集 中 連 載

text__阿部貴弘
Progression プロダクトマネージャー / デザイナー。Web デザイン会社、Web サービス会社などを経て、Progression プロジェクトの開発・運営を展開中。現在は講演活動と同時にグローバルな普及を目指しています。http://progression.jp/

Flash開発の新機軸「ActionScriptライブラリ」

SERIES #08

Progression クラススタイル編：ActionScript 3.0でサイト遷移構造を実装する

Progressionは、FlashサイトにHTMLサイトのようなページ遷移やディープリンクといったサイトの基本機能を実装するためのフレームワークです。その利用方法には、ユーザーのActionScriptスキルに合わせて「コンポーネント」「タイムライン」「クラス」の3つのスタイルが用意されています。シリーズ最終回となる今回は、その中でもActionScript 3.0を使うクラススタイルを解説します。Progressionを導入すれば、サイト構造と遷移機能を簡単に実装できるようになるため、これからActionScript 3.0によるサイト構築手法を習得したい人だけでなく、すでに独自でノウハウを習得している人にもおすすめしたい手法です。本記事では、クラススタイルの基本的なスクリプティングフローとサンプル事例を紹介します。

※コンポーネントスタイルについては、2009年2月号の本連載にて詳しく解説しています。タイムラインスタイルについては、P129のコラムを参照してください。

Progression™

```
/*
 *
 */
package myproject {
    import flash.display.StageAlign;
    import flash.display.StageQuality;
    import flash.display.StageScaleMode;
    import jp.progression.casts.*;
    import jp.progression.commands.*;
    import jp.progression.core.debug.Verbose;
    import jp.progression.events.*;
    import jp.progression.loader.*;
    import jp.progression.*;
    import jp.progression.scenes.*;

    /*=============================================
     * Index クラス
     */
    public class Index extends CastDocument {
```

Progressionが用意する流れの上に処理を設定する

ActionScriptはそのバージョン3.0で、より堅牢なオブジェクト指向プログラミングへと進化しました。Progressionの「クラススタイル」は、ActionScript 3.0の設計思想に準拠するように開発された制作スタイルです。コンポーネントスタイルやタイムラインスタイルはActionScript 3.0の一部の機能を扱いやすく加工したものですが、クラススタイルではすべての機能を直接操作することが可能となっており、ActionScript 3.0のパフォーマンスを最大限に引き出しながらサイトを構築することができます。

Progressionクラススタイルと従来のスクリプティング手法との最大の違いは、「処理の流れの扱い方」です。通常、Flashを制作する場合、「外部データの読み込み」や「アニメーション処理」などのさまざまな“時間の掛かる処理”をaddEventListener()メソッドによるイベント監視で繋ぎ合わせることで開発を進めます。しかし、個々の処理単位でイベント監視を行うとなると、スクリプトも複雑になり、管理の面でも非常に大変です。その管理や動作の整合性チェックに時間と手間がかかり、表現を追求する余裕がないという人も多いのではないでしょうか。

Progressionの場合は、「一つひとつの処理を繋げて流す」のではなく、「Progressionが用意する流れの上に処理を設定する」ように設計されています。こうすることで、流れの制御を気にせずにコンテンツ制作が行えるようになります。それでは、クラススタイルのスクリプティング手法を解説していきますが、その前に基本的なスクリプティングフローを把握しておきましょう。

Progressionには、サイトの基本機能を実装するためのクラス以外にも、エフェクトやコマンドなどのさまざまな機能があります。しかも、日本語による充実したAPIリファレンスが用意されています
http://asdoc.progression.jp/

