

Flash Quick Starts:Programming with ActionScript 3.0

Creating a simple ActionScript 3.0 の日本語訳

本ドキュメントは、Adobe 社のサイトにある“Creating a simple ActionScript 3.0 “をヒム・カンパニー 永井勝則が自主的に日本語に訳したものです。

<http://www.himco.jp/>

knagai@himco.jp

(2009/2/2)

本ドキュメントの原文は

http://www.adobe.com/devnet/flash/quickstart/creating_class_as3/

です。

簡単な ActionScript 3.0 クラスの作成

みなさんはおそらく、ActionScript コードを記述するときにはいつも、オブジェクトのインスタンスを作成し、プロパティの値の読み取りや設定を行い、メソッドを呼び出してアクションを実行し、オブジェクトによって送出されたイベントに反応する、オブジェクト指向プログラミングを使っておられることでしょう。しかしビルトインのオブジェクトに加え、みなさんは自分自身のオブジェクトのタイプ(クラスと呼ばれます)を定義することもできます。みなさんの ActionScript プロジェクトが単純な機能性を超えるようになってきたら、自分自身のクラスの使用は、ActionScript を組織化し、関連する機能をグループ化し、見る必要のないときにはその詳細を隠すことを可能にすることで、プログラミングを簡単にしてくれます。

クラスを作成する前に、相応の時間をかけてアプリケーションの設計とクラスの構造化について考えるようにします。クラスの組織化の方法に関する考え方のヒントは、「ActionScript 3.0 のプログラミング」の[「クラスの設計戦略」](#)を参照してください。

クラス的设计プランが固まるか、少なくとも、追跡する必要のある情報や実行する必要のある動作に関する考えがまとまりさえすれば、クラスを記述する実際のシンタックス(書き方)はいたって簡単です。

このクイックスタートでは、Greeter(あいさつする人のこと)クラスという名前の小さな ActionScript クラスを作成していきます。Greeter の構造は次の通りです(このサンプルの全コードはこのクイックスタートの終わりに掲載しています)。

- パッケージ: com.example.quickstart
- クラス: Greeter
- プロパティ:
 - name: あいさつ文で使用する名前を設定、取得
- メソッド:
 - Greeter()コンストラクタ: 新しい Greeter インスタンスを作成。オプションの String パラメータは name プロパティの値を設定する。
 - sayHello(): あいさつ文を含む String を返す。

Greeter クラスでは、最初に Greeter オブジェクトを作成するとき Greeter()コンストラクタ内かまたは、後から Greeter オブジェクトの name プロパティの値を変更することで、名前を指定することができます。名前を指定すると、Greeter オブジェクトの sayHello()メソッドを呼び出すことで、指定した名前を使ったあいさつを作成することができます。

ActionScript の記述では、Greeter クラスは次のようなものです（メソッド本体はプレースホルダー（代わり）のコードになっています）。

```
package com.example.quickstart{
    public class Greeter {

        // ----- コンストラクタ -----
        public function Greeter(initialName:String = "") {
            // 指定された場合、名前の値を設定
        }

        // ----- プロパティ -----
        public var name:String;

        // ----- メソッド -----
        public function sayHello():String {
            // あいさつ文を作成し、それを戻り値として渡す
        }
    }
}
```

以下はこれからみなさんが自分自身の ActionScript クラスを作成し使用していく上での手順です。

クラスファイルの作成

クラスとパッケージステートメントの作成

クラスへのプロパティの追加

クラスへのメソッドの追加

そのほかのクラス要素の追加

アプリケーションでのクラスの使用

クラスファイルの作成

ActionScript 3.0 クラスは、それを使用する Flash FLA や Flex MXML ファイルとは別の、テキストドキュメントとして記述します。クラスファイルには、.as 拡張子を加えて、そのクラスと同じ名前をつける必要があります。クラスファイルは、クラスのパッケージ構造に一致するフォルダ階層に含まれる必要があります。このサンプル

では、Greeter クラスを作成するので、このファイルは Greeter.as という名前のファイルに保存します。パッケージを含む完全なクラス名は “com.example.quickstart.Greeter” (Greeter クラスは “com.example.quickstart” パッケージに含まれます) なので、Greeter.as は /com/example/quickstart というフォルダ構造に保存する必要があります。“com” フォルダはこのアプリケーションのルートフォルダに含めることも、アプリケーションの ActionScript クラスパスの一部であるフォルダに含めることもできます。

Greeter クラスのクラスファイルを作成するには、

1. Flex Builder や Flash といった ActionScript 用のプログラムか、Dreamweaver のような汎用的なプログラミングツールまたは、プレーンテキストのドキュメントが記述できるプログラムで、新規テキストドキュメントを開きます。
2. ファイル名を “Greeter.as” にして、“/com/example/quickstart/” と同じフォルダ構造内に保存します。この “com” フォルダはこのアプリケーションのルートにあります。

