SASハッシュオブジェクトを利用して 医薬品開発に使用するプログラムを効率化する ー有害事象と併用薬、臨床検査値と途中変更のある施設基準値 のマッチングからSASプログラムコードの分析まで

森岡　裕

神田　悟志
ナイフィックス株式会社　バイオメトリクスグループ

Using SAS Hash Objects to Improve Program from Matching-Merge Clinical data to Analyzing your SAS Code

Yutaka Morioka

Satoshi Kanda
Biometorics Group, Niphix K K

要旨:

SAS9.1よりでハッシュオブジェクト(メモリに展開されるデータ構造体)機能がBase SASで利用可能となった。
ビックデータと相性がよく、うまく利用することで、処理時間を劇的に短縮したり、柔軟なコードが実現可能である

医薬品開発における、よくある処理を例にとって、ハッシュオブジェクトの基本から応用までを説明したい。

はじめに

今回、発表させていただくSASハッシュオブジェクトについての説明には、発表者の独自理解・独自解釈が含まれます。

一部SAS社の公式見解とは異なる理解や説明が含まれる可能性がございますので、正しく学習・理解するためには、次にあげるドキュメント等を参照してください。

本発表では、SASハッシュオブジェクトって、こんなもので、こういうことができるのか、面白いなぁぐらいに思っていただければ発表者としては大成功です。

ハッシュオブジェクトの学習の仕方について

SAS社の日本語オンラインヘルプ

http://www.sas.com/offices/asiapacific/japan/service/onlinehelp.html#section=2

SAS社Webページから

「サポート」タブ

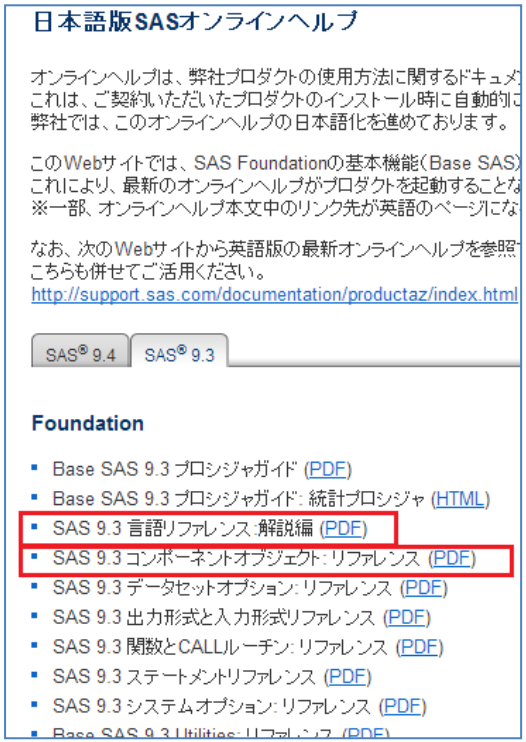
⇒「日本語版SASオンラインヘルプ」

ハッシュオブジェクトの学習の仕方について

SAS社の日本語オンラインヘルプ

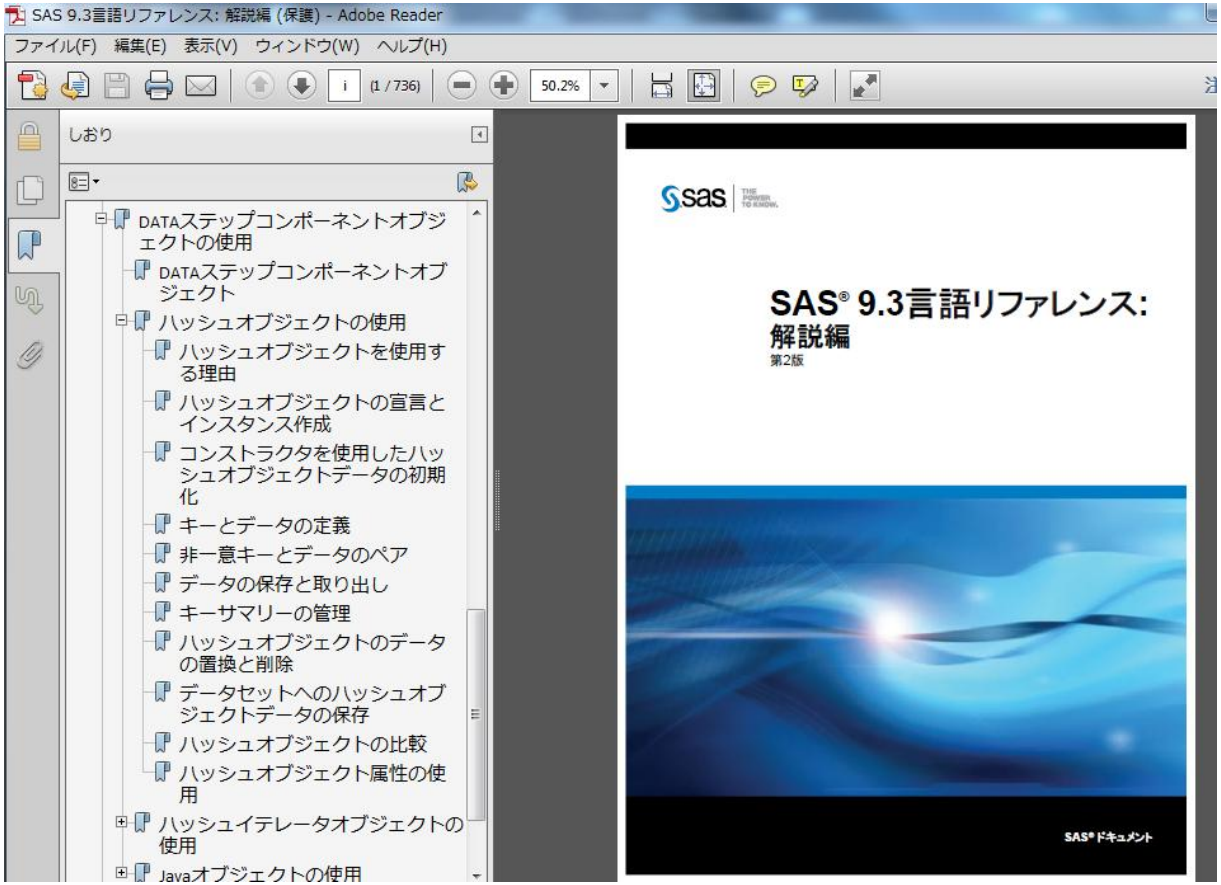
http://www.sas.com/offices/asiapacific/japan/service/onlinehelp.html#section=2

