 55
streetAddress=123 Thistlecrest ln
city=Cary
state=NC
zip=27511

Invalid address:  This one is invalid due to the length of the street...        423
XYYYSDDDDDTHHHHHHZZZJJJXQTYTRRRRRRRASDAOOIIKKKK CT JASDGTJ        UW 5290889
```

PCRXREPLACE パッケージの使用

PCRXREPLACE パッケージの概要

PCRXREPLACE パッケージを使用すると、テキストに対して置換または代入操作を実行できます。たとえば、機密性の高い名前や電話番号を REDACTED という単語に置き換えたり、プログラムでテキストを更新したりすることができます。

REPLACE パッケージは UTF-8 および Latin1 テキストをネイティブに処理できるため、これらの形式をトランスコードする必要はありません。

PCRXREPLACE パッケージの宣言とインスタンス化

DECLARE PACKAGE ステートメントを使用して、PCRXREPLACE パッケージを宣言します。正規表現を含めて、PCRXREPLACE パッケージを表現および置換テキストに関連付けることができます。パッケージ宣言中に正規表現を含めることはオプションです。新しい PCRXREPLACE パッケージを宣言した後、そのパッケージを使用して文字列の置換操作を実行できます。

PCRXREPLACE のインスタンスを構築する方法は 3 つあります。

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

  ```
  declare package pcrxreplace variable('/textToReplace/replacementText');
  ```

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

  ```
  declare package pcrxreplace variable;
  variable = _new_ pcrxreplace('/textToReplace/replacementText');
  ```

- 表現を指定せずに DECLARE PACKAGE ステートメントを使用します。

  ```
  declare package pcrxreplace variable;
  variable = _new_ pcrxreplace();
  ```

詳細については、“DECLARE PACKAGE ステートメント: PCRXREPLACE パッケージ” (*SAS DS2 言語リファレンス*)、および “_NEW_ 演算子: PCRXREPLACE パッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

例

データセット内の姓と名を入れ替える例を次に示します。

```
data names(overwrite=yes);
  dcl varchar(32) name;
    method init();
      /* create a dataset to work on */
```

```
        name = 'Jones, Fred'; output;
        name = 'Kavich, Kate'; output;
        name = 'Turley, Ron'; output;
        name = 'Dulix, Yolanda'; output;
      end;
enddata;
run;

data _null_;
  /* Variable declaration */
  dcl package pcrxReplace nameSwap;

  method init();
    nameSwap = _new_ pcrxReplace( 's/(\w+), (\w+)/$2 $1/g' );
  end;

  /* Method that switches the first and last names in
    a dataset formatted as follows: 'lastname, firstname' */
  method run();
    dcl varchar(32) fname lname replacementText outputText;
    dcl int rc;
    set names;
    outputText = nameSwap.apply(name);
    put name=;
   end;

enddata;
run;
```

次の行がログに出力されます。

```
name=Fred Jones
name=Kate Kavich
name=Ron Turley
name=Yolanda Dulix
```

SQLSTMT パッケージの使用

SQLSTMT パッケージの概要

注: SQLSTMT パッケージは、CAS サーバーではサポートされていません。

SQLSTMT パッケージは、FedSQL ステートメントを DBMS に渡して実行し、DBMS から返された結果セットにアクセスする方法を提供します。FedSQL ステートメントは、テーブルから行を作成、選択、変更、挿入、削除できます。FedSQL ステートメントがテーブルから行を選択する場合、SQLSTMT パッケージは、結果セットで返された行を問い合わせるためのメソッドを提供します。

SQLSTMT インスタンスが作成されると、FedSQL ステートメントが FedSQL 言語プロセッサに送信され、次に FedSQL 言語プロセッサがステートメントを DBMS に送

信して、準備されてインスタンスに格納されます。その後、インスタンスを使用して、FedSQL ステートメントを複数回効率的に実行できます。実行時までのステートメント準備の遅延により、FedSQL ステートメントは、DS2 プログラムの実行中に動的に構築およびカスタマイズできます。

Hadoop 分散: Hadoop ディストリビューションを使用している場合、SQLSTMT パッケージを使用するには Hive 0.13 以降が必要です。

次に、SQLSTMT を使用して Teradata テーブルに値を挿入する簡単な例を示します。

```
dcl package sqlstmt s('insert into td.testdata (x, y, z) values (?, ?, ?)',
   [x y z]);

  do i=1 to 5;
    x=i;
    y=i*1.1;
    z=i*10.01;
    s.execute();
  end;
end;
```

次の行がテーブルに挿入されます。

```
1 1.1 10.01
2 2.2 20.02
3 3.3 30.03
4 4.4 40.04
5 5.5 50.05
```

その他の例については、“DS2 サンプルプログラム” (*SAS DS2 言語リファレンス*)を参照してください。

SQLSTMT パッケージの宣言とインスタンス化

DECLARE PACKAGE ステートメントを使用して、SQLSTMT パッケージを宣言します。パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。

SQLSTMT パッケージのインスタンスを構築する方法は 3 つあります。

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。次の 2 つの構文形式があります。

DECLARE PACKAGE SQLSTMT *variable*[('*sql-txt*' [, \[*parameter-variable-list*\])];

DECLARE PACKAGE SQLSTMT *variable* [('*sql-txt*' [, *connection-string*])];

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。次の 2 つの構文形式があります。

DECLARE PACKAGE SQLSTMT *variable*;

variable = **_NEW_ SQLSTMT**('*sql-txt*' [, \[*parameter-variable-list*\]);

DECLARE PACKAGE SQLSTMT *variable*;

variable = **_NEW_ SQLSTMT** ('*sql-txt*' [, *connection-string*]);

注: DECLARE PACKAGE ステートメントは、_NEW_演算子が実行されるまで、SQLSTMT パッケージインスタンスを構築しません。_NEW_演算子が実行されるまで、SQL ステートメントの準備は行われません。

- SQL テキストを指定せずに DECLARE PACKAGE ステートメントを使用します。

DECLARE PACKAGE SQLSTMT *variable*( );

variable = **_NEW_ SQLSTMT** ( );

_NEW_演算子を使用すると、*sql-text* は、式から生成された文字列値または変数に格納された文字列値にすることができます。

DECLARE ステートメントのかっこ内に構築用の引数が含まれている場合(SQLSTMT パッケージでは引数の省略も有効)、パッケージインスタンスが割り当てられます。かっこが含まれていない場合、変数は作成されますが、パッケージインスタンスは割り当てられません。

複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

FedSQL ステートメントのパラメーター値の指定

FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は次のいずれかです。

- コンストラクターの DECLARE PACKAGE ステートメントまたは_NEW_演算子で指定された変数の現在の値

  詳細については、"DECLARE PACKAGE ステートメント: SQLSTMT パッケージ" (*SAS DS2 言語リファレンス*)および"_NEW_ 演算子: SQLSTMT パッケージ" (*SAS DS2 言語リファレンス*)を参照してください。

- BINDPARAMETERS メソッドで指定された変数の現在の値

  BINDPARAMETERS メソッドを使用して FedSQL ステートメントを実行すると、ステートメントの実行時にバインドされた変数の値が読み取られ、ステートメントのパラメーターの値として使用されます。バインドされた変数の型が対応するパラメーターの型と異なる場合、バインドされた変数の値はパラメーターの型に変換されます。詳細については、"BINDPARAMETERS メソッド" (*SAS DS2 言語リファレンス*)を参照してください。

注: SQLSTMT パッケージを使用して DS2 パッケージの METHOD ステートメントを実行する場合は、REGISTEROUTPARAMETER メソッドを使用して、FedSQL ステートメントの出力パラメーターを DS2 パッケージの METHOD ステートメントの IN_OUT パラメーターにマッピングします。パラメーターデータは、バインドされた変数で排他的に指定する必要があります。SET*type* メソッドと REGISTEROUTPARAMETER メソッドが呼び出されると、ランタイムエラーが発

生します。詳細については、"REGISTEROUTPARAMETER メソッド" (*SAS DS2 言語リファレンス*)を参照してください。

- SET*type* メソッドで指定された値

  これらのメソッドの詳細については、"DS2 SQLSTMT パッケージのメソッド、演算子、およびステートメント" (*SAS DS2 言語リファレンス*)を参照してください。

パラメーター値は、バインドされた変数または SET*type* メソッドで排他的に指定する必要があります。

注: FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL が DS2 ではなく SQL ステートメントを含む文字列を解析するために発生します。

接続文字列の指定

接続文字列は、データへの接続方法を定義します。接続文字列は、サブミットするクエリ言語と、1 つまたは複数のデータソースに接続するために必要な情報を識別します。

SQLSTMT パッケージを宣言してインスタンス化するときに、接続文字列を指定できます。接続文字列パラメーターは、主に SAS Federation Server で使用するために設計されています。完全に指定された接続文字列の作成の詳細については、*SAS Federation Server: 管理者ガイド*を参照してください。

接続文字列が提供されていない場合、SQLSTMT パッケージインスタンスは、現在割り当てられている libref の属性を使用して、HPDS2 または DS2 プロシジャによって生成された接続文字列を使用します。

FedSQL ステートメントの実行

EXECUTE メソッドは、FedSQL ステートメントを実行し、ステータスインジケーターを返します。正常に実行されるとゼロ、エラーが発生した場合は 1、データがない場合(NODATA)は 2 が返されます。NODATA 条件は、SQL UPDATE または DELETE ステートメントがどの行にも影響を与えない場合に存在します。

FedSQL ステートメントが実行されると、バインドされた変数の値が読み取られ、ステートメントのパラメーターの値として使用されます。

SQLSTMT インスタンスは、1 つの結果セットのみを維持します。前回の実行の結果セットがある場合は、FedSQL ステートメントが実行される前に解放されます。

FedSQL ステートメントは、実行時に動的に実行されます。ステートメントは実行時に準備されるため、DS2 プログラムの実行中に動的に構築およびカスタマイズできます。

結果セットデータへのアクセス

FETCH メソッドは、結果セットから次のデータ行を返します。ステータスインジケーターが返されます。正常に実行されるとゼロ、エラーが発生した場合は 1、データがない場合(NODATA)は 2 が返されます。フェッチされる次の行が結果セットの終了後にある場合、NODATA 条件が存在します。

変数が BINDRESULTS メソッドまたは FETCH メソッドで結果セットの列にバインドされている場合、各結果セットの列のフェッチされたデータは、その列にバインドされている変数に配置されます。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

注: 文字データの場合、結果セットの列データがすべて取得されるまで、GET*type* メソッドを繰り返し呼び出すことができます。数値データの場合、GET*type* メソッドを 1 回だけ呼び出して、結果セットの列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。詳細については、GET*type* メソッドを"DS2 SQLSTMT パッケージのメソッド、演算子、およびステートメント" (*SAS DS2 言語リファレンス*)で参照してください。

SQLSTMT インスタンスは、1 つの結果セットのみを維持します。CLOSERESULTS メソッドは、FedSQL ステートメントが実行または削除されると、結果セットを自動的に解放します。

FedSQL ステートメントの実行前に FETCH メソッドが呼び出されると、ランタイムエラーが発生します。

SQLSTMT パッケージと SQLEXEC 関数の比較

次の表は、SQLSTMT パッケージと SQLEXEC 関数を比較したものです。

SQLSTMT パッケージ	SQLEXEC 関数
FedSQL ステートメントが複数回実行される場合に適用可能	FedSQL ステートメントが 1 回だけ実行される場合に適用可能
実行時に FedSQL ステートメントを動的に割り当て、準備、実行、解放する	実行時に FedSQL ステートメントを動的に割り当て、準備、実行、解放する
パラメーターの受け渡しをサポートする	パラメーターの受け渡しをサポートしない
結果セットを生成する	結果セットを生成しない
ランタイム SELECT クエリ生成をサポートする	SELECT ステートメントでは使用できない

SQLSTMT パッケージ	SQLEXEC 関数
Java Database Connectivity (JDBC) PreparedStatement クラスと同様	SQL EXECUTE IMMEDIATE ステートメントまたは JDBC Statement.executeUpdate(String)メソッドと同様

TZ パッケージの使用

TZ パッケージの概要

DS2 は、他のデータソースで使用されている SQL スタイルの日付と時間の規則をサポートしています。データソースが SAS データセットでない場合、DS2 は DATE、TIME、および TIMESTAMP のデータ型を持つ日付と時間を処理できます。SAS の日付、時間、および日時の値は、TO_DATE、TO_TIME、および TO_TIMESTAMP 関数を使用して、DS2 の日付、時間、およびタイムスタンプの値に変換できます。ただし、これらの関数にはタイムゾーンが組み込まれていません。

TZ パッケージを使用すると、ローカルおよび国際的な時刻と日付の値を処理できます。

TZ パッケージの宣言とインスタンス化

TZ パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package tz tzpkg;
tzpkg = _new_ tz();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package tz tzpkg();
```

詳細については、“DECLARE PACKAGE ステートメント: TZ パッケージ” (*SAS DS2 言語リファレンス*)、および “_NEW_ 演算子: TZ パッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

時間とタイムゾーンの情報を返す

TZ パッケージを使用して、次の値を返すことができます。

- 現在のローカル時間
- 現在の協定世界時(UTC)時間

- 現在のタイムゾーン ID
- 現在のタイムゾーン名
- 指定されたローカル時間での UTC からのタイムゾーンのタイムゾーンオフセット
- 指定された UTC 時間での UTC からのタイムゾーンのタイムゾーンオフセット

TZ パッケージを使用して世界時計を取得する方法の例を次に示します。

```
data _null_ ;
  method init();

    declare package tz tzone() ;
    dcl double tokyo_time london_time new_york_time utc_time ;

    tokyo_time = tzone.getLocalTime('Asia/Tokyo') ;
    london_time = tzone.getLocalTime('Europe/London') ;
    new_york_time = tzone.getLocalTime('America/New_York') ;

    utc_time = tzone.getUTCTime();

    put utc_time = datetime. ;
    put tokyo_time = datetime. ;
    put london_time = datetime. ;
    put new_york_time = datetime. ;
  end ;
enddata ;
run;
```

次の行が SAS ログに書き込まれます。

```
utc_time=18NOV14:13:53:11
tokyo_time=18NOV14:22:53:11
london_time=18NOV14:13:53:11
new_york_time=18NOV14:08:53:11
```

これらのアクションを実行するためのメソッドの詳細については、"DS2 TZ パッケージのメソッド、演算子、およびステートメント" (*SAS DS2 言語リファレンス*)を参照してください。タイムゾーン ID と名前の詳細については、"タイムゾーン ID とタイムゾーン名" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照してください。

タイムゾーンオフセットを返す

TZ パッケージを使用して、指定されたローカル時間または指定された UTC 時間の UTC からのタイムゾーンオフセットを返すことができます。

タイムゾーンオフセットは、タイムゾーンと UTC の時間差と分差を+|–*hhmm* または+|–*hh*:*mm* の形式で示します。

'asia/tokyo'と'America/New_York'からのタイムゾーンオフセットを返す例を次に示します。

```
data _null_ ;
  method init();
```

```
 declare package tz tzone('asia/tokyo') ;
 dcl double new_york local_time;
 dcl char(40) cstr ;

 local_time = tzone.getOffset() ;
 put local_time time.;

 new_york = tzone.getOffset('America/New_York') ;
 put new_york time.;


 end;
 enddata ;
run;
```

次の行が SAS ログに書き込まれます。

```
9:00:00
-4:00:00
```

ローカル時間または UTC 時間の変換

TZ パッケージを使用して、ローカルを次のいずれかの時間形式に変換できます。

- ISO8601 (タイムゾーンオフセットありまたはなし)
- タイムゾーン ID を含む TIMESTAMP 文字列
- UTC 時間

さらに、UTC 時間をローカル時間に変換することもできます。

次に例を示します。

```
data _null_ ;
  method init();

    declare package tz tzone('asia/tokyo') ;
    dcl double local_time ;
    dcl char(35) local_time_iso local_time_utc local_time_tz;


    local_time = tzone.tolocaltime(15550) ;
    put local_time time.;

    local_time_iso = tzone.toiso8601(15500) ;
    put local_time_iso ;

    local_time_utc = tzone.toutctime(15500) ;
    put local_time_utc time.;

    local_time_tz =tzone.totimestampz(15500);
    put local_time_tz;
```

```
  end;
enddata ;
run;
```

次の行が SAS ログに書き込まれます。

```
13:19:10
1960-01-01T04:18:20.00+09:00
-4:41:40
1960-01-01 04:18:20.00 asia/tokyo
```

14

スレッド処理

スレッド処理の概要

注: DS2 が CAS サーバー上でどのように実行されるかについての詳細は、“CAS での DS2 の実行方法” (297 ページ)を参照してください。

通常、DS2 コードは順次実行されます。つまり、あるプロセスが完了するまで実行されてから、次のプロセスが開始されます。**スレッド処理**を使用して、複数のプロセスを同時に実行することができます。スレッド処理では、同時に実行されるコードの各セクションは、**スレッド**で実行されていると言われます。DS2 スレッドは、複数のコアを備えたマシンでも超並列処理(MPP)データベース内でもうまく機能します。

DS2 プログラムは、入力データを処理し、出力データを生成します。DS2 プログラムは、プログラムまたはスレッドとして、2 つの異なる方法で実行できます。DS2 プログラムをプログラムとして実行した場合の結果を次に示します。

- 入力データには、データベーステーブルの行と DS2 プログラムスレッドの行の両方を含めることができます。
- 出力データは、クライアントアプリケーションに返されるデータベーステーブルまたは行のいずれかになります。

DS2 プログラムをスレッドとして実行した場合の結果を次に示します。

- 入力データには、データベーステーブルの行のみを含めることができ、他のスレッドの行は含めることができません。

- 出力データには、スレッドを開始した DS2 プログラムに返される行が含まれます。

DS2 コードをスレッドで実行できるようにするには、次の操作を実行します。

1 DS2 コードを THREAD...ENDTHREAD ステートメントで囲んでスレッドを作成します。

2 DECLARE THREAD ステートメントを使用して、DS2 プログラム内にスレッドのインスタンスを 1 つ以上作成します。

3 SET FROM ステートメントを使用して 1 つまたは複数のスレッドを実行します。

この例では、THREAD ステートメントを使用して、非常に単純なスレッド T を作成します。

```
thread t;
  dcl int x;
  method init();
    dcl int i;
    do i = 1 to 3;
      x = i;
      output;
    end;
  end;
endthread;
```

この DS2 プログラムでは、T のインスタンスが宣言され、RUN メソッドの SET FROM ステートメントを使用して 2 つのスレッドが実行されます。2 つのスレッドのそれぞれが x に対して 3 つの行を生成し、出力テーブルに合計 6 つの行が生成されます。

```
 data;
   dcl thread t t_instance;
   method run();
    set from t_instance threads=2;
    put 'x= ' x ;
   end;
 enddata;
```

DS2 プログラムを実行すると、SAS ログに次の出力が表示される場合があります。スレッドの処理方法により、出力の順序が異なる可能性があります。

```
x= 1
x= 2
x= 3
x= 1
x= 2
x= 3
```

