BREW & EZアプリ
ゲームプログラミング

○　第７回　イベントの処理　○

イベント

BREW実行環境は、実行中のアプリケーションを監視しています。キーが押されたり、アプリケーションが中断されたりといったイベントが発生すると、該当するアプリケーションにイベントが起きたことを通知します。各アプリケーションは、送出されてきたイベントを正しく処理しないと、強制終了させられてしまうことがあります。

イベント	発　生　条　件
EVT_APP_START	アプリケーションが起動されたとき(AEEClsCreateInstance関数の実行後)
EVT_APP_STOP	アプリケーションが終了を要求したとき
EVT_APP_SUSPEND	アプリケーションの実行が一時中断されようとしているとき
EVT_APP_RESUME	中断されたアプリケーションが再開されようとしているとき
EVT_KEY_PRESS	キーが押されたとき
EVT_KEY_RELEASE	キーが離されたとき

▲おもなイベントと発生条件

イベントハンドラ

BREWアプリケーションは、イベントハンドラと呼ばれる特別な関数を準備しなければなりません。この関数は、BREW実行環境から送出されてくるイベントを処理するための関数です。*プロジェクト名*_HandleEvent関数が該当します。

*プロジェクト名*_HandleEvent関数は、以下のように送出されてきたイベントをswitch文で分岐させ、それぞれの処理を行います。イベントは十数種類ありますが、すべてのイベントを処理する必要はありません。必要のないイベントの場合は、FALSEを返してイベントハンドラから抜けます。イベントを処理した場合は、決められた戻り値を返します。

```
// イベントハンドラ(イベントを処理する関数)
static boolean プロジェクト名_HandleEvent(プロジェクト名* pMe, AEEEvent eCode,
                                          uint16 wParam, uint32 dwParam)
{
    switch(eCode) {
    case EVT_APP_START:
        ※アプリケーションが起動されたときの処理(変数の初期化など)
        return TRUE;

    case EVT_APP_STOP:
        ※アプリケーションを終了するときの処理
        return TRUE;

    case EVT_KEY_PRESS:
        ※キーが押されたときの処理
        return TRUE;

    default:
        break;
    }

    return FALSE;
}
```

イベントハンドラは、アプリケーションが何も処理していないときだけ実行されます。イベントの処理中だったり、タイマで指定された関数を実行中のときは、発生したイベントは処理待ち状態になります。処理待ち状態のイベントが一定時間処理されなかったり、一定数を超えた場合、BREW実行環境は、そのアプリケーションを強制終了させてしまいます。したがって、イベントやタイマ起動時の処理は、あまり長時間行うことはできません。目安としては、１秒程度といわれています。

Handle_Event関数

イベントハンドラ「*プロジェクト名*_HandleEvent関数」のプロトタイプは、以下のようになっています。

```
static boolean プロジェクト名_HandleEvent(プロジェクト名* pMe, AEEEvent eCode,
                                          uint16 wParam, uint32 dwParam);
```

最初のstaticは、ほかのソースファイルと関数名が重複してもエラーにならないようにあるだけで、動作が変わるなどの意味はありません。

イベントハンドラが呼び出されると、BREW実行環境から４つの引数が渡されます。それぞれ、以下のような情報が格納されています。

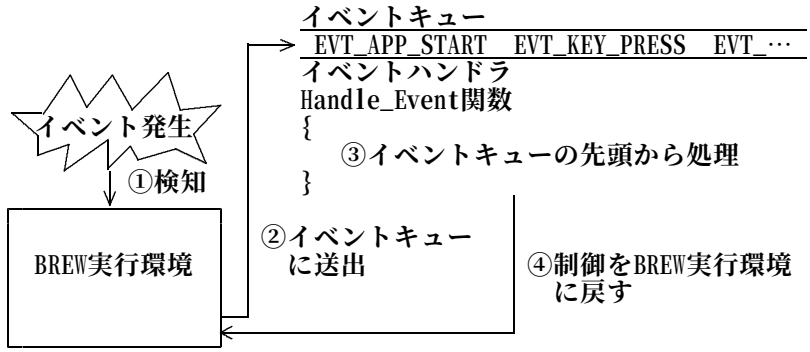
*プロジェクト名**	pMe……アプレット構造体(へのポインタ)
AEEEvent	eCode……送出されたイベントの種類
uint16	wParam …イベントの追加情報。内容はイベントによって異なります
uint32	dwParam…イベントの追加情報。内容はイベントによって異なります

BREWプログラミングは、イベントに応答しながら処理を行うことからイベントドリブン(イベント駆動型)プログラミングと呼ばれています。アプリケーションが独自に動作するのではなく、BREW実行環境から取得したイベントをトリガ(きっかけ)として動作するためです。

このような複雑な構造になっているのは、BREW実行環境が複数のアプリケーションを最適にスケジュールし、安定して協調動作させるためです(Javaは、イベントを処理するプログラムとメイン処理のプログラムを同時に実行できるマルチスレッド型なので、このような制限はありません)。

旧来のMS-DOS(コンソール)などのアプリケーションでは、main関数からプログラムの実行が始まり、プログラマが記述した流れにそってプログラムが実行されます。プログラムの中で文字を表示する必要があれば、そこでprintf関数などを実行することによって文字出力が行われます。MS-DOSアプリケーションでは、プログラマが記述したコードのとおりに処理が進んでいきます。
ところがBREWアプリケーションでは、プログラマが書くコードは、BREW実行環境から必要なときに呼び出されるようになっています。BREWプログラミングの世界では、プログラマはプログラムの流れを取り仕切るのではなく、イベントを受け取ったときやタイマが起動したときの動作という局部的なコードを記述するのです。「プログラムの流れはこうしよう」という考え方ではなく、「このイベントが発生したらこうしよう」という考え方になっています。