またはSAS社WebページTOPから「サポート」⇒「日本語版SASオンラインヘルプ」



SAS® 9.3言語リファレンス:解説編

22 章 DATA ステップコンポーネントオブジェクトの使用 p449～

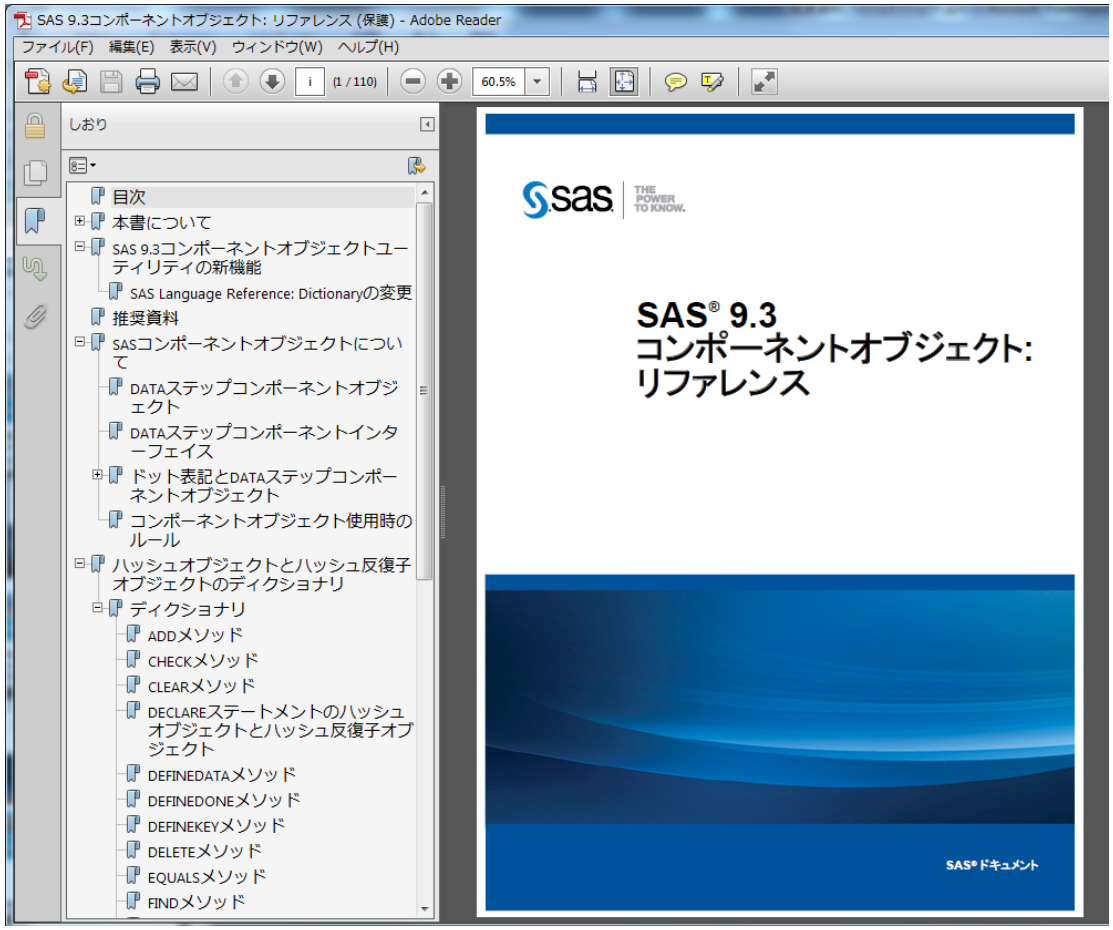


ハッシュオブジェクトの概略や基本的な使い方が説明されている。

詳しいメソッド（命令式）の1つ1つについては次にあげる資料を参照する。

SAS® 9.3コンポーネントオブジェクト: リファレンス

2章ハッシュオブジェクトとハッシュ反復子オブジェクトのディクショナリp5～



SASハッシュオブジェクトで使用可能な全メソッドが1つ1つヘルプ的に紹介されている

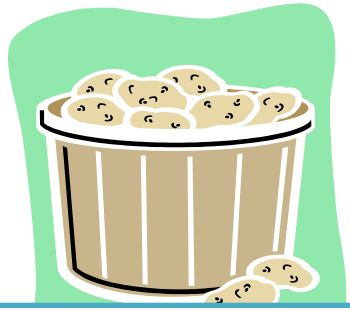
「SAS HASH」でWeb検索すれば、海外のPaperが山ほど、見つかり、実際どういった風に使われているかがわかるので、

リファレンスを読む前に
まずそれらを流し読みする方が理解が速くなるかもしれません

ハッシュってなに？①

もともと、細切れやごちゃ混ぜといった意味を持つ単語だが、ようはデータの検索や確認をするためのこまかいキーワードみたいなイメージ

ハッシュドビーフ



ハッシュドポテト

ハッシュってなに？②

ハッシュ関数等の言葉と混同してしまうが、少し意味が違うので忘れる

連想配列ともいう

乱暴にいうと、変数の値そのものを要素番号の代わりにできる検索テーブル

ARRAYによる配列と、ハッシュオブジェクトのイメージ

	商品コード	名前	値段
1	AA121	りんご	100
2	DS345	キウイ	130
3	WQ444	オレンジ	120

ARRAYのイメージ

商品コードの配列

CODE_1	CODE_2	CODE_3
AA121	DS345	WQ444
1	2	3

商品名の配列

NAME_1	NAME_2	NAME_3
りんご	キウイ	オレンジ
1	2	3

価格の配列

PRICE_1	PRICE_2	PRICE_3
100	130	120
1	2	3

キウイの情報をください

商品コードの配列の要素番号2
と
商品名の配列の要素番号2と
価格の配列の要素番号2に
該当する変数の値ですね。

ARRAYは変数を「配列」というグループで括り、**要素番号を振って管理する**。
プログラマは要素番号を指定することで、配列から求めるべき値をとりだすことができる

ハッシュオブジェクトのイメージ

Key

Data

	商品コード	名前	値段
1	AA121	りんご	100
2	DS345	キウイ	130
3	WQ444	オレン	120

キウイの情報をください

Keyが「DS345」のDataですね

ハッシュオブジェクトは、Keyとそれに結びつくDataで管理する。要素番号で値を得るので番号ではなく、Keyを使って、Dataになっている変数を全部持ってくるイメージ

データの結合を MERGE・SQL・HASHでやってみる

AE(有害事象)

VIEWTABLE: Work.Ae

	USUBJID	AEID	AETERM	CMID
1	T-001	1	不眠	1
2	T-001	2	頭痛	2
3	T-001	3	発熱	2
4	T-001	4	胃炎	3
5	T-002	1	腸炎	1
6	T-002	2	感冒	2
7	T-002	3	骨折	3
8	T-002	4	頭痛	3

CM(併用薬)

VIEWTABLE: Work.Cm

	USUBJID	CMID	CMTERM
1	T-001	1	睡眠導入薬
2	T-001	2	解熱鎮痛剤
3	T-001	3	胃腸薬
4	T-002	1	胃腸薬
5	T-002	2	感冒薬
6	T-002	3	解熱鎮痛剤

有害事象にどんな併用薬が使われたかが知りたい場合、AEのデータをベースにして、USUBJIDとCMIDをキーとして結合する

データの結合を MERGE・SQL・HASHでやってみる

MERGE

```
proc sort data=AE out=AE_1;
 by USUBJID CMID;
run;

proc sort data=CM out=CM_1;
 by USUBJID CMID;
run;

data MERGE_AE;
 merge AE_1(in=ina)
       CM_1;
if  ina;
run;
```

Mergeは事前に
ソートが必要

SQL

```
proc sql noprint;
 create table SQL_AE as
  select
AE.USUBJID,AEID,AETERM,AE.CMID,CMTERM
  from AE left outer join CM
  on AE.USUBJID=CM.USUBJID and
AE.CMID=CM.CMID;
quit;
```

コードの詳細解説は書ききれないので論文を

HASH（リターンコードなし）

```
data HASH_AE;
if _N_=0 then set CM;    －①
if _N_=1 then do;    －②
declare hash hcm(dataset:'CM');    －③
 hcm.definekey('USUBJID','CMID');    －④
 hcm.definedata('CMTERM');    －⑤
 hcm.definedone();   －⑥
end;
set AE;    －⑦
 hcm.find();   －⑧
run;
```

①は、変数の初期化のため
②はハッシュオブヘクトの宣言は一度でいいため
③で[hcm]という名前のハッシュオブジェクトを宣言している。(dataset:)で格納するデータセットを指定
④で何をKeyにするかを指定
⑤で何をデータにするかを指定
⑥で定義終了
⑦から通常のデータステップ、AEをセット
⑧はfindメソッド、これによりKeyであるUSUBJIDとCMIDに基づいてハッシュオブジェクトからDataであるCMTERMがやってくる