注: 1 つの計算スレッドが I/O スレッドに追いつくことができる場合、その 1 つのスレッドがすべての計算に使用されます。

注: 1 つのリーダーがすべてのスレッドにフィードします。スレッドプログラムの SET ステートメントは、その SET ステートメントの単一のリーダーを共有します。入力テーブルの各行は、正確に 1 つのスレッドに送信されます。

注: データプログラムとスレッドプログラムは型が一致している必要があります。10 進データ型の場合、精度とスケールの両方が一致する必要があります。文字列の場合、列サイズ、文字セット、および文字列が可変長であるか固定長であるかは、すべて一致する必要があります。

詳細については、"THREAD ステートメント" (*SAS DS2 言語リファレンス*)を参照してください。

スレッドと DS2 プログラム

シリアルプログラムと並列プログラムの概要

DS2 プログラムは、複数のデータ行に対して同時に操作を実行できるため、大きなデータセットの処理に要する時間が短縮されます。DS2 プログラムの構造に基づいて、DS2 コンパイラは、複数の行に対して同時に実行できる操作と、各行に順次適用する必要がある操作を決定します。DS2 プログラムは、シリアルプログラム、パラレルプログラム、またはパラレル-シリアルプログラムに分類されます。

シリアルプログラム
: 行間のデータ依存関係を持つ操作が含まれます。したがって、行は順次処理する必要があります。データプログラムはデータセットを処理し、結果セットを生成します。

パラレルプログラム
: 行間のデータ依存関係を持つ操作は含まれません。したがって、複数のデータ行を並列処理できます。各スレッドは、データセットのサブセットを処理し、結果セットのサブセットを生成します。

パラレル-シリアルプログラム
: 行間にデータ依存関係のある操作と、データ依存関係のない操作が含まれています。操作の処理は、パラレルステージとシリアルステージの 2 つのステージに分けられます。パラレルステージでは、各スレッドが入力データセットのサブセットを処理し、中間データセットのサブセットを生成します。シリアルステージでは、1 つのスレッドが完全な中間データセットを処理し、完全な結果セットを生成します。

注: 詳細については、"CAS での DS2 の実行方法" (297 ページ)を参照してください。

データ操作

DS2 データ操作は、シリアル操作または並列操作のいずれかに分類されます。

シリアル操作
行間でデータの依存関係を持つ操作。

- シリアル操作は、各データ行に順次適用する必要があります。
- シリアル操作は、DS2 データプログラム内のステートメントによって実装されます。

並列操作
行間でデータの依存関係がない操作。

- 並列操作は、複数のデータ行に並列に適用できます。
- 並列操作は、DS2 スレッドプログラム内のステートメントによって実装されます。

DS2 コンパイラは、DS2 プログラム内のステートメントの配置に基づいてステートメントを分類します。DS2 コンパイラは、スレッドプログラム内のすべてのステートメントが並列操作を実装し、データプログラム内のすべてのステートメントがシリアル操作を実装すると想定します。唯一の例外は、データの入出力ステートメントです。DS2 コンパイラは、SET、SET FROM、または OUTPUT をデータ操作ステートメントとは見なしません。

DS2 プログラムの概要

DS2 プログラムは、含まれるプログラムのタイプとデータ操作のタイプに基づいて、シリアルプログラム、並列プログラム、または並列-シリアルプログラムのいずれかに分類されます。

DS2 シリアルプログラム
シリアル操作のみを含むプログラム。

- データプログラムのみがあります(スレッドプログラムなし)。
- データプログラムには、シリアルデータ操作が含まれます。

DS2 並列プログラム
並列操作のみを含むプログラム。

- スレッドプログラムとデータプログラムがあります。
- スレッドプログラムには、並列データ操作が含まれています。
- データプログラムには、データ操作が含まれていません。つまり、データプログラムには、SET FROM と OUTPUT 以外のステートメントは含まれていません。

DS2 並列-シリアルプログラム
プログラムには、並列操作とシリアル操作の両方が含まれています。

- スレッドプログラム(パラレルステージ)とデータプログラム(シリアルステージ)があります。
- スレッドプログラムには、並列データ操作が含まれています。
- データプログラムには、シリアルデータ操作が含まれます。つまり、データプログラムには、SET FROM と OUTPUT 以外に少なくとも 1 つのステートメントが含まれています。

DS2 スレッドで役立つ自動変数

DS2 スレッド間で問題をサブセット化するために使用される自動変数がいくつかあります。これらの自動変数は、PUT ステートメントを使用してデバッグするときにコンテキストを提供するのにも役立ちます。

変数	説明
HOSTNAME	DS2 プログラムが実行されているワーカーノードまたはホストの名前。
LOCALNTHREADS	ローカルノードで実行されているプログラムの DS2 スレッドの総数。たとえば、THREADS=10 の SET FROM ステートメントを含む DS2 プログラムが、4 つのワーカーを持つ CAS サーバーで実行される場合、_LOCALNTHREADS_は 10 になります。単一ノード環境では、すべてのスレッドがローカルノードで実行されるため、_LOCALNTHREADS_は_NTHREADS_と同じ値になります。
LOCALTHREADID	ローカルノードの現在の DS2 スレッドに割り当てられた一意の数値識別子。たとえば、THREADS=10 の SET FROM ステートメントを含む DS2 プログラムが、4 つのワーカーを持つ CAS サーバーで実行される場合、_LOCALTHREADID_は 1 から 10 までの値を返します。単一ノード環境では、_LOCALTHREADID_は_THREADID_と同じ値になります。
NTHREADS	プログラムで実行されている DS2 スレッドの総数。並列環境では、_NTHREADS_は、DS2 プログラムが実行されているすべてのノードにわたる DS2 スレッドの総数です。たとえば、THREADS=10 の SET FROM ステートメントを含む DS2 プログラムが、4 つのワーカーを持つ CAS サーバーで実行される場合、_NTHREADS_は 40 になります。
THREADID	現在の DS2 スレッドに割り当てられた一意の数値識別子。並列環境では、_THREADID_は、DS2 プログラムが実行されているすべてのノードにあるすべての DS2 スレッド間で重複しません。たとえば、THREADS=10 の SET FROM ステートメントを含む DS2 プログラムが、4 つのワーカーを持つ CAS サーバーで実行される場合、_THREADID_は 1 から 40 までの値を返します。

_NTHREADS_と_LOCALNTHREADS_の両方が、THREAD ブロックで実行されているスレッドの数を参照します。DATA ブロックで実行されているスレッドはカウントされません。

THREADID_の値は 1 から_NTHREADS_までです。一方、_LOCALTHREADID_の値は 1 から_LOCALNTHREADS_までです。DATA ブロック内で実行されているスレッドに対して、_THREADID と_LOCALTHREADID_のどちらも 0 になります。

15

DS2 と FedSQL の使用

DS2 から FedSQL ステートメントを動的に実行する

注: DS2 アクションまたは PROC DS2 の SET ステートメントを除いて、DS2 プログラム内での FedSQL クエリの使用は、現在 CAS サーバーではサポートされていません。

DS2 プログラム内から FedSQL ステートメントを埋め込んで実行できます。次の場合に FedSQL を DS2 で使用できます。

- DS2 パッケージメソッド式を FedSQL SELECT ステートメントの関数として呼び出すことができます。

  詳細については、“式での DS2 パッケージの使用” (*SAS FedSQL 言語リファレンス*) を参照してください。

- SQLSTMT パッケージを使用して、FedSQL ステートメントを生成、準備、および実行し、実行時にテーブルから行を作成、選択、変更、挿入、削除できます。

  SQLSTMT パッケージは、複数回実行される FedSQL ステートメント、パラメーターを含むステートメント、または結果セットを生成するステートメントで使用することを目的としています。詳細については、“SQLSTMT パッケージの使用” (232 ページ)を参照してください。

- また、SQLEXEC 関数を使用して、FedSQL ステートメントを生成、準備、および実行し、実行時にテーブルから行を作成、選択、変更、挿入、削除できます。

  SQLEXEC 関数は、1 回だけ実行され、パラメーターを持たず、結果セットを生成しないステートメントで使用することを目的としています。

  詳細については、“SQLEXEC 関数” (*SAS DS2 言語リファレンス*)を参照してください。

- DECLARE PACKAGE ステートメントまたは DATASET メソッドで FedSQL SELECT ステートメントを使用して、実行時にデータをハッシュインスタンスにロードできます。

  詳細については、“FedSQL クエリとハッシュインスタンスを使用して実行時に動的に行を取得する” (206 ページ)、“DATASET メソッド” (*SAS DS2 言語リファレンス*)および“DECLARE PACKAGE ステートメント: ハッシュパッケージ” (*SAS DS2 言語リファレンス*)を参照してください。

- SET ステートメントを使用して、FedSQL SELECT ステートメントでデータを読み込むことができます。

  詳細については、“FedSQL が埋め込まれた SET ステートメント” (253 ページ)および“SET ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

16

DS2 入力と出力

DS2 入力と出力の概要

次の方法を使用して、DS2 データを読み取ることができます。

- DS2 プログラムのパッケージおよびメソッド内で変数または配列の割り当てを使用して、データを生成できます。
- SET ステートメントを使用して、既存のテーブルを読み取ることができます。

  詳細については、“SET ステートメントを使用したデータの読み取り” (250 ページ)および“SET ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。
- ハッシュパッケージを使用してデータを読み取ることができます。

  詳細については、“ハッシュパッケージを使用したデータの読み取り” (254 ページ)を参照してください。
- SELECT ステートメントを実行して結果セットにアクセスすることにより、SQLSTMT パッケージを使用してデータを取得できます。

注: SQLSTMT パッケージは、CAS サーバーではサポートされていません。

詳細については、“SQLSTMT パッケージと SQLEXEC 関数を使用したデータの読み取りと書き込み” (255 ページ)を参照してください。

次の方法を使用して、DS2 データを書き込むことができます。

- OUTPUT ステートメントを使用して DS2 データを書き込みます。OUTPUT ステートメントは、結果テーブルに行を書き込みます。

  詳細については、“OUTPUT ステートメントを使用したデータの書き込み” (255 ページ)を参照してください。

- ハッシュパッケージの OUTPUT メソッドを使用できます。

  詳細については、“OUTPUT メソッド” (*SAS DS2 言語リファレンス*)および“ハッシュパッケージデータをテーブルに保存する” (203 ページ)を参照してください。

- SQLEXEC 関数または SQLSTMT パッケージで FedSQL INSERT および UPDATE ステートメントを使用できます。

  注: SQLEXEC 関数と SQLSTMT パッケージは、CAS サーバーではサポートされていません。

  “SQLSTMT パッケージと SQLEXEC 関数を使用したデータの読み取りと書き込み” (255 ページ)、“SQLEXEC 関数” (*SAS DS2 言語リファレンス*)、および“結果セットデータへのアクセス” (236 ページ)。

注: MongoDB データソースには、テーブルを作成するための特別な要件があります。データに応じて、ルートテーブル、親テーブル、および子テーブルを作成することをお勧めします。DS2 ではルートテーブルのみを作成できます。親テーブルと子テーブルを作成するには、FedSQL を使用する必要があります。*SAS/ACCESS for Relational Databases: リファレンス*の MongoDB に関する情報を参照してくださ。

SET ステートメントを使用したデータの読み取り

SET ステートメントの概要

SET ステートメントは柔軟性があり、DS2 プログラミングでさまざまな用途があります。これらの使用法は、SET ステートメントで使用するオプションとステートメントによって決まります。

- DS2 プログラムでさらに処理するために、既存のテーブルから行と列を読み取る

- テーブルの連結とインターリーブ、およびテーブルの 1 対 1 の読み取りの実行

  詳細については、17 章, “テーブルの結合” (259 ページ)を参照してください。

SET ステートメントが実行されるたびに、1 行がプログラムデータベクトルに読み込まれます。SET は、特に指定しない限り、入力テーブルからすべての 列とすべての行を読み取ります。SET ステートメントには複数のテーブルを含めることができ、DS2 プログラムには、複数の SET ステートメントを含めることができます。

詳細については、“SET ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

注: スレッドプログラムの SET ステートメントは、その SET ステートメントの単一のリーダーを共有します。SET ステートメントが実行されるたびに、指定された入力テーブルから 1 行が読み取られます。ただし、すべてのスレッドが単一のリーダーを共有します。入力テーブルの各行は、正確に 1 つのスレッドに送信されます。INIT メソッドで SET ステートメントを使用する場合、最初のスレッドがすべての行を読み取ります。他のスレッドはファイルの終わりに到達し、関連する RUN メソッドに進まずに処理を終了します。したがって、スレッドプログラムの INIT または TERM メソッドでは、SET ステートメントを使用しないことをお勧めします。

注: SET ステートメントは、RUN メソッドの暗黙的なループ機能を利用できるように、RUN メソッドでの使用が適しています。

SET ステートメントのコンパイル

DS2 コンパイラは、SET ステートメントを評価するときに、各 *table-reference* の列情報を読み取ります。各テーブルの列ごとに、列の型属性と同じ型属性を持つグローバル変数が DS2 プログラムに作成されます。このようにして、SET ステートメントは関連する列変数のセットを作成します。グローバル変数がすでに存在し、その型がテーブル列の型と一致しない場合、エラーが発生します。これは、2 つのテーブルに同じ名前の列がある場合、それらの列の型に互換性がある必要があることを意味します。

SET ステートメントの実行

データプログラムでは、SET ステートメントが実行されると、最初のテーブルから最初の行が読み取られます。連続して実行すると、現在のテーブルが完全に読み取られるまで行が読み取られます。次に、次のテーブルの最初の行が読み取られます。行が読み取られるたびに、行の値が対応する列変数に割り当てられます。SET ステートメントによって作成された列変数は保持されます。SET ステートメントの実行は、すべてのテーブルのすべての 行が完全に読み取られると終了します。

スレッドプログラムの場合、特定のスレッドに入る行の順序は、BY ステートメントを使用しない限り未定義です。その場合でも、スレッドのスレッドプログラムでは、

すべての行が他のテーブルからの行の前に受信されると想定する方法はありません。

読み取られているテーブルに表示されない列変数は、SAS モードか ANSI モードかに応じて、SAS 欠損値または null 値に設定されます。SET ステートメントで宣言された変数は、次のように初期化されます。

- SAS モードの場合:
  - DOUBLE データ型は、SAS 数値欠損値(.)に初期化されます。
  - 固定幅の CHAR および NCHAR データ型は、SAS 文字欠損値(空白で埋められた文字列)に初期化されます。
  - VARCHAR や NVARCHAR など、その他すべてのデータ型は ANSI null に初期化されます。
- ANSI モードでは、すべての型が ANSI null に初期化されます。

注: 詳細については、7 章, “DS2 による null および SAS 欠損値の処理方法” (61 ページ)を参照してください。

次に例を示します。このプログラムでは、2 行目と 4 行目に列変数 *modifiers* が指定されていません。OUTPUT メソッドは、最後の OUTPUT ステートメントの後に値が割り当てられているかどうかにかかわらず、すべての定義済み変数の値を使用します。

```
proc ds2;
data test (overwrite=yes);
  dcl char string1 string2 modifiers having informat $char8. format $char8.;
  method init();
    string1='aBc'; string2='AbC'; modifiers='i'; output;
    string1='   abc'; string2='abc'; output;
    string1='   abc'; string2='abc';  modifiers='l'; output;
    string1=' abc'; string2='   abx'; output;
    string1=' abc'; string2='   abx'; modifiers='l';  output;
  end;
enddata;
run;

data test_out (overwrite=yes);
  dcl double result;
  method run();
    set test;
    result=compare(string1, string2, modifiers);
    put 'String 1= ' string1 'String 2= ' string2 'Modifier= ' modifiers
      'Result= ' result;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
String 1= aBc    String 2= AbC    Modifier= i     Result= 0
String 1=   abc  String 2= abc    Modifier= i     Result= -1
String 1=   abc  String 2= abc    Modifier= l     Result= 0
String 1= abc    String 2=   abx  Modifier= l     Result= -3
String 1= abc    String 2=   abx  Modifier= l     Result= -3
```

2 行目と 4 行目が *modifiers* 列変数に指定された値を持たないようにするには、変数を欠損値または null 値に設定する必要があります。

```
proc ds2;
data test (overwrite=yes);
  dcl char string1 string2 modifiers having informat $char8. format $char8.;
    method init();
      string1='aBc'; string2='AbC'; modifiers='i'; output;
      string1='   abc'; string2='abc'; modifiers=' '; output;
      string1='   abc'; string2='abc';  modifiers='l'; output;
      string1=' abc'; string2='   abx'; modifiers=' '; output;
      string1=' abc'; string2='   abx'; modifiers='l';  output;
    end;
enddata;
run;

data test_out (overwrite=yes);
  method run();
    set test;
    result=compare(string1, string2, modifiers);
    put 'String 1= ' string1 'String 2= ' string2 'Modifier= ' modifiers
      'Result= ' result;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
String 1= aBc    String 2= AbC    Modifier= i     Result= 0
String 1=   abc  String 2= abc    Modifier=       Result= -1
String 1=   abc  String 2= abc    Modifier= l     Result= 0
String 1= abc    String 2=   abx  Modifier=       Result= 2
String 1= abc    String 2=   abx  Modifier= l     Result= -3
```

FedSQL が埋め込まれた SET ステートメント

次の例のように、SET ステートメントは FedSQL コードを使用してテーブルを 読み取ることができます。

```
set {select * from catalog_base.investment};
```

テーブル名と埋め込み FedSQL を SET ステートメントにインターリーブすることができます。この例では、SET ステートメントは同じテーブルを 2 回読み取ります。

```
set {select * from catalog_base.investment} catalog_base.investment;
```

SET ステートメントで使用される埋め込み FedSQL は、有効な FedSQL コードである必要があり、テーブル行のセットに解決される必要があります。それ以外の場合は、エラーが発生します。

注: 埋め込み SQL に含まれる内容に関係なく、埋め込み SQL から表示される行の順序は保証されません。

```
set {select * from sql13a order by "X", "Y"};
```

一部の環境では、ORDER BY 句によって課された順序が保持される場合がありますが、他の環境では保持されません。DS2 は埋め込み SQL を追加のコードでラップします。また、DS2 は FIRST または LAST 情報を検出および生成できないため、ORDER BY 句は受け入れられません。よりよい方法は、埋め込み SQL テキストの外側で BY を使用してプログラムを作成することです。

```
set {select * from sql13a}; by "X" "Y";
```

CAS サーバーでは、ORDER BY 順序はデータプログラムまたはスレッドプログラムのいずれにも保持されません。これは、埋め込み SQL の結果が FedSQL アクションによって一時テーブルに保存されるために発生します。ORDER BY 順序は、一時テーブルが作成される前に FedSQL アクションによって適用されます。一時テーブルが DS2 アクションによって読み取られるとき、順序は暗示されません。

