デリゲート

関数ポインタ

C/C++ で低水準なシステムプログラムを開発したことがある人は
これらの言語が関数のエントリーポイントを扱う重要性をよくご存知だと思います
関数のエントリーポイントを表すアドレスを、一般に「関数ポインタ」と呼びましたが
この関数ポインタは、システムに呼び出してほしい関数の位置などを知らせることができました

C# でもこのような機能は必要であると考えられましたが
関数ポインタは型保証がなく、リスクが生じるのでオブジェクト指向には適しません
そこで、C# では**デリゲート**という機能が用いられます

デリゲートは関数ポインタに代わる機能で、コールバック関数などの作成に用いられます
さらに、当然オブジェクト思考という見解からインスタンスメソッドにも対応しなければなりません
デリゲートは、C++ では不完全だった型の保証も含めてコールバックを保証するのです

デリゲートを宣言するには **delegate** キーワードを用います
これでデリゲートを宣言し、メソッドの位置を格納するデリゲート型の参照型変数を生成します
まずは、デリゲートの宣言構文を紹介しましょう

[attributes] [modifiers] delegate result-type identifier ([formal-parameters]);

見てわかるように、ほとんどメソッドの宣言と同じです
attributes には属性、modifiers には修飾子、result-type には戻り値
identifier には識別子、formal-parameters には仮パラメータを指定します

デリゲートの宣言は名前空間やクラスのメンバ空間などで宣言できます
クラスのメンバ空間で宣言した場合も、インスタンスとの関わりはありません
new、public、protected、internal、private のいずれかの修飾子のみ指定できます

これが、関数ポインタの代わりにメソッドの位置を記憶するデリゲートの宣言です
この時点で、まずデリゲート型が作成されたことになります
この型を使用するには、**デリゲートのインスタンス**を生成します
デリゲートは、クラスのようにふるまうのでインスタンスが必要なのです

new 演算子を用いるということはクラスのインスタンス生成と同じです
しかしデリゲートは、いささか特殊な生成式を持っています

new delegate_type(expression);

delegate_type は生成するデリゲートの型を指定します
expression には、デリゲートが呼び出すべきメソッドを指定します
または、他のデリゲート型を指定してコピーを作る場合もあります

expression に指定するメソッドは、必ずデリゲート型と同じシグネチャでなければなりません
シグネチャが同じであれば、型が保証されるのでほかに制約されるものはありません
生成されたデリゲート型変数は、指定したメソッドの位置情報を格納しています

デリゲートを用いてメソッドにアクセスする方法はいたって簡単です
デリゲートの識別子と () を用いて、通常のメソッドのように呼び出すことができます

```
delegate void KittyCallback();

class Test {
 static void Main() {
  KittyCallback kitty = new KittyCallback(Kitty);
  kitty();
 }
 static void Kitty() {
  System.Console.WriteLine("Kitty on your lap");
 }
}
```

KittyCallback はデリゲートです
KittyCallback 型のインスタンスは戻り値が void、パラメータ無しのメソッドの位置を保有します
KittyCallback kitty = new KittyCallback(Kitty) に注目していただければわかるように
デリゲート型 kitty 変数は Main.Kitty() メソッドの位置情報を持っているのです

デリゲートの特殊な呼び出し規約で kitty() とすれば、そのまま Kitty() メソッドが呼び出されます
これを見れば、デリゲートがコールバックに適していることがよくわかるでしょう

デリゲート型はインスタンスを生成し、参照型の変数をもつということは
これも System.Object を基底とした一貫した型情報を持っていることを想像できます
実際にその通りで、デリゲート型とは **System.Delegate** の派生クラスを指します
重要なことですが、Delegate クラスはデリゲート型ではありません
Delegate クラスの**派生クラスがデリゲート型**なのです

public abstract class Delegate : ICloneable, ISerializable

デリゲート型は、全てこのクラスを拡張しているのです
必要であれば Object 型にキャストしたり、Delegate の機能を使うこともできます

このクラスの詳細については、.NET クラスライブラリのリファレンスを参照してください

また、デリゲート型は **暗黙的に sealed** となっています
デリゲート型からさらに派生クラスを生成することはできません

上のプログラムのデリゲートは静的なメソッドを参照しましたが
デリゲートはシグネチャが同じであれば、インスタンスのメソッドでもかまいません
この時、デリゲートはインスタンスの情報と共にメソッドの情報を格納します

```
delegate void KittyCallback();

class Kitty {
 private string str;
 public Kitty(string str) {
  this.str = str;
 }
 public void Write() {
  System.Console.WriteLine(str);
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty("Kitty on your lap");
  KittyCallback write = new KittyCallback(obj.Write);
  write();
 }
}
```

このプログラムは、KittyCallback デリゲートに Kitty のインスタンスメソッドを格納しています
デリゲートは、メソッドを呼び出す時にインスタンスを参照してアクセスします

また、デリゲートの宣言はクラスなどのメンバとして扱うことも可能です
基本的な扱いはメソッドと同じですが、オーバーライドはできません
ただし、基底クラスで同一の宣言がある場合 new で隠蔽することは可能です

```
class Test {
 delegate void KittyCallback();
 static void Main() {
  KittyCallback kitty = new KittyCallback(Kitty);
  kitty();
 }
 static void Kitty() {
  System.Console.WriteLine("Kitty on your lap");
 }
}
```

このプログラムの KittyCallback デリゲートは Test クラスで宣言されています
他のスコープからこれにアクセスする時は、静的に Test.KittyCallback とすることでアクセスできます
当然ですが、デリゲート型変数は通常の参照型変数としてフィールドに配置できます

これまでのサンプルでは戻り値が void で、パラメータがないシグネチャのデリゲートでしたが
コールバックの仕様などにあわせて、目的のシグネチャを指定できます

```
class Test {
 delegate void WriteCallback(System.Object str);
 static void Main() {
  WriteCallback write = new WriteCallback(System.Console.WriteLine);
  write("Kitty on your lap");
 }
}
```

このプログラムでは、第一パラメータに Object 型の参照を受ける
WriteCallback デリゲートを宣言しています
そして Console.WriteLine() メソッドを参照するインスタンスを作成しています

---

delegate

[attributes] [modifiers] delegate result-type identifier ([formal-parameters]);

デリゲートを宣言します

attributes - 属性を指定します
modifiers - 修飾子を指定します
result-type - 戻り値の型を指定します
identifier - 識別子を指定します
formal-parameters - パラメータを指定します