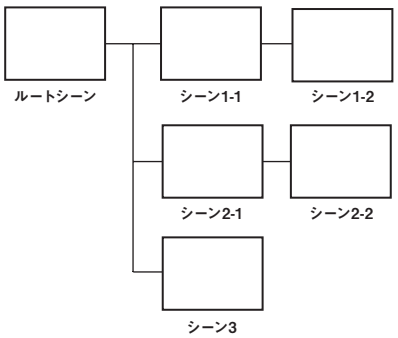
Progressionは、個人サイト、企業サイト、プロモーションサイトなど、すでに210以上のサイトで利用されています。Progression公式サイトのショーケースでは、導入事例の一部が紹介されています
http://progression.jp/ja/showcase/

クラススタイルでの基本的なスクリプティングフロー

Progressionでは、「シーン(SceneObject)」「キャストオブジェクト(CastObject)」「コマンド(Command)」というクラスを使って、サイトの基本機能(処理の流れ)を実装します。各クラスの役割は、右のように考えるとわかりやすいでしょう。スクリプティング方法は、作成するサイトによって変わってきますが、基本的なスクリプティングフローは、以下のようになります。まずは、このフローを覚えて、次のページから実際にスクリプトを書いてみましょう。

Progression = 監督
シーン(SceneObject) = 脚本上の舞台
キャストオブジェクト(CastObject) = 役者、大道具、小道具
コマンド(Command) = 脚本上の構成、流れ

01 シーンの作成



構築するシーン構造に合わせて、**SceneObjectクラスを使ってシーン構造を作成します。**個々のシーンは、単なる器なので、中は空っぽの状態です。なお、**SceneObjectクラスはActionScript 3.0のSpriteやMovieClipと同じようなものです。**そのため、以下の2パターンの設定方法が使用できます。

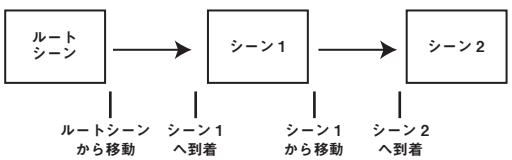
1. SceneObjectを継承して、シーンの数だけファイルを作成する。
2. new SceneObject()して、インスタンスのプロパティ操作で設定を行う。

02 キャストオブジェクトの作成とコマンド設定

※表示オブジェクトの作成やコマンド設定は、SceneObjectクラス内でも可能ですが、コンテンツや表現が複雑な場合は、キャストオブジェクトとして管理した方が効率的です。

Progressionで制御したい表示オブジェクトなどは、キャストオブジェクトとしてCastSpriteクラスやCastMovieClipクラスを使って作成します。キャストオブジェクトの作成もSceneObjectクラス同様に「継承するパターン」と「インスタンスに個性を設定するパターン」の2種類が利用可能です。また、キャストオブジェクト内にアニメーション処理などのコマンドを設定します。

03 イベント処理の設定

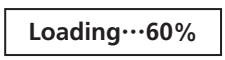


シーン構造とキャストオブジェクトを作成したら、次は「どのシーンでどのキャストオブジェクトを表示させるか」といった処理を設定します。該当シーンへ到着した瞬間、該当シーンから別のシーンへ移動した瞬間など、シーン移動時に発生する各種イベント時の処理(オブジェクトの表示・削除など)を設定します。

04 ボタンを作成

シーンに沿って画面を表示する処理まで完成したら、ユーザー操作に従って各シーン間を移動するナビゲーション処理を設定します。**ナビゲーションボタンには、CastButtonクラスを使用します。**CastButtonクラスもSceneObjectクラス同様に「継承するパターン」と「インスタンスに個性を設定するパターン」の2種類が利用可能です。