ハッシュパッケージを使用したデータの読み取り

ハッシュパッケージを使用して、テーブルからデータを読み取ることができます。

DECLARE PACKAGE ステートメント、_NEW_演算子、および DATASET メソッドは、次のいずれかの方法を使用してデータソースを識別します。

- テーブルを識別する名前
- テーブル行のセットに解決される有効な FedSQL SELECT ステートメント

  注: FedSQL クエリを使用してデータを選択することは、CAS サーバーではサポートされていません。

注: DS2 ハッシュテーブルから FCMP パッケージにデータをロードすることはできません。

詳細については、“ハッシュパッケージの使用” (194 ページ)を参照してください。

SQLSTMT パッケージと SQLEXEC 関数を使用したデータの読み取りと書き込み

注: SQLSTMT パッケージと SQLEXEC 関数は、CAS サーバーではサポートされていません。

SQLSTMT パッケージと SQLEXEC 関数を使用すると、DS2 プログラムで FedSQL ステートメントを動的に生成、準備、および実行して、テーブルの行を更新、挿入、または削除できます。SQLSTMT パッケージまたは SQLEXEC 関数のインスタンスを使用すると、FedSQL ステートメントの割り当て、準備、実行、解放が実行時に行われます。

さらに、SQLSTMT パッケージは、実行された FedSQL ステートメントによって生成される結果セットを調べることができます。

注意
SQLEXEC 関数および SQLSTMT パッケージで使用できるのは、FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

詳細については、"SQLSTMT パッケージの使用" (232 ページ)および"SQLEXEC 関数" (*SAS DS2 言語リファレンス*)を参照してください。

ヒント　SQLSTMT パッケージでパラメーター化された FedSQL クエリを使用することにより、KEY=を指定した DATA ステップの SET ステートメントの機能を実現できます。

OUTPUT ステートメントを使用したデータの書き込み

OUTPUT ステートメントは、OUTPUT ステートメントの実行時にグローバル変数に含まれる行の値を使用して、出力行を作成します。OUTPUT ステートメントは、DS2 プログラムの最後ではなく、すぐに現在の行をテーブルに書き込みます。OUTPUT ステートメントでテーブル名が指定されていない場合、行は DATA ステートメントにリストされている 1 つまたは複数のテーブルに書き込まれます。

DS2 は、値が既存のテーブルから読み取られたかプログラムで作成さたかに関係なく、コンパイラが DS2 プログラム内で検出した順序で値を追跡します。

OUTPUT ステートメントを指定しなかった場合、DS2 は RUN メソッドの最後に暗黙的に 1 つ追加し、作成中の 1 つまたは複数のテーブルに行を書き込みます。

DS2 プログラムに明示的な OUTPUT ステートメントを配置すると、自動出力がオーバーライドされ、明示的な OUTPUT ステートメントが実行された場合にのみテーブルに行が追加されます。ただし、OUTPUT ステートメントを使用して任意の 1 つのテーブルに行を書き込むと、RUN メソッドの最後に暗黙的な OUTPUT ステートメントは存在しなくなります。この状況では、DS2 プログラムは、明示的な OUTPUT が実行された場合にのみテーブルに行を書き込みます。OUTPUT ステートメントは、単独で使用することも、IF-THEN/ELSE または SELECT ステートメントの一部として使用することも、DO ループ処理で使用することもできます。

注: スレッドプログラムの OUTPUT ステートメントにテーブル名を含めることはできません。各出力行は、スレッドを開始したデータプログラムに返されます。

SAS 外部のデータソースを使用する場合の出力テーブルの列の順序

一部のデータソースは、DS2 からの出力テーブルの列の優先順序を 1 つ選択します。たとえば、Hive では、BYPARTITION 列は常にテーブルの最後に移動します。これは、さまざまなデータソースがパフォーマンスを最適化しようとする場合によくあることです。

DS2 プログラムでの宣言の順序は、データソースの列の順序として使用されない場合があります。たとえば、keep K1- - K4;を使用すると、CREATE TABLE ステートメントで **K1** が **K4** の後に表示されるため、期待どおりに列を取得できないか、エラーが発生する可能性があります。

NLS トランスコーディングの失敗

トランスコーディングは、文字データをあるエンコーディングから別のエンコーディングに変換するプロセスです。NLS トランスコーディングの失敗は、行の入力または出力操作中、または文字列の割り当て中に発生する可能性があります。デフォルトでは、このランタイムエラーにより、行の処理が停止します。PROC FEDSQL および PROC DS2 で XCODE=オプションを使用して、デフォルトの動作を変更できます。

このオプションを使用すると、エラーを無視して行の処理を続行することを選択できます。

詳細については、*Base SAS プロシジャガイド*を参照してください。

17

テーブルの結合

データ結合の定義

DS2 処理のコンテキストでは、データの結合には次の意味があります。

- 連結
- インターリーブ
- 1 対 1 の読み取り
- マッチマージ

テーブルの結合に使用される 2 つのステートメントは、MERGE と SET です。

テーブルの結合: 基本概念

複数のテーブルに保存されている情報を結合する前に知っておくべきこと

多くのアプリケーションでは、データを処理して意味のある結果を生成する前に、入力データを特定の形式にする必要があります。通常、データは複数のソースから取得され、さまざまな形式になっている場合があります。したがって、データを分析したり、データからレポートを作成したりする前に、データを論理的に関連付けて処理するための中間的な手順を踏まなければならないことがよくあります。

アプリケーションの要件はさまざまですが、データにアクセスし、結合し、処理するすべてのアプリケーションに共通する要素があります。出力をどのように表示するかを決定したら、次のタスクを実行する必要があります。

- 入力データがどのように関連しているかを判断します。
- 必要に応じて、データが適切に並べ替えられていることを確認します。
- 入力データを処理するための適切なアクセス方法を選択します。
- 適切なツールを選択してタスクを完了します。

データを関連付ける 4 つの方法

データリレーションシップのカテゴリ

入力データの複数のソース間のリレーションシップは、各ソースに物理レベルまたは論理レベルのいずれかで共通のデータが含まれている場合に存在します。たとえば、従業員データと部門データは、共通の値を共有する従業員 ID 列を介して関連付けることができます。別のテーブルには、部分的な値が行番号によって別のテーブルに論理的に関連付けられた数値シーケンス番号を含めることができます。

データ内の既存のリレーションシップを識別できる必要があります。この知識は、入力データを処理して目的の結果を生成する方法を理解するために不可欠です。関連するすべてのデータは、次の 4 つのカテゴリのいずれかに分類されます。これらのカテゴリは、テーブル間での行の関係によって特徴付けられます。

- 1 対 1
- 1 対多

- 多対 1
- 多対多

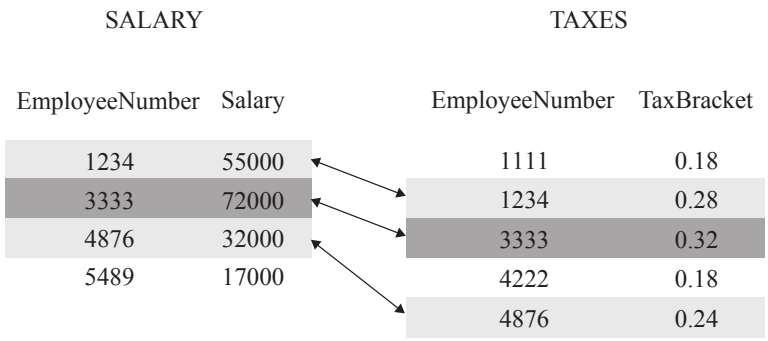
必要な結果を得るためには、これらの各方法でどのように行を結合するか、各方法で共通列の重複値をどのように扱うか、各方法で共通列の欠損値や不一致値をどのように扱うかを理解する必要があります。一部の方法では、テーブルを並べ替えて前処理する必要もあります。“テーブル結合方法” (263 ページ)の各方法の説明を参照してください。

1 対 1 リレーションシップ

1 対 1 リレーションシップでは、通常、選択した 1 つ以上の列の値に基づいて、1 つのテーブルの 1 つの行が別のテーブルの 1 つの行に関連付けられます。1 対 1 リレーションシップとは、選択した列の各値が各テーブルで 1 回しか出現しないことを意味します。選択した複数の列を操作する場合、このリレーションシップは、値の各組み合わせが各テーブルで 1 回しか出現しないことを意味します。

次の例では、Salary テーブルと Taxes テーブルの行が、EmployeeNumber の共通の値によって関連付けられています。

図 17.1　1 対 1 リレーションシップ

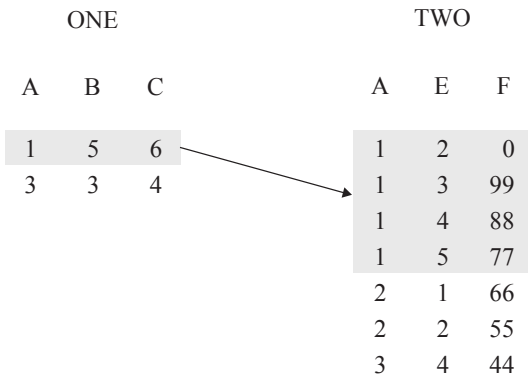


1 対多および多対 1 のリレーションシップ

入力テーブル間の 1 対多または多対 1 のリレーションシップは、1 つのテーブルには、選択された列の特定の値を持つ行が最大で 1 つあることを意味しますが、もう 1 つの入力テーブルには、各値が複数回出現する可能性があります。選択した複数の列を操作する場合、このリレーションシップは、値の各組み合わせが 1 つのテーブルで 1 回しか出現しないことを意味します。ただし、その組み合わせは他のテーブルで複数回出現する可能性があります。入力テーブルが処理される順序によって、リレーションシップが 1 対多か多対 1 かが決まります。

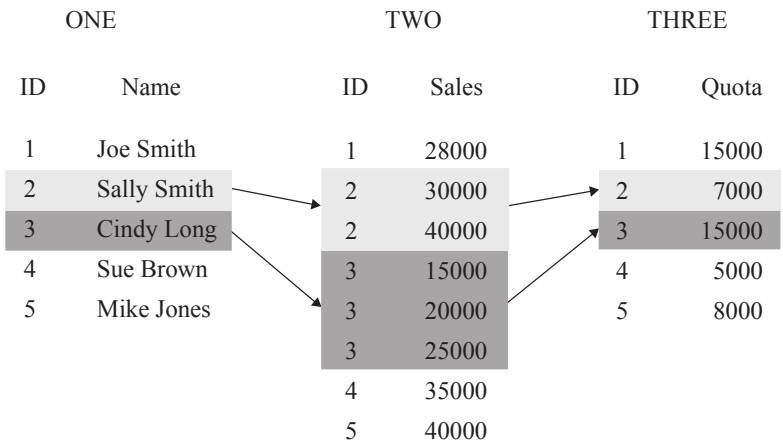
次の例では、テーブル One と Two の行は、列 A の共通の値によって関連付けられています。A の値は、テーブル One では一意ですが、テーブル Two では一意ではありません。

図 17.2　1 対多リレーションシップ



次の例では、テーブル One、Two、および Three の行が、列 ID の共通の値によって関連付けられています。ID の値は、テーブル One と Three では一意ですが、テーブル Two では一意ではありません。ID の値 2 と 3 の場合、テーブル One と Two の行の間には 1 対多リレーションシップが存在し、テーブル Two と Three の行の間には多対 1 リレーションシップが存在します。

図 17.3　1 対多および多対 1 のリレーションシップ

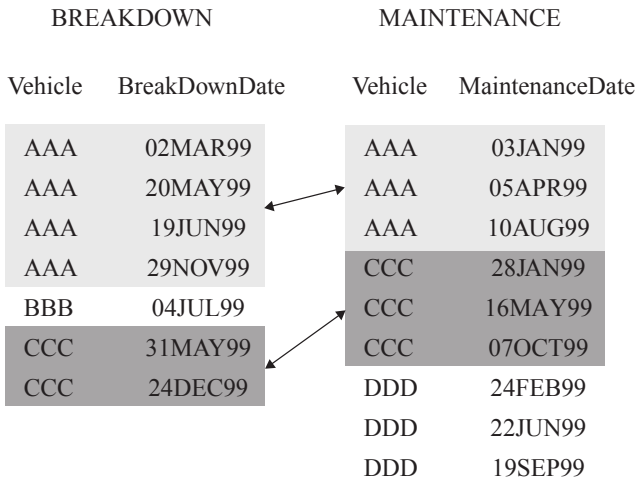


多対多リレーションシップ

多対多のカテゴリは、1 つ以上の共通列の値に基づいて、各入力テーブルの複数の行を関連付けることができることを意味します。

次の例では、BreakDown テーブルと Maintenance テーブルの行が、Vehicle 列の共通の値によって関連付けられています。Vehicle の値は、どちらのテーブルでも一意ではありません。Vehicle の値 AAA と CCC について、これらのテーブルの行間に多対多リレーションシップが存在します。

図 17.4 多対多リレーションシップ



テーブル結合方法の概要

テーブル結合方法

これらの方法を使用してテーブルを結合することができます。

- 連結
- インターリーブ
- 1 対 1 の読み取り
- マッチマージ

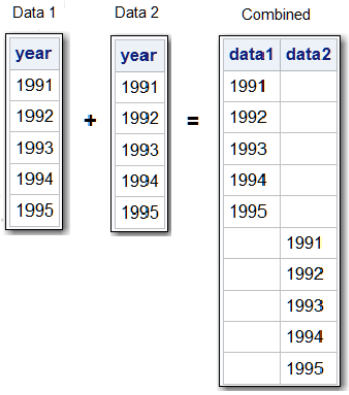
連結

次の図は、2 つのテーブルを連結した結果を示しています。テーブルを連結すると、あるテーブルの行が別のテーブルに追加されます。データプログラムは、すべての行が処理されるまで Data1 を順次読み取り、次に Data2 を読み取ります。テーブル Combined には、連結の結果が含まれます。テーブルは、SET ステートメントにリストされている順序で処理されることに注意してください。

```
data concatenate;
  method run();
    set data1 data2;
  end;
enddata;
run;
```

注: スレッドプログラムの場合、特定のスレッドに入る行の順序は、BY ステートメントを使用しない限り未定義です。その場合でも、スレッドのスレッドプログラムでは、すべての行が他のテーブルからの行の前に受信されると想定する方法はありません。

図 17.5　2 つのテーブルの連結



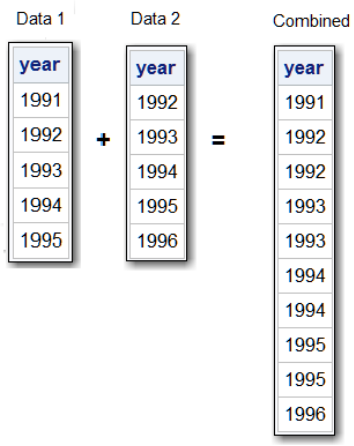
詳細については、“DS2 テーブルの結合: 方法” (270 ページ)を参照してください。

インターリーブ

次のデータプログラムは、2 つのテーブルをインターリーブします。インターリーブは、1 つ以上の共通の列に基づいて、2 つ以上のテーブルの行を散在させます。テーブル Combined は結果を示しています。

```
data interleave;
  method run();
    set data1 data2;
    by year;
  end;
enddata;
run;
```

図 17.6 2 つのテーブルのインターリーブ



詳細については、“DS2 テーブルの結合: 方法” (270 ページ)を参照してください。

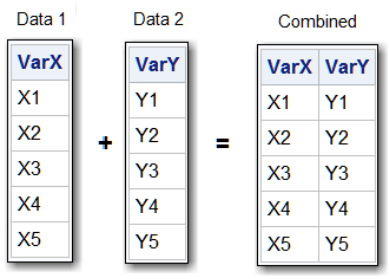
1 対 1 の読み取り

次の図は、1 対 1 の読み取りの結果を示しています。1 対 1 の読み取りでは、寄与する各テーブルのすべての 列を含む行を作成することにより、2 つ以上のテーブルの行を結合します。行は、各テーブル内の相対的な位置に基づいて結合されます。つまり、あるテーブルの最初の行と別のテーブルの最初の行というように結合されます。最小のテーブルから最後の行を読み取った後、データプログラムは停止します。テーブル Combined は結果を示しています。

```
data one2one;
  method run();
    set data1;
    set data2;
  end;
enddata;
run;
```

次のデータプログラムでは、2 つのテーブルが 1 対 1 で読み取られます。

図 17.7 1 対 1 の読み取り



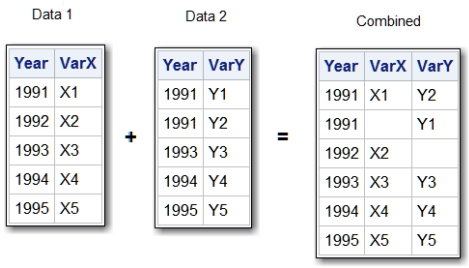
詳細については、“DS2 テーブルの結合: 方法” (270 ページ)を参照してください。

マッチマージ

次の図は、マッチマージの結果を示しています。マッチマージは、1 つ以上の共通列の値に基づいて、2 つ以上のテーブルの行を新しいテーブルの 1 つの行に結合します。次のデータプログラムでは、2 つのテーブルが 1 対 1 で読み取られます。テーブル Combined は結果を示しています。

```
data one2one;
  method run();
    merge data1 data2;
    by year;
  end;
enddata;
run;
```

図 17.8　2 つのテーブルのマッチマージ



詳細については、“DS2 テーブルの結合: 方法” (270 ページ)を参照してください。

テーブルの準備方法

テーブルを準備するためのガイドライン

テーブルを結合する前に、次のガイドラインに従って、必要な結果を生成してください。

- テーブルの構造と内容を理解します。
- 一般的な問題の原因を調べます。
- 行が正しい順序になっていること、または正しい順序で取得できること(たとえば、事前に並べ替えることによって)を確認してください。
- プログラムをテストします。

テーブルの構造と内容を理解する

データがどのように関連しているかを判断するには、テーブルの構造を調べてください。テーブル構造を表示するには、DATASETS プロシジャまたは CONTENTS プロシジャを実行して記述子情報を表示します。記述子情報には、各テーブルの行数、各列の名前と属性、および拡張属性(テーブルと列の拡張属性を含む)のアルファベット順リストが含まれます。行のサンプルを印刷するには、PRINT プロシジャまたは REPORT プロシジャを使用します。

VTYPE や VLENGTH などの関数を使用して、特定の記述子情報を表示することもできます。詳細については、“DS2 関数” (*SAS DS2 言語リファレンス*)を参照してください。

一般的な問題の原因を調べる

プログラムが正しく実行されない場合は、入力データに次のエラーがないか確認してください。

- 名前が同じで異なるデータを表す列

  DS2 は、指定された名前の列を 1 つだけ新しいテーブルに含めます。名前が同じでデータが異なる列を持つ 2 つのテーブルをマージする場合、最後に読み取られたテーブルの値が他のテーブルの値を上書きします。

  エラーを修正するには、SET または MERGE ステートメントで RENAME=テーブルオプションを使用して、テーブルを結合する前に列の名前を変更します。または、DATASETS プロシジャを使用できます。

- データは同じだが属性が異なる共通の列

  DS2 がこれらの違いを処理する方法は、どの属性が異なるかによって異なります。

  - 型属性

    型属性に互換性がない場合、DS2 はデータプログラムの処理を停止し、列に互換性がないことを示すエラーメッセージを発行します。

    このエラーを修正するには、別の DS2 データプログラムを使用して、必要に応じて型を変更する必要があります。使用する DS2 ステートメントは、列の性質によって異なります。次の表には、列を結合するときの結果属性が含まれています。

