ACTIONSCRIPT™ 3.0 のプログラミング

Adobe®

© 2007 Adobe Systems Incorporated. All rights reserved.

ActionScript™ 3.0 のプログラミング

本マニュアルが、エンドユーザー使用許諾契約付きのソフトウェアと一緒に配布される場合には、本マニュアル、および本マニュアルに記載されているソフトウェアは、ライセンスの所有者にのみ供給され、同ライセンスの条項に従っている場合にのみ使用またはコピーすることが許されます。当該エンドユーザー使用許諾契約により許可されている場合を除き、本マニュアルのいかなる部分といえども、Adobe Systems Incorporated (アドビ システムズ社) の書面による事前の許可なしに、電子的、機械的、録音、その他いかなる形式・手段であれ、複製、検索システムへの保存、または伝送を行うことはできません。本マニュアルの内容は、エンドユーザー使用許諾契約を含むソフトウェアと共に提供されていない場合であっても、著作権法により保護されていることにご留意ください。

本マニュアルに記載される内容は、あくまでも参照用としてのみ使用されること、また、なんら予告なしに変更されることを条件として、提供されるものであり、従って、当該情報が、アドビ システムズ社による確約として解釈されてはなりません。アドビ システムズ社は、本マニュアルにおけるいかなる誤りまたは不正確な記述に対しても、いかなる義務や責任を負うものではありません。

新しいアートワークを創作するためにテンプレートとして取り込もうとする既存のアートワークまたは画像は、著作権法により保護されている可能性のあるものであることをご留意ください。当該アートワークまたは画像を新しいアートワークに許可なく取り込んだ場合、著作権者の権利を侵害することがあります。従って、著作権者から必要なすべての許可を必ず取得してください。

例として使用されている会社名は、実在の会社・組織を示すものではありません。

Adobe、Adobe ロゴ、Flex、Flex Builder、および Flash Player は、アドビ システムズ社の米国ならびに他の国における登録商標または商標です。

ActiveX and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.Macintosh is a trademark of Apple Inc., registered in the United States and other countries.All other trademarks are the property of their respective owners.

Speech compression and decompression technology licensed from Nellymoser, Inc. (www.nellymoser.com).

Sorenson Spark™ video compression and decompression technology licensed from Sorenson Media, Inc.

Opera® browser Copyright © 1995-2002 Opera Software ASA and its suppliers. All rights reserved.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. government end users. The software and documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States.Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA.For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

目 次

本マニュアルについて 13
本マニュアルの使い方 13
ActionScript マニュアルへのアクセス 14
ActionScript の学習用リソース 16

第 1 章 : ActionScript 3.0 の概要 17
ActionScript について 17
ActionScript 3.0 の利点 18
ActionScript 3.0 の新機能 18
コア言語機能 18
Flash Player API 機能 20
旧バージョンとの互換性 22

第 2 章 : ActionScript の使用について 25
プログラミングの基礎 26
コンピュータプログラムの内容 26
変数と定数 26
データ型 27
オブジェクトの操作 28
プロパティ 29
メソッド 30
イベント 31
基本的なイベント処理 31
イベント処理プロセスの確認 32
イベント処理の例 36
オブジェクトインスタンスの作成 37
共通のプログラム要素 39
例 : アニメーションポートフォリオピース 41
ActionScript によるアプリケーションの構築 44
コードの構成オプション 44
適切なツールの選択 47
ActionScript の開発プロセス 49

カスタムクラスの作成 50
クラスの設計戦略. 50
クラスのコードの作成 51
クラスの構成に関する提案 53
例 : 基本的なアプリケーションの作成 53
後続の例の実行 60

第 3 章 : ActionScript 言語とシンタックス 63

言語の概要 64
オブジェクトとクラス 65
パッケージと名前空間 66
パッケージ. 66
名前空間. 71
変数 78
データ型 82
型チェック. 83
ダイナミッククラス 87
データ型の記述. 89
型変換. 92
シンタックス 97
演算子 102
条件演算 109
ループ 112
関数 115
基本的な関数の概念 115
関数パラメータ. 120
オブジェクトとしての関数 125
関数のスコープ. 127