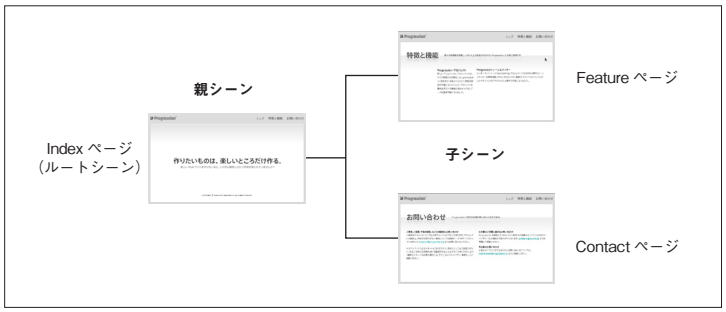
05 プリローダーの作成



最後にプリローダーを作成して完成です。**プリローダーは、Preloaderクラスを使って作成します。**

TUTORIAL　クラススタイルの基本コーディングを覚えよう

それでは、クラススタイルで右図にあるサンプルサイトを制作しながら、基本的なコーディングを解説していきましょう。サンプルは、ルートシーンと2つの子シーンを持つシンプルな構成です。なお、ここではProgressionのバージョン3をベースに解説しています。バージョン3を利用するにはFlash CS3以降が必要です。



STEP 01　Progressionをインストールする

ProgressionはFlashの拡張機能として組み込まれ、そのパッケージは公式サイト（http://progression.jp/）から入手します。インストールに成功すると、メニューの「ウインドウ→その他のパネル」に「**Progressionプロジェクト**」が追加されます。

パッケージは、「MXP」「JSFL」「SWC」の3つの形式を用意しています。**通常は、MXP形式を利用して下さい**

ダウンロードしたMXP形式ファイルをダブルクリックすると、Adobe Extension Managerが起動してインストールが開始されます

Progressionプロジェクト。制作スタイルの選択、コンテンツサイズなどの基本設定を行います

STEP 02　プロジェクトのセットアップ

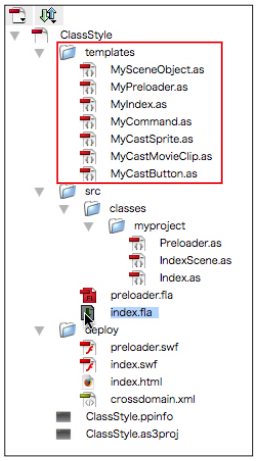
Progressionを使ったプロジェクトの作業を開始するには、まず「Progressionプロジェクト」パネルを開きます。今回はクラススタイルで作成するので、種類メニューから「クラス」を選び、その下にあるタブで基本設定を行います。設定後、**「新しく作成する」ボタンをクリックすると、プロジェクトに必要なファイル群が自動生成されます。**

プロジェクトの名前は、書き出されるフォルダ名やFlashを埋め込むHTMLのtitle要素に使用されます

「一般」タブでは、Flashサイトの幅や高さ、背景色などを設定します

「Flash」タブでは、フレームレートやウインドウモードを設定します

「ActionScript」タブでは、自動的に書き出されるasファイルに関連した設定を行います。今回はデフォルトのままにしておきます



各種設定後、「新しく作成する」ボタンをクリックするとプロジェクトに必要なファイル群が自動生成されます。今回、**シーンやキャストオブジェクトの定義は、「templates」フォルダ内にあるファイルをコピーして、そのファイルにスクリプトを記述します**

View & Download

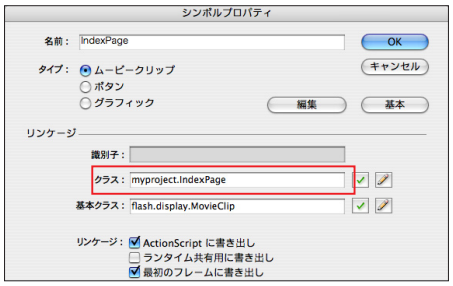
サンプルデータをダウンロードする際には、目次に掲載しているIDとパスワードが必要です。

http://book.mycom.co.jp/wd/

STEP 03 ページの各パーツを作成する

このサンプルサイトのページは、テキストや画像だけを配置した1つのMovieClipシンボルです。自動生成したindex.flaを開き、各ページのMovieClipシンボル（IndexPage、FeaturePage、ContactPage）と各ナビゲーションボタンのMovieClipシンボル（IndexButton、FeatureButton、ContactButton）を作成して、ライブラリに登録します。それぞれ、シンボルプロパティウインドウを開き、リンケージのクラス項目を設定します。

