華和梨 Bootcamp Elite
～わんもわせっ～

2008/5/5

華和梨開発チーム
さとー

Site: http://kawari.sourceforge.net
Blog: http://d.hatena.ne.jp/satos/
Mail: shobu@users.sourceforge.net

アウトライン

- 序
  - エントリツリーを活用しよう
- 急
  - タスクマネージャー“KTM”の仕様
- 破
  - タスクマネージャー“KTM”の実装と活用

序

エントリツリーを活用しよう

エントリツリーとは？

- 名前にピリオドを含むエントリ群を、ツリー構造に見立てたもの
- フォルダを“\”で区切るのと同じ考え方
  - エントリ例：System.Request.Reference0
  - フォルダ例：\WINDOWS\system32\user32.dll
- 華和梨は同じエントリツリー内のエントリ群をまとめて扱う機能がある

同じエントリツリーとは

- エントリ名のピリオドより前が共通のエントリ同士は、同じエントリツリーに属する
  - 例:
    System.Request.Reference0 エントリ
    System.Request.Status エントリ
    System.Request エントリ
    ・・・・・・どちらも”System.Request”ツリー内に属する
  - 例外:
    すべてのエントリは”.”エントリツリーに属する

エントリツリーを操作するコマンド

- 以下のコマンドはエントリツリーを操作可能
  - listtreeコマンド
    - あるエントリツリー以下のエントリを再帰的に列挙
  - listsubコマンド
    - あるエントリツリーのエントリを列挙
  - copytreeコマンド
    - エントリツリーを別のエントリツリーにコピー
  - movetreeコマンド
    - エントリツリーを別のエントリツリーに移動
  - cleartreeコマンド・・・・・・
    - エントリツリーを削除

Dosコマンド/ファイルに例えると

- listtree → “dir /S”
  （ディレクトリ以下全ファイル）
- listsub → “dir”
  （ディレクトリ内のファイル）
- copytree → “copy”
  （ディレクトリごとコピー）
- movetree → “move”
  （ディレクトリごとの移動）
- cleartree → “rmdir”
  （内容ごとディレクトリ削除）

エントリツリーの使い道1（構造体）

- 例：キャラクタのHP、MP、所持金をツリーで管理

```
=kis
setstr キャラクタ1.HP 30;
setstr キャラクタ1.MP 40;
setstr キャラクタ1.所持金 2000;
copytree 能力値デフォルト キャラクタ1;
対戦 キャラクタ1 キャラクタ2;
=end
```

- 複数の値（上の例ではデフォルトの能力値群）を一括して扱える

エントリツリーの使い道２（DB）

- 例：ツリー名で生物の分類をDB化

```
生物.脊椎動物網.哺乳類.霊長目.ヒト科.ホモ・サピエンス：さとー
生物.脊椎動物網.哺乳類.霊長目.ヒト科.ホモ・バグエンス：C.Ponapalt

=kis
function 分類検索 $(
    listtree @全種 生物;
    foreach @種 @全種 $(
      if $[ $(find @種 $(getcode @arg[1]) >=0 ] $(
                pushstr @種候補 ${@種};
      );
    );
    return $(join @種候補 “,”);
);
=end
```

- listtreeでインデックスを作成し、findで検索

急

タスクマネージャー“KTM”の仕様

”KTM”とは

- “Kawari Task Manager”の略
- 発動条件・優先関係が複雑に絡み合った「タスク」の実行を管理するスクリプト
- OpenKEEPSでOnSecondChangeイベントの機能実行スケジューリングに採用中

KTM誕生の背景

- OnSecondChangeイベントは派生イベントがある
  - 自発トーク、時報トーク、見切れ反応、重なり反応、ミドルウェア内部管理、etc…
- 派生イベント間には、優先順位関係がある
  - 例：内部＞＞時報＞自発＞見切れ＝重なり
- 優先順位をifで全て書くのは非常に面倒
  - 機能が増える度に条件判断箇所の書き換え発生
  - ミドルウェアはユーザ側まで管理しきれない

