マウスイベント

オーバーライドによるイベント処理

マウスは、今やパーソナルコンピュータの入力装置の必需品となっています
Windows はマウスが無くても操作できるようになっていますし
アプリケーションは、マウスが無くても制御できるように設計されなければなりません

しかし、ペイントソフトなどではポインティングデバイス無しには使えないでしょうし
多くのエンドユーザーはキーボードよりマウスを使うことに慣れているのが現状です
一般の GUI アプリケーションは、マウスの入力を上手に処理しなければなりません

.NET アプリケーションのイベントシステムは、極めて優れた設計になっています
筆者が知る限りでは、これ以上に優れたシステムはみたことがありません
Win32 API や Java、MFC よりも柔軟で、理想的な構造になっているのです

この柔軟性は**2つのイベント処理方法**を持つことで実現されています
それは、**デリゲートとオーバーライド**の2通りの処理です
まずは、このうちのオーバーライドを用いた実現方法から解説しましょう

OnPaint() メソッドは、無効矩形が作られると呼び出されるというイベントの一つです
これは Control クラスで定義されていて、オーバーライドすることによって
その機能を拡張し、イベント発生時に処理するコードを容易に指定することができます
その他のイベントも、これと同じ考え方で On***() という名前のメソッドで定義されています

この方法は非常に簡単にコントロールの機能を拡張する手段として使えます
メソッドをオーバーライドすれば、後は望み通りのタイミングで
システムがメソッドを呼び出してくれるので、面倒な手続きは一切ありません

マウスカーソルがコントロールのクライアント領域に侵入すると
Control.OnMouseEnter() メソッドが呼び出され
カーソルがコントロールの外に出ると **Control.OnMouseLeave()** が呼ばれます

protected virtual void OnMouseEnter(EventArgs e);
protected virtual void OnMouseHover(EventArgs e);

このメソッドをオーバーライドすることによって、それぞれのイベントを処理できます
e には、このイベントの情報を表す EventArgs オブジェクトが指定されます

パラメータ型は **System.EventArgs** クラスです
このクラスは、イベント発生時にメソッドに伝える情報をパッケージ化する基本クラスです

public class EventArgs

コンストラクタもデフォルトのものしか定義されておらず、特に説明することはありません
このクラスは、イベントメソッドのパラメータのルートクラスとして定義されています

マウスのボタンが押されると **Control.OnMouseDown()** メソッドが
離されると **Control.OnMouseUp()** メソッドが呼び出されます
これによって、マウスクリックを処理することができるようになります

protected virtual void OnMouseDown(MouseEventArgs e);
protected virtual void OnMouseUp(MouseEventArgs e);

e には、マウスの情報をパックした MouseEventArgs オブジェクトが指定されます
このクラス型については、すぐ後で詳しく説明します

```
using System.Windows.Forms;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }
 override protected void OnMouseDown(MouseEventArgs e) {
  Text = "Event = MouseDown";
 }
 override protected void OnMouseUp(MouseEventArgs e) {
  Text = "Event = MouseUp";
 }
 override protected void OnMouseEnter(System.EventArgs e) {
  Text = "Event = MouseEnter";
 }
 override protected void OnMouseLeave(System.EventArgs e) {
  Text = "Event = MouseLeave";
 }
}
```

このプログラムは、各マウスイベントに応じて
発生したイベントを表す文字列をウィンドウテキストに設定します

マウスイベントが発生した時に、マウスの情報をメソッドに届ける役割を持つのが

System.Windows.Forms.MouseEventArgs クラスです

```
Object
   EventArgs
      MouseEventArgs

public class MouseEventArgs : EventArgs
```

このクラスは、次のようなコンストラクタを定義しています

```
public MouseEventArgs(
 MouseButtons button , int clicks ,
 int x , int y , int delta
);
```

このコンストラクタの各引数こそ、マウスの状態を表す情報です
button には、押されたマウスボタンを表す MouseButtons 列挙型の値を
clicks にはマウスボタンのクリック回数を指定します

x と y には、イベント発生時のマウスの X 座標と Y 座標を
delta はホイールマウスのホイール回転量を表す値を指定します
このコンストラクタを使って明示的に MouseEventArgs 型インスタンスを作成すれば
システムではなくプログラムから、意図的にイベントを発生させることも可能です

マウスのボタンは System.Windows.Forms.MouseButtons 列挙型で表されます

public enum MouseButtons

この列挙型は、次のような意味のあるメンバを定義しています

メンバ	解説
Left	マウスの左ボタンが押された
Middle	マウスの中央ボタンが押された
None	マウスのボタンは押されなかった
Right	マウスの右ボタンが押された
XButton1	ファースト X ボタンが押された (Microsoft インテリマウス専用)
XButton2	セカンド X ボタンが押された (Microsoft インテリマウス専用)

MouseEventArgs クラスは、各種の情報を取得できるプロパティを公開しています
イベント発生時に押されたボタンは MouseEventArgs.Button で、
クリック数は MouseEventArgs.Clicks プロパティ、
ホイールの回転量は MouseEventArgs.Delta プロパティ、
座標は MouseEventArgs.X と MouseEventArgs.Y で得られます

public MouseButtons Button {get;}
public int Clicks {get;}
public int Delta {get;}
public int X {get;}
public int Y {get;}