各ページのMovieClipシンボル



シンボルプロパティウインドウを開いてリンケージのクラス項目に、各ページの場合は「myproject.IndexPage」「myproject.FeaturePage」「myproject.ContactPage」、各ナビゲーションボタンの場合は「myproject.IndexButton」「myproject.FeatureButton」「myproject.ContactButton」と設定します

STEP 04 初期設定をする

Progressionを使う上で必要となる初期設定を行います。ドキュメントクラスとなるIndexクラス※1を編集します。index.asファイルを開いて、54行目～60行目のコードを右のように変更します（赤字部分）。**prog = new Progression( )はProgressionインスタンスを作成するためのもので、Progressionを使用する上でもっとも重要なコードです。第一引数にはインスタンスを識別するためのユニークな識別子を、第二引数には関連付けたいstageインスタンスの参照を、第三引数にはルートシーン（ここではIndexScene）として使用したいクラスの参照を設定します。**

prog.syncはブラウザ同期※2を有効化するための設定です。今回はブラウザ同期をさせるので設定値をtrueに変更します。

prog.goto()メソッドでは、シーンの移動を行っています。特に理由がなければこのままの状態にしておきます。

```
省略

// Progression インスタンスを作成します。
prog = new Progression( "index", stage, IndexScene );

// ブラウザ再生時に URL 同期を有効化します。
prog.sync = true;

// 最初のシーンに移動します。
prog.goto( prog.firstSceneId );

省略
```

【Index.asファイルでの変更例】

※1 Indexクラスは、CastDocumentクラスを継承しています。CastDocumentクラスは、ドキュメントクラスとして必要となるさまざまな機能が追加されたMovieClipクラスのサブクラスです。

※2 ブラウザ同期とは、SWFAddressを使用した機能です。コンテンツに合わせたシーン構造を作成するだけで、ディープリンクの制御をまったく意識せずとも、各シーンとWebブラウザのURLが自動的に同期するようになります。

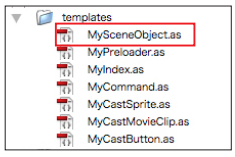
STEP 05 シーンを作成する

クラススタイルでシーンを扱うには、SceneObjectクラスを使用します。SceneObjectクラスをそのままnewして使用することもできますが、今回はSceneObjectクラスのサブクラスを作成して、その中に各種設定を行うことにしましょう。サブクラスを作成するには、プロジェクト内のtemplatesフォルダにあるMySceneObject.asというテンプレートファイルを使用するのが便利です。**MySceneObject.asをsrc/classes/myprojectフォルダ内にシーンの数だけコピーして、IndexScene.as、FeatureScene.as、ContactScene.asという3つのファイルを作成します。**

各ファイルを開き、**パッケージ名を「myproject」に、クラス名とコンストラクタ名をファイル名に対応したものに変更します**（赤字部分）。これでシーン構造を作成する準備が整いました。

```
package myproject {

中略

/*=============================================*//**
                * IndexScene クラス
                */
               public class IndexScene extends SceneObject {


/*=============================================*//**
                               * コンストラクタ
                               */
                              public function IndexScene() {

                              }
```

【IndexScene.asファイルでの変更例】



templatesフォルダ内にあるMySceneObject.asをsrc/classes/myprojectフォルダ内にシーンの数だけコピーします

シーン構造の作成は、STEP04でProgressionインスタンスを作成する際に第三引数に設定したルートシーン（IndexScene）から始めます。IndexScene.asを開いて、**コンストラクタに子シーンとなるFeatureSceneとContactSceneを作成するコードを追記します**（赤字部分）。

