コードレビュー Vol.3

CODE Review

USP 研究所技術研究員
written by 大内智明

今回の記事で取り上げるのは、発注処理や伝票処理といった業務で発生する情報・連絡を集積し、本部および店舗の従業員が使用するPCなどにそれらの内容を通知する処理です。シンプルなシステムを開発する際の解決策として、ぜひ参考にしてください。

常駐型シェルスクリプトの紹介

まずは、概要図（図 1）をご覧ください。業務全般に言えますが、締め日時などの特定の時期になると、トランザクションが集中的に発生します。その結果、データを集積するサーバー側の処理に負荷がかかります。この影響が、他の業務と密に連携を行っている処理やサービスに及ぶケースもあります。

そこでシステム負荷を小さくするためユニケージ開発手法でよく用いられる常駐型シェルスクリプトのコードを紹介したいと思います。

紹介するセクションは、大きく 3 つの要素から成り立っています。

[1] 常駐型シェルスクリプトを実行する処理
[2] 処理 [1] を継続するかどうかの判定処理
[3] 定期的に実行する集積、加工、配信処理

リスト 1 のコードは、図 2 の赤色で示した処理フローに該当します。

図 1　発注処理・伝票処理の構成例

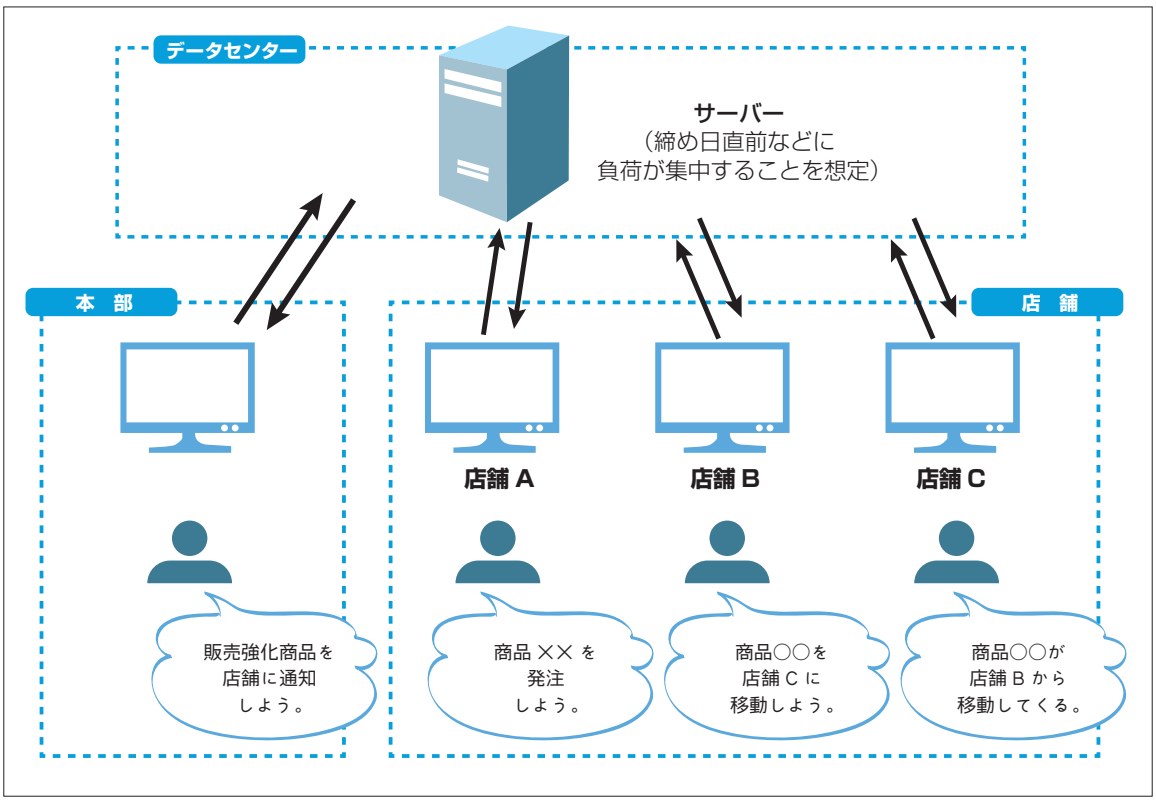
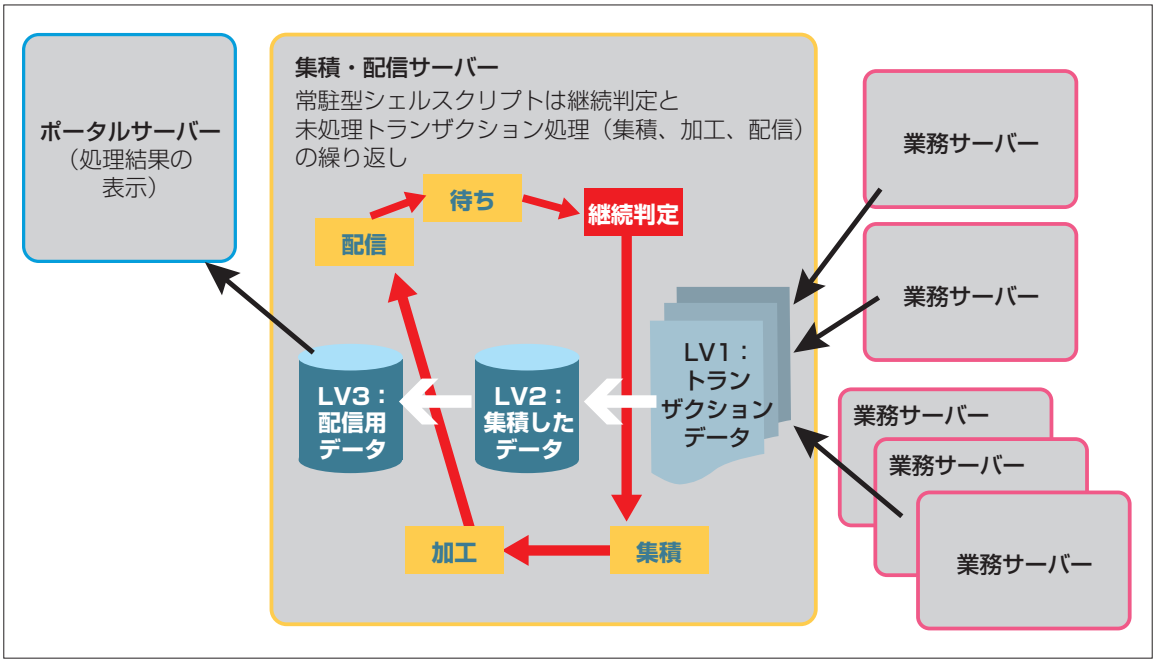


図2　常駐型シェルスクリプトの処理フロー



処理フロー内に書かれているファイルの種類「LV1、LV2、LV3」について説明します。LV1、LV2、LV3 とはユニケージ開発手法で使用されるデータのレベルごとに付した名称です（ほかに LV4 と LV5 があります）。LV1 は、システムに入力された「生データ」（原始データ）を示します。アプリケーションから入力されるデータ、外部システムからインターフェースされるデータがその例です。
LV2 は、LV1 をユニケージ上で使いやすいように変換、整形したデータを指します。LV3 は、LV2 のデータを業務上使いやすい形式に整理したデータのことです。「店別」や「商品別」といった項目や種類でデータを分類しているのが特徴です。