第 4 章 : ActionScript のオブジェクト指向プログラミング 129

オブジェクト指向プログラミングの基礎. 130
クラス 132
クラス定義. 132
クラスプロパティの属性 135
変数. 138
メソッド. 139
クラスによる列挙. 146
埋め込みアセットクラス 148
インターフェイス 148
継承 152
高度なトピック 161
例 : GeometricShapes 170

第 5 章 : 日付と時刻の操作 179
日付と時刻の基礎 179
カレンダー日付と時刻の管理 180
タイマー間隔の制御 184
例 : 単純なアナログ時計 186

第 6 章 : ストリングの操作 191
ストリングの基礎 192
ストリングの作成 194
length プロパティ 195
ストリング内の文字の操作 196
ストリングの比較 197
他のオブジェクトのストリング表現の取得 197
ストリングの連結 198
ストリング内のサブストリングとパターンの検索 198
ストリングの大文字と小文字の変換 203
例 : ASCII アート 204

第 7 章 : 配列の操作 211
配列の基礎 212
インデックス配列 214
結合配列 222
多次元配列 226
配列のコピー 229
高度なトピック 230
Example: PlayList 235

第 8 章 : エラー処理 241
エラー処理の基礎 242
エラーの種類 245
ActionScript 3.0 におけるエラー処理 247
ActionScript 3.0 のエラー処理要素 247
エラー処理のアプローチ 248
デバッガ版の Flash Player の操作 249
アプリケーションの同期エラーの処理 249
独自のエラークラスの作成 255
エラーイベントへの応答とステータス 256
Error クラスの比較 259
ECMAScript コア Error クラス 260
ActionScript コア Error クラス 262
flash.error パッケージ Error クラス 263
例 : CustomErrors アプリケーション 265

第 9 章 : 正規表現の使用 273
正規表現の基礎 274
正規表現のシンタックス 276
正規表現のインスタンスの作成 277
文字、メタ文字、およびメタシーケンス 278
文字クラス 281
繰り返し制御文字 283
選択制御 284
グループ 285
フラグとプロパティ 288
ストリングに対して正規表現を使用するメソッド 292
例 : Wiki パーサー 293

第 10 章 : イベントの処理 299
イベント処理の基本 300
ActionScript 3.0 イベント処理の以前のバージョンとの違い 303
イベントフロー 306
イベントオブジェクト 308
イベントリスナー 313
例 : Alarm Clock 320

第 11 章 : XML の操作 327
XML の基礎 328
E4X による XML 処理 332
XML オブジェクト 334
XMLList オブジェクト 336
XML 変数の初期化 338
XML オブジェクトの構築と変形 339
XML 階層構造内の移動 340
XML 名前空間の使用 346
XML の型変換 347
外部 XML ドキュメントの読み込み 349
インターネットから RSS データをロードする 349

第 12 章 : 表示のプログラミング 353
表示のプログラミングの基本 354
コア表示クラス 358
表示リストによるアプローチの利点 360

表示オブジェクトの操作 363
DisplayObject クラスのプロパティとメソッド 363
表示リストへの表示オブジェクトの追加 364
表示オブジェクトコンテナの操作 364
表示リスト内の移動 368
Stage プロパティの設定 370
表示オブジェクトのイベントの処理 374
DisplayObject サブクラスの選択 375
表示オブジェクトの操作 376
位置の変更 376
表示オブジェクトのパンとスクロール 382
サイズの変更とオブジェクトの拡大 / 縮小 383
拡大 / 縮小時の歪みの制御 385
表示オブジェクトのキャッシュ 387
キャッシュを有効にするシナリオ 388
ビットマップキャッシュの有効化 389
不透明な背景色の設定 390
ブレンドモードの適用 390
DisplayObject の色の調整 391
コードを使用したカラー値の設定 392
コードを使用したカラーおよび明度効果の変更 393
オブジェクトの回転 394
オブジェクトのフェード 394
表示オブジェクトのマスク 395
オブジェクトのアニメーション化 398
表示コンテンツの動的ロード 400
表示オブジェクトのロード 400
ロードの進行状況の監視 401
ロードするコンテンツの指定 402
例 : SpriteArranger 404

第 13 章 : ジオメトリの操作 411

ジオメトリの基本 411
Point オブジェクトの使用 414
Rectangle オブジェクトの使用 416
Matrix オブジェクトの使用 420
例 : マトリックス変換の表示オブジェクトへの適用 421