    表 17.1 型プロモーション

左側	右側	結果
BIGINT	BIGINT	BIGINT
BIGINT	INTEGER	BIGINT
BIGINT	SMALLINT	BIGINT
	TINYINT	BIGINT

左側	右側	結果
BIGINT		
INTEGER	BIGINT	BIGINT
INTEGER	INTEGER	INTEGER
INTEGER	SMALLINT	INTEGER
INTEGER	TINYINT	INTEGER
SMALLINT	BIGINT	BIGINT
SMALLINT	INTEGER	INTEGER
SMALLINT	SMALLINT	SMALLINT
SMALLINT	TINYINT	INTEGER
TINYINT	BIGINT	BIGINT
TINYINT	INTEGER	INTEGER
TINYINT	SMALLINT	INTEGER
TINYINT	TINYINT	TINYINT
DOUBLE	DOUBLE	DOUBLE
DOUBLE	FLOAT	DOUBLE
INTEGER	DECIMAL	DOUBLE
REAL	REAL	DOUBLE
BIGINT	DOUBLE	DOUBLE
DECIMAL (X,Y)	DECIMAL (X,Z)	DOUBLE
TIME[1]	TIME[1]	TIME[1]
TIMESTAMP	TIMESTAMP	TIMESTAMP
TIMESTAMP	DATE	ERROR
DATE	DATE	DATE
VARBINARY[1]	VARBINARY	VARBINARY
BINARY	BINARY	BINARY
BINARY	VARBINARY	VARBINARY
CHAR(N)	CHAR(N)	CHAR(N)
CHAR(N)[2]	CHAR(N)[2]	VARCHAR(N)[2]
VARCHAR(N)	VARCHAR(N)	VARCHAR(N)
VARCHAR(N)	VARCHAR(M)	VARCHAR(MAX (N, M))
CHAR(N)	CHAR(M)	VARCHAR (MAX (N, M))
VARCHAR(N)	CHAR(M)	VARCHAR (MAX (N, M))
NCHAR(M)	VARCHAR(M)	VARCHAR(M)

左側	右側	結果
NCHAR(M)	NCHAR(M)	NCHAR(M)
INTEGER	CHARACTER	ERROR
DATE	DOUBLE	ERROR

1 Hive のサポートなし
2 DB2、Impala、ODBC、Oracle の場合

注: ベストプラクティスは、スレッドプログラムですべての変数を宣言することです。これにより、データソース間の型の不一致を回避できます。

- 長さ属性

  長さ属性が異なる場合、DS2 は最大長の列を含むテーブルから長さを取得します。次の例では、MERGE ステートメントにリストされているすべてのテーブルに列 Mileage が含まれています。Quarter1 では、列 Mileage の長さは 4 バイトです。Quarter2 では 8 バイト、Quarter3 と Quarter4 では 6 バイトです。出力テーブル Yearly では、列 Mileage の長さは 8 バイトで、これは Quarter2 から派生した長さです。

```
data yearly;
  dcl char(4) quarter1 char(8) quarter2 char(6) quarter3 quarter4;
  method run();
    merge quarter1 quarter2 quarter3 quarter4;
    by Account;
  end;
enddata;
run;
```

- ラベル、出力形式、入力形式属性

  これらの属性のいずれかが異なる場合、DS2 は、その属性を持つ列を含む最初のテーブルから属性を取得します。ただし、明示的に指定したラベル、出力形式、または入力形式はデフォルトをオーバーライドします。すべてのテーブルに明示的に指定された属性が含まれている場合、最初のテーブルで指定された属性が他の属性をオーバーライドします。

  VLABEL などの関数を使用して、特定の記述子情報を表示することもできます。詳細については、“DS2 関数” (*SAS DS2 言語リファレンス*)を参照してください。

プログラムのテスト

テーブルを準備する最後のステップとして、プログラムをテストする必要があります。プログラムのすべてのロジックをテストする行のサンプルを含む小さな一時テーブルを作成します。ロジックに問題があり、予期しない出力が得られた場合は、プログラムをデバッグできます。

DS2 テーブルの結合: 方法

連結

定義

テーブルの連結とは、2 つ以上のテーブルを次々に組み合わせて 1 つのテーブルにすることです。新しいテーブルの行数は、元のテーブルの行数の合計です。行の順序はシーケンシャルです。最初のテーブルのすべての 行の後に、2 番目のテーブルのすべての行が続きます。

最も単純なケースでは、すべての入力テーブルに同じ列が含まれます。入力テーブルに異なる列が含まれている場合、あるテーブルの行には、他のテーブルでのみ定義されている列の欠損値があります。いずれの場合も、新しいテーブルの列は古いテーブルの列と同じです。

構文

テーブルを連結するには、次の形式の SET ステートメントを使用します。

SET *table(s)*;

引数

table(s)
有効なテーブル名を指定します。

詳細については、“SET ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

連結中の DS2 処理

コンパイルフェーズ
DS2 は、SET ステートメントで指定された各テーブルの記述子情報を読み取り、すべてのテーブルのすべての 列とデータプログラムによって 作成された列を含むプログラムデータベクトルを作成します。

実行 - ステップ 1
DS2 は、最初のテーブルの最初の行をプログラムデータベクトルに読み込みます。最初の行を処理し、データプログラムの他のステートメントを実行します。次に、プログラムデータベクトルの内容を新しいテーブルに書き込みます。

SET ステートメントは、データプログラム中に値が計算または割り当てられる列を除いて、プログラムデータベクトルの値を欠損値にリセットしません。データプログラムによって作成された列は、保持されない限り、データプログラムの各反復の開始時に欠損に設定されます。テーブルから読み取られる変数はそうではありません。

実行 - ステップ 2
データプログラムでは、DS2 は、ファイルの終わりインジケーターが見つかるまで、最初のテーブルから一度に 1 行ずつ読み取り続けます。次に、プログラムデータベクトルの列の値が欠損に設定され、データプログラムは 2 番目のテーブルから行の読み取りを開始し、すべてのテーブルからすべての 行を読み取るまで、これを繰り返していきます。スレッドプログラムの場合、特定のスレッドに入る行の順序は、BY ステートメントを使用しない限り未定義です。その場合でも、スレッドのスレッドプログラムでは、すべての行が他のテーブルからの行の前に受信されると想定する方法はありません。

例 1: データプログラムを使用した連結

この例では、各テーブルに列 Common と列 Number が含まれており、行は Common の値の順に並べられています。通常、同じ列を持つテーブルを連結します。この場合、各テーブルには、テーブルの結合による効果をより明確に示すための一意の列も含まれています。次のプログラムは、SET ステートメントを使用してテーブルを連結し、結果を出力します。

```
data animal(overwrite=yes);
  dcl varchar(10) common animal number;
  method init();
    common='a'; animal='Ant'; number='5'; output;
    common='b'; animal='Bird'; number=''; output;
    common='c'; animal='Cat'; number='17'; output;
    common='d'; animal='Dog'; number='9';  output;
    common='e'; animal='Eagle'; number=''; output;
    common='f'; animal='Frog';  number='76'; output;
  end;
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant number;
  method init();
    common='g'; plant='Grape'; number='69'; output;
    common='h'; plant='Hazelnut'; number='55'; output;
    common='i'; plant='Indigo'; number=''; output;
    common='j'; plant='Jicama';  number='14'; output;
    common='k'; plant='Kale'; number='5'; output;
    common='l'; plant='Lentil';  number='77'; output;
  end;
enddata;
run;


/*  set concatenates */
```

```
data concatenate (overwrite=yes);
  method run();
    set animal plant;
  end;
enddata;
run;

proc print data=concatenate;
run;
quit;
```

アウトプット 17.1　連結テーブル(SET ステートメント)

Animal

common	animal	number
a	Ant	5
b	Bird	
c	Cat	17
d	Dog	9
e	Eagle	
f	Frog	76

+

Plant

common	plant	number
g	Grape	69
h	Hazelnut	55
i	Indigo	
j	Jicama	14
k	Kale	5
l	Lentil	77

=

Concatenate

common	animal	number	plant
a	Ant	5	
b	Bird		
c	Cat	17	
d	Dog	9	
e	Eagle		
f	Frog	76	
g		69	Grape
h		55	Hazelnut
i			Indigo
j		14	Jicama
k		5	Kale
l		77	Lentil

結果のテーブル Concatenate には 12 行あります。これは、結合されたテーブルの行の合計です。プログラムデータベクトルには、すべてのテーブルのすべての 列が含まれます。あるテーブルでは見つかったが別のテーブルでは見つからなかった列の値は、欠損に設定されます。

例 2: SQL を使用した連結

SQL 言語を使用してテーブルを連結することもできます。この例では、SQL は両方のテーブルの各行を読み取り、Combined という名前の新しいテーブルを作成します。YEAR1 および YEAR2 入力テーブルを次に示します。

次の SQL コードは、テーブル Combined を作成して出力します。

```
proc sql;
  create table combined as
    select * from animal
        union all
      select * from plant;
      select * from combined;
quit;
```

```
proc print data=combined;
run;
```

出力は、SET ステートメントを使用した場合の出力とまったく同じです。

インターリーブ

定義

インターリーブでは、SET ステートメントと BY ステートメントを使用して、複数のテーブルを 1 つの新しいテーブルに結合します。これは、BY グループ処理とも呼ばれます。BY グループ処理は、1 つ以上の共通列の値によってグループ化または順序付けされた 1 つ以上のテーブルの行を結合する方法です。SET ステートメントの直後に BY ステートメントを指定すると、SET ステートメントは入力テーブルの行を並べ替え順にインターリーブします。並べ替え順序または並べ替えキーは、BY ステートメントの列名によって指定されます。新しいテーブルの行数は、元のテーブルの行数の合計です。

注: 文字データ型の列を持つテーブルがあると仮定します。DECLARE ステートメントを使用した入力時に列を数値データ型に変更した場合、結果の列の並べ替え順は数値ではありません。たとえば、次の文字列(CHAR)がテーブルに存在するとします: 9、10、500。SET および BY ステートメントを使用してテーブルを読み取るときに s を数値列(DOUBLE)として宣言すると、データは英数字順、つまり 10、500、9 で出力として生成されます。SET ステートメントは、文字列が数値データ型に変換される前に、行を英数字順に並べ替えます。

列を昇順ではなく降順で並べ替えるには、BY ステートメントの列名の前にキーワード DESCENDING を使用します。

構文

BY 変数を使用する場合、次の形式の SET ステートメントを使用してテーブルをインターリーブします。

SET *table(s)*;

BY [DESCENDING]*column* [...[DESCENDING] *column*];

引数

table
テーブル名を指定します。

DESCENDING
指定された列によってテーブルが降順で並べ替えられることを指定します。DESCENDING は、数値列の場合は最大から最小へ、文字列の場合はアルファベットの逆順になります。

column
テーブルを並べ替えるための各列を指定します。これらの列は、現在のデータプログラムの BY 変数と呼ばれます。

並べ替え要件

BY ステートメントを使用すると、内部的に DS2 は並べ替えられた順序で行を要求します。行がすでに並べ替えられている場合は、データの"再並べ替え"が必要になることがあります。

インターリーブ中の DS2 処理

コンパイルフェーズ

- DS2 は、SET ステートメントで指定された各テーブルの記述子情報を読み取り、すべてのテーブルのすべての 列とデータプログラムによって作成された列を含むプログラムデータベクトルを作成します。
- DS2 プログラムでは、SAS は BY 列ごとに FIRST.*variable* と LAST.*variable* の 2 つの一時変数を作成することにより、各 BY グループの開始と終了を識別します。これらの値は、行に次の特性があるかどうかを示します。
  - BY グループの最初
  - BY グループの最後
  - BY グループの最初でも最後でもない
  - BY グループに行が 1 つしかない場合のように、最初と最後の両方

  行が BY グループの最初の行である場合、値が変更された列、および BY ステートメント内の後続のすべての列に対して、SAS は FIRST.*variable* の値を 1 に設定します。BY グループの他のすべての行では、FIRST.variable の値は 0 です。同様に、その行が BY グループの最後の行である場合、次の行で値が変更される列、および BY ステートメント内の後続のすべての列に対して、SAS は LAST.variable の値を 1 に設定します。BY グループの他のすべての行では、LAST.variable の値は 0 です。テーブルの最後の行では、すべての LAST.variable 変数の値が 1 に設定されます。これらの一時変数は DS2 プログラミングで使用できますが、出力テーブルには追加されません。

  BY グループの最初の行を処理しているか、最後の行を処理しているかに基づいて、条件付きでアクションを実行できます。

  注: BY グループ処理を示す例については、*“テーブルのインターリーブ” (SAS DS2 言語リファレンス)*を参照してください。

注: SPD Engine データセットの場合、ユーティリティファイルは、余分なスペースを必要とする特定の操作に使用されます。BY ステートメントにはユーティリティファイルが必要で、SAS UTILLOC=システムオプションはそのユーティリティファイルにスペースを 割り当てます。詳細については、*SAS システムオプション: リファレンス*の SAS UTILLOC=システムオプションを参照してください。

実行 - ステップ 1
DS2 は、SET ステートメントで指定された各テーブルの最初の行を比較して、どの BY グループを新しいテーブルに最初に出現させるかを決定します。選択したテーブルの最初の BY グループからすべての 行を読み取ります。この BY グループが複数のテーブルに出現する場合、SET ステートメントに出現する順序でテーブルから読み取ります。プログラムデータベクトル内の列の値は、DS2 が新しいテーブルの読み取りを開始するたびに、また BY グループが変更されるたびに、欠損に設定されます。

実行 - ステップ 2
DS2 は、各テーブルの次の行を比較して次の BY グループを決定し、この BY グループの行を含む SET ステートメントで選択されたテーブルから行の読み取りを開始します。DS2 は、すべてのテーブルからすべての 行を読み取るまで続行します。

例 1: 最も単純なケースでのインターリーブ

この例では、各テーブルに BY 変数 Common が含まれており、行は BY 変数の値の順に並べられています。次の例では、Animal および Plant 入力テーブルを作成します。

```
data animal(overwrite=yes);
  dcl varchar(10) common animal number;
  method init();
    common='a'; animal='Ant'; number='5'; output;
    common='b'; animal='Bird'; number=''; output;
    common='c'; animal='Cat'; number='17'; output;
    common='d'; animal='Dog'; number='9';  output;
    common='e'; animal='Eagle'; number=''; output;
    common='f'; animal='Frog';  number='76'; output;
  end;
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant number;
  method init();
    common='a'; plant=''; number='69'; output;
    common='b'; plant='Bamboo'; number='55'; output;
    common='c'; plant='Cabbage'; number=''; output;
    common='d'; plant='Daffodil';  number='14'; output;
    common='e'; plant='Eucalyptus; number='5'; output;
    common='f'; plant='Fig';  number='77'; output;
  end;
```

```
enddata;
run;
```

次のプログラムは、SET ステートメントと BY ステートメントを使用してテーブルをインターリーブし、結果を出力します。

```
/*  set with by interleaves */
data interleave (overwrite=yes);
  method run();
    set animal plant; by common;
  end;
enddata;
run;

proc print data=interleave;
run;
```

アウトプット 17.2　インターリーブテーブル(SET ステートメント)

Animal

common	animal	number
a	Ant	5
b	Bird	
c	Cat	17
d	Dog	9
e	Eagle	
f	Frog	76

+

Plant

common	plant	number
a	Azalea	69
b	Bamboo	55
c	Cabbage	
d	Daffodil	14
e	Eucalyptus	5
f	Fig	77

=

Interleave

common	animal	number	plant
a	Ant	5	
a		69	Azalea
b	Bird		
b		55	Bamboo
c	Cat	17	
c			Cabbage
d	Dog	9	
d		14	Daffodil
e	Eagle		
e		5	Eucalyptus
f	Frog	76	
f		77	Fig

結果のテーブル Interleave は 12 行あります。これは、結合されたテーブルの行の合計です。新しいテーブルには、両方のテーブルのすべての 列が含まれます。一方のテーブルで見つかったが、もう一方のテーブルでは見つからなかった列の値は、欠損に設定され、行は BY 変数の値で並べられます。

例 2: BY 変数の重複値によるインターリーブ

テーブルに BY 変数の重複値が含まれている場合、行は元のテーブルに出現した順序で新しいテーブルに書き込まれます。この例には、BY 変数 Common の重複値が含まれています。次のプログラムでは、Animal および Plant 入力テーブルを作成します。

```
data animal(overwrite=yes);
  dcl varchar(10) common animal number;
  method init();
```

```
    common='a'; animal='Ant'; number='5'; output;
    common='a'; animal='Bird'; number=''; output;
    common='b'; animal='Cat'; number='17'; output;
    common='c'; animal='Dog'; number='9';  output;
    common='d'; animal='Eagle'; number=''; output;
    common='e'; animal='Frog';  number='76'; output;
  end;
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant number;
  method init();
    common='a'; plant='Grape'; number='69'; output;
    common='b'; plant='Hazelnut'; number='55'; output;
    common='c'; plant='Indigo'; number=''; output;
    common='c'; plant='Jicama';  number='14'; output;
    common='d'; plant='Kale'; number='5'; output;
    common='e'; plant='Lentil';  number='77'; output;
  end;
enddata;
run;
```

次のプログラムは、SET ステートメントと BY ステートメントを使用してテーブルをインターリーブし、結果を出力します。

```
/*  set with by interleaves */
data interleave (overwrite=yes);
  method run();
    set animal plant; by common;
  end;
enddata;
run;

proc print data=interleave;
run;
quit;
```

アウトプット 17.3　複数の BY 変数を含むインターリーブテーブル(SET ステートメント)

Obs	common	animal	number	plant
1	a	Bird		
2	a	Ant	5	
3	a		69	Grape
4	b	Cat	17	
5	b		55	Hazelnut
6	c	Dog	9	
7	c		14	Jicama
8	c			Indigo
9	d	Eagle		
10	d		5	Kale
11	e	Frog	76	
12	e		77	Lentil

新しいテーブルの行数は、すべてのテーブルの 行の合計です。行は、元のテーブルに出現した順序で新しいテーブルに書き込まれます。

例 3: 各テーブルで異なる BY 値を使用したインターリーブ

テーブル Animal と Plant の両方に、一方のテーブルには存在するが他方のテーブルには存在しない値が含まれています。次のプログラムでは、Animal および Plant 入力テーブルを作成します。

```
data animal(overwrite=yes);
  dcl varchar(10) common animal number;
  method init();
    common='a'; animal='Ant'; number='5'; output;
    common='c'; animal='Bird'; number=''; output;
    common='d'; animal='Cat'; number='17'; output;
    common='e'; animal='Dog'; number='9';  output;
  end;
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant number;
  method init();
    common='a'; plant='Grape'; number='69'; output;
    common='b'; plant='Hazelnut'; number='55'; output;
```

```
    common='c'; plant='Indigo'; number=''; output;
    common='d'; plant='Jicama';  number='14'; output;
    common='e'; plant='Kale'; number='5'; output;
    common='f'; plant='Lentil';  number='77'; output;
  end;
enddata;
run;
```