→簡単に派生イベントを管理したい！

先行例の調査

- 同様の問題を解決した例は？
  - OSがまさに「発動条件・優先関係が複雑に絡み合ったタスクの実行を管理」している

→華和梨上に「OS」を実装すればいい！
（かなり大変そう・・・）

でも、やるんだよ！

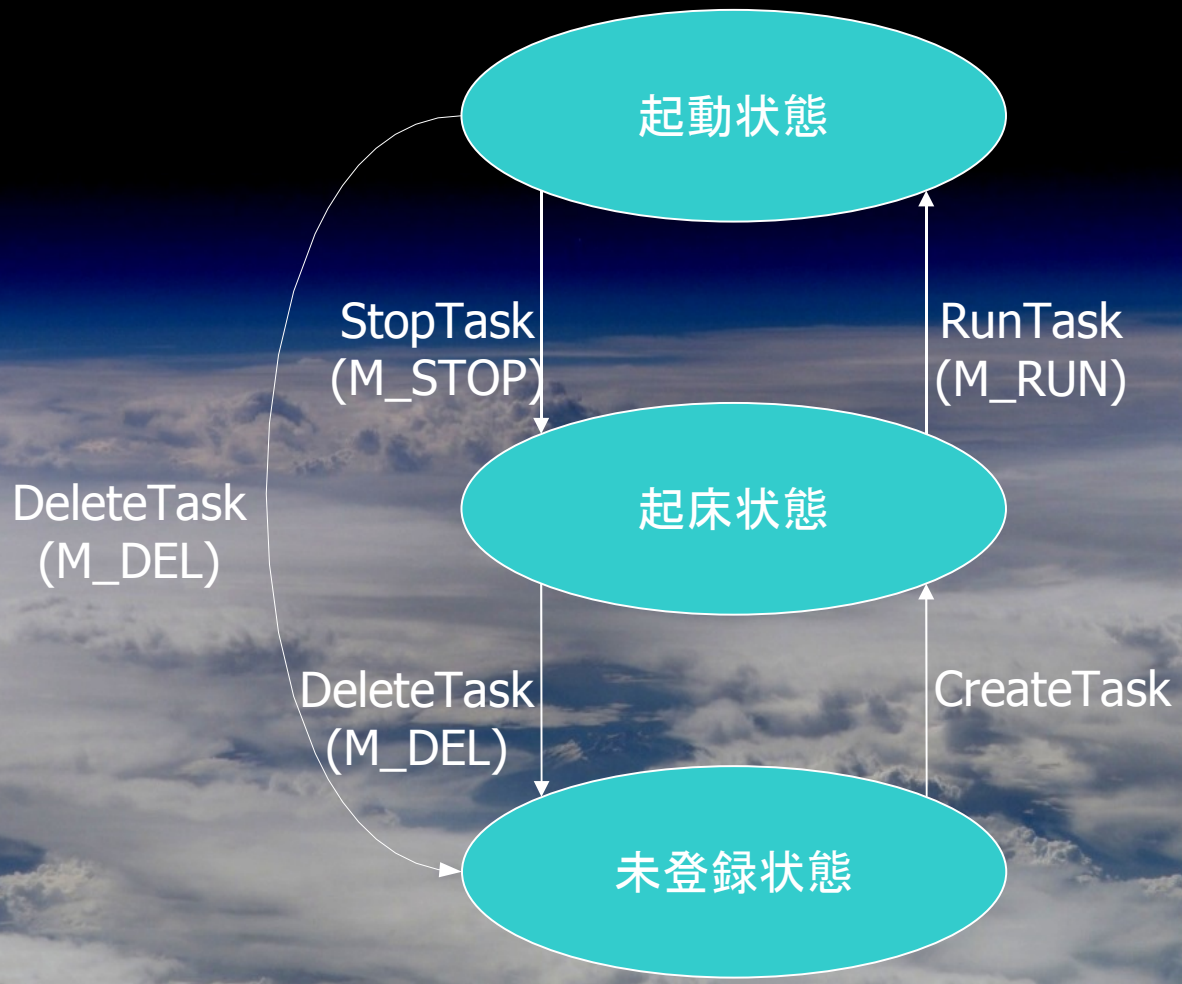
「OS」への要求仕様の整理

- OnSecondChange毎に機能
- タスク毎に、以下の情報を管理すること
  - タスクの発動条件（カウンタが0になったetc…）
  - タスク自体の内容
  - タスクの初期化方法（カウンタを60で初期化etc…）
  - タスクの終了方法（フラグ類を開放etc…）
  - 優先順位
- タスクの実行順序を保証できること（Aタスク後にBタスクを実行を保証できる）
- タスクの優先順位を動的に変更できること
- タスク間で通信する方法があること

先行例を参考にして仕様決定

- T-Kernel(TRON系OS)への要求仕様と似ているため、実装の参考とする
- API名はWin32 API類似のものにする
- タスク毎にID（tid）を発行し、これでタスクを管理する