第 14 章 : 描画 API の使用 427
描画 API の使用の基礎. 428
Graphics クラスについて 429
線と曲線の描画 430
ビルトインメソッドを使用したシェイプの描画. 433
グラデーションの線と塗りの作成 434
描画メソッドでの Math クラスの使用 438
描画 API でのアニメーション化 439
例 : Algorithmic Visual Generator. 440

第 15 章 : 表示オブジェクトのフィルタ処理 443
表示オブジェクトのフィルタ処理の基礎. 443
フィルタの作成と適用 445
新しいフィルタの作成 445
フィルタの適用. 446
フィルタのしくみ. 448
フィルタ操作の潜在的な問題 448
使用可能な表示フィルタ 450
ベベルフィルタ. 451
ぼかしフィルタ. 452
ドロップシャドウフィルタ 453
グローフィルタ. 453
グラデーションベベルフィルタ 454
グラデーショングローフィルタ 455
例 : 基本的なフィルタの組み合わせ 456
カラーマトリックスフィルタ 459
畳み込みフィルタ. 459
置き換えマップフィルタ 462
例 : フィルタワークベンチ. 468

第 16 章 : ムービークリップの操作 469
ムービークリップの基礎 470
MovieClip オブジェクトの操作 472
ムービークリップの再生の制御 472
シーンの操作. 475
ActionScript での MovieClip オブジェクトの作成 475
ActionScript に対するライブラリシンボルの書き出し 476
外部の SWF ファイルのロード 479
例 : RuntimeAssetsExplorer 480

第 17 章 : テキストの操作 485
テキストの操作の基礎 486
テキストの表示 489
テキストのタイプ 489
テキストフィールドのコンテンツの変更 489
HTML テキストの表示. 490
テキストフィールドでのイメージの使用 491
テキストフィールド内のテキストのスクロール 491
テキストの選択と操作 493
テキスト入力のキャプチャ 494
テキスト入力の制限 496
テキストのフォーマット 496
テキストフォーマットの割り当て 497
CSS の適用 497
外部 CSS ファイルのロード 499
テキストフィールド内のテキストの範囲のフォーマット 500
高度なテキストレンダリング 501
静止テキストの操作 504
例 : 新聞形式のテキストフォーマット 505
外部 CSS ファイルの読み取り 506
ページへの記事要素の配置 508
フィールドサイズに合わせるためのフォントサイズ変更 509
複数の段へのテキストの分割 511

第 18 章 : ビットマップの操作 515
ビットマップの操作の基礎 516
Bitmap および BitmapData クラス 519
ピクセルの操作 521
個々のピクセルの操作 521
ピクセルレベルの衝突検出 523
ビットマップデータのコピー 524
ノイズ関数によるテクスチャの作成 525
ビットマップのスクロール 527
例 : 画面外ビットマップを使用してスプライトをアニメーション化する 528

第 19 章 : ビデオの操作 529
ビデオの基礎 530
FLV 形式について 533
Video クラスについて 534
ビデオファイルのロード 534

ビデオ再生の制御 535
ビデオストリームの末尾の検知 537
ビデオファイルのストリーミング 538
キューポイントについて 538
onCuePoint および onMetaData のコールバックメソッドの記述 539
NetStream オブジェクトの client プロパティをオブジェクトに設定する 540
カスタムクラスの作成とコールバックメソッドを処理するメソッドの定義 541
NetStream クラスの拡張とコールバックメソッドを処理するメソッドの追加 542
NetStream クラスの拡張と動的化 543
NetStream オブジェクトの client プロパティに対する this の設定 544
キューポイントの使用 545
ビデオメタデータの使用 546
カメラ入力のキャプチャ 549
Camera クラスについて 549
画面上でのカメラの表示 550
カメラアプリケーションの設計 550
ユーザーのカメラへの接続 551
カメラのインストールの確認 551
カメラのアクセス許可の検知 552
ビデオ品質の最大化 554
再生状況の監視 555
サーバーへのビデオ送信 556
高度なトピック 557
Flash Player とエンコードされた FLV ファイルとの互換性 557
サーバーでホスティングするための FLV ファイルの設定 557
Macintosh のローカル FLV ファイルのターゲット化 558
例 : ビデオジュークボックス 559