このサンプルでは、孫シーン（子シーンの子シーン）がないため、FeatureScene.asとContactScene.asではコードを追記する必要はありません。**孫シーンがある場合は、該当する子シーンにコードを記述します。**

以上でシーン構造ができました。次にキャストオブジェクトを作成します。

```
public function IndexScene() {
        // FeatureScene を作成
        var feature:FeatureScene = new FeatureScene();
        feature.name = "feature";
        addScene( feature );

        // ContactScene を作成
        var contact:ContactScene = new ContactScene();
        contact.name = "contact";
        addScene( contact );
}
```

【IndexScene.asファイルでの変更例】

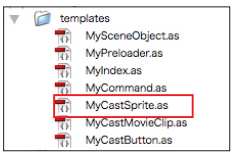
STEP 06 キャストオブジェクトを作成する

Progressionで制御したい表示オブジェクトはキャストオブジェクトとして作成します。このサンプルでは、各シーンに表示するオブジェクトは1つのMovieClipシンボルです。ここでは、シーン作成時と同様にテンプレートから作成します。**templatesフォルダ内にあるMyCastSprite.asをsrc/classes/myprojectフォルダ内にコピーして、各シーンに対応するようにIndexPage.as、FeaturePage.as、ContactPage.asの3つのファイルを作成します。**

そして、各ファイルを開き、**パッケージ名を「myproject」に、クラス名とコンストラクタ名をファイル名に対応したものに変更します**（赤字部分）。

```
package myproject {

中略

/*=============================================*//**
         * IndexPage クラス
         */
        public class IndexPage extends CastSprite {

/*=============================================*//**
                 * コンストラクタ
                 */
                public function IndexPage( initObject:Object = null ) {
                        super( initObject );
                }
```

【IndexPage.asファイルでの変更例】



templatesフォルダ内にあるMyCastSprite.asをsrc/classes/myprojectフォルダ内にシーンの数だけコピーします

STEP 07 キャストオブジェクトにコマンドを設定する

キャストオブジェクトを作成したら、演出となるコマンドを設定しましょう。Progressionには、さまざまな種類のコマンドが用意されています。このサンプルでは、DoTweenerクラス※3を使ってアニメーション処理を設定します。ここで設定しているのは、オブジェクトを表示／非表示する際に透明度を変化させるアニメーションです。

また、**コマンドを設定するには、キャストオブジェクト側のイベントについて理解する必要があります。**キャストオブジェクトインスタンスで発生する代表的なイベントは以下の2つです。

※3 DoTweenerクラスは、名前からわかるようにTweenerを使うためのコマンドです。Tweenerと同じようなフォーマットで処理を記述することができます。

```
// キャストオブジェクトがディスプレイリストに追加されたときの処理
protected override function _onCastAdded():void {
        // ページを透明にする
        alpha = 0;

        // 実行したいコマンドを登録
        addCommand(
                // ページを1秒かけて不透明にする
                new DoTweener( this, { alpha:1, time:1 } )
        );
}

// キャストオブジェクトがディスプレイリストから削除されたときの処理
protected override function _onCastRemoved():void {
        // 実行したいコマンドを登録
        addCommand(
                // ページを1秒かけて透明にする
                new DoTweener( this, { alpha:0, time:1 } )
        );
}
```

【IndexPage.asファイルでの変更例】

イベント	説明
CastEvent.CAST_ADDED	キャストオブジェクトがAddChild／AddChildAt／AddChildAtAboveコマンド経由でディスプレイリストに追加された瞬間に発生。このイベントは、onCastAddedイベントハンドラメソッドや_onCastAdded()オーバーライド・イベントハンドラメソッドでの設定も可能。
CastEvent.CAST_REMOVED	キャストオブジェクトがRemoveChild／RemoveAllChildrenコマンド経由でディスプレイリストから削除された瞬間に発生。このイベントはonCastRemovedイベントハンドラメソッドや_onCastRemoved()オーバーライド・イベントハンドラメソッドでの設定も可能。

