ゲームプログラミング

▶　DirectInput－第７回　ゲームパッドのデータ取得　◀

ゲームパッドの状態とデバイスバッファは、キーボードやマウスと同じメソッドで取得することができますが、データ取得前にポーリングをしなければなりません。

ポーリング

ポーリングとは、デバイスに新しいデータがないかを調査し、デバイスが最新の入力データを提供するための動作をいいます(正確には、ゲームパッドからデータを取得するために必要なハードウェア割り込みを生成する動作)。ほとんどのゲームパッドは、ポーリング対象デバイスになっており、定期的にポーリングをしなければデバイスからからデータを取得することができません。
ポーリングは、デバイスオブジェクトのPollメソッドで行います。ゲームパッドからデータを取得する前に、必ずこのメソッドを呼び出す必要があります。なお、このメソッドはキーボードやマウスといったポーリングをする必要がないデバイスで呼び出しても、何の効果も害もありません。

```
// ポーリング
LPDIRECTINPUTDEVICE8   lpDIDGamePad;     // ゲームパッドのデバイスオブジェクト(初期化済みとする)
lpDIDGamePad->Poll();
```

ゲームパッドの状態取得

ゲームパッドの状態は、GetDeviceStateメソッドで取得します。このメソッドがゲームパッドから取得する情報は、軸の位置または変化量、ボタンの状態です。これらの情報はDIJOYSTATE構造体またはDIJOYSTATE2構造体に格納されます。
ゲームパッドの状態を取得する場合、GetDeviceStateメソッドの１つ目の引数はsizeof演算子を使って構造体のサイズを指定し、２つ目の引数は状態を格納する構造体変数のアドレスを指定します。

```
// ゲームパッド状態の取得
LPDIRECTINPUTDEVICE8   lpDIDGamePad;     // ゲームパッドのデバイスオブジェクト(初期化済みとする)
DIJOYSTATE             diGamePadState;   // ゲームパッドの状態を格納する構造体変数
lpDIDGamePad->Poll();                    // ポーリング
lpDIDGamePad->GetDeviceState(sizeof(diGamePadState), &diGamePadState);
```

上記のようにGetDeviceStateメソッドを呼び出すと、DIJOYSTATE構造体にゲームパッドの状態が格納されます。この構造体のメンバと格納される情報は以下のようになっています。

```
struct DIJOYSTATE {
    LONG    lX;                   // x軸の傾き
    LONG    lY;                   // y軸の傾き
    LONG    lZ;                   // z軸の傾き
    LONG    lRx;                  // x軸回転
    LONG    lRy;                  // y軸回転
    LONG    lRz;                  // z軸回転
    LONG    rglSlider[2];         // ２つの追加軸
    DWORD   rgdwPOV[4];           // 視点ハットなどの方向コントローラ
    BYTE    rgbButtons[32];       // ボタンの状態
};
```

軸は中心位置をもとにどの方向に傾いているかを判断します。ボタンはキーボードやマウスの場合と同じように、BYTE配列になっており、特定のボタンを指定するにはボタンの番号を添字にします。「押されている」「押されていない」という情報は上位１ビットで判定します。

```
// ゲームパッドのボタン０が押されているかを調べる
if(0 != (diGamePadState.rgbButtons[0] & 0x80))
    // ボタン０は押されている
```

デバイスバッファの取得と消去

ゲームパッドのデバイスバッファに格納された情報の取得と消去は、マウスやキーボードと同じですが、GetDeviceDataメソッドを呼び出す前にPollメソッドを呼び出しておきます。

○ 課題 ○

ゲームパッドからデータを取得する関数を作成しましょう。

(１)以下のプログラムは、ゲームパッドの状態を取得するDIGamePadState関数です。関数の仕様をよく読み、プログラムを完成させましょう。

DIGetGamePadState関数
－説明－
DIGetGamePadState関数は、ゲームパッドの状態を取得し、指定された領域に格納します。

－パラメータ－
const DIDEV did…状態を取得するゲームパッド。DIDEV列挙型(DID_GAMEPAD1など)を指定
DIJOYSTATE& diGamePadState…ゲームパッドの状態を格納するDIJOYSTATE構造体変数

－戻り値－
取得に成功した場合はtrue、それ以外はfalseを返す。

```
/*****************************************************************************************************/
/*                                          ゲームパッド状態取得                                       */
/*****************************************************************************************************/
bool DIGetGamePadState(const DIDEV did, DIJOYSTATE& diGamePadState)
{
    ZeroMemory(&diGamePadState, sizeof(diGamePadState));
    diGamePadState.lX  = diGamePadState.lY  = diGamePadState.lZ  = DIAXIS_RANGE_CENTER;
    diGamePadState.lRx = diGamePadState.lRy = diGamePadState.lRz = DIAXIS_RANGE_CENTER;

#ifdef _DEBUG
    if(DID_GAMEPAD1 > did) {
        OutputDebugString("*** Error - デバイス指定エラー(DIGetGamePadState)\n");
        return false;
    }
#endif
    if(NULL == g_lpDIDevice8[did]) {
        OutputDebugString("*** Error - ゲームパッド未初期化(DIGetGamePadState)\n");
        return false;
    }

    ※ここは各自考えましょう
    const HRESULT  hr = g_lpDIDevice8[did]->※ここは各自考えましょう(ヒント：ゲームパッドの状態取得)
    if(DI_OK != hr) {
        if(DIERR_INPUTLOST == hr)
            g_lpDIDevice8[did]->Acquire();
        OutputDebugString("*** Error - ゲームパッド状態取得(DIGetGamePadState)\n");
        return false;
    }

    return true;
}
```

(２)DIGetDeviceBuffer関数をゲームパッド対応に変更しましょう。

ヒント：指定されたデバイスがゲームパッドの場合は、バッファ取得前にポーリングを行います。

(３)DIFlushDeviceBuffer関数をゲームパッド対応に変更しましょう。

(４)キャラクターをゲームパッドで移動してみましょう。