第 20 章 : サウンドの操作 565

サウンドの操作の基礎 566
サウンドアーキテクチャについて 568
外部サウンドファイルのロード 570
埋め込みサウンドの操作 573
ストリーミングサウンドファイルの操作 574
サウンドの再生 575
サウンドの一時停止と再開 576
再生の監視 576
ストリーミングサウンドの停止 578
サウンドのロード時および再生時のセキュリティ重要事項 578

サウンドボリュームとパンの制御 580
サウンドメタデータの操作 582
生のサウンドデータへのアクセス 583
サウンド入力のキャプチャ 586
マイクへのアクセス 586
マイクオーディオのローカルスピーカーへのルーティング 588
マイクオーディオの変更 588
マイクのアクティビティの検知 589
メディアサーバーとのオーディオの送受信 590
例 : Podcast Player. 591
ポッドキャストチャンネルの RSS データの読み取り 592
SoundFacade クラスを使用した、サウンドのロードと再生の簡便化 . . . 592
再生の進行状況の表示 597
再生の一時停止と再開 597
Podcast Player の拡張例 598

第 21 章 : ユーザー入力のキャプチャ 599

ユーザー入力の基礎 599
キーボード入力のキャプチャ 601
マウス入力のキャプチャ 603
例 : WordSearch. 608

第 22 章 : ネットワーキングとコミュニケーション 613

ネットワーキングとコミュニケーションの基本. 614
外部データの操作 617
他の Flash Player インスタンスへの接続 623
ソケットの接続 629
ローカルデータの保存 635
ファイルのアップロードおよびダウンロード操作. 638
例 : Telnet クライアントの構築 649
例 : ファイルのアップロードとダウンロード 652

第 23 章 : クライアントのシステム環境 661

クライアントのシステム環境の基礎. 662
System クラスの使用 664
Capabilities クラスの使用. 665
ApplicationDomain クラスの使用 666
IME クラスの使用. 670
例 : システム機能の検出 675

第 24 章 : プリント 681
プリントの基本 682
ページのプリント 683
Flash Player タスクとシステムプリント 684
サイズ、拡大率、用紙の向きの設定 687
例 : 複数ページのプリント 690
例 : 拡大 / 縮小、トリミング、および応答 692

第 25 章 : External API の使用 695
External API 使用の基本 696
External API の要件と利点 699
ExternalInterface クラスの使用 700
外部コンテナに関する情報の取得 700
ActionScript からの外部コードの呼び出し 701
コンテナからの ActionScript コードの呼び出し 702
External API の XML フォーマット 703
例 : Web ページコンテナに対する External API の使用 705
例 : ActiveX コンテナに対する External API の使用 712

第 26 章 : Flash Player セキュリティ 719
Flash Player セキュリティの概要 720
アクセス許可管理の概要 722
セキュリティサンドボックス 732
ネットワーク API の制限 735
フルスクリーンモードのセキュリティ 737
コンテンツのロード 738
クロススクリプト 742
ロードされたメディアへのデータとしてのアクセス 746
データのロード 749
セキュリティドメインに読み込まれた SWF ファイルからの
埋め込みコンテンツのロード 752
古いコンテンツの操作 752
LocalConnection 許可の設定 753
ホスト Web ページのスクリプトへのアクセスの制御 754
共有オブジェクト 755
カメラ、マイク、クリップボード、マウス、キーボードのアクセス 757

索 引 759

本マニュアルについて

本マニュアルでは、ActionScript™ 3.0 でアプリケーションを開発するための基本情報を提供します。ここで説明する考え方や手法を理解するには、データ型、変数、ループ、関数などの一般的なプログラミング概念についての知識が前提となります。また、クラスや継承などのオブジェクト指向のプログラミング概念についても理解している必要があります。ActionScript 1.0 や ActionScript 2.0 に関する知識は有効ですが、必須ではありません。