このようなシステムからのメッセージに応答するようなプログラム構造は、キーボードやマウスからの入力がない限り処理を行わないワープロやメーラーといったビジネスアプリケーションと非常に相性が良いのですが、常にリアルタイムで動作しなければならないゲームプログラムと相性が良いとはいえません。



▲BREWアプリケーションの基本プログラミングモデル

課　題

変数やデータの初期化および解放を行う関数と、アプリケーションが休止および復帰状態になったときの関数を作成しましょう。

(１)変数やデータの初期化を行う関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
// ゲームデータ初期化
boolean プロジェクト名_InitGameData(プロジェクト名* pMe)
{
    return TRUE;
}
```

※「プロジェクト名」は、作成したプロジェクトの名前にそって置き換えてください

(２)(１)のプロトタイプを宣言します。以下のプログラムを適切な場所に追加しましょう。

```
boolean プロジェクト名_InitGameData(プロジェクト名* pMe);
```

(３)変数やデータの解放を行う関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
// ゲームデータ解放
void プロジェクト名_FreeGameData(プロジェクト名* pMe)
{
}
```

(４)(３)のプロトタイプを適切な場所に作成しましょう。

(５)アプリケーションの起動時に、変数やデータを初期化します。

アプリケーションが使用する変数やデータは、未初期化バグを防ぐため、アプリケーションが起動したときに0やNULLなどで初期化しておきます。
アプリケーションが起動し、AEEClsCreateInstance関数から抜けると、EVT_APP_STARTイベントが発生します。変数やデータは、このイベントで初期化します。

EVT_APP_STARTイベントを処理する以下の関数を完成させ、適切な場所に追加しましょう。

```
// アプリケーション起動
static boolean プロジェクト名_OnAppStart(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    if(※ここは各自考えましょう(pMe) == FALSE)   // 初期化
        return FALSE;

    プロジェクト名_MainLoop(pMe);                  // メインループ
    return TRUE;
}
```

(６)(５)のプロトタイプを宣言します。以下のプログラムを適切な場所に追加しましょう。

```
static boolean プロジェクト名_OnAppStart(プロジェクト名* pMe, uint16 wParam, uint32 dwParam);
```

(７)アプリケーションの終了時に、変数やデータを解放します。

アプリケーションが使用した変数やデータは、アプリケーション終了時に自動的に解放されますが、独自にメモリを確保したり、画像を読み込んだ場合はプログラマが解放しなければなりません。
アプリケーションが終了するときは、プロジェクト名_FreeAppData関数が呼び出される前に、EVT_APP_STOPイベントが発生します。メモリや画像の解放は、このイベントで行います。

EVT_APP_STOPイベントを処理する以下の関数を完成させ、適切な場所に追加しましょう。

```
// アプリケーション終了
static boolean プロジェクト名_OnAppStop(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    ※ここは各自考えましょう;
```

```
    return TRUE;
}
```

(８)(７)のプロトタイプを適切な場所に作成しましょう。

(９)イベントハンドラでEVT_APP_STARTイベントとEVT_APP_STOPイベントが処理されるようにします。
*プロジェクト名*_HandleEvent関数を以下のように変更しましょう。

```
static boolean プロジェクト名_HandleEvent(プロジェクト名* pMe, AEEEvent eCode,
                                          uint16 wParam, uint32 dwParam)
{
    switch (eCode) {
      case EVT_APP_START:   return プロジェクト名_OnAppStart(pMe, wParam, dwParam);
      case EVT_APP_STOP:    return プロジェクト名_OnAppStop (pMe, wParam, dwParam);
   }

   return FALSE;
}
```

(10)アプリケーションの一時中断とその復帰に対応させます。

実行中のBREWアプリケーションは、電話がかかってきたり、メールを着信したりすると、実行が一時中断される場合があります。一時中断された状態では、ゲームを進めることができないので、タイマを停止し、ゲームが進行しないようにします。

アプリケーションが一時中断されたり中断から復帰した場合は、イベントよって通知されます。アプリケーションは、これらのイベントを利用することにより、一時中断とその復帰に対応させることができます。

アプリケーションが一時中断されたときのイベントを処理する以下の関数を完成させ、適切な場所に追加しましょう。

```
// アプリケーション中断
static boolean プロジェクト名_OnAppSuspend(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    // タイマキャンセル
    ISHELL_???????????(pMe->pIShell, プロジェクト名_MainLoop, pMe);

    return TRUE;
}
```

(11)(10)のプロトタイプを適切な場所に作成しましょう。

(12)アプリケーションが中断から復帰したときのイベントを処理する以下の関数を完成させ、適切な場所に追加しましょう。

```
// アプリケーション再開
static boolean プロジェクト名_OnAppResume(プロジェクト名* pMe, uint16 wParam, uint32 dwParam)
{
    // タイマ設定
    ISHELL_????????(pMe->pIShell, 0, プロジェクト名_MainLoop, pMe);

    return TRUE;
}
```

(13)(12)のプロトタイプを適切な場所に作成しましょう。

(14)イベントハンドラでアプリケーションの一時中断とその復帰イベントが処理されるようにします。
以下のプログラムを完成させ、適切な場所に追加しましょう。

```
    case EVT_???_???????: return プロジェクト名_OnAppSuspend(pMe, wParam, dwParam);
    case EVT_???_??????:  return プロジェクト名_OnAppResume (pMe, wParam, dwParam);
```