全て同じ結果を得ることができました

	USUBJID	CMID	CMTERM	AEID	AETERM
1	T-001	1	睡眠導入薬	1	不眠
2	T-001	2	解熱鎮痛剤	2	頭痛
3	T-001	2	解熱鎮痛剤	3	発熱
4	T-001	3	胃腸薬	4	胃炎
5	T-002	1	胃腸薬	1	腸炎
6	T-002	2	感冒薬	2	感冒
7	T-002	3	解熱鎮痛剤	3	骨折
8	T-002	3	解熱鎮痛剤	4	頭痛

declearステートメントでハッシュオブジェクトを作ることができます

ハッシュオブジェクトを宣言する

```
declare hash ハッシュ名(オプション);
```

宣言時にSASデータセットをハッシュに格納する場合

```
declare hash ハッシュ名(dataset:'データセット名');
```

作成されたハッシュオブジェクトに対しては、メソッドという方法でしか操作できない。
通常のSASのステートメントや関数その他、一切無効なので、覚えるしかない

基本構文

```
ハッシュオブジェクト名.メソッド(変数やオプションの指定);
```

先ほどのコードで使ったメソッドを紹介

Definekeyメソッド：ハッシュオブジェクトにKeyを設定する

```
hcm.definekey('USUBJID','CMID');
```

Definedataメソッド：ハッシュオブジェクトにdataを設定する

```
hcm.definedata('CMTERM');
```

Definedoneメソッド：ハッシュオブジェクトの作成を完了する

```
hcm.definedone();
```

findメソッド：ハッシュオブジェクトからkeyに基づいて、dataを取得する

```
hcm.find();
```

変数の指定など、全てシングルコーテーションが必要なところに注意！

リターンコードの話

AE(有害事象)

VIEWTABLE: Work.Ae

	USUBJID	AEID	AETERM	CMID
1	T-001	1	不眠	1
2	T-001	2	頭痛	2
3	T-001	3	発熱	2
4	T-001	4	胃炎	3
5	T-002	1	腸炎	1
6	T-002	2	感冒	2
7	T-002	3	骨折	3
8	T-002	4	頭痛	3

_CM
(CMからCMID=1を削除)

	USUBJID	CMID	CMTERM
1	T-001	2	解熱鎮痛剤
2	T-001	3	胃腸薬
3	T-002	2	感冒薬
4	T-002	3	解熱鎮痛剤

CMID＝1の値がCMから無くなった。MERGEやSQLなら、問題にならないが、果たしてHASHは、、、、？

```
NOTE: データセット WORK._CM から 4 オブザベーションを読み込みました。
ERROR: Key not found.
ERROR: Key not found.
NOTE: エラーが発生したため、このステップの処理を中止しました。
NOTE: データセット WORK.AE から 8 オブザベーションを読み込みました。
WARNING: データセット WORK.HASH_AE は未完成です。このステップは、 8 オブザベーション、5 変数で停止しました。
NOTE: DATA ステートメント 処理 (合計処理時間):
      処理時間                0.32 秒
      CPU 時間                0.04 秒
```

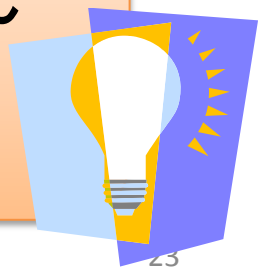
先ほどと同じコードを実行したら、エラーになりました！！

Findメソッドは、与えられた値からハッシュオブジェクトのKeyを検索して、見つからない場合はエラーを返す

↓

メソッドからエラーを渡されたSASは、ログにエラーを出して、処理を中断する

メソッドのエラーを制御する仕組みがあればよい。



HASH（リターンコードあり）

```
data HASH_AE;
if _N_=0 then set _CM;
if _N_=1 then do;
declare hash hcm(dataset:'_CM');
 hcm.definekey('USUBJID','CMID');
 hcm.definedata('CMTERM');
 hcm.definedone();
end;
set AE;
 rc=hcm.find();
run;
```

メソッドが返すエラー（或いは正常に実行できた際の0値）を、rcという変数に割り当てることで、この変数で吸収して、ログにエラーを返さないようにする。

結果、処理は止まらない

```
NOTE: データセット WORK._CM から 4 オブザベーションを読み込みました。
NOTE: データセット WORK.AE から 8 オブザベーションを読み込みました。
NOTE: データセット WORK.HASH_AE は 8 オブザベーション、 6 変数です。
NOTE: DATA ステートメント 処理 (合計処理時間):
      処理時間                0.01 秒
      CPU 時間                0.01 秒
```

	USUBJID	CMID	CMTERM	AEID	AETERM	rc
1	T-001	1		1	不眠	160038
2	T-001	2	解熱鎮痛剤	2	頭痛	0
3	T-001	2	解熱鎮痛剤	3	発熱	0
4	T-001	3	胃腸薬	4	胃炎	0
5	T-002	1	胃腸薬	1	腸炎	160038
6	T-002	2	感冒薬	2	感冒	0
7	T-002	3	解熱鎮痛剤	3	骨折	0
8	T-002	3	解熱鎮痛剤	4	頭痛	0

メソッドが失敗した箇所には0でない数字が入る（数字はエラーメッセージごとに固有）

先ほど、リターンコードなしの際にERRORが2つ上がっていたが、その二つが右図の160038に対応している

これだけでは結果は正しくない！

もう一工夫！

	USUBJID	CMID	CMTERM	AEID	AETERM	rc
1	T-001	1		1	不眠	160038
2	T-001	2	解熱鎮痛剤	2	頭痛	0
3	T-001	2	解熱鎮痛剤	3	発熱	0
4	T-001	3	胃腸薬	4	胃炎	0
5	T-002	1	胃腸薬	1	腸炎	160038
6	T-002	2	感冒薬	2	感冒	0
7	T-002	3	解熱鎮痛剤	3	骨折	0
8	T-002	3	解熱鎮痛剤	4	頭痛	0

Keyがみつからず、findメソッドのエラー箇所のはずなのに、なぜか値が入っている？

よく見ると、前のオブザベーションの値が引き延ばされている！！

findメソッドは失敗すると、最後にうまくいった値を引きずる！！

敢えてその性質を利用することもあるが、今回は間違いなので次コードで対応する

HASH（リターンコードあり＋エラー処理）

```
data HASH_AE;
 if _N_=0 then set _CM;
 if _N_=1 then do;
 declare hash hcm(dataset:'_CM');
  hcm.definekey('USUBJID','CMID');
  hcm.definedata('CMTERM');
  hcm.definedone();
 end;
set AE;

  rc=hcm.find();
   if rc^=0 then CMTRM=”;
run;
```

rcの値が0でない場合が、findメソッドがkeyを見つけられなかった値というのならば、その時には、引き延ばされたCMTRMを消してあげればいい

ハッシュオブジェクトにaddメソッドでにデータを格納する（特定の事象名にフラグを立てる処理）

```
data HASH_PAIN;
 length AETERM $4. FLAG $1.; －①
if _N_=1 then do;
declare hash pain();     －②
 rc=pain.definekey('AETERM');
 rc=pain.definedata('FLAG');
 rc=pain.definedone();
 rc=pain.add(key:'頭痛',data:'Y'); －③
rc=pain.add(key:'骨折',data:'Y');
call missing(AETERM,FLAG);     －④
end;
set AE;
 rc=pain.find();
 if rc^=0 then FLAG='N';
 drop rc;
run;
```