このプログラムは、SET ステートメントと BY ステートメントを使用してこれらのテーブルをインターリーブし、結果を出力します。

```
data interleave (overwrite=yes);
  method run();
    set animal plant; by common;
  end;
enddata;
run;

proc print data=interleave;
run;
quit;
```

アウトプット 17.4　異なる BY 変数を含むインターリーブテーブル(SET ステートメント)

Obs	common	animal	number	plant
1	a	Ant	5	
2	a		69	Grape
3	b		55	Hazelnut
4	c	Bird		
5	c			Indigo
6	d	Cat	17	
7	d		14	Jicama
8	e	Dog	9	
9	e		5	Kale
10	f		77	Lentil

結果のテーブルには、BY 変数の値で並べられた 10 行が含まれます。

コメントと比較

- 他の言語では、マージという用語はしばしばインターリーブを意味するために使用されます。DS2 では、マージという用語は、2 つ以上のテーブルの行を 1 つの行に結合する操作に予約されています。インターリーブされたテーブルの行は結合されません。これらは、BY 変数の値の順序で元のテーブルからコピーされます。

- 1 つのテーブルに同じ BY 値を持つ複数の行がある場合、DATA ステップはそれらの行の順序を結果に保持します。
- データプログラムを使用するには、入力テーブルを適切に並べ替える必要があります。SQL では、入力テーブルが順番に並んでいる必要はありません。

1 対 1 の読み取り

定義

1 対 1 の読み取りでは、2 つ以上の SET ステートメントを使用して各テーブルから行を個別に読み取ることにより、2 つ以上のテーブルの行を 1 つの行に結合します。このプロセスは、1 対 1 のマッチングとも呼ばれます。新しいテーブルには、すべての入力テーブルのすべての 列が含まれます。新しいテーブルの行数は、最小の元のテーブルの行数です。テーブルに共通の列が含まれている場合、最後のテーブルから読み込まれた値は、以前のテーブルから読み込まれた値を置き換えます。

構文

1 対 1 の読み取りには、次の形式の SET ステートメントを使用します。

SET *table-1*;

SET *table-2*;

引数

table-1
テーブル名を指定します。*table-1* は、データプログラムが読み取る最初のテーブルです。

table-2
テーブル名を指定します。*table-2* は、データプログラムが読み取る 2 番目のテーブルです。

注意
複数の SET ステートメントでテーブルを結合する場合は注意してください。 複数の SET ステートメントを使用して行を結合すると、望ましくない結果が生じる可能性があります。この方法を使用してテーブルを結合する前に、テーブルの代表的なサンプルでプログラムをテストしてください。

詳細については、“SET ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

1 対 1 の読み取り中の DS2 処理

コンパイルフェーズ
: DS2 は、SET ステートメントで指定された各テーブルの記述子情報を読み取り、すべてのテーブルのすべての 列とデータプログラムによって 作成された列を含むプログラムデータベクトルを作成します。

実行 - ステップ 1
: DS2 が最初の SET ステートメントを実行すると、DS2 は最初のテーブルの最初の行をプログラムデータベクトルに読み込みます。2 番目の SET ステートメントは、2 番目のテーブルの最初の行をプログラムデータベクトルに読み込みます。両方のテーブルに同じ列が含まれている場合、値が欠損していても、最初のテーブルの値が 2 番目のテーブルの値に置き換えられます。最後のテーブルから最初の行を読み取り、データプログラム内の他のステートメントを実行した後、DS2 はプログラムデータベクトルの内容を新しいテーブルに書き込みます。SET ステートメントは、データプログラム中に作成された列または値が割り当てられた列を除いて、プログラムデータベクトルの値を欠損値にリセットしません。

実行 - ステップ 2
: DS2 は、いずれかのテーブルでファイルの終わりインジケーターを検出するまで、一方のテーブルからの読み取りを続け、次に他方のテーブルからの読み取りを続けます。DS2 は、最も短いテーブルの最後の行で処理を停止し、長いテーブルから残りの行を読み取りません。

例 1: 1 対 1 の読み取り: 等しい行数の処理

テーブル Animal とテーブル Plant の両方に列 Common が含まれており、その列の値によって 並べられています。次のプログラムでは、Animal および Plant 入力テーブルを作成します。

```
data animal(overwrite=yes);
  dcl varchar(10) common animal number;
  method init();
    common='a'; animal='Ant';  output;
    common='b'; animal='Bird';  output;
    common='c'; animal='Cat';  output;
    common='d'; animal='Dog';   output;
    common='e'; animal='Eagle';  output;
    common='f'; animal='Frog';   output;
  end;
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant number;
  method init();
    common='a'; plant='Grape'; output;
```

```
    common='b'; plant='Hazelnut'; output;
    common='c'; plant='Indigo';  output;
    common='d'; plant='Jicama';  output;
    common='e'; plant='Kale';  output;
    common='g'; plant='Lentil';  output;
  end;
enddata;
run;
```

次のプログラムは、2 つの SET ステートメントを使用して Animal と Plant の行を結合し、結果を出力します。

```
data one2one (overwrite=yes);
  method run();
    set animal;
    set plant;
  end;
enddata;
run;

proc print data=one2one;
run;
quit;
```

アウトプット 17.5　等しい行数での 1 対 1 の読み取り(SET ステートメント)

Obs	common	animal	plant
1	a	Ant	Grape
2	b	Bird	Hazelnut
3	c	Cat	Indigo
4	d	Dog	Jicama
5	e	Eagle	Kale
6	g	Frog	Lentil

新しいテーブルの各行には、すべてのテーブルのすべての 列が含まれています。ただし、6 行目の Common 列の値には"g"が含まれていることに注意してください。Animal テーブルの 6 行目の Common の値は、Plant の値によって上書きされました。Plant は、DS2 が最後に読み取ったテーブルです。

行数が不均等な 1 対 1 の読み取り

テーブル Animal とテーブル Plant の両方に列 Common が含まれており、その列の値によって並べられています。テーブルの行数が異なります。次のプログラムでは、Animal および Plant 入力テーブルを作成します。

```
data animal(overwrite=yes);
```

```
  dcl varchar(10) common animal;
  method init();
    common='a'; animal='Ant'; output;
    common='a'; animal='Bird';  output;
    common='b'; animal='Cat';  output;
    common='c'; animal='Dog';  output;
    common='d'; animal='Eagle';  output;
    common='e'; animal='Frog';  output;
  end;
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant;
  method init();
    common='a'; plant='Grape';output;
    common='b'; plant='Hazelnut';  output;
    common='c'; plant='Indigo'; output;
    common='d'; plant='Jicama';  output;
  end;
enddata;
run;
```

次のプログラムは、2 つの SET ステートメントを使用して Animal と Plant の行を結合し、結果を出力します。

```
data one2onerowsnotequal (overwrite=yes);
  method run();
    set animal;
    set plant;
  end;
enddata;
run;
quit;

proc print data=one2onerowsnotequal;
run;
```

アウトプット 17.6　行数が不均等な 1 対 1 の読み取り(SET ステートメント)

Obs	common	animal	plant
1	a	Ant	Grape
2	b	Bird	Hazelnut
3	c	Cat	Indigo
4	d	Dog	Jicama

DS2 は最も短いテーブルの最後の行で処理を停止するため、結果テーブルには 4 つの行しか含まれません。

コメントと比較

- 複数の SET ステートメントを他の DS2 ステートメントとともに使用すると、次の応用が可能になります。
  - 1 つの行を多数の行とマージする
  - 行の条件付きマージ
  - 同じテーブルから 2 回読み取る

マッチマージ

定義

マッチマージは、共通の列の値に従って、2 つ以上のテーブルの行を新しいテーブルの 1 つの行に結合します。新しいテーブルの行数は、すべてのテーブルの 各 BY グループの最大行数の合計です。マッチマージを実行するには、必要な BY ステートメントとともに MERGE ステートメントを使用します。マッチマージを実行すると、すべてのテーブルが BY ステートメントで指定した列によって並べ替えられます。

構文

テーブルをマッチマージするには、次の形式の MERGE ステートメントを使用します。

MERGE *table(s)*;

BY *column(s)*;

引数

table
行が読み取られる既存のテーブルを少なくとも 2 つ指定します。

column
テーブルを並べ替えるための各列の名前を指定します。これらの列は BY 変数と呼ばれます。

詳細については、“MERGE ステートメント” (*SAS DS2 言語リファレンス*)および“BY ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

マッチマージ中の DS2 処理

コンパイルフェーズ
DS2 は、MERGE ステートメントで指定された各テーブルの記述子情報を読み取り、すべてのテーブルのすべての 行とデータプログラムによって 作成された行を含むプログラムデータベクトルを作成します。

実行 - ステップ 1
DS2 は、MERGE ステートメントで指定された各テーブルの最初の BY グループを調べて、どの BY グループを新しいテーブルに最初に出現させるかを決定します。データプログラムは、各テーブルからその BY グループの最初の行をプログラムデータベクトルに読み込み、MERGE ステートメントに記述された順序でテーブルを読み取ります。テーブルの BY グループに行がない場合、プログラムデータベクトルには、そのテーブルに固有の行の欠損値が含まれます。

実行 - ステップ 2
入力テーブルの各行は、出力テーブルで 1 回だけ使用されます。テーブルに固有の列は、BY グループの作成中にそのテーブルが使い果たされると、欠損値または null 値で埋められます。

実行 - ステップ 3
DS2 は、すべてのテーブルのすべての BY グループからすべての 行を読み取るまで、これらのステップを繰り返します。

注意
DECIMAL または NUMERIC データ型を持つ DS2 マージの BY 変数は、DOUBLE データ型に変換されます。 一致する DECIMAL 列が BY 変数でない場合、DECIMAL 列は DECIMAL データ型のままになります。

注意
テーブル間で DECIMAL または NUMERIC データ型の列間に型、スケール、または精度の不一致がある場合、その列は DOUBLE データ型に変換されます。

RETAIN を使用して DATA ステップマージを作成する

MERGE ステートメントで RETAIN 引数を使用して、DATA ステップマージと同様の多対多のマッチマージを生成できます。ただし、BY グループ内の行の順序はほとんどのデータソースで定義されていないため、RETAIN を使用した MERGE には多くの正解がある可能性があります。したがって、同じプログラムから異なる結果が得られる可能性があります。どちらも正しいものです。

RETAIN 引数を指定すると、特定の BY グループのデータセットの最後の行が、寄与しているデータセットのいずれにも行がなくなるまで繰り返し使用されます。

詳細については、“MERGE ステートメント” (*SAS DS2 言語リファレンス*)を参照してください。

例 1: 行の結合

テーブル Animal と Plant にはそれぞれ BY 変数 Common が含まれており、行は BY 変数の値の順に並べられています。次のプログラムでは、Animal および Plant 入力テーブルを作成します。

```
data animal(overwrite=yes);
  dcl varchar(10) common animal;
  method init();
    common='a'; animal='Ant';  output;
    common='b'; animal='Bird';  output;
    common='c'; animal='Cat';  output;
    common='d'; animal='Dog';   output;
    common='e'; animal='Eagle';  output;
    common='f'; animal='Frog';   output;
  end;
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant;
  method init();
    common='a'; plant='Grape'; output;
    common='b'; plant='Hazelnut'; output;
    common='c'; plant='Indigo';  output;
    common='d'; plant='Jicama';  output;
    common='e'; plant='Kale';  output;
    common='f'; plant='Lentil';  output;
  end;
enddata;
run;
```

次のプログラムは、BY 変数 Common の値に従ってテーブルをマージし、結果を出力します。

```
data mmerge (overwrite=yes);
  method run();
    merge animal plant;
    by common;
  end;
enddata;
run;
quit;

proc print data=mmerge;
run;
quit;
```

アウトプット 17.7　単純マッチマージ(MERGE ステートメント)

Obs	common	animal	plant
1	a	Ant	Grape
2	b	Bird	Hazelnut
3	c	Cat	Indigo
4	d	Dog	Jicama
5	e	Eagle	Kale
6	f	Frog	Lentil

新しいテーブルの各行には、すべてのテーブルのすべての 列が含まれています。

例 2: BY 変数の値が重複するマッチマージ

次の例では、テーブル Animal と Plant に BY 変数 Common の重複値が含まれています。次のプログラムでは、Animal および Plant 入力テーブルを作成します。

```
data animal(overwrite=yes);
  dcl varchar(10) common animal;
  method init();
    common='a'; animal='Ant'; output;
    common='a'; animal='Ape';  output;
    common='b'; animal='Bird';  output;
    common='c'; animal='Cat';  output;
    common='d'; animal='Dog';  output;
    common='e'; animal='Eagle';  output;
  end;
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant;
  method init();
    common='a'; plant='Apple';output;
    common='b'; plant='Banana';  output;
    common='c'; plant='Coconut'; output;
    common='c'; plant='Celery'; output;
    common='d'; plant='Dewberry';  output;
    common='e'; plant='Eggplant'; output;
  end;
enddata;
run;
```

次のプログラムは、マージされたテーブル MATCH1 を生成し、結果を出力します。

```
data mmdiffby (overwrite=yes);
  method run();
    merge animal plant;
    by common;
```

```
  end;
enddata;
run;
quit;

proc print data=mmdiffby;
run;
quit;
```

アウトプット 17.8　重複する BY 変数を使用したマッチマージ(MERGE ステートメント)

Obs	common	animal	plant
1	a	Ape	Apple
2	a	Ant	
3	b	Bird	Banana
4	c	Cat	Coconut
5	c		Celery
6	d	Dog	Dewberry
7	e	Eagle	Eggplant

出力の 2 行目では、列 Plant の値は保持されません。マッチマージでは、5 行目の Animal の値も重複しませんでした。

注: MERGE ステートメントは、多対多のマッチマージでデカルト積を生成しません。かわりに、少なくとも 1 つのテーブルに BY グループの行があるときに、1 対 1 マージを実行します。BY グループのすべての行が 1 つのテーブルから読み取られ、別のテーブルにまだ行がある場合、DS2 は列を欠損値または null 値で埋めます。

例 3: 不一致行でのマッチマージ

DS2 が入力テーブル内の不一致行でマッチマージを実行する場合、値が欠損していても、DS2 はプログラムデータベクトル内のすべての列の値を保持します。テーブル Animal および Plant には、BY 変数 Common のすべての値が含まれているわけではありません。次のプログラムでは、Animal および Plant 入力テーブルを作成します。

```
data animal(overwrite=yes);
  dcl varchar(10) common animal;
  method init();
    common='a'; animal='Ant'; output;
    common='c'; animal='Cat';  output;
    common='d'; animal='Dog';  output;
    common='e'; animal='Eagle';  output;
  end;
```

```
enddata;
run;


data plant(overwrite=yes);
  dcl varchar(10) common plant;
  method init();
    common='a'; plant='Apple';output;
    common='b'; plant='Banana';  output;
    common='c'; plant='Coconut'; output;
    common='e'; plant='Eggplant'; output;
    common='f'; plant='Fig';  output;end;
enddata;
run;
```

次のプログラムは、マージされたテーブル Mmnomrow を生成し、結果を出力します。

```
data mmnomrow (overwrite=yes);
  method run();
    merge animal plant;
    by common;
  end;
enddata;
run;
quit;

proc print data=mmnomrow;
run;
quit;
```

アウトプット 17.9　不一致行でのマッチマージ(MERGE ステートメント)

Obs	common	animal	plant
1	a	Ant	Apple
2	b		Banana
3	c	Cat	Coconut
4	d	Dog	
5	e	Eagle	Eggplant
6	f		Fig

出力が示すように、列 Common のすべての値が新しいテーブルに表示されます。これには、一方のテーブルにあって他方のテーブルにない列の欠損値が含まれます。

18

予約語

DS2 言語の予約語

次の単語は DS2 言語キーワードとして予約されており、変数名として使用したり、本来の用途と異なる方法で使用したりすることはできません。

注: 予約語は、二重引用符で囲むと変数名として使用できます。詳細と例については、“区切り識別子” (58 ページ)を参照してください。

表 18.1　DS2 予約語

特殊文字	A	B	C	D
___KPLIST	ABORT	BIGINT	CALL	DATA
ALL	AND	BINARY	CATALOG	DATE
HOSTNAME	AS	BY	CHAR	DCL
NEW	ASM		CHARACTER	DECIMAL
LOCALNTHREADS			COMMIT	DECLARE
LOCALTHREADID			CONTINUE	DELETE
NTHREADS				DESCENDING
NULL				DIM
_RC_				DO
ROWSET				DOUBLE
TEMPORARY				DROP
THREADID				DS2_OPTIONS

E	**F**	**G**	**H**	**I**
ELIF	FLOAT	GE	HAVING	IDENTITY
ELSE	FORMAT	GLOBAL		IF
END	FORTRAN	GOTO		IN
ENDDATA	FORWARD	GROUP		INDSNAME
ENDMODULE	FROM	GT		INDSNUM
ENDPACKAGE	FUNCTION			INFORMAT
ENDSTAGE				INLINE
ENDTABLE				INPUT
ENDTHREAD				INT
EQ				INTEGER
ERROR				IN_OUT
ESCAPE				

K	**L**	**M**	**N**	**O**
KEEP	LABEL	MERGE	NATIONAL	ODS
	LE	METHOD	NCHAR	OF
	LEAVE	MISSING	NE	OR
	LIKE	MODIFY	NG	ORDER
	LIST	MODULE	NL	OTHER
	LT		NOT	OTHERWISE
			NULL	OUTPUT
			NUMERIC	OVERWRITE
			NVARCHAR	

P	**R**	**S**	**T**	**U**
PACKAGE	REAL	SELECT	TABLE	UNTIL
PARTITION	REMOVE	SET	THEN	UPDATE
PRECISION	RENAME	SMALLINT	THIS	
PRIVATE	REPLACE	SQLSUB	THREAD	
PROGRAM	REQUIRE	STAGE	THREADS	
PUT	RETAIN	STOP	TIME	
	RETURN	STORED	TIMESTAMP	
	RETURNS	SUBSTR	TINYINT	
	ROLLBACK	SYSTEM	TO	
			TRANSACTION	
			T_UDF	
			TSPL_OPTIONS	

V	**W**			
VARARRAY	WHEN			
VARBINARY	WHERE			
VARCHAR	WHILE			
VARLIST				
VARYING				

3 部

DS2 と CAS

19

CAS の DS2: 概念

CAS での DS2 の実行方法

CAS の DS2

DS2 プログラムは、CAS でも SAS クライアントでも実行できます。CAS は、分散並列実行を可能にするクラウドベースのランタイム環境です。CAS で実行される DS2 プログラムは、通常、複数のマシンに分散された複数のスレッドを使用して、インメモリ CAS テーブルで動作します。DS2 プログラムを CAS で実行する方法は、SAS クライアントで実行する方法とほとんど同じです。CAS の DS2 は、同じ言語要素のほとんどをサポートし、SAS クライアントの DS2 とほとんど同じ機能を提供します

CAS で DS2 プログラムを実行する理由

単一のマシンでのビッグデータ処理は低速です。DS2 プログラムを CAS で実行すると、複数のマシンでプログラムを実行し、そのマシンのスレッド間で処理ワークロードを分割することにより、並列実行によってビッグデータの処理が高速化されます。