リスト1　処理負荷を軽減する常駐型プログラム

```
 1 #!/bin/ush -xve                                          ユニケージ独自シェル
 2 # システム名     : ＵＳＰシステム                          ush（ユーシェル）
 3 # サブシステム名: 連絡アラートＬＶ３公開テーブルの配信(日次)
 4 # 業務名         : 連絡アラート
 5 # プログラム名   : 連絡アラートＬＶ３公開テーブルの配信(日次)
 6 # 備考(Usage)    : DATAMASTER.LOOP.ALERT_TSUCHI_RCV [YYYYMMDD]
 7 # シェル名       : DATAMASTER.LOOP.ALERT_TSUCHI_RCV
 8 # 作成日         : 2013/09/06
 9 # 会社名         : USP
10 # 作成者         : Y.Shinmi
11
12 #////////////////////////////////////////////////////////////////////////////
13 # 初期設定
14 #////////////////////////////////////////////////////////////////////////////
15
16 # 走行ログの記録
17 logd="${HOME}/LOG"                                        # ログディレクトリ
18 logf="${logd}/LOG.$(basename $0).$(date +%Y%m%d)_$(date +%H%M%S)_$$"  # ログファイル名
19 echo  "${logf}" > /dev/null 2>&1
20 log 2> ${logf}                                           log は ush 専用コマンド。exec の
21                                                          代わりに走行ログを出力する。
22 # パスの定義
23 PATH=/home/UTL:/home/TOOL:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin:${PATH}
24 LANG=ja_JP.UTF-8
25
26 #-----------------------------------------------------------------------------
```

次ページへ続く→

```
27 # 変数の定義
28 #-------------------------------------------------------------------------------
29 tmp=/tmp/$$-$(basename $0)_$(date +%Y%m%d%H%M%S)          # 一時ファイル
30 hostname="$(HOSTNAME)"                                   # TODO サーバー名
31 semd="${HOME}/SEMAPHORE"                                 # セマフォディレクトリ
32 shld=${HOME}/SYS
33
34 sday=$(date +%Y%m%d)   # デフォルト
35
36 # 引き数の確認
37 [ $# -eq 1 ] && sday=$1
38
39 limitday="${sday}"                                       # 同期期限日
40 limithhmm="2358"                                         # 同期期限時分
41
42 # エラー時の終了処理定義（ushエラーハンドラ）
43 err.ERROR_EXIT(){
44   touch ${semd}/$(basename $0).${hostname}.ERROR.${sday}
45   echo "${hostname} $(basename $0)_${sday} ERROR $(date +%Y%m%d%H%M%S) ${logf}" >> ${logd}/UPCNT
46   exit 1
47 }
48
49 # 前回セマフォの消去
50 rm -f ${semd}/$(basename $0).${hostname}.*.${sday}  >/dev/null 2>&1
51
52 # 起動時刻の記録
53 echo "${hostname} $(basename $0)_${sday} START $(date +%Y%m%d%H%M%S)" >> ${logd}/UPCNT
54 touch ${semd}/$(basename $0).${hostname}.START.${sday}
55
56 #////////////////////////////////////////////////////////////////////////////
57 # データ処理部
58 #////////////////////////////////////////////////////////////////////////////
59
60 set +x
61 while true ; do
62
63   #-------------------------------------
64   #
65   # 終了条件判定部
66   #
67   #-------------------------------------
68
69   # 開始セマフォのSTART -> STOP とすると強制終了
70   if [ -e ${semd}/$(basename $0).${hostname}.STOP.${sday} ] ; then
71     mv ${semd}/$(basename $0).${hostname}.STOP.${sday} ¥
72        ${semd}/$(basename $0).${hostname}.STOP_DONE.${sday}
73     break
74   fi
75
76   # 終了時刻がきたら終了する
77   [ $(date +%Y%m%d%H%M) -gt ${limitday}${limithhmm} ] && break
78
79   # 業務閉塞の確認
80   if [ ¥( -e ${semd}/DATAMASTER.DAY.ALERT_TSUCHI_RCV.${hostname}.START.${sday} ¥) ¥
81     -a ¥( ! -e ${semd}/DATAMASTER.DAY.ALERT_TSUCHI_RCV.${hostname}.END.${sday} ¥) ¥
82     -a ¥( ! -e ${semd}/DATAMASTER.DAY.ALERT_TSUCHI_RCV.${hostname}.ERROR.${sday} ¥) ] ; then
83         sleep 60
84         continue
85   fi
```