目次

本マニュアルの使い方 13
ActionScript マニュアルへのアクセス 14
ActionScript の学習用リソース 16

本マニュアルの使い方

本マニュアルの章は、ActionScript マニュアルの関連箇所を簡単に見つけられるように、次の論理グループに分割されています。

章	説明
第 1 ～ 4 章、ActionScript のプログラミングの概要	言語のシンタックス、ステートメントと演算子、ECMAScript Edition 4 言語仕様案、オブジェクト指向の ActionScript プログラミング、Adobe® Flash® Player 9 表示リストでの表示オブジェクトの新しい管理方法など、ActionScript 3.0 の主要な概念について説明します。
第 5 ～ 10 章、ActionScript 3.0 の主要なデータ型とクラス	ECMAScript 仕様案の一部でもある ActionScript 3.0 の最上位のデータ型について説明します。
第 11 ～ 26 章、Flash Player API	イベント処理、ネットワーキングおよび通信、ファイル入力および出力、外部インターフェイス、アプリケーションセキュリティモデルなど、Adobe Flash Player 9 に固有のパッケージやクラスに実装された重要な機能について説明します。

また、本マニュアルには、重要なクラスやよく使われるクラスについて、アプリケーションプログラミングの概念を示す多くのサンプルファイルが含まれています。サンプルファイルは、Adobe® Flash® CS3 Professional に簡単にロードして使用できるようにパッケージ化されており、ラッパーファイルが含まれている場合もあります。ただし、中心的に使用するサンプルコードは ActionScript 3.0 で作成されたもので、任意の開発環境で使用できます。

ActionScript 3.0 は、次のようないくつかの方法で記述し、コンパイルできます。

- Adobe Flex Builder 2 開発環境を使用する
- Flex Builder 2 に付属のテキストエディタやコマンドラインコンパイラなどを使用する
- Adobe® Flash® CS3 Professional オーサリングツールを使用する

ActionScript 開発環境の詳細については、第 1 章の「ActionScript 3.0 の概要」を参照してください。

本マニュアルのサンプルコードを理解するために、Flex Builder や Flash オーサリングツールなど、ActionScript の統合開発環境の使用経験は必要ありません。ただし、これらのツールのマニュアルを参照し、これらのツールを使用して ActionScript 3.0 コードの記述とコンパイルの方法を学習することもできます。詳細については、14 ページの「ActionScript マニュアルへのアクセス」を参照してください。

ActionScript マニュアルへのアクセス

本マニュアルでは、リッチでパワフルなオブジェクト指向プログラミング言語である ActionScript 3.0 の説明に重点を置いているので、特定のツールのアプリケーション開発プロセスやワークフロー、またはサーバーアーキテクチャについては広く取り上げません。したがって、ActionScript 3.0 アプリケーションの設計、開発、テスト、およびデプロイの際には、『ActionScript 3.0 のプログラミング』に加えて、他のマニュアルも参照してください。

ActionScript 3.0 のマニュアル

本マニュアルでは、ActionScript 3.0 プログラミング言語の概念を示し、実装の詳細と重要な言語機能を示すサンプルについて説明します。ただし、本マニュアルは完全な言語リファレンスではありません。言語リファレンスとしては、ActionScript 3.0 コンポーネントリファレンスガイド を参照してください。この言語のすべてのクラス、メソッド、プロパティ、およびイベントが説明されています。ActionScript 3.0 コンポーネントリファレンスガイド には、コア言語、Flash のコンポーネント (fl パッケージ)、および Flash Player API (flash パッケージ) に関する詳細な参照情報が記載されています。