処理モード

- シングルスレッド

  プログラムは、単一のマシン上の単一のスレッドで実行されます。単一のスレッドがすべてのデータを処理します。非常に大きなデータセットの場合、シングルスレッドでの実行はお勧めしません。

- マルチスレッド

  プログラムは、複数のマシンに分散された複数のスレッドで並列して実行されます。各スレッドは、データの一部を処理します。

DS2 スレッドプログラムで SET FROM ステートメントを使用する DS2 プログラムのみがマルチスレッドで実行されます。他のすべての DS2 プログラムは、単一のマシン上の単一のスレッドで実行されます。単一のスレッドがすべてのデータを順次処理します。

DS2 スレッドプログラムで SET FROM ステートメントを使用する DS2 プログラムは、部分的または完全にマルチスレッドで実行されます。DS2 スレッドプログラムは常にマルチスレッドで実行されますが、DS2 データプログラムは、それに含まれるステートメントが SET FROM ステートメントと、オプションとして OUTPUT ステートメントだけである場合にのみマルチスレッドで実行されます。DS2 データプログラムに他のステートメントが含まれている場合、DS2 データプログラムはシングルスレッドで実行されます。DS2 スレッドプログラムがマルチスレッドで実行され、DS2 データプログラムがシングルスレッドで実行される DS2 プログラムは、部分的にマルチスレッドです。

注: SET FROM ステートメントで明示的なスレッド数が指定されている場合、DS2 プログラムは、各ワーカーで指定された数のスレッドで実行されます。たとえば、THREADS=4 で CAS セッションに 3 つのワーカーが含まれている場合、3 つの CAS ワーカーはそれぞれ 4 つのスレッドを使用し、合計 12 のスレッドが DS2 プログラムを実行します。SET FROM ステートメントでスレッド数が明示的に指定されていない場合、各 CAS ワーカーのスレッド数はデフォルトでライセンスされたスレッド制限になります。

マルチスレッド処理

データプログラムが DS2 スレッドから設定され、データプログラムに SET FROM ステートメントと、オプションとして OUTPUT ステートメントのみが含まれている場合、DS2 プログラムはマルチスレッドで実行されます。DS2 プログラムがマルチスレッドで実行される場合、オブザベーションは複数のスレッドで処理され、すべての CAS ワーカーに並列に分散されます。マルチスレッド実行中、各 CAS ワーカーは入力テーブルのサブセットを処理し、出力テーブルのサブセットを生成します。CAS データ管理は、各 CAS ワーカーをサポートし、入力テーブルのサブセットを並列に読み取り、出力テーブルのサブセットを並列に書き込みます。

DS2 プログラムをマルチスレッドで実行する方法を次に示します。

- 各 CAS ワーカーの各スレッドで、DS2 スレッドプログラムは次のことを行います。
  - 入力テーブルからデータのサブセットを読み取ります。
  - データのサブセットを処理します。
  - データのサブセットを DS2 データプログラムに送信します。
- 各 CAS ワーカーの各スレッドで、DS2 データプログラムは次のことを行います。
  - DS2 スレッドプログラムからデータのサブセットを受け取ります。
  - データのサブセットを出力テーブルに書き込みます。

注: DS2 データプログラムは、データを出力テーブルに書き込む以外に、データの追加処理を実行しません。

マルチスレッドの例

次の例では、DS2 プログラムは **cars** をフィルタリングして、**msrp** が$100,000 より大きい車だけを表示します。プログラムは複数のスレッドで実行され、各スレッドが cars テーブルからレコードのサブセットを読み取ります。また、各スレッドは、レコードのサブセットを **cars_luxury** テーブルに書き込みます。

```
cas casauto; 1
proc casutil sessref=casauto; 2
  load data=sashelp.cars;
run;

proc ds2 sessref=casauto; 3
thread cars_thd / overwrite=yes; 4
  method run();
    set cars;
    if (msrp > 100000) then do;
      output;
```

```
  end;
end;
endthread;

data cars_luxury / overwrite=yes; 5
  dcl thread cars_thd t;
  method run();
    set from t;
  end;
enddata;
run;
quit;
```

1 CAS サーバーで **casauto** という名前のセッションを開始します。

2 **casauto** CAS セッションで、sashelp.cars を SAS クライアントから **cars** インメモリテーブルにロードします。

3 **casauto** CAS セッションで DS2 プログラムを実行します。CAS で DS2 プログラムを実行するには、SESSREF=オプションが必要です。

4 各 CAS ワーカーの各スレッドで、スレッドプログラムは **cars** インメモリテーブルから行のサブセットを読み取ります。次に、スレッドプログラムは、**msrp** が $100,000 より大きい行のサブセットをデータプログラムに送信します。

5 各 CAS ワーカーの各スレッドで、データプログラムはスレッドプログラムから、**msrp** が$100,000 より大きい行のサブセットを受け取ります。次に、データプログラムは、**msrp** が$100,000 より大きい行のサブセットを、**cars_luxury** インメモリテーブルに書き込みます。

DS2 プログラムの実行が終了すると、**cars_luxury** テーブルには、 **msrp** が$100,000 より大きい完全な車のセットが含まれます。

シングルスレッド処理

DS2 スレッドプログラムで SET FROM ステートメントを使用しない DS2 プログラムは、シングルスレッドで実行されます。DS2 プログラムがシングルスレッドで実行される場合、プログラムは単一の CAS ワーカーの 1 つのスレッドで実行されます。単一のスレッドは入力テーブルを処理し、出力テーブルを生成します。すべてのデータオブザベーションは順次処理されます。

単一の CAS ワーカーの単一のスレッドで、DS2 プログラムは次のことを行います。

- 入力テーブルからすべてのデータを 読み取ります。
- すべてのデータを処理します。
- すべてのデータを 出力テーブルに書き込みます。

注: CAS でプログラムをシングルスレッドで実行すると、CAS セッションで使用可能なワーカーの数に関係なく、処理は単一のワーカーにローカライズされます。

シングルスレッドの例

次の例では、DS2 プログラムは、**sashelp.cars** テーブル内のすべての車の平均 **msrp** を計算します。プログラムは、データプログラムで SET FROM ステートメントを使用しないため、シングルスレッドで実行されます。CAS セッション内のワーカーの数に関係なく、すべてのデータ処理は単一の CAS ワーカー内の単一のスレッドにローカライズされます。

```
cas casauto; 1
proc casutil sessref=casauto; 2
  load data=sashelp.cars;
run;

proc ds2 sessref=casauto; 3
data _null_; 4
  dcl double sum_msrp n;
  retain sum_msrp n;
  method init();
    sum_msrp = 0;
    n = 0;
  end;
  method run();
    set cars;
    sum_msrp = sum_msrp + msrp;
    n = n + 1;
  end;
  method term();
    dcl double mean_msrp having format dollar8.;
    mean_msrp = sum_msrp / n;
    put 'Mean MSRP: ' mean_msrp;
end;
enddata;
run;
quit;
```

1 CAS サーバーで **casauto** という名前のセッションを開始します。

2 **casauto** CAS セッションで、**sashelp.cars** を SAS クライアントから cars インメモリテーブルにロードします。

3 **casauto** CAS セッションで DS2 プログラムを実行します。CAS で DS2 プログラムを実行するには、SESSREF=オプションが必要です。

4 単一の CAS ワーカーの単一のスレッドで、データプログラムは **cars** インメモリテーブルからすべての 行を読み取り、すべての行の **msrp** 値を合計し、**msrp** 値の平均を計算して、平均を出力します。

SAS ログには、次のような出力が含まれます。

```
Mean MSRP:   $32,775
NOTE: Running DATA program on one node
```

部分的なマルチスレッド処理

DS2 スレッドプログラムで SET FROM ステートメントを使用する DS2 プログラムは、DS2 スレッドプログラムがマルチスレッドで実行され、DS2 データプログラムがシングルスレッドで実行される場合、部分的にマルチスレッドで実行されます。SET FROM または OUTPUT 以外のステートメントを含むデータプログラムは、シングルスレッドで実行されます。部分的なマルチスレッド実行には、次の 2 つのステージがあります。

パラレルステージ
: すべての CAS ワーカーに分散された複数のスレッドで DS2 スレッドプログラムを実行します。

シリアルステージ
: 単一の CAS ワーカー内の単一のスレッドで DS2 データプログラムを実行します。

パラレルステージでは、各 CAS ワーカーが入力テーブルのサブセットを処理し、中間テーブルのサブセットを生成します。シリアルステージでは、単一の CAS ワーカーが完全な中間テーブルを処理し、完全な出力テーブルを生成します。

DS2 プログラムを部分的にマルチスレッドで実行する方法を次に示します。

- パラレルステージ

  各 CAS ワーカーの各スレッドで、DS2 スレッドプログラムは次のことを行います。

  - 入力テーブルからデータのサブセットを読み取ります。
  - データのサブセットを処理します。
  - データのサブセットを一時テーブルに書き込みます。

- シリアルステージ

  単一の CAS ワーカーの単一のスレッドで、DS2 データプログラムは次のことを行います。

  - 一時テーブルからすべてのデータを 読み取ります。
  - すべてのデータを処理します。
  - すべてのデータを出力テーブルに書き込みます。

部分的にマルチスレッドで実行する理由

並列分散実行は、複数のマシンでプログラムを実行し、そのマシンのスレッド間で処理ワークロードを分割することにより、ビッグデータの処理を高速化します。マルチスレッド実行では、データ処理の各分割が独立している必要があります。データを集計または結合するデータ処理アルゴリズムは、完全なマルチスレッド実行のためのデータ処理の独立性要件に違反しています。データ処理アルゴリズムをデー

タ処理の分割適用結合モデルにリファクタリングできる場合、処理を部分的に並列化できます。分割適用結合戦略では、アルゴリズムはデータを分割し、他の分割とは関係なく各データ分割にデータ処理を適用してから、分割を結合します。MapReduce は、分割適用結合戦略の一般的な特殊化です。

部分的にマルチスレッドで実行される DS2 プログラムは、スレッドプログラムがデータ処理をデータ分割に並列に適用することにより、分割適用結合戦略を実装します。データプログラムは、データ処理の結果をシリアルに結合します。最高のパフォーマンスを得るには、できるだけ多くのデータ処理をスレッドプログラムで実行する必要があります。同期またはシリアル化を必要とするデータ処理のみをデータプログラムで実行する必要があります。

注: スレッドプログラムでのデータ処理のみが、CAS セッションで使用可能なワーカー全体に分散されます。データプログラムでのデータ処理は、CAS セッションで使用可能なワーカーの数に関係なく、常に単一のワーカーにローカライズされます。

部分的なマルチスレッドの例

次の例では、DS2 プログラムは、**sashelp.cars** テーブル内のすべての車の平均 **msrp** を計算します。DS2 スレッドとデータプログラムからのセットは SET FROM ステートメントと OUTPUT ステートメントに限定されないため、プログラムは部分的にマルチスレッドで実行されます。

```
cas casauto; 1

proc casutil sessref=casauto; 2
  load data=sashelp.cars;
run;

proc ds2 sessref=casauto; 3
thread cars_thd / overwrite=yes; 4
  dcl double partial_sum_msrp partial_n;
  retain partial_sum_msrp partial_n;
  keep partial_sum_msrp partial_n;
  method init();
    partial_sum_msrp = 0;
    partial_n = 0;
  end;
  method run();
    set cars;
    partial_sum_msrp = partial_sum_msrp + msrp;
    partial_n = partial_n + 1;
  end;
  method term();
    output;
  end;
endthread;

data _null_; 5
  dcl thread cars_thd t;
  dcl double sum_msrp n;
```

```
  retain sum_msrp n;
  method init();
    sum_msrp = 0;
    n = 0;
  end;
  method run();
    set from t;
   sum_msrp = sum_msrp + partial_sum_msrp;
   n = n + partial_n;
 end;
 method term();
   dcl double mean_msrp having format dollar8.;
   mean_msrp = sum_msrp / n;
   put 'Mean MSRP: ' mean_msrp;
  end;
enddata;
run;
quit;
```

1. CAS サーバーで **casauto** という名前のセッションを開始します。
2. **casauto** CAS セッションで、**sashelp.cars** を SAS クライアントから cars インメモリテーブルにロードします。
3. **casauto** CAS セッションで DS2 プログラムを実行します。CAS で DS2 プログラムを実行するには、SESSREF=オプションが必要です。
4. 各 CAS ワーカーの各スレッドで、スレッドプログラムは **cars** インメモリテーブルから行のサブセットを読み取ります。スレッドプログラムは、行のサブセットの **msrp** を合計し、**msrp** の部分合計を計算します。次に、スレッドプログラムは、 **msrp** の部分合計を一時インメモリテーブルに書き込みます。
5. 単一の CAS ワーカーの単一のスレッドで、データプログラムは一時インメモリテーブルから **msrp** のすべての部分合計を読み取り、部分合計を合計してすべての **msrp** 値の合計を求め、**msrp** 値の平均を計算して、平均を出力します。

SAS ログには、次のような出力が含まれます。

```
NOTE: Running THREAD program on all nodes
Mean MSRP:   $32,775
NOTE: Running DATA program on one node
```

この部分的なマルチスレッドの例は、シングルスレッドの例のリファクターであることに注意してください。シングルスレッドの実行を必要とするシリアルアルゴリズムを分割適用結合戦略に変換できる場合、アルゴリズムをリファクタリングすると、多くの場合、CAS などの分散並列環境で処理されるビッグデータのランタイムパフォーマンスが大幅に向上します。

CAS の DS2 スレッドプログラム

CAS の分析の基本的な計算モデルは、分散コンピューティングとマルチスレッドコンピューティングを組み合わせたものです。データを渡す CAS アクションは通常、複数のスレッドによって行われます。各スレッドはデータの一部を受け取り、入力テーブルの各レコードは 1 つのスレッドによってのみ消費されます。これらのスレ

ッドは、実行の同時性を強調するために *ワーカースレッド*と呼ばれます。使用できる同時ワーカースレッドの最大数は、ソフトウェアライセンスによって決まります。

許可されている最大数よりも多いスレッドを DS2 プログラムで指定している場合、DS2 はスレッド数をその最大数に減らします。DS2 プログラムで 0 スレッドが指定されているか、スレッド数が指定されていない場合、DS2 は、ソフトウェアライセンスで決定されている、許可されている最大スレッド数を使用します。

Liref と Caslib

DS2 プログラムが CAS で実行される場合、DS2 プログラムは、SAS クライアントで作成された libref にアクセスできません。DS2 プログラムで使用できるのは、CAS セッションで作成された caslib だけです。

CAS で実行される DS2 プログラムでは、次の形式で caslib の CAS テーブルを指定します。

caslib.table-name

caslib にかっこまたはその他の特殊文字が含まれている場合は、caslib を二重引用符で囲む必要があります。次に例を示します。

```
set "caslib(janedoe)".mytable;
```

コメント

/* message */コンストラクトを使用して DS2 プログラムにコメントを追加し、プログラムの目的を文書化したり、プログラムの異常な部分を説明したり、複雑なプログラムや計算のステップを説明したりできます。

> 重要　ステートメント内または DS2 プログラムで 1 つの空白が表示される場所ならどこにでもコメントを書くことができますが、/* message */コンストラクトを使用するコメントはネストできません。

CAS でサポートされていない DS2 言語要素

CAS でサポートされていない DS2 言語要素がいくつかあります。これらの言語要素は、"この *language-element* は CAS サーバーではサポートされていません。"という制限付きで *SAS DS2 言語リファレンス*で文書化されています。

CAS での BY グループ処理

SAS クライアントでは、DS2 BY グループ処理によって入力テーブルの行がグループ化され、BY ステートメントの 1 つ以上の列の値によって行が順序付けされます。

SAS クライアントでは、事前に並べ替えられたデータが順序付けられて DS2 に到着し、BY グループが BY グループブロック全体としてスレッドに分散されます。CAS では、データはパーティション上で本質的に順序付けられていません。BY ステートメントがない場合、各スレッドはノードに対してローカルなデータを受け取ります。スレッドプログラムまたはデータプログラムのいずれかで BY ステートメントが使用されている場合、各スレッドが 1 つ以上の BY グループ全体を参照できるように、テーブル行がワーカー間で送信されます。グリッドを介したデータ転送はリソースを大量に消費する可能性があるため、データプログラムで BY 処理を使用して結果を取得する前に、スレッドプログラムでハッシュテーブルを使用して各パーティションに存在する生データを集計するのが最善かどうかを検討する必要があります。入力行に比べて個別のデータグループが比較的少ない場合は、データプログラムで BY ステートメントを使用してデータを再分散する前に、スレッドプログラムでデータをそれらの少数の個別のグループにまとめる方が望ましい場合があります。

CAS サーバーでの整数型変換

DS2 は、SAS プラットフォームで実行する場合、BIGINT、INTEGER、SMALLINT、TINYINT 整数データ型をサポートします。CAS サーバーテーブルは、BIGINT (INT64)および INTEGER (INT32)の整数データ型のみをサポートします。

CAS runDS2 アクションは、SMALLINT または TINYINT データ型の変数を宣言する DS2 プログラムを CAS で実行できます。プログラムが変数を CAS テーブルに書き込む場合、書き込まれた SMALLINT または TINYINT 値は、CAS テーブルの INTEGER (INT32)データ型に変換されます。

-2147483647 の値は、CAS テーブルの INTEGER (INT32)列の SAS 欠損値を表します。-9223372036854775808 の値は、CAS テーブルの BIGINT (INT64)列の SAS 欠損値を表します。-2147483647 は CAS テーブルの BIGINT (INT64)列の有効な値であるため、-2147483647 はどこでも SAS 欠損値を表すわけではないことに注意してください。

DS2 では、-2147483647 は BIGINT 変数に対して有効ですが、INTEGER、SMALLINT、および TINYINT 変数の範囲外です。DS2 では、-9223372036854775808 は、BIGINT、INTEGER、SMALLINT、TINYINT 変数の範囲外です。範囲外の値が整数変数に割り当てられているが、変数の型でサポートされている場合、SAS 欠損値または ANSI null が変数に割り当てられます。したがって、-2147483647 または-9876543210 が INTEGER、SMALLINT、または TINYINT

変数に割り当てられている場合、その変数には SAS 欠損値または ANSI null 値が割り当てられます。

CAS テーブルの INTEGER (INT32)列が DS2 プログラムに読み込まれ、オブザベーションの INTEGER (INT32)列の値が-2147483647 である場合、その値は、入力時に SAS 欠損値の DS2 の内部表現に変換されます。CAS テーブルの BIGINT (INT64)列が DS2 プログラムに読み込まれ、オブザベーションの BIGINT (INT64)列の値が-2147483647 である場合、値は変換されずにそのまま読み取られます。

CAS サーバーでの DS2 ログ

DS2 と CAS は、SAS ログ機能をサポートしています。SAS Viya プラットフォームでのログの詳細については、*SAS Viya プラットフォーム: 管理*の"ログ"を参照してください。