シェル変数の定義について
・グローバル変数なので、上書き防止のためにも初期化場所は基本的にシェルスクリプトの冒頭にする。
・大文字にするとシェルの予約変数（HOME、USER、SHELL など）と混合する懸念があるので小文字にする。
・命名ルールとしてディレクトリ名は“d”で終わる名称にする。
semd="${HOME}/SEMAPHORE"
shld=${HOME}/SYS

err 関数 {}{} は、エラーが発生すると自動的に呼び出される。ush 独自の仕様である。

外部からこの常駐シェルスクリプトを停止させることができる。

この常駐型プログラムは 1 日 1 回起動し、終了時刻になると終了する。

[2] 処理 [1] を継続して実行するかどうかの判定処理（70 ～ 85 行目）

[1] 常駐型プログラムの実行範囲（61 ～ 104 行目）

業務サーバーのステータスをチェックしている。

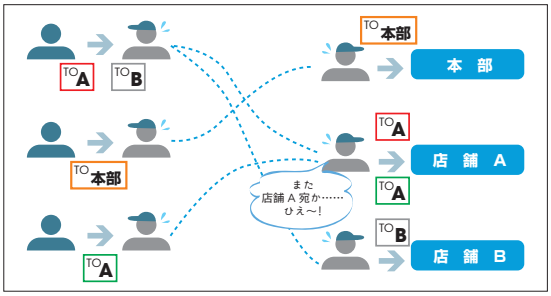
```
 86
 87   #------------------------------------
 88   #
 89   # 処理起動
 90   #
 91   #------------------------------------
 92
 93 # 送信済みデータ集積
 94 ${shld}/LV2MAKE.TIMES.ALERT_TSUCHI ${sday}
 95
 96 # LV3データ作成
 97 ${shld}/LV3MAKE.TIMES.ALERT_TSUCHI ${sday}
 98
 99 # LV3データのサーバー間送信
100 ${shld}/DISTRI_LV3TBL.TIMES.ALERT_TSUCHI ${sday}
101
102   sleep 300
103
104 done
105
106 set -x
107
108 #////////////////////////////////////////////////////////////////////////
109 # 終了処理
110 #////////////////////////////////////////////////////////////////////////
111 # 終了時刻の記録
112 echo "${hostname} $(basename $0)_${sday} END $(date +%Y%m%d%H%M%S)" >> ${logd}/UPCNT
113 touch ${semd}/$(basename $0).${hostname}.END.${sday}
114
115 # 終了
116 rm -f ${tmp}-*
117 echo "$(basename $0) exit 0"
118 exit 0
```

[3] 定期的に実行する集積、加工、配信処理

技術的な概要

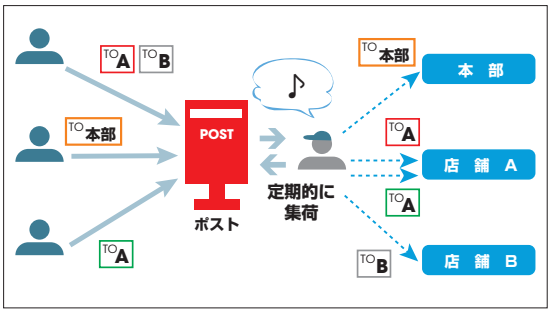
期限がある業務では一般に、締めの直前にトランザクションのピークが発生することで使用されるリソースが逼迫し、すぐにCPUコア数の上限に達します。例えば、特定業務サーバーからの処理にリソースが割かれ、他システムとの連携に影響が及ぶことが考えられます（図3-A）。