クラスとパッケージステートメントの作成

クラス名の定義は class ステートメントを使って行います。クラスは、そのほかのコードがそのクラスのインスタンスを作成するために “public” として指定しなければなりません (特に、2, 3 の特別な環境をのぞいて ActionScript 3.0 では必ず public にします)。完全なステートメントは、public class クラス名です。このステートメントにはつづけて、そのクラスの中身を囲む2つの中かっこをつけます。

class ステートメントは、クラスのありかを示すパッケージ名を示す package ステートメントで囲む必要があります。package ステートメントのシンタックスは class ステートメントのシンタックスと似ており、package の後にパッケージ名をつづけて、パッケージ内容を囲む2つの中かっこをつけます。

Greeter クラスの class と package ステートメントを追加するには：

- ActionScript ファイルで、次のコードを追加します。

```
package com.example.quickstart{
    public class Greeter extends Sprite {
        // クラス内容(メソッドやプロパティ)をここに記述する
    }
}
```

このコードは package ステートメント (package com.example.quickstart) を中かっこ内に含み、その内部に中かっこをつけた class ステートメントを含んでいます。

クラスへのプロパティの追加

クラスのプロパティは、クラス本体内で `var` ステートメントを使って定義します。そのシンタックスは、`public` 指定子を加えるところ以外は変数を宣言するときに使用するシンタックスと同じです (`class` ステートメントで使ったときと同じ方法で)。

Greeter クラスに `name` プロパティを追加するには：

- `class` ステートメントの開き中かっこと閉じ中かっこの間で、空行を追加して次のコードを追加します。

```
public var name:String;
```

このコード行では、`String` 値を保持する `name` プロパティを定義しています。プロパティではオプションで、変数に初期値を指定するのと同じ方法でデフォルト値を指定することができます (たとえば、`public var name:String = "Fred";`)。しかしこの Quick Start では初期値はコンストラクタメソッド内で設定します。

また次のように、指定子を `pubic` でなく `private` を使ってプライベートなプロパティを定義することもできます。

```
private var secretValue:Number;
```

プライベートのプロパティはそのクラス定義内のコードからはアクセスできる変数ですが、クラス定義の外からはアクセスできない変数です。

クラスへのメソッドの追加

クラスのメソッドは、関数を定義するのと同じシンタックスを使い、アクセス指定子 (`public` や `private`) を加えて定義します。メソッドの定義には次の要素があります。

- アクセス指定子 (通常は `public` か `private`)
- 用語 `function`
- 関数名
- メソッドが受け取るパラメータの定義を含む開いたかっこと閉じかっこ
- コロンとその後にメソッドが返す値のデータ型 (またはメソッドが値を返さない場合は `void`)
- メソッドが実行するステートメントを囲む開いた中かっこと閉じる中かっこ

パラメータを受け取らず String 値を返す sayHello()メソッドを作成するには:

- class ステートメントの開く中かっこと閉じる中かっこの中に、次のコード行を入力します。

```
public function sayHello():String {
    var result:String;
    if (name != null && name.length > 0) {
        result = "Hello there, " + name + ".";
    } else {
        result = "Hello there, anonymous.";
    }
    return result;
}
```

コンストラクタメソッドは、クラスのインスタンスを作成する処理の一部として呼び出される特別なメソッドです。コンストラクタメソッドを使用すると、プロパティの初期値の指定やそのほかの細かな設定の実行を行うことができます。コンストラクタメソッドの特殊な特徴の一部として、コンストラクタを定義するときは、メソッド名はクラス名と正確に一致する必要があり、メソッドは public として定義しなければなりません。また戻り値のデータ型(void も)は指定しません。

Greeter クラスのコンストラクタメソッドを作成するには:

- class ステートメントの開く中かっこと閉じる中かっこの間に、次のコード行を入力します。

```
public function Greeter(initialName:String = "") {
    name = initialName;
}
```

このコードでは、オプションで単一の String パラメータを受け取るコンストラクタを定義しています。このパラメータに関して値が渡される場合は、その値が name プロパティに代入されます。それ以外の場合は name プロパティは、"" (空の文字列)がメソッドパラメータのデフォルト値であるため、""に設定されます。

クラス内でコンストラクタメソッドを定義しない場合は(たとえばセットアップ操作を何も実行する必要がない場合)、コンパイラが自動的にクラス用の空のコンストラクタ(パラメータとステートメントを何も持たない)を作成します。

そのほかのクラス要素の追加

定義できるクラス要素はもう少しあります。これらの要素を作成する処理はメソッドやプロパティの作成よりも複雑になるので、この Quick Start では取り上げません。

- アクセサはメソッドとプロパティの中間の特殊なものです。クラスを定義するコードを記述するときは、複数のアクションを実行できるように(単なる値の読み取りや代入ではなく、プロパティを定義するときはこれができることのすべてですが)アクセサをメソッドのように記述します。しかしクラスのインスタンスを作成すると、アクセサは、値の読み取りや値の代入にその名前を使うだけで、プロパティのように扱うことができます。詳細については[ActionScript 3.0 のプログラミング]の[“get および set アクセサメソッド”](#)を参照してください。
- ActionScript のイベントは特定のシンタックスでは定義しません。クラスのイベントは、イベントリスナーを追跡し、イベントリスナーのイベントを通知する EventDispatcher クラスの機能性を使って定義します。自作のクラスにイベントを作成する詳細については、[ActionScript 3.0 のプログラミング]の[“イベントの処理”](#)を参照してください

