

Flash Quick Starts:Programming with ActionScript 3.0

Display list programming の日本語訳

本ドキュメントは、Adobe 社のサイトにある“Display list programming “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

(2009/1/10)

本ドキュメントの原文は

http://www.adobe.com/devnet/flash/quickstart/display_list_programming_as3/

です。

表示リストのプログラミング

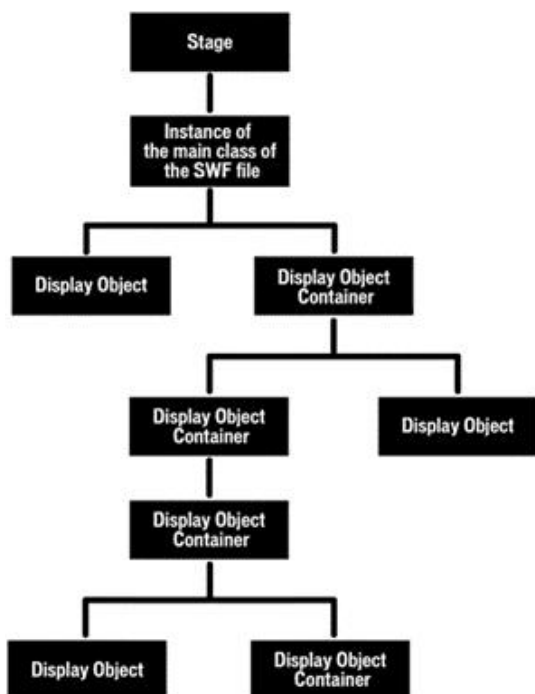
ActionScript 3.0 で構築されたアプリケーションはそれぞれ、表示リストと呼ばれる表示されるオブジェクトの階層を持っています。表示リストはアプリケーション内の目に見えるすべての要素を含みます。表示要素は1つかまたは複数の以下のグループに分けられます。

- **Stage:** Stage は表示オブジェクトの基本コンテナです。各アプリケーションは、画面上のすべての表示オブジェクトを含む Stage オブジェクトを1つ持ちます。ステージは最上位のコンテナで表示階層の一番上にあります。各 SWF ファイルは、SWF ファイルのメインクラスと呼ばれる、関連づけられた ActionScript クラスを持ちます。Flash Player は、HTML ページの SWF ファイルを開くと、そのクラスのコンストラクタ関数を呼び出し、作成されたインスタンス(これは常に表示オブジェクトのタイプになります)が Stage オブジェクトの子として追加されます。SWF ファイルのメインクラスは常に Sprite クラスを拡張します。Stage には DisplayObject インスタンスの stage プロパティを通してアクセスできます。
- **表示オブジェクト:** ActionScript 3.0 では、アプリケーション内の画面に表示されるすべての要素は表示オブジェクトのタイプになります。flash.display パッケージには、そのほかの多数のクラスによって拡張される基本クラスの DisplayObject クラスが含まれます。これらの異なるクラスは、例を挙げると、ベクターシェイプ、ムービークリップ、テキストフィールドといった異なる表示オブジェクトのタイプを表します。
- **表示オブジェクトコンテナ:** 表示オブジェクトコンテナは、それ自体を視覚的に表すことに加え、表示オブジェクトである子オブジェクトを含むこともできるという、特殊な表示オブジェクトのタイプです。DisplayObjectContainer クラスは DisplayObject クラスのサブクラスです。DisplayObjectContainer オブジェクトはその子リストに複数の表示オブジェクトを含むことができます。

すべての視覚的な表示オブジェクトは DisplayObject クラスから継承しますが、それぞれの型は DisplayObject クラスの特定のサブクラスの型です。作成できるのは、定義された視覚的表現を持つ特定の表示オブジェクトのインスタンスのみです。たとえば適切なコンストラクタメソッドを使って、Sprite クラスや Shape クラス、Video クラスのインスタンスを作成できますが、DisplayObject()コンストラクタを呼び出して DisplayObject クラスのインスタンスを作成することはできません。