①は、変数定義のため
②で今回は、データセットから読み込まないので、空括弧なことに注目
③で直接、keyとdataを指定してハッシュオブジェクトに格納している
④call missingしないと変数が初期化されていませんになる

	AETERM	FLAG	USUBJID	AEID	CMID
1	不眠	N	T-001	1	1
2	頭痛	Y	T-001	2	2
3	発熱	N	T-001	3	2
4	胃炎	N	T-001	4	3
5	腸炎	N	T-002	1	1
6	感冒	N	T-002	2	2
7	骨折	Y	T-002	3	3
8	頭痛	Y	T-002	4	3

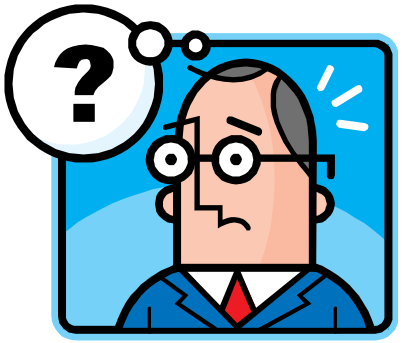
先ほどのコードで使ったメソッドを紹介

Addメソッド：ハッシュオブジェクトにデータを挿入する

```
pain.add(key:'頭痛',data:'Y');
```

Definedoneメソッドで、一度ハッシュオブジェクトを作成してから、入れる

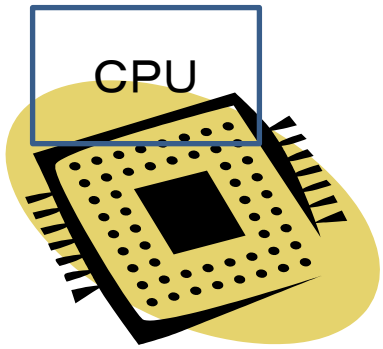
なんのためにわざわざハッシュオブジェクトなんて使う必要があるんだろう



プログラム処理時間の高速化

ステップ数の大幅な削減・コードのスリム化

マスターすれば、ハッシュの中でデータ加工やその他、データステップでできるようなことを全てできるので、超絶技巧コードを生み出せる

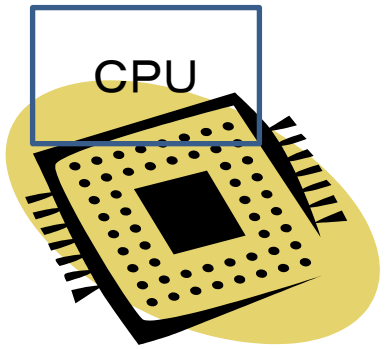


CPUは処理を行う頭脳。通常SASの処理はハードディスクからデータを読み込んで、処理する

メモリ:
データへのアクセスは超高速で可能だが、容量が小さい

ハードディスク:
容量は膨大だが、データへのアクセスには時間がかかる

一昔前までは、メモリ容量が貧弱だったため、メモリは処理のための一時的な作業領域で、使ったらすぐに解放。
メモリ上にデータを格納することはありえなかった



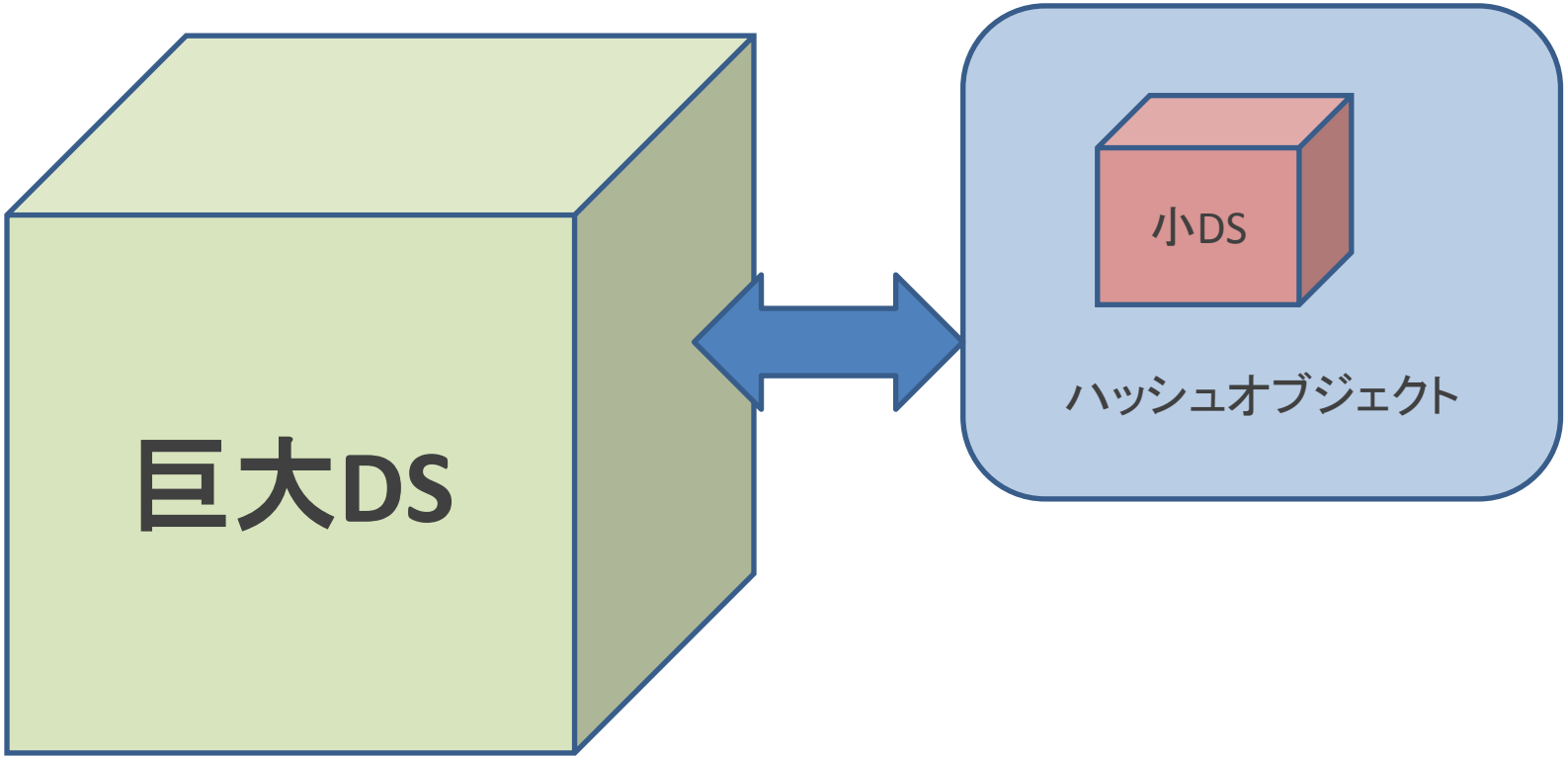
メモリ

ハードディスク

ハッシュオブジェクトはメモリ上に展開され、
ステップの終了までメモリ上に居座る。
そのため、データのアクセスが高速

ハッシュオブジェクト

ハッシュオブジェクトが向いているのは、
巨大なデータセットから小さいデータセットを参照するようなケース！！
（いくらメモリ容量が昔に比べて大きくなったとはいえ、あまり大きなデータをメモリ上に展開できないので）



大きいデータセット（マクロ変数の値をどんどん大きくしていく）	小さいデータセット

```
%let size=10;

data BIG;
 do ID=1 to &size;
  do RID=1 to 100;
   array dummy(5) $100.;
        output;
  end;
 end;
run;
```

```
data SMALL;
 do RID=100 to 1 by -1;
   array dummy(5) $100.;
        FLAG='Y';
output;
  end;
run;
```