CAS サーバーでの DS2 セッションエンコーディング

CAS サーバーでコンパイルおよび実行される DS2 プログラムの DS2 セッションエンコーディングは、常に UTF-8 です。

20

CAS での DS2 プログラムの実行

CAS で DS2 プログラムを実行する方法

CAS で DS2 プログラムを実行するには、次の 2 つの方法があります。

- PROC DS2

```
cas casauto;
caslib _all_ assign;

proc cas;
  session casauto;
/* This  sets the active caslib to casdata.*/
table.addCaslib /
    caslib="casdata"
    dataSource={srcType="path"}
    path="path-to-your-data";

 /* Load source data (cars) into a table.*/
  table.loadTable /
     caslib="casdata"
     path="cars.sashdat"
     casOut={name="cars", replace=true};
run;
quit;

proc ds2 sessref=casauto;
thread cars_thd / overwrite=yes;
  method run();
     set casdata.cars;
     if (msrp > 100000) then do;
        put make= model= msrp=;
        output;
     end;
```

```
  end;
endthread;

data cars_luxury / overwrite=yes;
  dcl thread cars_thd t;
  method run();
    set from t threads=4;
  end;
enddata;
run;
quit;
```

このプログラムの説明については、“DS2 プログラムの手順説明” (311 ページ)を参照してください。

ROC DS2 の詳細については、“DS2 プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。

- DS2 runDS2 アクション

```
cas casauto;
caslib _all_ assign;

proc cas;
 session casauto;
/* This  sets the active caslib to casdata.*/
  table.addCaslib /
    caslib="casdata"
    dataSource={srcType="path"}
    path="path-to-your-data";

/* Load source data (cars) into a table.*/
  table.loadTable /
    caslib="casdata"
    path="cars.sashdat"
    casOut={name="cars", replace=true};
run;

/*DS2 program to search for cars over $100K */
  ds2.runDS2 program="thread cars_thd / overwrite=yes;
  method run();
    set casdata.cars;
    if (msrp > 100000) then do;
      put make= model= msrp=;
      output;
    end;
  end;
endthread;

data cars_luxury / overwrite=yes;
  dcl thread cars_thd t;
  method run();
    set from t threads=4;
  end;
enddata;";
run;
quit;
```

詳細については、“DS2 Action Set” (*SAS Viya Platform: System Programming Guide*)を参照してください。

> ヒント　Lua または Python を使用していない場合は、PROC DS2 を使用して DS2 プログラムを実行することをお勧めします。PROC DS2 を使用する利点があります。詳細については、“DS2 Action Set Details” (*SAS Viya Platform: System Programming Guide*)を参照してください。

DS2 プログラムの手順説明

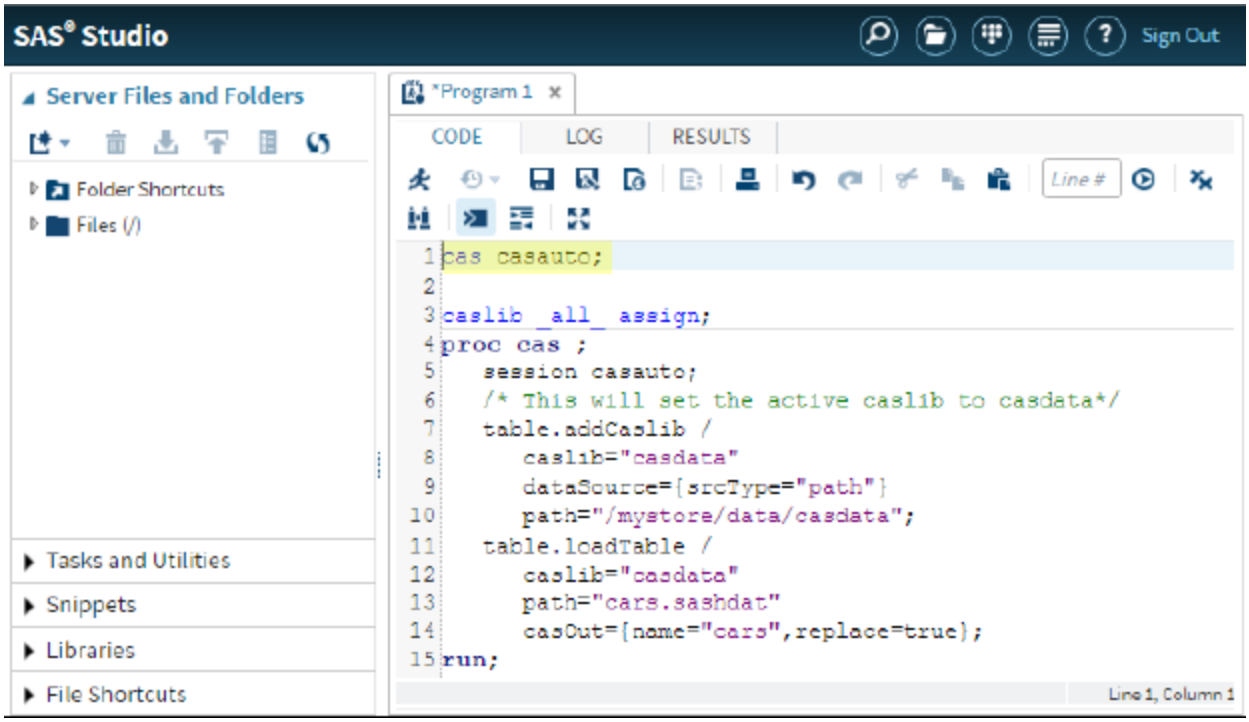
CAS で DS2 プログラムを実行するには、次の操作を行います。

1 CAS セッションを開始します。

CAS にアクセスするには、CAS セッションを開始してアクティブにします。CAS ステートメントで CAS セッションの名前を指定します。

```
cas casauto;
```

SAS Studio コードタブのビューは次のとおりです。



SAS ログには、次のような注が含まれます。

```
NOTE: The session CASAUTO connected successfully to Cloud Analytic Services cloud.example.com using
port 5570. The UUID is
    session-UUID. The user is casdemo and the active caslib is CASUSER(casdemo).
NOTE: The SAS option SESSREF was updated with the value CASAUTO.
NOTE: The SAS macro _SESSREF_ was updated with the value CASAUTO.
NOTE: The session is using nnn workers.
```

2 caslib を関連付けます。

CASLIB ステートメントを使用して、デフォルトの CASUSER caslib を、作成するすべての SAS libref に関連付けることができます。

```
caslib _all_ assign;
```

SAS ログに表示される注の部分的なリストを次に示します。

```
NOTE: CASLIB CASUSER(casdemo) for session CASAUTO will be mapped to SAS Library CASUSER.
```

3 caslib を作成し、データをロードします。

ファイルシステム上のファイルへのアクセスを提供する caslib を作成するには、table.addCaslib アクションを、**path** に設定された dataSource オプションとともに使用します。この例では、CASDATA caslib は、CASAUTO セッションと/mystore/data/casdata にあるソーステーブルとの間のインターフェイスです。

```
proc cas ;
  session casauto;
  /* This will set the active caslib to casdata*/
  table.addCaslib /
    caslib="casdata"
    dataSource={srcType="path"}
    path="/mystore/data/casdata";
```

詳細については、*"caslib" (SAS Cloud Analytic Servcies: 基本)*を参照してください。

ソースデータ(cars)を CAS のインメモリテーブルにロードするには、table.loadTable アクションを使用できます。

```
table.loadTable /
  caslib="casdata"
  path="cars.sashdat"
  casOut={name="cars",replace=true};
run;
```

CASDATA caslib を説明する結果は、SAS Studio の**結果**タブに表示されます。

Results from table.addCaslib

CAS Library Information

Library	Type	Path	Sub-directories included	Session local	Active
casdata	PATH	/mystore/data/casdata/	No	Yes	Yes

4 CAS テーブルを表示します。

table.fetch アクションを使用して、インメモリテーブル(cars)からデータの行を表示できます。

```
table.fetch / table="cars" to=10;
run;
```

インメモリーブルから最初の 10 行をフェッチした結果を次に示します。

Selected Rows from Table CARS

Index	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
1	BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3	6	225	16	23	4023	110	180
2	Honda	S2000 convertible 2dr	Sports	Asia	Rear	$33,260	$29,965	2.2	4	240	20	25	2835	95	162
3	Nissan	Altima S 4dr	Sedan	Asia	Front	$19,240	$18,030	2.5	4	175	21	26	3039	110	192
4	Audi	A6 4.2 Quattro 4dr	Sedan	Europe	All	$49,690	$44,936	4.2	8	300	17	24	4024	109	193
5	Honda	Civic LX 4dr	Sedan	Asia	Front	$15,890	$14,531	1.7	4	115	32	38	2513	103	175
6	Mitsubishi	Eclipse GTS 2dr	Sports	Asia	Front	$25,092	$23,456	3	6	210	21	28	3241	101	177
7	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
8	GMC	Safari SLE	Sedan	USA	Rear	$25,640	$23,215	4.3	6	190	16	20	4339	111	190
9	Mercury	Marauder 4dr	Sedan	USA	Rear	$34,495	$31,558	4.6	8	302	17	23	4195	115	212
10	Volvo	S80 2.5T 4dr	Sedan	Europe	All	$37,895	$35,688	2.5	5	194	20	27	3691	110	190

5 DS2 プログラムを実行します。

PROC DS2 を sessref=オプションとともに使用すると、インメモリ CAS テーブルを使用してプログラムを CAS で実行できます。

```
proc ds2 sessref=casauto;          1

thread cars_thd / overwrite=yes;   2
  method run();
    set casdata.cars;              3
    if (msrp > 100000) then do;
      put make= model= msrp=;
      output;
    end;
  end;
endthread;

data cars_luxury / overwrite=yes;  4
  dcl thread cars_thd t;
   method run();
    set from t threads=4;
   end;
enddata;

run;
quit;
```

1 PROC DS2 ステートメントに sessref=オプションを追加して、プログラムを CAS で実行します。

2 プログラムのパラレルステージでの操作は、複数のデータ行に並列に適用されます。

3 インメモリテーブルから行を読み取ります。

4 スレッドインスタンスが作成され、実行されます。

SAS ログに表示される結果は次のとおりです。

```
 NOTE: Running THREAD program on all nodes
Make=Porsche       Model= 911 GT2 2dr                      MSRP=$192,465
Make=Mercedes-Benz Model= CL600 2dr                        MSRP=$128,420
Make=Mercedes-Benz Model= SL55 AMG 2dr                     MSRP=$121,770
Make=Mercedes-Benz Model= SL600 convertible 2dr            MSRP=$126,670
NOTE: Created thread cars_thd in data set "casdata".cars_thd.
NOTE: Running THREAD program on all nodes
NOTE: Running DATA program on all nodes
NOTE: Execution succeeded. 4 rows affected.
104       quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time          4.89 seconds
      cpu time           0.02 seconds
```

21

モデルのパブリッシュとスコアリング

モデルのパブリッシュとスコアリングの概要

SCOREACCEL プロシジャと CAS アクションは、モデルのパブリッシュとスコアリングのために、CAS サーバーと外部データソースへのインターフェイスを提供します。詳細については、“In-Database Model Scoring” (*SAS In-Database Products: User's Guide*)を参照してください。

モデルのパブリッシュとスコアリングのための PROC SCOREACCEL および CAS アクション

構文と例については、次のドキュメントを参照してください。

- “SCOREACCEL Procedure” (*SAS In-Database Products: User's Guide*)
- “Run model program” (*SAS Viya Platform: System Programming Guide*)

- “モデルのパブリッシュとスコアリングアクションセット” (*SAS Visual Analytics: プログラミングガイド*)
- “SAS Embedded Process for Spark アクションセット” (*SAS Visual Analytics: プログラミングガイド*)

4 部

付録

付録 1

式オペランドの DS2 型変換

式オペランドの DS2 自動型変換

次の表に、式オペランドの自動型変換を示します。最初の列は"変換元"の型です。最初の行は"変換先"の型です。

表 A1.1　割り当てステートメントの DS2 自動型変換

変換元/変換先	TinyInt	SmallInt	Integer	BigInt	Decimal/Numeric	Real	Double	Date	Time	Timestamp	Char	Varchar	NChar	NVarchar	Binary	Varbinary
TinyInt	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
SmallInt	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
Integer	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
BigInt	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
Decimal/Numeric	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
Real	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
Double	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
Date	N	N	N	N	N	N	N	Y	N	N	Y	Y	Y	Y	N	N
Time	N	N	N	N	N	N	N	N	Y	N	Y	Y	Y	Y	N	N
Timestamp	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	N	N
Char	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
Varchar	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
NChar	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
NVarchar	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	N	N
Binary	N	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y

変換元/変換先	TinyInt	SmallInt	Integer	BigInt	Decimal / Numeric	Real	Double	Date	Time	Timestamp	Char	Varchar	NChar	NVarchar	Binary	Varbinary
Varbinary	N	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y

付録 2

DS2 ロガー

DS2 ロガーの概要

SAS ログ機能は、SAS サーバーおよび SAS プログラミング環境でログメッセージを分類およびフィルタリングし、ログメッセージをさまざまな出力デバイスに書き込むフレームワークです。サーバー環境では、管理メッセージの Admin、アプリケーションメッセージの App、パフォーマンスメッセージの Perf など、定義済みのメッセージカテゴリに基づいてログ機能がメッセージをログに記録します。カテゴリのメッセージは、ファイル、コンソール、およびその他のシステム宛先に同時に書き込むことができます。ログ機能により、TRACE、DEBUG、INFO、WARN、ERROR、FATAL のしきい値に基づいてメッセージをフィルタリングすることもできます。

DS2 には、構成情報とランタイム情報の両方を報告するためのロガーがいくつか用意されています。さらに、DS2 は HTTP パッケージと JSON パッケージ用のロガーを提供します。一般に、INFO は最小限の情報を提供し、DEBUG はフィールドの問題をデバッグするために必要なほとんど(おそらくすべて)の情報を提供します。TRACE レベルは、関心のあるあらゆるものを提供します。

ロガーと SAS ログ機能の詳細については、*SAS Viya ログ機能: 構成とリファレンス*を参照してください。SAS Viya プラットフォームで使用する DS2 ロガーを定義する方法については、“SAS Viya の SAS ログ機能への DS2 ロガーの追加” (333 ページ)を参照してください。

注: SAS Viya プラットフォームの SAS ログ機能は、32000 バイトを超えるログメッセージを 32000 バイトに切り捨てます。

構成ロガー

構成ロガーは、DS2 コンパイラの起動時に情報を追跡し、ユーザーのコードを実際に実行するためのコンテキストを提供します。

次の構成ロガーを使用できます。

App.TableServices.DS2.Config.Options
: DS2 コンパイラに提供されるオプションを示します。

App.TableServices.DS2.Config.Source
: DS2 コンパイラによって処理された DS2 ソースコードを示します。

App.TableServices.DS2.Config.Version
: DS2 コンパイラによってロードされたすべてのスレッドカーネル 拡張のバージョン情報を示します。

DS2 の問題を診断する場合、App.TableServices.DS2.Config.*ロガーは祖先から構成を継承しないことを理解することが重要です。App.TableServices.DS2.Config.*ロガーのデフォルトレベルは OFF です。App.TableServices.DS2.Config.*ロガーイベントをサーバーのログにキャプチャする場合は、該当する各 App.TableServices.DS2.Config.*ロガーを明示的に設定する必要があります。追加のログトラフィックがパフォーマンスに影響するため、DS2 の問題を診断する場合にのみ構成することをお勧めします。

ランタイムロガー

ランタイムロガーは実際の実行を追跡します。追跡される情報の一部は、処理される行ごとに生成されます。したがって、大きな入力テーブルは非常に大量のデータを生成する可能性があります。

次のランタイムロガーを使用できます。

App.TableServices.DS2.Runtime.Calls
: 実行中のすべてのメソッド呼び出しのトレースを示します。

App.TableServices.DS2.Runtime.SQL
DS2 コンパイラによって準備された SQL ステートメント、DS2 コンパイラによって実行された SQL ステートメント、またはその両方のすべての SQL ステートメントを示します。

App.TableServices.DS2.Runtime.Timing
コードのコンパイルと実行に費やされた時間を示します。要求された情報のレベル(INFO、DEBUG、TRACE)に応じて、DS2 実行中のさまざまな時点でタイミング情報が提供されます。例としては、解析時間、コンパイル時間、さまざまな監査および変換パス時間、INIT、RUN、および TERM メソッドの実行時間があります。

App.TableServices.DS2.Runtime.Put
すべての PUT ステートメント出力を記録します。

App.TableServices.DS2.Runtime.Log
SAS セッションで SAS ログに送信されるすべてのメッセージを 記録します。SAS Federation Server では、これらのメッセージもデフォルトで診断レコードとして ODBC ステートメントハンドルに追加されます。

App.TableServices.DS2.Runtime.TraceVariables
DEBUG または TRACE に設定すると、DS2 プログラムの実行中に変数値の変更が記録されます。トレース情報は、次の DS2 ステートメントおよび関数によって変更される変数についてのみ報告されます。

- 割り当てステートメント(=)
- Array 割り当てステートメント(:=)
- DO ステートメントによるインデックス変数割り当て
- SET または MERGE ステートメント
- 擬似 SUBSTR 関数(割り当ての左辺に用いた場合)
- SUM ステートメント(variable + expression; variable – expression;)

次の方法で変更された変数については、変数のトレース情報はレポートされません。

- ビルトイン DS2 パッケージによって変更される変数データ。たとえば、HASH.FIND メソッドがデータ変数を変更する場合、その変数の変更はトレースで報告されません。
- DS2 自動変数の暗黙の更新。たとえば、N 自動変数の暗黙的な増分はトレースで報告されません。
- 変数の作成中に、変数を欠損または null に暗黙的に初期化。
- RUN メソッドの各反復前に、保持されていない変数を欠損または null に暗黙的に再初期化。

注: App.TableServices.DS2.Runtime.TraceVariables ロガーは、TRACEVARIABLES オプションとともに使用されます。TRACEVARIABLES オプションは、“DS2_OPTIONS ステートメント” (*SAS DS2 言語リファレンス*)で設定されます。TRACEVARIABLES オプションが有効でない場合、変数のトレース情報は報告されません。TRACEVARIABLES オプションの設定方法の例については、“例: トレース変数のログ” (329 ページ)を参照してください。

DS2 の問題を診断する場合、App.TableServices.DS2.Runtime.*ロガーは祖先から構成を継承しないことを理解することが重要です。
App.TableServices.DS2.Runtime.*ロガーのデフォルトレベルは OFF です。
App.TableServices.DS2.Runtime.*ロガーイベントをサーバーのログにキャプチャする場合は、該当する各 App.TableServices.DS2.Runtime.*ロガーを明示的に設定する必要があります。追加のログトラフィックがパフォーマンスに影響するため、DS2 の問題を診断する場合にのみロガーを構成することをお勧めします。

HTTP パッケージロガー

HTTP クライアントは、SAS ログ機能による HTTP トラフィックのログをサポートしています。

ロガーの名前は App.TableServices.d2pkg.HTTP です。ロガーは、次のログ記録レベルをサポートしています。

INFO
: Web サーバーへの接続と切断、要求情報、ステータス情報などの一般的なトラフィック情報を表示します。

DEBUG
: すべての要求と応答のヘッダーと、本文データの最初の 64 バイトを示します。これにより、サーバーに送信されるすべてのデータを表示しなくても、クライアントとサーバーが何を行っているかを確認できます。

