オブジェクト指向と ゲームプログラミング

▶ DirectInput編 － 第７回　ゲームパッドのデータ取得 ◀

ゲームパッドのデータ取得

ゲームパッドの状態やデバイスバッファの取得は、キーボードやマウスと同じメソッドで行うことができますが、データ取得前にポーリングをしなければならない場合があります。

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、デバイス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のことです)。いくつかのゲームパッドは、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、IDirectInputDevice8::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、このメソッドを呼び出すしておきます。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、何の効果も不具合もありません。

```
// ポーリング(pDIDGamePadは、初期化済みのゲームパッドのデバイスオブジェクト)
pDIDGamePad->Poll();
```

ポーリングが必要であるかどうかは、Pollメソッドの戻り値で調べることができます。Pollメソッドが「DI_NOEFFECT」を返したときは、ポーリングが不要であることを表します。

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
DIJOYSTATE   GamePadState;      // ゲームパッドの状態を格納する構造体変数
pDIDGamePad->Poll();            // ポーリング
pDIDGamePad->GetDeviceState(sizeof(GamePadState), &GamePadState);
```

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 != (GamePadState.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ようにGetDeviceDataメソッドで行いますが、この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機能を追加しましょう。

(１)以下のプロトタイプをCGamePadクラス定義の適切な場所に追加しましょう。

```
    virtual bool GetState(LPVOID pState);
    virtual DWORD GetBuffer(DIDEVICEOBJECTDATA outDIData[], const DWORD inElements);
    virtual void FlushBuffer();
```

(２)CGamePadクラスの各関数を実装します。以下のプログラムの足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          状態取得                                                 */
/*****************************************************************************************************/
bool CGamePad::GetState(LPVOID pState)
{
    m_pDIDevice->????();
    return CInputDevice::????????(sizeof(DIJOYSTATE), pState);
}

/*****************************************************************************************************/
/*                                          バッファ取得                                             */
/*****************************************************************************************************/
DWORD CGamePad::GetBuffer(DIDEVICEOBJECTDATA outBufData[], const DWORD inElements)
{
    m_pDIDevice->????();
    return ???????????::?????????(outBufData, inElements);
}

/*****************************************************************************************************/
/*                                          バッファ消去                                             */
/*****************************************************************************************************/
void CGamePad::FlushBuffer()
{
    m_pDIDevice->????();
    CInputDevice::???????????();
}
```

(３)キャラクタをゲームパッドで移動してみましょう。

(４)キーボードとゲームパッドの両方でキャラクタが移動できるようにプログラムを変更しましょう。