2つのデータセットをMerge SQL HASHの3種類の方法で結合する処理について実行時間を比較する。大きいデータセットについては、どんどん大きくしていく。

Merge

```
proc sort data=BIG
out=BIG_1;
 by RID;
run;
proc sort data=SMALL
out=SMALL_1;
 by RID;
run;
data MERGE_DS;
 merge BIG_1(in=ina)
       SMALL_1;
 by RID;
 if ina;
run;
```

SQL

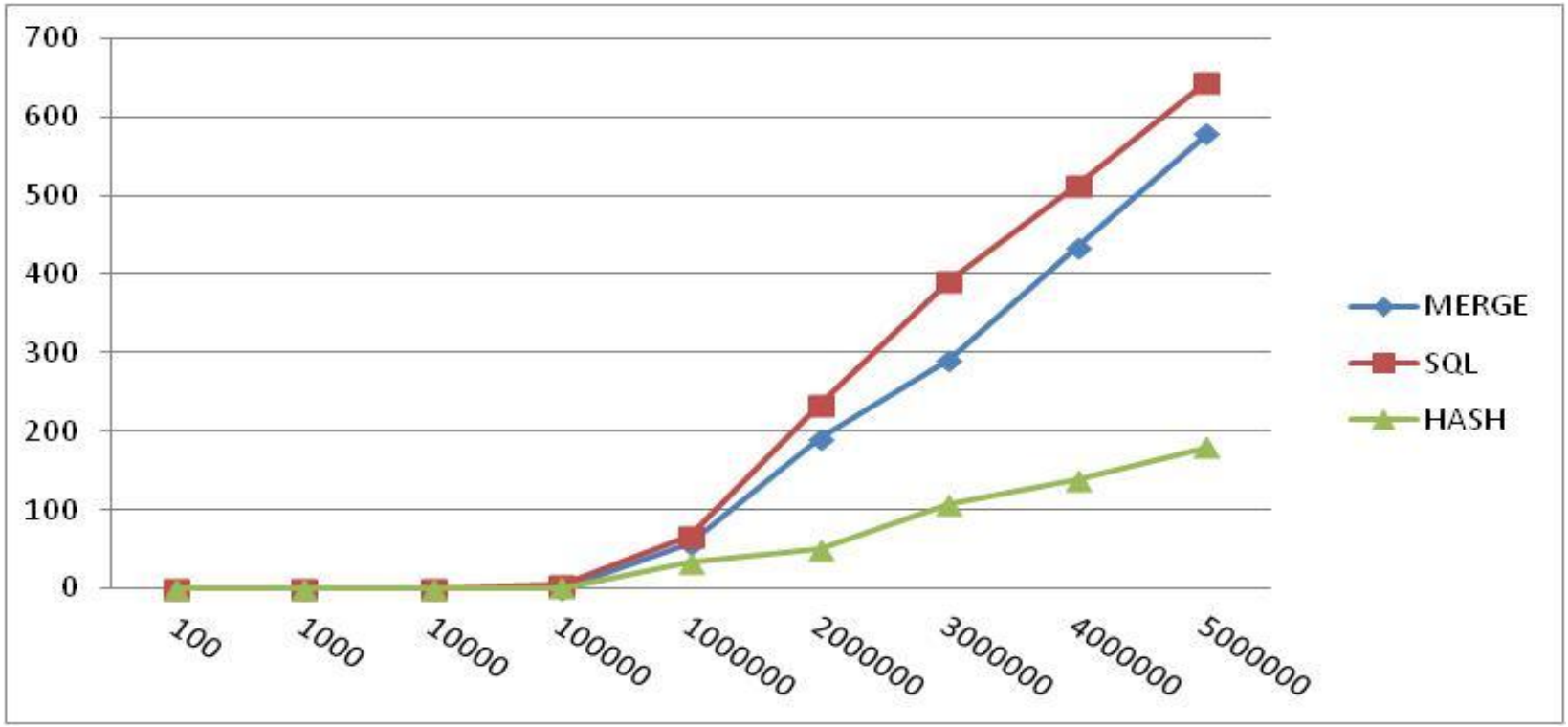
```
proc sql noprint;
 create table SQL_DS as
  select BIG.*,FLAG
   from BIG left outer join SMALL
    on BIG.RID=SMALL.RID;
quit;
```

HASH

```
data HASH_DS;
 if _N_=0 then set SMALL;
 if _N_=1 then do;
  declare hash HS(dataset:'SMALL');
   rc=HS.definekey('RID');
   rc=HS.definedata('FLAG');
   rc=HS.definedone();
  end;
 set BIG;
  rc=HS.find();
  if rc^=0 then FLAG='';
run;
```

方法	データセット「BIG」のOBS数								
	100	1000	10000	100000	1000000	2000000	3000000	4000000	5000000
MERGE	0.02	0.03	0.06	0.44	56.29	190.75	290.6	434.64	578.81
SQL	0.01	0.03	0.08	4.96	66.03	233.86	391.44	513.17	643.74
HASH	0.02	0.02	0.05	0.7	32.29	49.34	107.91	137.33	179.73

（秒）



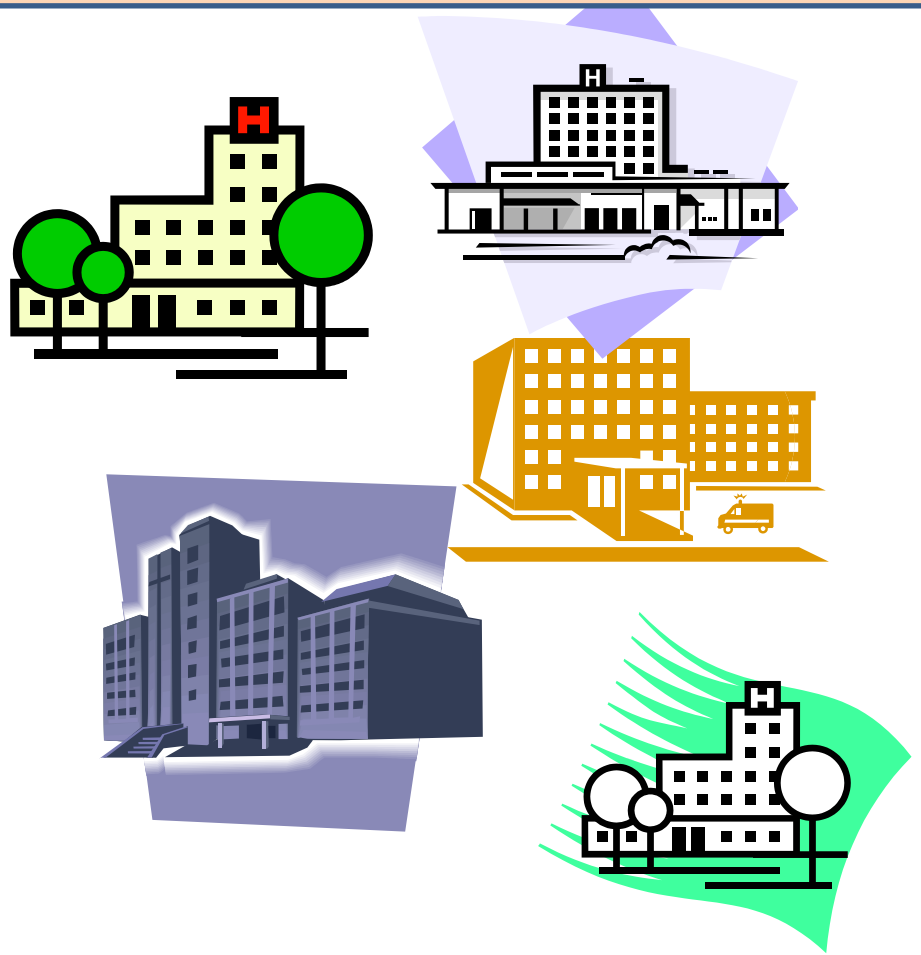
データセット」のサイズが小さいうちはどの方法も実行時間に差はないが、ある程度巨大になってくると、処理時間の増加率に差がでる。
HASHの場合、データの増加に対して処理効率がよい。