図3-A　業務サーバー（左側の人）から送られるデータを逐次、宛先に届ける同期処理のイメージ



そこでシステム負荷が上限に達しないように、データ配信（サーバー間送信）のタイミングを、ある程度、トランザクションデータを集積させてから、まとめて行う非同期処理を活用します。ここで常駐型シェルスクリプトが活躍します（図3-B）（→補足1）。

図3-B　業務サーバーから送られるデータをいったん集約し、配信を効率化させる

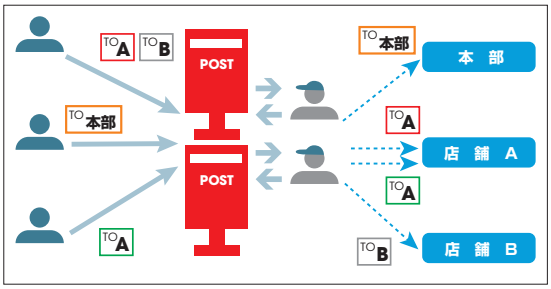


【補足1】常駐型シェルスクリプトは、非同期で動作しているのでリアルタイム性に欠けることがあります（具体的にはおよそ数秒〜数分）。要件仕様においてリアルタイム性が要求される場合は別の手法を選択する必要があります。

常駐型シェルスクリプトは基本的に 1 プロセス（シリアル）で実行されています。つまり、トランザクションが発生した順に処理していくために、仕組みが単純になります。具体的には一定間隔ごとに本スクリプトの処理継続の判定、未処理トランザクションの処理（集積、加工、配信）を行っています。

今回は紹介していませんが、発生する件数が多く、1 プロセスの常駐シェルスクリプトでは処理が追いつかない場合、業務の種類（発注、伝票、その他の業務）ごとにプロセスを分散させて対応します（図 3-C）。分散させるといっても、プロセス数はたかだか数個なのでシステム負荷はさほど大きくはなりません。

図 3-C　増加した業務サーバーの種類にあわせてプロセスを分散させれば負荷を平準化できる



コードの見どころ

今回のコードを前述した 3 つの観点から説明します。

[1] 常駐型プログラムの実行範囲は、while 文が一定間隔で繰り返しループする範囲になります。開始位置は 61 行目、終了位置は 104 行目になります。

[2] 処理を継続して実行するかどうかの判定を 70 ～ 85 行目で行っています。[1] のコードはそのままでは無限ループしますから、終了条件をここで設けています。終了条件の 1 つは外部から終了フラグを立てる場合、もう 1 つは当日のシステム時刻が 23:58 になる場合です（→補足 2）。

【補足 2】開始時刻と終了時刻のパターンとして、朝から当日の夜まで実行させるパターン、朝から翌日の朝まで実行させるパターン（例：午前 5 時～翌日午前 4 時）、24 時間 365 日実行させるパターンなど、いろいろあります。

[3] ループの中で定期的な処理を実行しているコードは、94 ～ 102 行目です。データの「集積」「加工」「配信」の処理を行います。この段階で、データを yyyymmdd 形式（1 日単位）で保存します。随時発生するトランザクションデータを日毎でまとめておくという保持の仕方は、ユニケージ開発手法ではよく用いられます。1 日単位で保存しておくと後日、週単位、月単位で累計したい場合に対応しやすいことが理由です。

まとめ

常駐型シェルスクリプトを採用すると、発生した多数のトランザクションをシーケンシャルに処理していくことになるので、次の効果が得られます（→補足 3）。

- **排他制御が不要になる。**
- **コードが簡単になる。**
- **システム負荷も小さくなる。**

今回は常駐型シェルスクリプトの概要と処理フローのコードを説明しました。次回はこの処理フローの一部である「集積処理」を担当するコードを説明する予定です。

【補足 3】常駐型シェルスクリプトを採用しない場合、cron を使って定期的に処理を実行する方法などがあります。ただし、その方法ではトランザクションの重複処理とプロセスの 2 重起動が起こる可能性がありますので、対処が必要です。