ActionScript アプリケーションの表示リストはアプリケーション内のすべての視覚的オブジェクトを含みます。表示リストは、Stage をそのほかのすべてのコンテンツの後ろにあるオブジェクトとして持つ階層またはツリー構造と見なすことができます。Stage は SWF ファイルのメインクラスのインスタンスを含みます(たとえば、

Flash の SWF のメインのタイムラインや Flex の SWF のアプリケーションコンポーネント)。このメインのオブジェクトは1つまたはそれ以上の表示オブジェクトや表示コンテナを含むことができ、そのコンテナはどれも子オブジェクトを含むことができます。次の図は SWF 内の架空の表示リスト階層の例です。



子のコンテンツ(コンテナ内の表示オブジェクト)は常にその親のコンテナの前面に表示されます。表示オブジェクトコンテナが複数の子である表示オブジェクトを含んでいる場合は、その前面に順に重なります。表示オブジェクトコンテナはその重なり順を覚えています。表示オブジェクトコンテナ内の子である表示オブジェクトの順番はよく、表示オブジェクトの深度、重なり順、インデックスと呼ばれます(各表示オブジェクトの深度は、配列のインデックスと同じように、一番下の子の位置は 0、その上の子は 1 というように、整数のインデックスとして保持されます)。

表示リスト上の表示オブジェクトで行えることは多くありますが、このクリックスタートでは、表示リストを扱うとき実行したいと思われるような以下の一般的な作業を取り上げます。

オブジェクトの表示リストへの追加

表示リスト内のオブジェクトの深度の変更

表示リストからのオブジェクトの削除

オブジェクトの表示リストへの追加

行いたいと思われる最も一般的なことは、表示リストに表示オブジェクトを追加することでしょう。そのためには、子の表示オブジェクトを作成し、それを表示オブジェクトコンテナの `addChild()` メソッドを呼び出すことでそのコンテナに追加します。デフォルトでは、表示オブジェクトはコンテナの子の前面に追加されます。子オブジェクトの重なりの中に挿入したい場合には、`addChildAt()` メソッドを使用します。

次のサンプルでは3つの表示オブジェクト (`Shape` クラスのインスタンス) を作成し、それらを `container` という名前のオブジェクトの子として表示リストに追加します。最初の2つのオブジェクト (`circle` と `triangle`) は `addChild()` で追加するので、この2つは 0 と 1 のインデックス位置に追加されます。3つめの子 (`square`) は `addChildAt()` で 1 のインデックス位置に追加します。これによって `square` は 1 のインデックス位置に配置され、そこにあった `triangle` は位置 2 に (前面に) 押し上げられます。

サンプルを単純にするため、以下のコードでは `createShape()` 関数を呼び出して、`Shape` インスタンスを作成しています。表示リストにオブジェクトを追加する処理は以下のメインコードで行います。

```
// このコードは Sprite や MovieClip インスタンスなどの
// DisplayObjectContainer に割り当てる必要がある
import flash.display.Shape;

var circle:Shape = createShape("circle");
this.addChild(circle);

var triangle:Shape = createShape("triangle");
this.addChild(triangle);

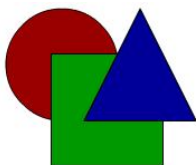
var square:Shape = createShape("square");
this.addChildAt(square, 1);

function createShape(shapeType:String):Shape {
    var size:int = 100;
    var result:Shape = new Shape();
    result.graphics.lineStyle(1, 0x000000);

    switch (shapeType) {
        case "circle" :
            result.graphics.beginFill(0x990000);
            result.graphics.drawCircle(size / 2, size / 2, size / 2);
```

```
        result.graphics.endFill();
        result.x = 10;
        result.y = 10;
        break;
    case "square" :
        result.graphics.beginFill(0x009900);
        result.graphics.drawRect(0, 0, size, size);
        result.graphics.endFill();
        result.x = 50;
        result.y = 50;
        break;
    case "triangle" :
        result.graphics.beginFill(0x000099);
        result.graphics.moveTo(size / 2, 0);
        result.graphics.lineTo(size, size);
        result.graphics.lineTo(0, size);
        result.graphics.lineTo(size / 2, 0);
        result.graphics.endFill();
        result.x = 80;
        result.y = 10;
        break;
    }
    return result;
}
```