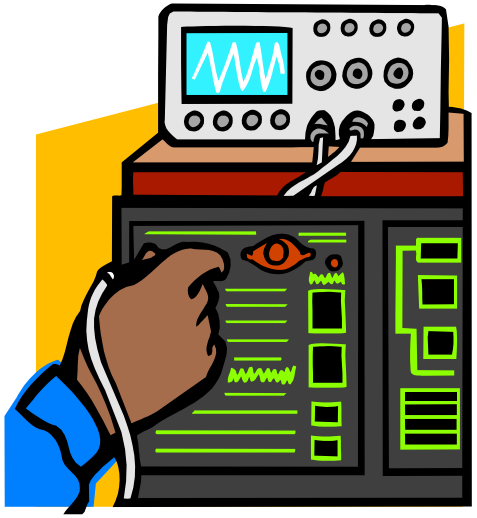
ただし！ハッシュオブジェクトは実行するハードのCPU性能やメモリ容量等で実行時間が大きく変わる!!

また実行する処理の内容や、使用するデータセットサイズ、ハッシュオブジェクトに格納するデータサイズなど、様々な要因が絡むので、今回の実験結果は参考程度に！！

実際に使う場合は、自分の実行環境で、実データか或いはそれに似たサンプルデータで検証をして、ハッシュの方が良いと確信を得てから使ってください！！
ハッシュを使って余計に遅くなる場合もありえます

多施設で行われる臨床試験の場合、別の施設で測定された臨床検査値を一つにまとめて解析するが、各施設それぞれに臨床検査基準値が存在するため、それを考慮する必要がある





LB(各被験者の検査値データ)

	SITE	USUBJID	LBDT	PARAM	AVAL
1	A施設	T-001	2013/01/15	RBC	380
2	A施設	T-001	2013/01/15	WBC	4.2
3	A施設	T-001	2013/02/15	RBC	372
4	A施設	T-001	2013/02/15	WBC	3450
5	A施設	T-001	2013/03/15	RBC	373
6	A施設	T-001	2013/03/15	WBC	3450
7	B施設	T-002	2013/01/15	RBC	500
8	B施設	T-002	2013/01/15	WBC	3.2
9	B施設	T-002	2013/02/15	RBC	380
10	B施設	T-002	2013/02/15	WBC	4.2
11	B施設	T-002	2013/03/15	RBC	371
12	B施設	T-002	2013/03/15	WBC	1.8

LBSS(各施設の基準値データ)

	SITE	USUBJID	LBDT	PARAM	AVAL
1	A施設	T-001	2013/01/15	RBC	380
2	A施設	T-001	2013/01/15	WBC	4.2
3	A施設	T-001	2013/02/15	RBC	372
4	A施設	T-001	2013/02/15	WBC	3450
5	A施設	T-001	2013/03/15	RBC	373
6	A施設	T-001	2013/03/15	WBC	3450
7	B施設	T-002	2013/01/15	RBC	500
8	B施設	T-002	2013/01/15	WBC	3.2
9	B施設	T-002	2013/02/15	RBC	380
10	B施設	T-002	2013/02/15	WBC	4.2
11	B施設	T-002	2013/03/15	RBC	371
12	B施設	T-002	2013/03/15	WBC	1.8

各被験者の検査データは、試験が進むにつれ、どんどん巨大化していくが、基準値のデータは、変更などにより多少増えることはあっても、基本的にはそれほど大きくならない

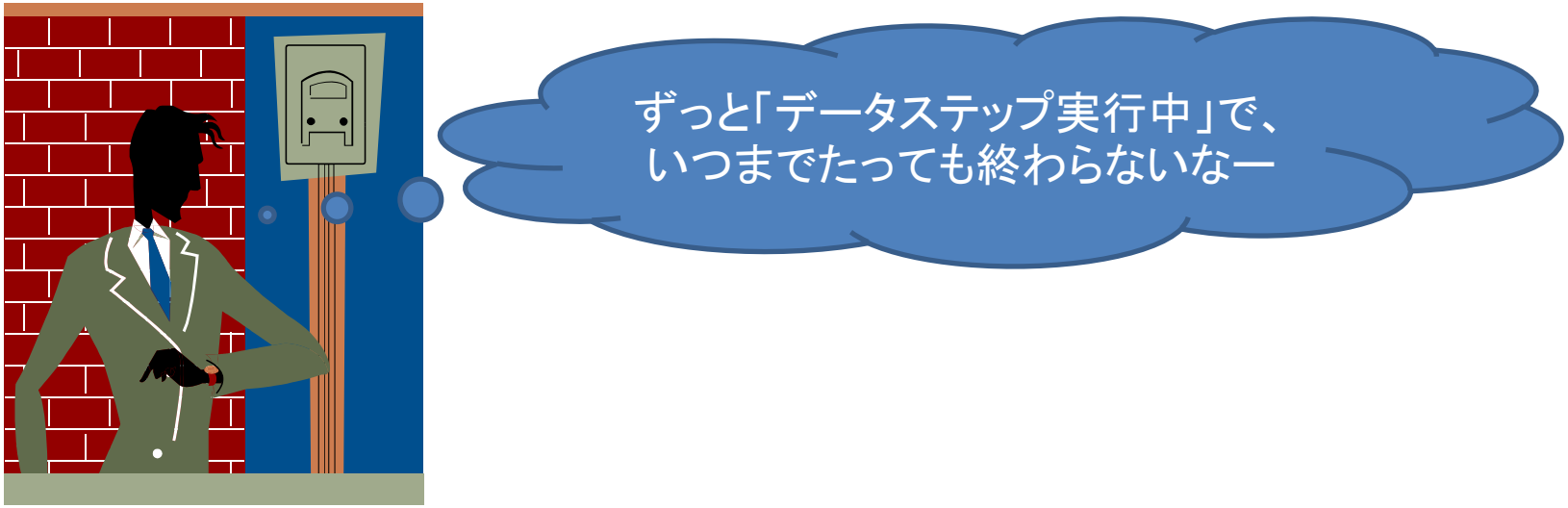
```
proc sort data=LBSS;
 by SITE PARAM STDT;
run;
proc transpose data=LBSS
out=LBSS_DT(drop=_NAME_) prefix=STDT_;
 var STDT;
 by SITE PARAM;
run;
proc transpose data=LBSS
out=LBSS_UNI(drop=_NAME_) prefix=UNI_;
 var UNI;
 by SITE PARAM;
run;
proc transpose data=LBSS
out=LBSS_LOW(drop=_NAME_) prefix=LOW_;
 var LOW;
 by SITE PARAM;
run;
proc transpose data=LBSS
out=LBSS_HI(drop=_NAME_) prefix=HI_;
 var HI;
 by SITE PARAM;
Run;
data LBSS_1;
 merge LBSS_DT LBSS_UNI LBSS_LOW LBSS_HI;
  by SITE PARAM;
run;
```

```
proc sort data=LB;
 by SITE PARAM;
run;
data LB_1;
 merge LB(in=ina)
       LBSS_1;
  by SITE PARAM;
  if ina;
run;

data LB_2;
 set LB_1;
  array ARDT(*) STDT_:;
  array ARUNI(*) UNI_:;
  array ARLOW(*) LOW_:;
  array ARHI(*) HI_:;
  do i=1 to dim(ARDT);
   if LBDT>=ARDT(i)>. then do;
    STDT=ARDT(i);
    UNI=ARUNI(i);
                LOW=ARLOW(i);
                HI=ARHI(i);
   end;
  end;
  format STDT yymmdds10.;
 drop STDT_: UNI_: LOW_: HI_: i;
run;
```