Flash のマニュアル

Flash 開発環境を使用する場合、以下のマニュアルを参照できます。

マニュアル名	説明
Flash ユーザーガイド	Flash オーサリング環境で動的な Web アプリケーションを開発する方法について説明します。
ActionScript 3.0 のプログラミング	ActionScript 3.0 言語とコア Flash Player API の具体的な使用方法について説明します。
ActionScript 3.0 コンポーネントリファレンスガイド	Flash コンポーネントと ActionScript 3.0 API のシンタックス、用途、およびコード例を提供します。
ActionScript 3.0 コンポーネントガイド	コンポーネントを使用して Flash アプリケーションを開発する方法の詳細について説明します。
Adobe Flash ActionScript 2.0 の学習	ActionScript 2.0 のシンタックスの概要を示し、ActionScript 2.0 を使用してさまざまなタイプのオブジェクトを操作する方法について説明します。
ActionScript 2.0 リファレンスガイド	Flash コンポーネントと ActionScript 2.0 API のシンタックス、用途、およびコード例を提供します。
ActionScript 2.0 コンポーネントガイド	ActionScript 2.0 コンポーネントを使用して Flash アプリケーションを開発する方法について詳しく説明します。
ActionScript 2.0 コンポーネントリファレンスガイド	Adobe Component Architecture Version 2 で使用可能な各コンポーネントについて、API と共に説明します。
Flash の拡張機能	JavaScript API で使用可能なオブジェクト、メソッド、およびプロパティについて説明します。
Flash Lite 2.x ファーストステップガイド	Adobe® Flash® Lite™ 2.x を使用してアプリケーションを開発する方法について説明し、Flash Lite 2.x で使用可能な ActionScript 機能のシンタックス、用途、およびコード例を提供します。
Flash Lite 2.x アプリケーションの開発	Flash Lite 2.x アプリケーションの開発方法について説明します。
Flash Lite 2.x ActionScript 入門	Flash Lite 2.x でアプリケーションを開発する方法の概要、および Flash Lite 2.x の開発者が使用できるすべての ActionScript 機能について説明します。
Flash Lite 2.x ActionScript リファレンスガイド	Flash Lite 2.x で使用可能な ActionScript 2.0 API のシンタックス、用途、およびコード例を提供します。

マニュアル名	説明
Flash Lite 1.x ファーストステップガイド	Flash Lite 1.x の概要を示し、Adobe® Device Central CS3 エミュレータを使用してコンテンツをテストする方法について説明します。
Flash Lite 1.x アプリケーションの開発	Flash Lite 1.x を使用してモバイルデバイス向けのアプリケーションを開発する方法について説明します。
Flash Lite 1.x ActionScript の学習	Flash Lite 1.x アプリケーションで ActionScript を使用する方法、および Flash Lite 1.x で使用可能なすべての ActionScript 機能について説明します。
Flash Lite 1.x ActionScript リファレンスガイド	Flash Lite 1.x で使用可能な ActionScript 要素のシンタックスと用途について説明します。

ActionScript の学習用リソース

これらのマニュアルの内容に加えて、記事、デザイン案、サンプルを定期的に更新し、アドビデベロッパーセンターとアドビデザインセンターで提供しています。

アドビデベロッパーセンター

アドビデベロッパーセンターでは、ActionScript の最新情報、実際のアプリケーション開発に関する記事、新たに発生した重要な問題に関する情報を提供しています。デベロッパーセンター (www.adobe.com/jp/devnet/) を参照してください。

アドビデザインセンター

最新のデジタルデザインやモーショングラフィックを紹介しています。このサイトでは、最先端のアーティストによる作品を参照し、新しいデザインのトレンドを見つけることができます。また、チュートリアル、主要なワークフロー、高度な手法を参考にしてスキルを磨くこともできます。チュートリアルと記事、またインスピレーションが得られるギャラリーの作品を月に 2 回更新しています。デザインセンター (www.adobe.com/jp/designcenter/) を参照してください。

第 1 章

ActionScript 3.0 の概要

1

この章では、ActionScript の最も画期的な最新バージョンである ActionScript 3.0 の概要を説明します。

目次

ActionScript について 17
ActionScript 3.0 の利点 18
ActionScript 3.0 の新機能 18
旧バージョンとの互換性 22

ActionScript について

ActionScript は、Adobe Flash Player 実行時環境用のプログラミング言語で、Flash コンテンツおよびアプリケーションでのユーザー操作やデータ処理などを可能にします。

ActionScript は、Flash Player の一部である ActionScript 仮想マシン (AVM) により実行されます。通常、ActionScript コードは、Adobe Flash CS3 Professional や Adobe® Flex™ Builder™ に内蔵されているコンパイラや、Adobe® Flex™ SDK や Flex™ Data Services で使用可能なコンパイラにより、" バイトコード形式 " (コンピュータにより書き込まれ、理解される一種のプログラミング言語) にコンパイルされます。バイトコードは、Flash Player 実行時環境で実行される SWF ファイルに埋め込まれます。

ActionScript 3.0 は、オブジェクト指向プログラミングの基礎知識を持つ開発者にはなじみのある堅牢なプログラミングモデルを提供します。ActionScript 3.0 には次のような機能があります。