結果



この SWF ファイルは作成した3つのシェイプを表示し、表示リストでの位置がその表示にどのような影響を与えるかを示しています。リスト内の位置 0 にある円はそのほかのものより下にあり、位置 2 にある三角形は最前面にあります。

表示リスト内のオブジェクトの深度の変更

親であるコンテナ内にある表示オブジェクトの重なりの中で、表示オブジェクトを前方または後方に移動させる（深度を変更する）テクニックはいくつかあります。表示オブジェクトコンテナでは、その `setChildIndex()` メソッドを使って子であるオブジェクトのインデックス位置を変えることができます。またその `swapChildren()` メソッド（両方の子のオブジェクトへの参照を持っている場合）や、`swapChildrenAt()` メソッド（各子のインデックス位置が分かっている場合）を使って2つの子の深度を入れ替えることができます。オブジェクトのインデックスを知る必要がある場合には、そのコンテナの `getChildIndex()` メソッドを使用します。また別の方法として、そのオブジェクトのインデックスが分かっている場合、そのオブジェクトへの参照が必要な場合には、その `getChildAt()` メソッドが使用できます。

次のサンプルでは2つの円、2つの正方形、1つの三角形の5個のシェイプを作成しています。シェイプのどれかをクリックすると、どのラジオボタンが選ばれていたかによって、表示リスト上でのその深度が変わります。ラジオボタンの値は、“最背面へ移動”、“背面へ移動”、“前面へ移動”、“最前面へ移動”です。“最背面へ移動”と“最前面へ移動”操作を実行するには、`container.setChildIndex()` メソッドをインデックス 0 で使ってオブジェクトを最背面へ移動させ、インデックス `numChildren - 1` を使ってオブジェクトを最前面に移動させます。“背面へ移動”と“前面へ移動”操作を実行するには、`container.getChildIndex()` を使って、クリックされたオブジェクトのインデックス値を取得し、その値よりも1大きいか1小さいインデックス値を持つオブジェクトを `container.getChildAt()` メソッドで参照して、`container.swapChildren()` メソッドでその深度を入れ替えます。

Note: このコードでは、各シェイプのクラス定義を行っておく必要があります。つまり `RedCircle`、`GreenSquare`、`BlueTriangle`、`YellowCircle`、`PurpleSquare` という名前で各シェイプのムービークリップを作成し、リンケージ設定を行っておく必要があります。

訳者注:

本文中では“シェイプ”と言っていますが、`Shape` クラスのオブジェクトではなく、実際にはオーサリング時に作成してあるムービークリップの形（シェイプ）です。コード内でこれらのムービークリップを使用するため、各ムービークリップのリンケージ設定を行う必要があります。また `RadioButton` コンポーネントもオーサリング時にステージに配置し、インスタンス名などの設定を行います。詳しくはダウンロードファイルをご覧ください。