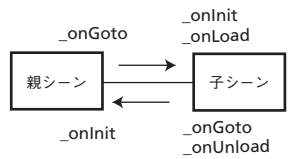
STEP 08 イベントを設定する

シーン構造とキャストオブジェクトを作成したら、次は「どのシーンでどのキャストオブジェクトを表示させるか」という処理を設定しなければなりません。**各シーンで必要となる処理は、シーンの移動状態に応じて発生するイベントに対して設定する必要があります。**シーン移動時に SceneObject インスタンスで発生する代表的なイベントは右の4つです。また、これらのイベントは下図のタイミングで発生します。

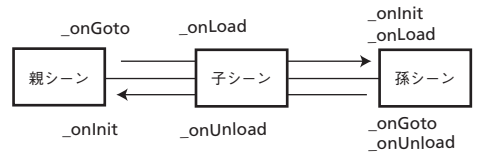
このサンプルでは「各シーンで、対応する MovieClip シンボルを表示／削除する」処理を作成します。まずは、IndexScene クラスに対して右のようなコードを追記します（赤字部分）。同様に FeatureScene クラスと ContactScene クラスにも追記します（詳しくはダウンロードファイルを参照）。これでイベントの設定は完了です。

イベント	説明
SceneEvent.INIT	移動先として指定したシーンに到着した瞬間に発生。主にそのシーンでのみ表示したい画面処理などで使用します。このイベントは onInit イベントハンドラメソッドや _onInit() オーバーライド・イベントハンドラメソッドでの設定も可能。
SceneEvent.GOTO	移動処理が開始され、現在のシーンから移動する瞬間に発生。主に SceneEvent.INIT で表示した画面の消去処理などで使用します。このイベントは onGoto イベントハンドラメソッドや _onGoto() オーバーライド・イベントハンドラメソッドでの設定も可能。
SceneEvent.LOAD	移動目的地のシーンが自身もしくは子シーンを示している場合、移動処理時に階層が変更された瞬間に発生。主に階層を下っても継続的に表示される画面処理などで使用します。このイベントは onLoad イベントハンドラメソッドや _onLoad() オーバーライド・イベントハンドラメソッドでの設定も可能。
SceneEvent.UNLOAD	移動目的地のシーンが親シーンを示している場合、移動処理時に階層が変更された瞬間に発生。主に SceneEvent.LOAD イベントで表示した画面の消去処理などで使用します。このイベントは onUnload イベントハンドラメソッドや _onUnload() オーバーライド・イベントハンドラメソッドでの設定も可能。

●親シーンと子シーン間の移動時に発生するイベント



●親シーンと孫シーン間の移動時に発生するイベント




```
public var page:IndexPage;

public function IndexScene() {
        // FeatureScene を作成
        var feature:FeatureScene = new FeatureScene();
        feature.name = "feature";
        addScene( feature );

        // ContactScene を作成
        var contact:ContactScene = new ContactScene();
        contact.name = "contact";
        addScene( contact );

        // IndexPage を作成
        // あらかじめライブラリにシンボルとして登録済み
        page = new IndexPage();
}

protected override function _onInit():void {
  addCommand(
   // 画面にシンボルを表示
    new AddChild( progression.container, page )
  );
}

protected override function _onGoto():void {
  addCommand(
   // 画面からシンボルを削除
    new RemoveChild( progression.container, page )
  );
}
```

【IndexScene.as ファイルでの変更例】

STEP 09 ボタンを作成する

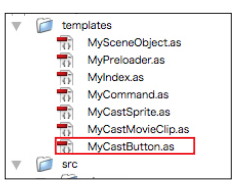
ここまででシーンに沿って画面を表示する処理まで完成しました。あとはユーザー操作に従って各シーン間を移動できるようになれば、Web サイトとして必要な機能を一通り揃えることができます。**Progression では、HTML での <a> 要素に相当する機能であるボタン機能が用意されています。**