前スライドのコードのように、施設基準値を転置して、検査データと結合し、doループで検査日と施設基準値の適応開始日を比較して、対応する基準値を取得することは容易。

しかし、コードは単純であっても、ステップ数がどうしても多くなりがち。
巨大なデータになればなるほど、1ステップに数十秒かかることも珍しくないため全体としての実行時間がながくなる



```
proc sort data=LBSS;
 by SITE PARAM STDT;
run;
data HASH_PATTERN_1
(rename=(STDT_=STDT UNI_=UNI HI_=HI LOW_=LOW));
declare hash hlbss(ordered: 'Y'); －①
  hlbss.definekey('SITE','PARAM','COUNTER'); －②
  hlbss.definedata('STDT','UNI','LOW','HI'); －③
hlbss.definedone();
 do until(ENDLBSS); －④
  set LBSS end=ENDLBSS; －⑤
   by SITE PARAM STDT; －⑥
   if first.PARAM then COUNTER=0; －⑦
    COUNTER+1; －⑧
    rc=hlbss.add();－⑨
 end;
 do until(ENDLB); －⑩
  set LB end=ENDLB;
   COUNTER=1; －⑪
   rc=hlbss.find();－⑫
    do while(rc=0); －⑬
     if LBDT>=STDT then do; －⑭
      STDT_=STDT; －⑮
      UNI_=UNI;
      LOW_=LOW;
      HI_=HI;
     end;
     COUNTER+1; －⑯
     rc=hlbss.find();－⑰
    end;
  output; －⑱
 end;
drop STDT UNI LOW HI COUNTER;
format STDT_ yymmdds10.;
run;
```

ハッシュならコードは長いが、2ステップで完了。

コードの説明は論文を参照してください。

おおまかにいうと、この方法は縦版ARRAYのような処理。

ハッシュオブジェクトに臨床検査値を格納する際にカウンターとなる変数を作ってそれを使って縦版ループのようなことをしている。

カウンター変数を作っている分、個人的に美しくない

```
proc sort data=LBSS;
 by SITE PARAM descending STDT;
run;

data HASH_PATTERN_2;
 if _N_=0 then do;
  set LBSS;
 end;
 if _N_=1 then do;
  declare hash h_lbss(dataset:'LBSS', multidata: 'Y');-①
   h_lbss.definekey('SITE','PARAM'); －②
   h_lbss.definedata('STDT','UNI','LOW','HI');
   h_lbss.definedone();
 end;

 do until(ENDLB);
  set LB end=ENDLB;
   call missing(STDT,UNI,LOW,HI);
   rc=h_lbss.find();

    do while(STDT>LBDT); －③
     rc=h_lbss.find_next();－④
    end;
   output;
 end;

format STDT yymmdds10.;
run;
```

9.2Phaze2以上でないとだめなコードだが、先ほどよりコンパクトにすることができた。

①でmultidataオプションを使い、ハッシュオブジェクト内のkey重複を許容する。

Findメソッドで一度基準値を取得した後、③で日付判定を入れ、④のfind_nextメソッドで、次に一致するkeyを検索する。

ハッシュ格納データの降順ソートがハッシュ内で保持されていることを当て込んだコード

先のコードは全て、あくまでも説明用サンプルデータに対して正しく処理できる説明用のサンプルコードです。

実際は、該当する基準値が存在しない等の、起こりうる様々な条件分岐を考慮する必要があります。

そのままコピーペーストで実装せずに、必ずカスタマイズしてテストを行ってから、使ってください。

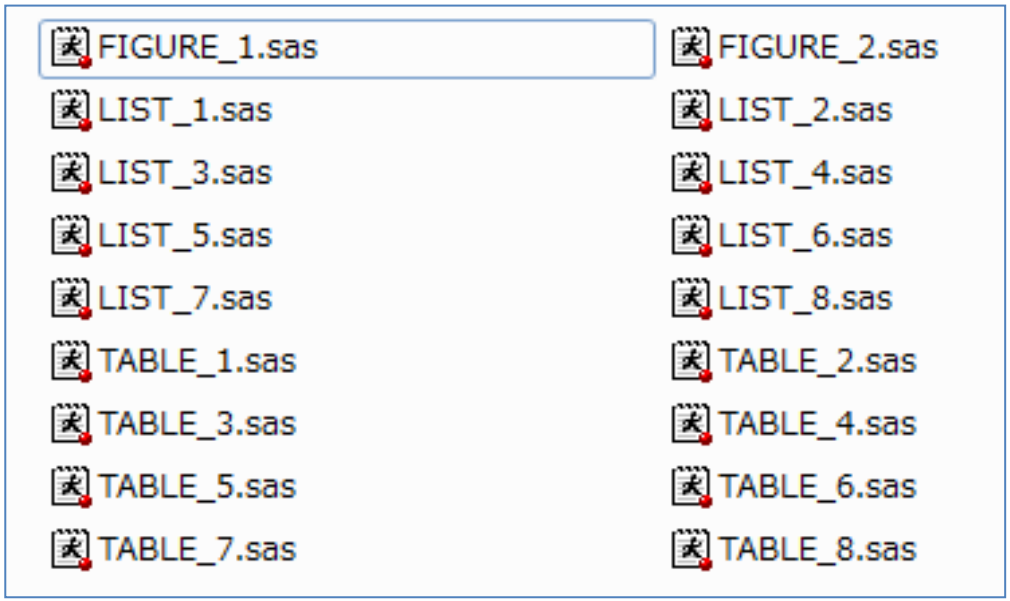
発表者もハッシュオブジェクト初心者なので信用しないで

おまけ（詳細は論文）

自分や他人が、どんな関数やプロシージャを使っているかを分析したい

ハッシュ反復子オブジェクトという方法と、Perl正規表現を組み合わせることで、そういったテキストマイニングを簡単にできる

フォルダ「C:¥解析Program」の中身



.sasは所詮ただの
テキストファイルなのでテキストファイルを読み込む
処理を書けばよい

```
%let place =C:¥解析Program;
filename FOLDER pipe "dir / b &place."; －①
data FILELIST;
  length FNAME $200 FPATH $1000;
  infile FOLDER dlm = '09'x;
  input FNAME;
  if index(FNAME,".sas") > 0;
   FPATH    =    trim(left("&place."))    ||    "¥"    ||
trim(left(FNAME));
run;
filename INP dummy; －②

③
data CODE;
  length LINE $1000.;
  set FILELIST;
  infile INP
 filevar=FPATH end=FLG dlm='09'X missover lrecl =
1000;
  do until(FLG);
    input LINE $;
     output;
  end;
run;
```