サンプル

```
// このコードは、Sprite や MovieClip インスタンスなどの
// DisplayObjectContainer に割り当てる必要がある
import flash.display.DisplayObject;
import flash.display.Sprite;
import flash.events.MouseEvent;
```

```
// Sprite の container の作成
var container:Sprite = new Sprite();
this.addChild(container);

// 子のシェイプを作成し、位置を決め、container に追加
var redCircle:RedCircle = new RedCircle();
redCircle.x = 20;
redCircle.y = 10;
container.addChild(redCircle);

var greenSquare:GreenSquare = new GreenSquare();
greenSquare.x = 90;
greenSquare.y = 20;
container.addChild(greenSquare);

var blueTriangle:BlueTriangle = new BlueTriangle();
blueTriangle.x = 110;
blueTriangle.y = 50;
container.addChild(blueTriangle);

var yellowCircle:YellowCircle = new YellowCircle();
yellowCircle.x = 70;
yellowCircle.y = 110;
container.addChild(yellowCircle);

var purpleSquare:PurpleSquare = new PurpleSquare();
purpleSquare.x = 10;
purpleSquare.y = 70;
container.addChild(purpleSquare);

// シェイプにイベントハンドラ関数を登録
redCircle.addEventListener(MouseEvent.CLICK, shapeClickHandler);
greenSquare.addEventListener(MouseEvent.CLICK, shapeClickHandler);
blueTriangle.addEventListener(MouseEvent.CLICK, shapeClickHandler);
yellowCircle.addEventListener(MouseEvent.CLICK, shapeClickHandler);
purpleSquare.addEventListener(MouseEvent.CLICK, shapeClickHandler);
```

```
// この関数は、シェイプのどれかがクリックされたら呼び出される
function shapeClickHandler(event:MouseEvent):void {
    var clickedShape:DisplayObject = event.target as DisplayObject;

    // 選択されているラジオボタンにしたがって適切なアクションを実行
    if (rbMoveToBack.selected) {
        moveToBack(clickedShape);
    } else if (rbMoveBackward.selected) {
        moveBackward(clickedShape);
    } else if (rbMoveForward.selected) {
        moveForward(clickedShape);
    } else if (rbMoveToFront.selected) {
        moveToFront(clickedShape);
    }
}

function moveToBack(shapeToMove:DisplayObject):void {
    // container の子リストで 0 のインデックスに設定することで
    //シェイプを最背面に移動
    container.setChildIndex(shapeToMove, 0);
}

function moveBackward(shapeToMove:DisplayObject):void {
    // クリックされたシェイプのインデックスを取得し、行き先インデックスを決める
    // 背面へ移動するので、行き先インデックスは 1 引いた値
    var shapeIndex:int = container.getChildIndex(shapeToMove);
    var destinationIndex:int = shapeIndex - 1;

    // まだ一番下のシェイプでないことを確認
    if (destinationIndex >= 0) {
        // 行き先インデックスでシェイプへの参照を取得
        var destination:DisplayObject = container.getChildAt(destinationIndex);
        // シェイプの入れ替え
        container.swapChildren(shapeToMove, destination);
    }
}
```

```

}

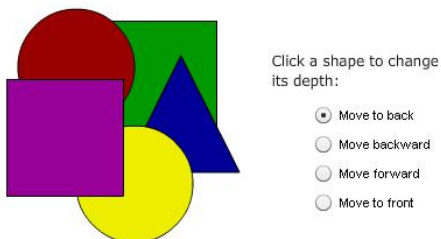
function moveForward(shapeToMove:DisplayObject):void {
    // クリックされたシェイプのインデックスを取得し、行き先インデックスを決める
    // 前面へ移動するので、行き先インデックスは 1 足した値
    var shapeIndex:int = container.getChildIndex(shapeToMove);
    var destinationIndex:int = shapeIndex + 1;

    // まだ一番上のシェイプでないことを確認
    if (destinationIndex < container.numChildren) {
        // 行き先インデックスでシェイプへの参照を取得
        var destination:DisplayObject = container.getChildAt(destinationIndex);
        // シェイプの入れ替え
        container.swapChildren(shapeToMove, destination);
    }
}

function moveToFront(shapeToMove:DisplayObject):void {
    // シェイプを一番前のインデックス(この値は常に numChildren - 1 になる)
    // に移動させることで最前面に移動させる
    container.setChildIndex(shapeToMove, container.numChildren - 1);
}