状態遷移



タスクに設定可能な項目

- タスクの初期化方法（コンストラクタ）
- タスクの起動条件
  - 特定時刻に実行
  - 特定周期で実行
  - 特定条件式が真の時に実行
- タスク本体
- タスクの優先順位
- タスクの終了方法（デストラクタ）

- タスク毎のローカルエントリ

タスクの実行順序保証法

- 以下の機構を提供する
  - Mutex（1個だけロック可能）
  - セマフォ（n個ロック可能）
- Mutex、セマフォはタスクが取得する主体で、tidと名前で管理する。
- ロックに失敗したらタスクが実行を諦めることで機能する、擬似Mutexである。実質的にフラグ同等。
  - 華和梨自身にOSのMutex弄る機能とマルチタスク機能がないので…

タスク間通信

- メールボックス機能を提供する
- タスク単位ではなく、汎用のメールボックス
- タスク間通信で使用する場合、タスク間でメールボックス名称を事前に設定しておく
- 投函時刻、投函タスク、投函内容を取得できる

- APIの一部も、システムへのメールで実行できるようにする（システムへのメール＝システムメッセージ）

サービス機能

- 登録済みタスク一覧
- 起動済みタスク一覧
- タスクの優先順位変更
- タスクの優先順位参照
- タスクの実行状態をセーブ（コアダンプ）

破

タスクマネージャー“KTM”の実装と活用

実装方針

- 全てを”KTM.”以下のエントリツリーに実装
  - KTM.task.[tid]: タスク
  - KTM.mutex.[Mutex名]: Mutex
  - KTM.semaphore.[セマフォ名]: セマフォ
  - KTM.mbox.[メールボックス名]: メールボックス
- この構成により、タスク・Mutex等を一つの塊（オブジェクト）として操作可能

OnSecondChangeでのタスク実行

- 事前に優先順位別の起動状態のタスクリストを持っておく
- 優先順位が高い順に、各タスクの起動条件を評価し、条件が真ならタスク実行
- 全タスクを評価後、SYSTEMメールボックスに来ているシステムメッセージを順に実行

OnSecondChange実行用のコード（抜粋）①

```
# 優先順位別に、条件評価と実行
loop 5 $(
        .setstr @tasklist KTM.activetask.rank${-1};
        if $(.size ${@tasklist}) $(
                foreach KTM.tid ${@tasklist} $(
                        # cond節を評価し、真だった場合はproc節を実行
                        if $(.get ${KTM.task.${KTM.tid}.cond}) $(
                                .get ${KTM.task.${KTM.tid}.proc};
                        );
                );
        );
);

# システムへのメッセージ処理
while $(KTM.GetMailNo "SYSTEM") $(
        .clear @msg;
        .split @msg $(KTM.GetMailMessage "SYSTEM") " ";
        if $(.size "KTM.SYSMAIL."$@msg[0]) $(
                .setstr KTM.sysmailparam $@msg[1];
                .setstr KTM.tid $(KTM.GetMailSender "SYSTEM");
                .get "KTM.SYSMAIL."$@msg[0];
        );
        KTM.DeleteMail "SYSTEM";
);
```

優先順位別
起動中タスクリスト

タスクの
起動条件

タスク本体

システムメッセージ用メ
ールボックス

この場合、**tid**が入る

システムメッセージ
実行用エントリ群
呼び出し
（②参照）

OnSecondChange実行用のコード（抜粋）②

```
# 処理するメッセージの実行部
KTM.SYSMAIL.M_RUN      : $(KTM.RunTask ${KTM.sysmailparam})
KTM.SYSMAIL.M_STOP     : $(KTM.StopTask ${KTM.sysmailparam})
KTM.SYSMAIL.M_DEL      : $(KTM.DeleteTask ${KTM.sysmailparam})
KTM.SYSMAIL.M_LCK_MTX  : $(KTM.LockMutex ${KTM.sysmailparam})
KTM.SYSMAIL.M_RLS_MTX  : $(KTM.ReleaseMutex ${KTM.sysmailparam})
KTM.SYSMAIL.M_GET_SEM  : $(KTM.GetSemaphore ${KTM.sysmailparam})
KTM.SYSMAIL.M_RLS_SEM  : $(KTM.ReleaseSemaphore ${KTM.sysmailparam})
KTM.SYSMAIL.M_EXE_PRC  : $(.get ${KTM.sysmailparam})
KTM.SYSMAIL.M_EXE_SELF : $(.get ${KTM.task.${KTM.tid}.proc})
```