イベントを処理する時にこれらのプロパティを用いれば
ボタンが押された時の座標に対する処理などを行うことができるようになります

マウスがコントロールの上で移動すると Control.OnMouseMove() が発生します
このイベントと、先ほどのマウスの情報を使ってマウスの位置を表示させてみましょう

protected virtual void OnMouseMove(MouseEventArgs e);

e に、イベント発生時のマウス情報を持つ MouseEventArgs オブジェクトが指定されます
ここから、X と Y プロパティを使ってマウスの座標を取得します

```
using System.Windows.Forms;

class WinMain : Form {
 public static void Main(string[] args) {
  Application.Run(new WinMain());
 }
 override protected void OnMouseMove(MouseEventArgs e) {
  Text = "X = " + e.X + " : Y = " + e.Y;
 }
}
```

このプログラムは、ウィンドウのタイトルバーにカーソルの座標を表示します
カーソルの座標は、コントロールのクライアント座標であることに注意してください

デリゲートによるイベントの実装

オーバーライドによるイベントの処理は、極めて簡単に実現する手段ですが
イベントを処理するために、元のコントロールを継承する必要があります
また、On**() メソッドは protected なので外部からアクセスすることもできません

しかし、実は高度なアプリケーションとなるとイベントは単一コントロールに終わりません
Adobe 社の Photoshop 等の高度なペインとソフトを例に挙げると
ユーザーがタブレットで、キャンバスに絵を描くと、当然そのウィンドウはイベントを処理します
そして、表示されているCGは描画イベントを発生させてビットマップを再度表示するでしょう

ただし、Photoshop はこの他にプレビューウィンドウやレイヤーウィンドウ
ヒストリーなどのツールウィンドウがあり、これもイベントに合わせて更新する必要があります
ところが、On***() でこれを作ると考えると、入力されたウィンドウはイベントが発生しますが
その他のツールウィンドウにも、このイベントを通知するとなると様々な工夫が必要です

Java は MVC アーキテクチャやインターフェイスを駆使してイベントの登録などを行い
コントロール(コンポーネント)がこれを適切に処理して呼び出す必要がありました

.NET の場合は、デリゲートとイベントという素晴らしい機能が存在するため
イベントを受けるコントロールに、呼び出してほしいメソッドを要求できます
すばらしいことに、このメソッドはインスタンスにこだわることはありません

Control クラスは、On**() メソッドに対応した公開インスタンスイベントを定義しています
イベントの名前は、On**() メソッドの On を省略した名前に統一されています
例えば　OnPaint() メソッドに対して **Control.Paint** イベントが定義されています

マウスイベントであれば、**Control.MouseDown**、**Control.MouseUp**、
Control.MouseEnter、**Control.MouseLeave**、
そして、**Control.MouseMove** イベントという形で定義されています

public event MouseEventHandler MouseDown;
public event MouseEventHandler MouseUp;
public event EventHandler MouseEnter;
public event EventHandler MouseLeave;
public event MouseEventHandler MouseMove;

イベント発生のタイミングは、On**() というときの場合と同じです
マルチキャストが可能なため、イベントに対して簡単にプラグインを導入できます

イベントの型は **System.Windows.Forms.MouseEventHandler** と
System.EventHandler デリゲート型であり、次のように定義されています

```
public delegate void MouseEventHandler(
 object sender , MouseEventArgs e
);
public delegate void EventHandler(
 object sender , EventArgs e
);
```

sender にはこのイベントの送り主、すなわち発生元のコントロールが
e には、On***() メソッドのパラメータ同様に、イベント情報を持つオブジェクトが指定されます
イベントの送り主が確定しているのであれば、アップキャストして重要なメンバに
外部のメソッドからアクセスするという事が可能になります

```
using System.Windows.Forms;

class WinMain : Form {
 public static void Main(string[] args) {
  WinMain win = new WinMain();
  win.MouseMove += new MouseEventHandler(_MouseMove);
  Application.Run(win);
 }

 static protected void _MouseMove(object obj , MouseEventArgs e) {
  ((Control)obj).Text = "X = " + e.X + " : Y = " + e.Y;
 }
}
```

このプログラムは、先ほどの OnMouseMove() を処理するサンプルプログラムを参考に
まったく同じ処理を、Control が提供するイベントメンバを使って実現したものです
.NET のデリゲートは、メソッドのインスタンスにこだわらないという特徴を持っているため
このように静的なメソッドでコントロールのイベントを受け取るという事が可能です

ここで重要なのは、イベントメンバはシステムではなくプログラムから呼ばれます
実は、イベントメンバを呼び出しているのは Control クラスの On***() メソッドです
そのため、On***() メソッドをオーバーライドしてしまうと、イベントが呼び出せません

そこで、通常は On***() イベントをオーバーライドした場合
処理の最後に基底クラスの On***() メソッドを呼び出すようにしましょう
