イベント

事象駆動

C# は、これまでの言語と異なり言語仕様でイベントをサポートします
これは、過去の言語には見られない稀な性質です

他の一般的な言語では、イベントの登録や解除、メッセージの送信などは
インターフェイスを通してメソッドを呼び出す方法などが考えられるでしょう
しかし、C# 言語はこれらの動作をまとめる方法を提供しています

イベントとは主に GUI システムで用いられるイベント駆動型を指します
これまで、コンソールプログラムを中心にプログラムしてきたので
コンソールプログラムしか経験のない方には、いささか理解し難いかもしれません
ですが、このイベント機能を覚えれば、C# の GUI プログラムを学習する時に役に立つでしょう

C# のイベントは、デリゲートを発展させた機能です
イベントを宣言するには **event** キーワードを用います

[attributes] [modifiers] event type declarator;

attributes には属性、modifiers には修飾子
type にはイベントの型、declarator には変数宣言子群を指定します

イベントの型は、**必ずデリゲート型**でなければなりません
修飾子には new、static、virtual、override、abstract、及びアクセス修飾子を指定できます

イベントメンバは、ほぼ**デリゲート型変数と同じ扱い**ができます
イベントメンバを左オペランドにして += 演算子でデリゲートを加算すれば
マルチキャストデリゲートとして、指定デリゲートをイベントに追加できます

では、そもそもデリゲートとして扱えるならばイベントの意義とはなんでしょう？
実は、イベントはデリゲートと違って**呼び出しに制限**があるのです
デリゲート型メンバ変数は、公開することによってコールバックメソッドの登録ができますが
他のクラスから**直接操作される危険性**もあり、これはオブジェクト指向に反します
イベントの発生によるコールバックメソッドの呼び出しは、そのオブジェクトが行うべきです

イベントメンバは、そのクラスの内部ではデリゲートのように扱えますが
外部からは **+=か -=の左辺**にしか指定できないという制限があり
これによって、外部コードがデリゲートにアクセスするのを防ぐのです
当然、この理想上 += や -= の演算結果は代入されたデリゲートに問わず void を返します

```
delegate void KittyCallback(string str);

class Kitty {
 private string str;
 public event KittyCallback KittyListener;
 public Kitty(string str) {
  this.str = str;
 }
 public void StartEvent() {
  KittyListener(str);  //Kitty内からは呼び出せる
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty("Kitty on your lap");
  obj.KittyListener += new KittyCallback(Action);
  //obj.KittyListener();  //呼び出せない
  obj.StartEvent();
 }
 static void Action(string str) {
  System.Console.WriteLine(str);
 }
}
```

このプログラムは、イベントの発生をコンソールで再現しているものです
Main() メソッドは Kitty のインスタンスを生成し、イベント KittyListener に
Test.Action() メソッドをコールバックメソッドとして登録します

しかし、外部から KittyListener へのアクセスは += または -= に限られます
obj.KittyListener() としてイベントを発生させたくても
外部から直接アクセスできないように設計されているのです
こうすることで、イベントを公開しながら外部コードによるデータの調査を逃れることができます

ちなみに、このようにイベントによって呼び出してほしいメソッドを
一般に**イベントハンドラ**と呼び、通常は独自のスレッドを持ちます
(スレッドについては、後記します)

イベントアクセッサ

イベントの登録や解除の過程に特殊な処理が必要な場合も少なくありません
イベント登録の管理を独自に手動で行う場合は特に必要となります

一般に、イベントの登録はイベント登録用メソッド(例えば addMouseListener())を用い
これによって、コールバックメソッドの管理を行います
しかし、イベント機能を言語仕様でサポートしている C# はより高度な
イベントアクセッサという、より直感的な方法を提供します

イベントアクセッサは、簡単に説明するならばプロパティのイベント版です
+= でイベントを追加された時と -= で解除された時に
自動的に指定のメソッドを呼び出す暗黙的なメソッド実行方法の一つです

[attributes] [modifiers] event type member-name {accessor-declarations};

member-name と accessor-declarations 以外は、先ほどの event の宣言と同じです
member-name にはアクセッサメンバの識別子
accessor-declarations には、イベントアクセッサ宣言子群を指定します

イベントアクセッサ宣言子群は
add アクセッサ宣言 と **remove アクセッサ宣言** があります
それぞれ、宣言する順番は関係ありません

[attributes] add { ...
[attributes] remove {...

attributes には、属性郡を指定することができます
+= で呼び出された場合は add が、-= で呼び出された時は remove ブロックが実行されます
この性質から、イベントアクセッサがプロパティと考え方は同じということがわかるでしょう
MSIL レベルでは、単純にメソッドを呼び出しているだけにすぎないのです

add も remove も、戻り値が void 型でイベント型の value パラメータを受けます
value には += や -= で指定した右オペランドが格納されています

```
using cout = System.Console;
delegate void KittyCallback();

class Kitty {
 private static KittyCallback kitty_listener;
 public static event KittyCallback KittyListener {
  add {
   cout.WriteLine("add : " + value);
   kitty_listener += value;
  }
  remove {
   cout.WriteLine("remove : " + value);
   kitty_listener -= value;
  }
 }
}

class Test {
 static void Main() {
  KittyCallback call = new KittyCallback(Action);
  Kitty.KittyListener += call;
  Kitty.KittyListener -= call;
 }
 static void Action() {}
}
```

このプログラムは、イベントアクセッサ KittyListener を宣言し
Main() メソッドでこれにデリゲート KittyCallback 型の変数 call を渡しています
アクセッサは、暗黙的に value というパラメータでこれを受け取っています

このアクセッサを用いれば、イベントハンドラの登録などを制御し
イベントに応じて独自のアルゴリズムで登録と解除の処理を行えます
しかも、利用者は単純に代入するだけでよいという簡易性を保ったままでです

---

event

[attributes] [modifiers] event type declarator;

イベントを宣言します

attributes - 属性を指定します
modifiers - 修飾子を指定します
type - イベントのデリゲート型を指定します
declarator - 変数宣言子群を指定します

[attributes] [modifiers] event type member-name {accessor-declarations};

イベントアクセッサを宣言します

attributes - 属性を指定します
modifiers - 修飾子を指定します
type - イベントアクセッサのデリゲート型を指定します
member-name - イベントアクセッサの識別子を指定します
accessor-declarations - イベントアクセッサ宣言子群を指定します