ボタンの作成もシーン作成時と同様に**templates フォルダ内にある MyCastButton.as をコピーして、各シーンに対応するように IndexButton.as、FeatureButton.as、ContactButton.as の3つのファイルを作成します。**

そして、各ファイルを開き、**パッケージ名を「myproject」に、クラス名とコンストラクタ名をファイル名に対応したものに変更します**（赤字部分）。

```
package myproject {

中略

/*=================================================*//**
        * IndexButton クラス
        */
        public class IndexButton extends CastButton {

/*=================================================*//**
                * コンストラクタ
                */
        public function IndexButton( initObject:Object = null ) {
                super( initObject );
        }
```

【IndexButton.as ファイルでの変更例】



templates フォルダ内にある MyCastButton.as を src/classes/myproject フォルダ内にボタンの数だけコピーします

次にIndexButton.asを開いて、**コンストラクタに右のような移動先を設定するコードを追記します**（赤字部分）。同様にFeatureButtonクラス、ContactButtonクラスにも表にある移動先を追記します。コンストラクタで指定している**CastButtonクラスのSceneIdプロパティが、ボタンがクリックされた際の移動先シーンとなります。この設定は特に必要がない限りはコンストラクタで行うようにしてください。**

```
public function IndexButton( initObject:Object = null ) {
        super( initObject );

        // クリック時の移動先を設定する
        sceneId = new SceneId( "/index" );
}
```

【IndexButton.asファイルでの変更例】

クラス名	SceneIdの値
IndexButton	new SceneId("/index")
FeatureButton	new SceneId("/index/feature")
ContactButton	new SceneId("/index/contact")

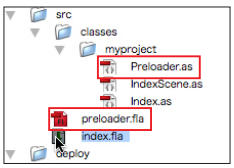
すべてのボタンクラスが完成したらIndexクラス（index.as）を開いて、ボタンを画面に表示させるコードを追記します（赤字部分）。ボタンが表示できたらindex.flaをプレビューし、実際にボタンが機能すること、シーンを移動すること、画面が正しく変更されることを確認してください。

```
protected override function _onInit():void {

中略

        // IndexButton を作成
        var indexButton:IndexButton = new IndexButton();
        indexButton.x = 514;
        addChild( indexButton );

        // FeatureButton を作成
        var featureButton:FeatureButton = new FeatureButton();
        featureButton.x = 568;
        addChild( featureButton );

        // ContactButton を作成
        var contactButton:ContactButton = new ContactButton();
        contactButton.x = 657;
        addChild( contactButton );
}
```

【Index.asファイルでの変更例】

STEP 10 プリローダーを作成する

プリローダーを作成するには、プロジェクト内のsrcフォルダ内にあるpreloader.fla を編集します。Preloaderクラスは CastPreloader クラスを継承しており、Indexクラスと同様に_onInit() イベントハンドラメソッドで初期化処理を設定することができます。Preloader.asを開き、このタイミングで読み込み状態を表示するためのテキストフィールドを作成します。



プリローダーを作成するには、preloader.flaとpreloader.asを編集します

```
public var _loadedPerField:TextField;


protected override function _onInit():void {
        // stage の初期設定
        align = StageAlign.TOP_LEFT;
        quality = StageQuality.HIGH;
        scaleMode = StageScaleMode.NO_SCALE;


        // TextField を作成
        _loadedPerField = new TextField();
        _loadedPerField.x = 280;
        _loadedPerField.y = -100;
        _loadedPerField.width = 200;
        _loadedPerField.autoSize = TextFieldAutoSize.CENTER;
        _loadedPerField.text = "Loading ... 0%";
        addChild( _loadedPerField );
}
```