アプリケーションでのクラスの使用

自作したクラスのインスタンスは、ほかの ActionScript クラスと同じように、作成し使用することができます。まずコンパイラにクラスを探す場所を教えるために、パッケージとクラス名を使って、import ステートメントを使う必要があります。Greeter クラスでは import ステートメントは次のようになります。

```
import com.example.quickstart.Greeter;
```

import ステートメントを追加すると、そのクラスをほかのクラスと同じように使用できるようになります。たとえばデータ型が作成したクラスである変数を宣言し、そのクラスのインスタンスを作成して、その変数に保持することができます。

```
var michaelGreeter:Greeter = new Greeter("Michael");
```

また値をそのインスタンスに代入し、そのメソッドを呼び出すこともできます。

```
michaelGreeter.name = "Mike"  
trace(michaelGreeter.sayHello());
```

サンプル

次のサンプルでは、Greeter クラスのインスタンスを作成し、その name プロパティを“Steve”に設定した後、sayHello()メソッドを呼び出し、その結果のあいさつ文を画面上のテキストフィールド(Flash)か Text コンポーネント(Flex)に書き出しています。

このサンプルは、Greeter.as ファイル内の Greeter クラス(この Quick Start で作成してきたファイル)と、Greeter クラスのインスタンスを作成し使用するアプリケーションファイル(Flash の FLA ファイルか Flex の MXML ファイル)の2つから構成されます。次のコードブロックは Greeter クラスの完成版です。サンプルアプリケーションファイルの Flash 版と Flex 版を作成する方法はこの後に記述しています。

```
package com.example.quickstart{

    public class Greeter {

        public function Greeter(initialName:String = "") {
            name = initialName;
        }

        public var name:String;

        public function sayHello():String {
            var result:String;

            if (name != null && name.length > 0) {
                result = "Hello there, " + name + ".";
            } else {
                result = "Hello there, anonymous.";
            }
            return result;
        }
    }
}
```

このサンプルを Flash CS3 Professional で作成するには：

1. 新規 Flash ドキュメントを作成します。

2. テキストツールを使ってステージにダイナミックテキストフィールドを作成し、インスタンス名を output にします。
3. フレーム1のキーフレームを選択し、アクションパネルを開いて以下のコードを記述します。

```
import com.example.quickstart.Greeter;

// Greeter インスタンスを、“Steve”で作成
var myGreeter:Greeter = new Greeter(“Steve”);
//Steve にあいさつ
output.text = myGreeter.sayHello();
// カーソルを次の行に移す
output.text += “\n”;
// Greeter インスタンスの名前を“Harold”に設定
myGreeter.name = “Harold”;
// Harold にあいさつ
output.text += myGreeter.sayHello();
```

4. メインメニューから[制御]→[ムービープレビュー]を選択し、結果を確認します。

Flex Bulder でサンプルを作成するには：

1. 新規 Flex プロジェクトを作成します。
2. ソースビューで、initialize=“initApp();”というイベントハンドラ属性を mx:Application タグに追加します。mx:Application タグは次のようになります。

```
<mx:Application xmlns:mx=“http://www.adobe.com/2006/mxml” layout=“absolute”
initialize=“initApp();”>
</mx:Application>
```

3. < mx:Application >タグ内に、“output”のidでTextコンポーネントを追加します。width 属性は 100% に、textAlign 属性は“center”に設定します。コードは次のようになります。

```
<mx:Application xmlns:mx=“http://www.adobe.com/2006/mxml” layout=“absolute”
initialize=“initApp();”>
<mx:Text id=“output” width=“100%” textAlign=“center” />
</mx:Application>
```

4. mx:Script ブロックをアプリケーションの MXML ファイルに追加します。コードは次のようになります。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
initialize="initApp();">
<mx:Script>
  <![CDATA[
    import com.example.quickstart.Greeter;
    private function initApp():void{
      // Greeter インスタンスを、"Steve"で作成
      var myGreeter:Greeter = new Greeter("Steve");
      //Steve にあいさつ
      output.text = myGreeter.sayHello();
      // カーソルを次の行に移す
      output.text += "\n";
      // Greeter インスタンスの名前を"Harold"に設定
      myGreeter.name = "Harold";
      // Harold にあいさつ
      output.text += myGreeter.sayHello();
    }
  ]]>
</mx:Script>
<mx:Text id="output" width="100%" textAlign="center" />
</mx:Application>
```

5. ツールバーの実行ボタンをクリックして結果を確認します。

結果



```
Hello there, Steve.
Hello there, Harold.
```