```

結果



このサンプルのソースファイルをダウンロードするには[ここ](#)をクリックしてください。

表示リストからのオブジェクトの削除

実行したいもう1つの作業は、表示リストから表示オブジェクトを削除することでしょう。コンテナの子リストか

ら表示オブジェクトを削除するには、そのコンテナの `removeChild()` メソッドを呼び出し、パラメータとして削除する表示オブジェクトを渡します。

次のサンプルでは、表示リストから子である表示オブジェクトを削除する例を示しています。ここでは SWF のメインクラスのインスタンスに割り当てられた9つの正方形(表示オブジェクト)があり、星型の表示オブジェクトがそのどれかの下に隠されています。その正方形のどれかの上でクリックすると、表示オブジェクトコンテナの `removeChild()` メソッドが呼び出され、パラメータとしてその正方形が渡されます。その結果表示リストからその正方形が削除されます(正方形が削除されると、その後に星型が隠れていたかどうかは分かりません)。

```
import flash.display.DisplayObject;
import flash.events.MouseEvent;

// 開始値の設定
var numGuesses:int = 0;
var gameOver:Boolean = false;
var correctAnswer:int;
var correctAnswerSquare:DisplayObject;

// "You Win"アニメーションの一時停止
rewardAnimation.gotoAndStop(1);

init();

// 正方形に適切なリスナー関数を登録
one.addEventListener(MouseEvent.CLICK, pickSquare);
two.addEventListener(MouseEvent.CLICK, pickSquare);
three.addEventListener(MouseEvent.CLICK, pickSquare);
four.addEventListener(MouseEvent.CLICK, pickSquare);
five.addEventListener(MouseEvent.CLICK, pickSquare);
six.addEventListener(MouseEvent.CLICK, pickSquare);
seven.addEventListener(MouseEvent.CLICK, pickSquare);
eight.addEventListener(MouseEvent.CLICK, pickSquare);
nine.addEventListener(MouseEvent.CLICK, pickSquare);

// この関数は正方形がクリックされると呼び出される
```

```
function pickSquare(event:MouseEvent):void {
    // ゲームが終わっていたらクリックを無視
    if (gameOver) {
        return;
    }

    // クリックされた正方形の参照を取得
    var clickedSquare:DisplayObject = event.target as DisplayObject;

    // 表示リストから正方形を削除
    this.removeChild(clickedSquare);

    // 解答(クリック)した回数の更新
    numGuesses++;
    numGuessesDisplay.text = numGuesses.toString();

    // 隠れていた星型を見つけたかどうかを調べる
    checkAnswer(clickedSquare);
}

// この関数はゲームのセットアップで呼び出される
function init():void {
    // 星型の場所をランダムに選択(1から9の間)
    correctAnswer = Math.round((Math.random() * 8) + 1);

    // どの正方形が星型を隠すのかを特定
    switch (correctAnswer) {
        case 1 :
            correctAnswerSquare = one;
            break;
        case 2 :
            correctAnswerSquare = two;
            break;
        case 3 :
            correctAnswerSquare = three;
```

```

        break;
    case 4 :
        correctAnswerSquare = four;
        break;
    case 5 :
        correctAnswerSquare = five;
        break;
    case 6 :
        correctAnswerSquare = six;
        break;
    case 7 :
        correctAnswerSquare = seven;
        break;
    case 8 :
        correctAnswerSquare = eight;
        break;
    case 9 :
        correctAnswerSquare = nine;
        break;
}
// 星型の作成と配置、表示リストへの追加
var star:Star = new Star();

var xOffset:Number = one.width * ((correctAnswer - 1) % 3);
var yOffset:Number = one.height * (Math.ceil(correctAnswer / 3) - 1);
star.x = (one.width / 2) + xOffset;
star.y = (one.height / 2) + yOffset;

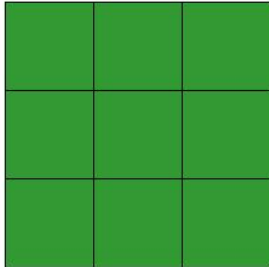
this.addChildAt(star, 0);
}

// この関数はユーザーがクリックして解答したとき呼び出される
// その解答が正しいかどうかを確認する
function checkAnswer(clickedSquare:DisplayObject):void {
    if (clickedSquare == correctAnswerSquare) {
        rewardAnimation.play();
    }
}

```

```
    gameOver = true;
  }
}
```

結果



Guess which square the
star is hiding under!

Guesses:

このサンプルのソースファイルをダウンロードするには[ここ](#)をクリックしてください。