TRACE
: Web サーバーとの間で送受信されるすべてのデータを表示します。

JSON パッケージロガー

JSON パッケージは、SAS ログ機能によるログをサポートしています。

ロガーの名前は App.tk.DS2PKG.JSON です。ロガーは、DEBUG および TRACE ログ記録レベルをサポートします。たとえば、システムの起動時にロガーレベルを TRACE に設定する場合は、次の行をログ構成ファイルに追加します。

```
<logger name="App.tk.DS2PKG.JSON> <level value="trace"/> </logger>
```

そのログ構成ファイルを使用する他のユーザーのログ記録レベル設定に影響を与えないようにするには、SAS ログ機能を使用して自分のセッションを変更します。

```
%log4sas();
%log4sas_logger(App.tk.D2PKG.JSON, 'level=trace');
```

JSON パッケージは、DEBUG または TRACE 情報をほとんど提供しません。GETNEXTTOKEN メソッドは、いくつかの TRACE 情報を提供します。次のログの抜粋は一例です。

```
INFO App.Program - 1   %log4sas();
INFO App.Program - 2   %log4sas_logger(App.tk.D2PKG.JSON, 'level=trace');
INFO App.Program - 3   proc ds2;
INFO App.Program - 4   ds2_options sas;
INFO App.Program - 5   data _null_;
INFO App.Program - 6      dcl package json j();
INFO App.Program - 7      dcl int i rc ttype parseFlags;
INFO App.Program - 8      dcl varchar(256) x;
INFO App.Program - 9      dcl varchar(256) token;
INFO App.Program - 10     method init();
INFO App.Program - 11        x = '[{"labl 1":[{"labl 2":" a string"}]},
                      {"labl 3":[["", 3.14, null, true, false]]}]';
INFO App.Program - 12        put x=;
INFO App.Program - 13        rc = j.createParser( x );
INFO App.Program - 14        i = 0;
INFO App.Program - 15        do while (rc < 101 );
INFO App.Program - 16          i = i + 1;
INFO App.Program - 17          token = '';
INFO App.Program - 2    The SAS System    16:33 Friday, October 21, 2016
INFO App.Program -
INFO App.Program - 18          j.getNextToken( rc, token, ttype, parseFlags );
INFO App.Program - 19        end;
INFO App.Program - 20      end;
INFO App.Program - 21      method term();
INFO App.Program - 22        rc = j.destroyParser();
INFO App.Program - 23      end;
INFO App.Program - 24   enddata;
INFO App.Program - 25   run;
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 1, type: 16,
                  flags: 0x0, token: [
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 2, type: 64,
                  flags: 0x0, token: {
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 10, type: 256,
                  flags: 0x1, token: labl 1
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 12, type: 16,
                  flags: 0x0, token: [
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 13, type: 64,
                  flags: 0x0, token: {
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 21, type: 256,
                  flags: 0x1, token: labl 2
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 33, type: 256,
                  flags: 0x0, token:  a string
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 34, type: 128,
                  flags: 0x0, token: }
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 35, type: 32,
                  flags: 0x0, token: ]
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 36, type: 128,
                  flags: 0x0, token: }
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 38, type: 64,
                  flags: 0x0, token: {
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 46, type: 256,
                  flags: 0x1, token: labl 3
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 48, type: 16,
                  flags: 0x0, token: [
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 49, type: 16,
                  flags: 0x0, token: [
```

```
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 51, type: 256,
                flags: 0x0, token:
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 57, type: 512,
                flags: 0x4, token: 3.14
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 63, type: 1024,
                flags: 0x0, token: null
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 69, type: 4,
                flags: 0x0, token: true
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 76, type: 8,
                flags: 0x0, token: false
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 77, type: 32,
                flags: 0x0, token: ]
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 78, type: 32,
                flags: 0x0, token: ]
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 79, type: 128,
                flags: 0x0, token: }
TRACE App.tk.D2PKG.JSON - d2jsnGetNextToken: line: 1, column: 80, type: 32,
                flags: 0x0, token: ]
INFO App.Program - x=[{"labl 1":[{"labl 2":" a string"}]},{"labl 3":[["", 3.14,
                null, true, false]]}]
INFO App.Program - NOTE: Execution succeeded. No rows affected.
INFO App.Program - 26        quit;
```

例: すべての SQL 操作のログ

次の例では、すべての SQL コマンドをログに記録するログ構成ファイルを作成します。構成ファイルの作成後、LOGCONFIGLOC=システムオプションを設定して、SAS ログ機能の初期化に使用される構成ファイルの名前を指定します。

次のコードは、ログ構成ファイルを作成します。

```
<?xml version="1.0"?>
  <logging:configuration xmlns:logging="http://www.sas.com/xml/logging/1.0/">
  <logger name="App.TableServices.DS2.Runtime.SQL">
     <level value="trace"/>
  </logger>
  <logger name="App.TableServices.DS2" additivity="false">
     <appender-ref ref="DetailedOutput"/>
  </logger>
  <root>
     <appender-ref ref="RootLogger"/>
     <level value="info"/>
  </root>
  <appender name="DetailedOutput" class="FileAppender">
     <param name="append" value="false"/>
     <param name="FileNamePattern" value="sql.%S{App.Log}"/>
     <layout>
        <param name="ConversionPattern" value="%-5p:%sn:[%c{3}]:%m"/>
     </layout>
  </appender>
  <appender name="RootLogger" class="FileAppender">
```

```
    <param name="Append" value="false"/>
    <param name="ImmediateFlush" value="true"/>
    <param name="FileNamePattern" value="%S{App.Log}"/>
    <layout>
      <param name="ConversionPattern" value="%m"/>
    </layout>
  </appender>
  </logging:configuration>
```

LOGCONFIGLOC=システムオプションは、構成ファイルを **config.14s** として参照するように設定されています。

```
sas -log test.log -logconfigloc config.14s
```

このシステムオプションの詳細については、*SAS Viya ログ機能: 構成とリファレンス* を参照してください。

これにより、次のような一連のメッセージを含むファイル **sql.test.log** が生成されます。

```
DEBUG:00000084:[DS2.Runtime.SQL]:Found 0 NOCHANGE columns
  INFO :00000085:[DS2.Runtime.SQL]:0x0afce4b0:exec-direct:CREATE TABLE WORK.outp
    ("i" DOUBLE, "j" DOUBLE )
  DEBUG:00000086:[DS2.Runtime.SQL]:0x0afce4b0:exec-direct:passed:rc=0x00000000
  DEBUG:00000087:[DS2.Runtime.SQL]:Found 0 NOCHANGE columns
  INFO :00000088:[DS2.Runtime.SQL]:0x0afce4b0:exec-direct:SELECT * FROM WORK.outp
  DEBUG:00000089:[DS2.Runtime.SQL]:0x0afce4b0:exec-direct:passed:rc=0x00000000
  INFO :00000095:[DS2.Runtime.SQL]:stmt=0x0afae060:prepare:SELECT * FROM WORK.outp
  DEBUG:00000096:[DS2.Runtime.SQL]:stmt=0x0afae060:prepare:passed:rc=0x00000000
  INFO :00000097:[DS2.Runtime.SQL]:stmt=0x0afae060:execute
  DEBUG:00000098:[DS2.Runtime.SQL]:stmt=0x0afae060:execute:passed:rc=0x00000000
```

出力ファイルの正確な内容は、DetailedOutput アペンダー内のレイアウトの ConversionPattern パラメーターによって定義されます。DetailedOutput アペンダーは、App.TableServices.DS2 ロガーの定義に関連付けられています。これにより、App.TableServices.DS2 をルートとする階層内のすべてのロガーが同じファイルにログ記録されます。の additivity="false"修飾子は、ログメッセージが上に移動するのを防ぎます。

例: トレース変数のログ

TraceVariables ロガーの呼び出し

トレース変数のログを有効にするには、次の操作を行います。

1 DEBUG または TRACE ログ記録レベルのいずれかを指定して、App.TableServices.DS2.Runtime.TraceVariables ロガーを構成します。

2 DS2 プログラムで TRACEVARIABLES オプションを指定します。
TRACEVARIABLES オプションは、“DS2_OPTIONS ステートメント” (*SAS DS2 言語リファレンス*)で設定されます。

注: 両方のステップを完了する必要があります。これらのタスクのいずれかが実行されていない場合、変数のトレース情報は SAS ログ機能に報告されません。

ロガーを構成するには、ログ構成ファイルに次の行を追加します。この例では、DEBUG のログ記録レベルを設定します。

```
<logger name="App.TableServices.DS2.Runtime.TraceVariables"><level value="debug"/></logger>
```

TRACEVARIABLES オプションの設定方法の例を次に示します。変数をトレースするプログラムブロックの直前に DS2_OPTIONS ステートメントを指定します。DS2_OPTIONS ステートメントは、調べるプログラムブロックごとに設定する必要があります。

```
proc ds2;
ds2_options tracevariables;
package swapper / overwrite=yes;
  method swap(in_out double x, in_out double y);
    dcl double t;
    t = x;
    x = y;
    y = t;
  end;
endpackage;

ds2_options tracevariables;
data _null_;
  method init();
    dcl package swapper s();
    dcl double a[2];
    a := (10, 20);
    put 'before swap:' a[*]=;
    s.swap(a[1], a[2]);
    put 'after swap:' a[*]=;
  end;
enddata;
run;
quit;
```

ロガーメッセージについて

前述のプログラムの SAS ログは次のとおりです。

```
1         proc ds2;
2         ds2_options tracevariables;
3         package swapper / overwrite=yes;
4           method swap(in_out double x, in_out double y);
5             dcl double t;
6             t = x;
7             x = y;
8             y = t;
9           end;
10        endpackage;
11
12        ds2_options tracevariables;
13        data _null_;
14          method init();
15            dcl package swapper s();
16            dcl double a[2];
17            a := (10, 20);
18            put 'before swap:' a[*]=;
19            s.swap(a[1], a[2]);
20            put 'after swap:' a[*]=;
21          end;
22        enddata;
23        run;
before swap: a[1]=10 a[2]=20
after swap: a[1]=20 a[2]=10
NOTE: Created package swapper in data set work.swapper.
NOTE: Execution succeeded.No rows affected.
24        quit;
```

App.TableServices.DS2.Runtime.TraceVariables ロガーによって SAS ログ機能に報告されるトレースメッセージを次に示します。

```
2021-01-12T13:33:35,063 DEBUG [00000007] - Line 17 (data block line 5): a[1]=10
2021-01-12T13:33:35,063 DEBUG [00000007] - Line 17 (data block line 5): a[2]=20
2021-01-12T13:33:35,063 DEBUG [00000007] - Line 6 (package swapper line 4): t=10
2021-01-12T13:33:35,064 DEBUG [00000007] - Line 7 (package swapper line 5): x=20
2021-01-12T13:33:35,064 DEBUG [00000007] - Line 8 (package swapper line 6): y=10
```

トレースメッセージの形式は次のとおりです。

Line n1 (program_block_type program_block_name line n2): variable_name=variable_value

n1
: SAS ログでアクティビティが 発生したステートメントの行番号。

program_block_type
: ステートメントを含むプログラムブロックのタイプ("package"、"thread"、または"data")。

program_block_name
: ステートメントを含むプログラムブロックの名前。パッケージおよびスレッドプログラムブロックの場合、パッケージまたはスレッドのユーザー定義名が名前になります。データプログラムブロックの場合、"block"が名前です。

n2
: プログラムブロックの行番号。プログラムブロックの行番号は、プログラムブロックを開始する PACKAGE、THREAD、または DATA ステートメントで 1 から始まり、プログラムブロックを終了する ENDPACKAGE、ENDTHREAD、または ENDDATA ステートメントに到達するまで順番に続きます。n2 は、パッケージ、スレッド、またはデータブロックの先頭を基準とした、アクティビティが 発生したステートメントの行番号です。

SAS ログの行番号は、現在の SAS セッションで他に何が実行されたかによって、実行ごとに変わる可能性があるため、この番号が提供されます。プログラムブロックの先頭からのステートメントの行番号は変更されません。プログラムブロックの行番号がわかると、ストアド DS2 パッケージまたはストアド DS2 スレッドが SAS ログのソースリストにない場合に役立ちます。

この例の詳細は次のとおりです。

- ロガー出力の最初の 2 つのメッセージは、SAS ログの 17 行目のデータプログラムブロックからの Array 割り当てステートメントを参照しています。

  ```
  2021-01-12T13:33:35,063 DEBUG [00000007] - Line 17 (data block line 5): a[1]=10
  2021-01-12T13:33:35,063 DEBUG [00000007] - Line 17 (data block line 5): a[2]=20
  ```

  17 行目のステートメント a := (10, 20)は、値 10 と 20 を含む定数リストを配列 a に割り当てます。最初のメッセージ a[1]=10 は、配列 a の最初の要素への値 10 の割り当てを記録します。2 番目のメッセージ a[2]=20 は、配列 a の 2 番目の要素への値 20 の割り当てを記録します。各メッセージの 5 行目への参照は、Array 割り当てステートメントの場所を DATA ステートメントからのオフセットとして再度示しています。ステートメント a := (10, 20)は、DATA ステートメントの 4 行下にあります。

- 残りのメッセージは、swapper パッケージで発生した変換の結果として生じる変数の割り当てについて説明しています。

  ```
  2021-01-12T13:33:35,063 DEBUG [00000007] - Line 6 (package swapper line 4): t=10
  2021-01-12T13:33:35,064 DEBUG [00000007] - Line 7 (package swapper line 5): x=20
  2021-01-12T13:33:35,064 DEBUG [00000007] - Line 8 (package swapper line 6): y=10
  ```

  6、7、および 8 行目には、変数 t、x、および y に値を割り当てる変数割り当てステートメントが含まれています。

  割り当てステートメント t=x では、t に x の値が割り当てられます。パラメーター x には、19 行目の s.swap メソッド呼び出しの第 1 引数 a[1]によって値 10 が渡されました。したがって、値 10 が t に割り当てられます。つまり、t=10 です。

  割り当てステートメント x=y では、x に y の値が割り当てられます。パラメーター y には、第 2 引数 a[2]によって値 20 が渡されました。したがって、値 20 が x に割り当てられます。つまり、x=20 です。

  割り当てステートメント y=t では、y に t の値が割り当てられます。変数 t には、以前の割り当てステートメント t=x によって割り当てられた値 10 がまだ含まれています。したがって、y=10 です。

注: 変数トレースは、特に DS2 プログラムが大きい場合、または DS2 プログラムによって入力されるデータセットが大きい場合に、大量の SAS ログメッセージを生成する可能性があります。したがって、追加のログトラフィックがパフォーマンスに悪影響を与えるため、DS2 の問題を診断する場合にのみログを構成することをお勧めします。

SAS Viya の SAS ログ機能への DS2 ロガーの追加

SAS Viya プラットフォームでは、プログラミングロガーは SAS Environment Manager のログ機能に定義されます。SAS Environment Manager を使用するには、管理アクセスが必要です。

コンテナーで実行されているアプリケーションは、ログをモニタリングツールから利用できるようにするために出力を stdout (コンソール)に送信する必要があります。DS2 ロガーの出力をコンソールにルーティングするには、sas.compute.server logconfig 構成スペースで DS2 ロガーを計算サーバーに対して宣言する必要があります。詳細については、SAS Compute Server と計算サービスを参照してください。sas.compute.server: logconfig インスタンスを更新する手順を参照してください。

付録 3

DS2 エラーの解決

DS2 エラーの種類

DS2 は、要求を完了できない場合、SAS ログにエラーを書き込みます。ほとんどのエラーは、次のように分類できます。

- 使用エラー

  不正または無効な要求が原因で、使用エラーが発生します。DS2 使用エラーは通常、DS2 プログラムの構文、意味、または論理エラーによって発生します。コンパイル時に構文エラーと意味エラーが検出されると、DS2 プログラムのコンパイルが妨げられます。正常にコンパイルされた DS2 プログラムには、実行時に表面化する論理エラーが含まれている可能性があります。

- リソースエラー

  リソースサービス要求を実行できない場合、リソースエラーが発生します。さまざまな条件がこれらの失敗につながります。たとえば、リソースにアクセスできない、リソースのしきい値にすでに達している、リソースにアクセスする権限がないなどの可能性があります。

DS2 エラーを解決する方法

エラーを解決するには、次の操作を行います。

1 SAS ログでエラーメッセージを確認します。これらのメッセージは、問題の原因を理解したり、エラーの解決に役立つアイデアを提供したりするのに役立つ場合があります。

問題の内容を示す例を次に示します。この例では、使用エラーが発生しました。このエラーメッセージは、DS2 プログラムの 7 行目またはその近くに構文エラーがあることを示しています。

```
ERROR: Parse encountered END when expecting ';'.
ERROR: Line 7: Parse failed:  >>> end <<<;
```

2 SAS ログでは、まず一番最初のエラーメッセージに注目します。場合によっては、1 つの問題が失敗の連鎖につながり、最初のエラーを解決すると、後続のエラーも解決することがあります。

3 エラーメッセージがリソースエラーを示している場合は、システム管理者に問い合わせて、システムリソースを監視してください。システムリソースの例には、メモリ使用量とディスク容量が含まれます。システムリソースが少ない場合は、リソースを増やす方法を探すか、アプリケーションがそれらをより効率的に使用できる方法を検討してください。

リソースエラーを示す例を次に示します。このメッセージは、メモリしきい値に達したことを示しています。

```
ERROR: Memory allocation failed.
```

4 前の 3 つのステップを使用してもエラーを解決できない場合は、SAS テクニカルサポートにお問い合わせください。提供する必要がある情報のリストは次のとおりです: SAS テクニカルサポートに問い合わせる際に覚えておくべき 4 つのヒント。

SAS ログに内部エラーコードを含むエラーメッセージが含まれている場合は、SAS テクニカルサポート担当者にコードを提供してください。内部エラーコードは、SAS が問題の原因をより迅速に特定するのに役立つ 16 進数です。

内部エラーコード(ハイライト表示)を含むエラーメッセージの例を次に示します。

```
ERROR: DS2 internal error code 0xECCC8C8BF7C049FD. Refer to DS2 documentation
    for resolving errors.
```

DS2 ロガー

DS2 ロガーは、エラーの解決にも役立ちます。ロガーは、実行中の DS2 プログラムから情報を取得する別のメカニズムを提供します。次に例を示します。

- App.TableServices.DS2.Config.Options ロガーは、多くの構成設定を報告します。そのうちの 1 つは DS2 セッションエンコーディングです。DS2 プログラムの実行に使用されているセッションエンコーディングが不明な場合は、このロガーを有効にして DS2 セッションエンコーディングを報告できます。
- HTTP パッケージロガーは、DS2 HTTP パッケージで生成された HTTP 要求の問題をデバッグするのに役立つ情報を提供します。

- DS2 プログラムの実行中に変数値の変更をトレースする TraceVariables ロガーは、問題の修復に役立ち、DS2 コードの監査を提供できます。

これらおよびその他の DS2 ロガーの詳細については、付録 2, “DS2 ロガー” (323 ページ)を参照してください。