- AVM2 という新しい ActionScript 仮想マシン。新しいバイトコード命令セットを使用し、パフォーマンスを大幅に向上させます。
- 最新のコンパイラコードベース。ECMAScript (ECMA 262) 規格にさらに忠実に準拠し、旧バージョンのコンパイラより深いレベルで最適化を実行します。
- 拡張および改良されたアプリケーションプログラミングインターフェイス (API)。オブジェクトの低レベルコントロールと真のオブジェクト指向モデルを備えています。

- ECMAScript (ECMA-262) Edition 4 言語仕様提案に準拠したコア言語。
- ECMAScript for XML (E4X) 仕様 (ECMA-357 edition 2) に準拠した XML API。E4X は、言語のネイティブデータ型として XML を追加する ECMAScript 言語拡張です。
- ドキュメントオブジェクトモデル (DOM) Level 3 Events 仕様に準拠したイベントモデル。

ActionScript 3.0 の利点

ActionScript 3.0 は、旧バージョンの ActionScript のスクリプト機能を上回っています。大容量のデータセットおよびオブジェクト指向の再利用可能なコードベースを使用する非常に複雑なアプリケーションを容易に作成できるように設計されています。ActionScript 3.0 は、Adobe Flash Player 9 で実行されるコンテンツには必要ありませんが、新しい仮想マシン AVM2 でのみ可能なパフォーマンスの向上を実現します。ActionScript 3.0 コードは、従来の ActionScript コードより最大で 10 倍高速に実行できます。

旧バージョンの ActionScript 仮想マシン AVM1 は、ActionScript 1.0 および ActionScript 2.0 のコードを実行します。AVM1 は、Flash Player 9 によって既存のコンテンツおよび古いコンテンツとの後方互換性がサポートされています。詳細については、22 ページの「旧バージョンとの互換性」を参照してください。

ActionScript 3.0 の新機能

ActionScript 3.0 には ActionScript のプログラマにわかりやすい多くのクラスおよび機能が含まれていますが、ActionScript 3.0 は旧バージョンの ActionScript とはアーキテクチャ上および概念上異なります。ActionScript 3.0 の強化点には、コア言語の新機能および低レベルオブジェクトの制御を高める改良された Flash Player API があります。

コア言語機能

コア言語は、ステートメント、式、条件、ループ、型などのプログラミング言語の基本的な要素を定義します。ActionScript 3.0 には、開発プロセスの効率化を促す多くの新機能が備わっています。

ランタイム例外

ActionScript 3.0 では、旧バージョンの ActionScript より多くのエラー状態が報告されます。ランタイム例外を一般的なエラー状態に使用すると、デバッグの操作性が向上し、エラーを確実に処理するアプリケーションを開発することができます。ランタイムエラーには、ソースファイルおよび行番号情報で注釈を付けたスタックトレースがあり、エラーをすばやく特定できます。

ランタイム型

ActionScript 2.0 では、型注釈は主に開発者を補助するもので、実行時にすべての値が動的に入力されました。ActionScript 3.0 では、型情報は実行時に維持され、さまざまな目的に使用されます。Flash Player 9 はランタイム型チェックを行い、システムの型安全性を向上させます。型情報はネイティブのマシン表現で変数を表すためにも使用され、パフォーマンスを向上させ、メモリの使用量を削減します。

sealed クラス

ActionScript 3.0 には、sealed クラスの概念が導入されています。sealed クラスには、コンパイル時に定義された一定のプロパティとメソッドだけが含まれ、その他のプロパティやメソッドを追加することはできません。このため、より厳密にコンパイル時のチェックを行うことができ、堅牢性の高いプログラムを作成できます。また、各オブジェクトインスタンスの内部ハッシュテーブルが不要になり、メモリの使用量を削減できます。ダイナミッククラスも `dynamic` キーワードを使用して使用可能です。ActionScript 3.0 のクラスはすべてデフォルトで sealed クラスになりますが、`dynamic` キーワードを使用してダイナミッククラスとして宣言することもできます。

メソッドクロージャ

ActionScript 3.0 では、元のオブジェクトインスタンスを自動的に記憶するメソッドクロージャが有効です。この機能はイベント処理に役立ちます。ActionScript 2.0 では、メソッドクロージャに抽出元のオブジェクトインスタンスが記憶されず、メソッドクロージャが呼び出されたときに予期しない動作が発生しました。mx.utils.Delegate クラスは一般的な方法でしたが、不要になりました。