この場合は**tid**が入る

システムメッセージ

KTMの効果

- 独自関数によるイレギュラーな処理がなくなった
- OnSecondChangeでの発話を、すべてミドルウェアで統一的に把握・管理できるようになった
- より柔軟なトーク制御が可能になった
  - バルーンにトーク表示中に他イベントが発生した場合、トーク表示完了5秒後に割り込みイベントのトークを表示
  - inductionモード等のロック解除し忘れによるフリーズを、自己診断して回避

KTMの採用例

- OpenKEEPS
  - 自発トーク、時報、見切れ・重なり等で使用
- ゴースト「言葉」in "Orangewords::"
  - Web上からの定期的情報取得等で使用
- ゴースト「晶子＆みたぺん」in "私設華和梨応援団"
  - 怒りフラグの一定時間以後の解除に使用

KTMの使用例 - フリーズ回避（1）(OpenKEEPS)

```
# 起動後、ずっとフリーズしっぱなしになることを回避するタスク
# フリーズ監視タスクを起動
System.Callback.OnLoad : $(
        KTM.RunTask $(
                KTM.CreateTask
                "FreezeCheck"
                KTM.TRUE
                task.freezecheck.proc
                KTM.RANK4
                task.freezecheck.init
        )
)
# タスクの初期化
task.freezecheck.init : $(
        if $[ $(GetInteger closemode) == 3 ] $(
                # 栞リロード後なのが明らかなので、すぐにフリーズを解除する
                .setstr $(KTM.Local counter) 58;
        ) else if $[ $(GetInteger closemode) == 2 ] $(
                # インストールかネットワーク更新で再起動した
                InheritCond;
                .dec count.boot 1 0;
        ) else $(
                .setstr $(KTM.Local counter) 0;
        );
)
```

タスク生成→即起動の場合、**CreateTask**の戻り値を**RunTask**に与えるのが常道

タスク本体等は、エントリ名を与えることで指定する

Local関数を使うと、タスク固有のローカルエントリ名が得られる

KTMの使用例 - フリーズ回避（2）(OpenKEEPS)

```
# タスク本体
task.freezecheck.proc : $(
	if $[ $(isNotFreezing) || $(KTM.GetMailNo FreezeCheck) ] $(
		# メールボックスに通知がきたら、内容問わず「フリーズしてない」と見る
		KTM.PostMail "SYSTEM" "M_STOP" ${KTM.tid};
		# 再度タスクが起床するのはキャッシュから出た時と推定できる
		# それに備え、タスク状態を初期化する
		.setstr $(KTM.Local counter) 0;
		while $(KTM.GetMailNo FreezeCheck) $(KTM.DeleteMail FreezeCheck);
		return;
	);
	.inc $(KTM.Local counter);
	if $[ ${$(KTM.Local counter)} == 60 ] $(
		# 起動から60秒間フリーズしっぱなし、又はリロード後だった
		# フリーズ解除と起動イベント相当の処理を行う
		resetFreeze;
		InheritCond;
		return;
	);
)
```

"FreezeCheck"メールフォルダを、他のタスクとの通信用に設定

"SYSTEM"メールフォルダ経由で、自タスクの停止を依頼

Local関数で得られるエントリは、タスク固有でかつ、タスク呼び出しを超えて内容が持続する

最後に

- KTMには、他にもできることがたくさんあります
  - キャラクタの感情や体調の実装
    - 徐々に怒りが収まる
    - 次第に空腹になる
    - 撫でられた後だと感じやすくなる
  - ネットワーク等の状態監視
    - 定期的に監視して、希望の状態になっていたらイベントを発生させる
- 色々応用できるので、チャレンジしてみましょう

御清聴ありがとうございました

Copyright

- 背景画像 ”**A Blue Crescent Moon from Space**“ (NASA)
  - ・・・パブリックドメイン（非営利の場合）
- 上記以外の本資料の内容
  - ・・・Creative Commonsライセンスの「表示 2.1 日本」と同じ扱いとします

参考文献

- プログラマブル準AI 華和梨
  - http://kawari.sourceforge.net/
- OpenKEEPS
  - http://keeps.sourceforge.jp/
- KTM、ゴースト「晶子＆みたぺん」
  - http://shobu.hp.infoseek.co.jp/
- ゴースト「言葉」 - Orangewords::.
  - http://orangewords.dw.land.to/