コードをそのままSASデータセットにした状態

	LINE	FNAME
861	call symputx('obs',_N_);	LIST_4.sas
862	run;	LIST_4.sas
863		LIST_4.sas
864	data OUTPUT_1;	LIST_4.sas
865	merge OUTPUT_0(in=ina)	LIST_4.sas
866	ADSL(keep=SUBJID REGAGE SEX);	LIST_4.sas
867	by SUBJID;	LIST_4.sas
868	if ina;	LIST_4.sas
869	run;	LIST_4.sas
870	proc sort;	LIST_4.sas
871	by SUBJID VISITNO;	LIST_4.sas
872	run;	LIST_4.sas
873	data OUTPUT;	LIST_4.sas
874	set OUTPUT_1;	LIST_4.sas
875	by SUBJID VISITNO;	LIST_4.sas
876	if first.SUBJID then do;	LIST_4.sas
877	SEX_=kscan(put(SEX,HSEX.),2,'');	LIST_4.sas
878	AGE_=REGAGE;	LIST_4.sas
879	end;	LIST_4.sas
880	run;	LIST_4.sas
881		LIST_4.sas
882	/*==========	LIST_4.sas

```
proc sort data=SASHELP.VFUNC－①
out=FUNC_DIC(keep=FNCNAME) nodupkey; －②
     where FNCNAME^=''; by FNCNAME;
run; －③
data FUNCTION(keep=FNCNAME LINE FNAME)
     PROCEDUR(keep=PRONAME LINE FNAME);
 if _N_=0 then do;
  set FUNC_DIC;
 end;
 declare hash FHUSH(dataset:'FUNC_DIC',ordered:'Y'); －④
  declare hiter FHITER('FHUSH');    －⑤
   FHUSH.definekey('FNCNAME');    －⑥
   FHUSH.definedata('FNCNAME');    －⑦
   FHUSH.definedone();
 do until(ENDLINE);
  set CODE end=ENDLINE;
   rc=FHITER.first();   －⑧
    do until(rc); －⑨
     prx1=compress('m/¥W'||lowcase(FNCNAME)||'¥(/');    －⑩
     if prxmatch(prx1,lowcase(LINE))>0 then output FUNCTION; ⑪
     rc=FHITER.next();   －⑫
    end;
   prx2=prxparse('s/proc (.+?)( |;).*/$1/');   －⑬
   pos=prxmatch(prx2,lowcase(LINE));    －⑭
   if pos>0 then do;
    PRONAME=prxposn(prx2,1,lowcase(LINE));    －⑮
     output PROCEDUR;
    end;
  end;
stop;
run;
```

④でハッシュオブジェクトFHUSHを宣言した後、⑤でそのFHUSHを使って
declare hiter FHITER('FHUSH');

としているが、これがハッシュ反復子オブジェクトの宣言。

ハッシュオブジェクトを指定してハッシュ反復子オブジェクトの中にいれることで、key-dataに依存しないメソッドが利用可能になるため　柔軟性がアップする。

ハッシュ反復子オブジェクト専用のメソッドが多くあって、それを使えるようにすること、key-dataの構造的縛りからコードを解放できることがメリット。

⑬以降がPerl正規表現を利用してコードからプロシジャ名と関数名を分離抽出している箇所

抽出されたプロシジャ名と使用コード行

	LINE	FNAME	PRONAME
1	proc lifetest	FIGURE_1.sas	lifetest
2	proc sort data=BASE_0;	FIGURE_1.sas	sort
3	proc sql noprint;	FIGURE_1.sas	sql
4	proc sort data=base_2;by axis_time pfs_m;run;	FIGURE_1.sas	sort
5	proc sort data=layout;	FIGURE_1.sas	sort
6	proc sort data=base4_1;	FIGURE_1.sas	sort
7	proc template;	FIGURE_1.sas	template
8	proc sgrender data=base template=FE002;	FIGURE_1.sas	sgrender
9	proc lifetest	FIGURE_2.sas	lifetest
10	proc sort data=BASE_0;	FIGURE_2.sas	sort
11	proc sql noprint;	FIGURE_2.sas	sql
12	proc sort data=base_2;by axis_time OS_m;run;	FIGURE_2.sas	sort
13	proc sort data=layout;	FIGURE_2.sas	sort
14	proc sort data=base4_1;	FIGURE_2.sas	sort
15	proc template;	FIGURE_2.sas	template
16	proc sgrender data=base template=FE001;	FIGURE_2.sas	sgrender
17	proc sort data=CRF.PERFOM out=PERFOM;	LIST_1.sas	sort
18	proc sort data=CRF.PS out=PS;	LIST_1.sas	sort
19	proc sort;	LIST_1.sas	sort
20	proc sort data=LAB;	LIST_2.sas	sort
	proc transpose data=LAB		

抽出された関数名と使用コード行

32	KSCAN	UNIT=kscan(PARAM,2,'()');	LIST_2.sas
33	KSCAN	PARAM=kscan(PARAM,1,'(');	LIST_2.sas
34	IN	where SUBJID='RCC-ZIG04' and COL_11 not in('2011/06/04','2011/07/01');	LIST_2.sas
35	COMPRESS	if IRGDTC^='' then COL_11=compress(COL_11||'('||IRG	LIST_2.sas
36	SYMPUTX	call symputx('obs',_N_);	LIST_2.sas
37	KSCAN	JIKI=kscan(put(VISITNO,GNV.),2,'.')|	LIST_2.sas
38	INDEX	if index(PARAM,'検査日')>0 then do;	LIST_2.sas
39	KSCAN	VISITNO_=kscan(put(VISITNO,GNV.	LIST_3.sas
40	KSCAN	SJIKI_=kscan(put(SJIKI,GNM.),2,'.');	LIST_3.sas
41	KSCAN	KOUHAN_=kscan(put(KOUHAN,GN	LIST_3.sas
42	ROUND	if DT^=. then DT_=put(round(DT,0.01),8.2);	LIST_3.sas
43	SYMPUTX	call symputx('obs',_N_);	LIST_3.sas
44	KSCAN	VISITNO_=kscan(put(VISITNO,GNV.	LIST_4.sas
45	KSCAN	ING_=kscan(put(ING,GNF.),2,'.');	LIST_4.sas
46	SYMPUTX	call symputx('obs',_N_);	LIST_4.sas
47	KSCAN	SEX_=kscan(put(SEX,HSEX.),2,'.');	LIST_4.sas
48	KSCAN	VISITNO_=kscan(put(VISITNO,GNV.	LIST_5.sas
49	SYMPUTX	call symputx('obs',_N_);	LIST_5.sas
50	COMPRESS	/* if i in(1,2,5) then	LIST_6.sas

上記の方法を使うことで、コードレビューが楽になる。

例えば、プログラム仕様書にないプロシジャや関数を使って解析している場合に指摘できる。

また、他人のコードから新しいプロシジャや関数を学びたい場合に、どこを見ればいいかわかるので効率がよい。

さらに

上記のデータセットを解析すれば、そのプログラマや、或いはそのプログラマが所属する組織のコードの傾向をしることができる。

使っているプロシジャや関数の傾向の類似度を定量化できれば、誰が誰からSASを教わったかを調べることもできるかも？

SASプログラミングの傾向遺伝などについて今後研究してみたい。

おわりに

SASハッシュオブジェクトは通常のSASプログラミングの書き方と大きく異なります。

そのため、ハッシュについて学び始めると、まず間違いなく、こんな面倒な書き方を勉強してまでやるぐらいなら、普通の書き方でも書けるからもういいわ！となります。

しかし、それを乗り越えて、もう少し勉強したならば、必ずハッシュオブジェクトの特徴がわかり、その使いどころや柔軟性の高さに気づくはずです。ハッシュオブジェクトと言えばイコール高速化と思われがちですが、ハッシュオブジェクトのいいところは必ずしもそこだけにあるわけではありません。

Proc sqlなども最初は特定の処理の高速化ができるということで取り上げられてきましたが、SQLの集合指向なコード体系がSASの苦手な部分をカバーできるということは既に周知の事実として受け入れられていると思います。

SQLやIML、HASHなど、違う方向性をもった技術をうまくまとめることでSASプログラムはより進化していきます。

今後も新手一生精神で、新しいアイデアを発表していきたいです