次にポイントとなるのは _onCastLoadStart() イベントハンドラメソッドです。**このメソッドはプリローダーがメインのコンテンツファイルの読み込みを開始した瞬間に実行され、このメソッドの実行中はシーンオブジェクトやキャストオブジェクトと同様に addCommand() メソッドによるコマンド処理が実行可能です。このイベントは、主にプログレスバー表示の際のアニメーション処理などを行うものとして設計されています。**今回は右のように設定します。

```
protected override function _onCastLoadStart():void {
        // 実行したいコマンドを登録
        addCommand(
                // テキストフィールドを画面内に移動
                new DoTweener( _loadedPerField, { y:200, time:1 } )
        );
}
```

_onCastLoadStart() イベントハンドラメソッドで登録したコマンドがすべて実行完了すると、実際に読み込み処理が監視されます。読み込みが開始されると _onProgress() イベントハンドラメソッドが呼ばれるようになります。ただし、このイベント中には _onCastLoadStart() イベントハンドラメソッドのように addCommand() メソッドを使用することはできません。これは _onProgress() メソッドが Event.ENTER_FRAME のように逐次呼ばれるイベントであるため、時間のかかる非同期処理を設定しても意味がないための仕様です。今回は右のようにテキストフィールドに表示している読み込み状態を更新していくことにしましょう。

```
protected override function _onProgress():void {
        // テキストフィールドの表示内容を、読み込み状態に応じて変化させる
        _loadedPerField.text = "Loading ... " + Math.round( bytesLoaded / bytesTotal *
100 ) + "%";
}
```

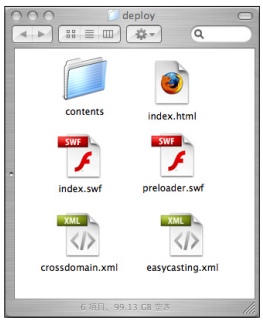
最後に _onCastLoadComplete() イベントハンドラメソッドを使用して、読み込み完了後の処理を設定します。このイベントの際には _onCastLoadStart() メソッドの時と同様に addCommand() メソッドが使用可能になります。読み込まれた swf ファイルは、_onCastLoadComplete() イベントハンドラメソッドが完了し、登録されたコマンドがすべて実行終了したタイミングで画面に表示されます。

```
protected override function _onCastLoadComplete():void {
        // 実行したいコマンドを登録
        addCommand(
                // テキストフィールドを画面外に移動
                new DoTweener( _loadedPerField, { y:500, time:1 } ),

                // テキストフィールドを画面から削除
                new RemoveChild( this, _loadedPerField )
        );
}
```

STEP 11 プロジェクトを公開する

制作作業は完了したら、あとは公開するだけです。まず、index.fla と preloader.fla の両方のファイルにおいて、「Progression プロジェクト」パネルからパブリッシュを行います。あとは、プロジェクト内の deploy フォルダの中身一式をそのままサーバにアップロードすれば作業は完了です。

「Progression プロジェクト」パネルからパブリッシュしてください



プロジェクト内の deploy フォルダの中身全部をサーバにアップロードします

AS 2.0 以前のスタイルで開発したければ「タイムラインスタイル」

column

Progression のクラススタイルは ActionScript 3.0 をベースとした開発手法ですが、まだ 3.0 に移行していない人もいるでしょう。そういう方のために用意されているのが「タイムラインスタイル」です。タイムラインスタイルでは、フレームアクションを基本として、タイムライン・アニメーションと組み合せながら、ActionScript 2.0 のような感覚で開発を行うことができます。

タイムラインスタイルによるスクリプティング手法の詳細については、Progression 公式サイトのドキュメントを参照してください。また、cellfusion さんがタイムスタイルを使った素晴らしいサンプルサイトのデータを公開していますので、こちらも参考にするといいでしょう。

Progression 公式サイトにあるタイムラインスタイルガイド
http://progression.jp/ja/doc/tutorial/timeline/

cellfusion さんによるタイムラインスタイルのサンプル
http://trac.progression.jp/browser/examples/cellfusion