ECMAScript for XML (E4X)

ActionScript 3.0 は、最近 ECMA-357 として標準化された ECMAScript for XML (E4X) を実装しています。E4X は、XML を操作するための自然で滑らかな言語コンストラクトを提供します。従来の XML 解析 API とは異なり、E4X の XML は、その言語のネイティブデータ型のように動作します。E4X は、必要なコード量を大幅に削減して、XML を操作するアプリケーションの開発を簡略化します。E4X の ActionScript 3.0 実装の詳細については、327 ページ、第 11 章の「XML の操作」を参照してください。

ECMA の E4X 仕様を確認するには、www.ecma-international.org を参照してください。

正規表現

ActionScript 3.0 では正規表現がネイティブにサポートされているので、ストリングをすばやく検索し、操作できます。ActionScript 3.0 は、ECMAScript Edition 3 言語仕様 (ECMA-262) で定義されている正規表現のサポートを実装しています。

名前空間

名前空間は、宣言 (`public`、`private`、`protected`) の可視性を制御するために使用する通常のアクセス指定子に似ています。名前空間は、カスタムアクセス指定子として動作し、任意の名前を指定できます。名前空間には、競合を回避するために URI (Universal Resource Identifier) が割り当てられています。また、E4X を使用するときに XML 名前空間を表すためにも使用します。

新しいプリミティブ型

ActionScript 2.0 の数値型は、倍精度浮動小数点数の Number 型のみですが、ActionScript 3.0 には、int 型と uint 型が含まれます。int 型は、ActionScript コードが CPU の高速整数演算機能を活用できる 32 ビット符号付き整数で、整数が使用されるループカウンタや変数に便利です。uint 型は符号なしの 32 ビット整数型で、RGB カラー値、バイトカウントなどに便利です。

Flash Player API 機能

ActionScript 3.0 の Flash Player API には、オブジェクトを低いレベルで制御できる多数の新しいクラスが追加されています。ActionScript 3.0 のアーキテクチャはまったく新しく、より直観的です。新しく追加されたクラスが多いため、ここでは詳しく説明しませんが、次のセクションでいくつかの大幅な変更について取り上げます。

DOM3 イベントモデル

DOM3 (Document Object Model Level 3) イベントモデルを使用すると、アプリケーション内のオブジェクトが互いにやり取りでき、その状態を保持し、変更に応答できるように、イベントメッセージを生成および処理できます。W3C (World Wide Web Consortium) DOM Level 3 Events 仕様に従って作成されたこのモデルは、旧バージョンの ActionScript で使用できるイベントシステムよりわかりやすく効率的なメカニズムを備えています。

イベントおよびエラーイベントは、flash.events パッケージにあります。Flash コンポーネントフレームワークでは Flash Player API と同じイベントモデルを使用するため、イベントシステムは Flash プラットフォーム全体で統一されています。

表示リスト API

Flash Player 表示リスト、つまり Flash アプリケーションのビジュアルエレメントを含むツリーにアクセスするための API は、Flash のビジュアルプリミティブを操作するためのクラスで構成されます。

新しい Sprite クラスは、MovieClip クラスによく似た軽量の構築ブロックですが、UI コンポーネントの基本クラスとして適しています。新しい Shape クラスは、未加工のベクターシェイプを表します。これらのクラスは、`new` 演算子で自然にインスタンス化でき、いつでも動的に再度親にすることができます。

深度の管理は、自動化されて Flash Player に組み込まれているため、深度番号を割り当てる必要がなくなりました。オブジェクトの z 順序を指定し管理するための新しいメソッドが用意されています。

動的なデータおよびコンテンツの処理

ActionScript 3.0 は、Flash アプリケーションでアセットおよびデータをロードし処理するための直観的で API 全体で一貫したメカニズムを備えています。新しい Loader クラスは、SWF ファイルおよびイメージアセットをロードするためのメカニズムを備え、ロードされたコンテンツに関する詳細情報にアクセスするための手段を提供します。URLLoader クラスには、データ駆動アプリケーションでテキストおよびバイナリデータをロードするための個別のメカニズムがあります。Socket クラスは、サーバーソケットに対して任意の形式でバイナリデータを読み取り、書き込む手段を提供します。

低レベルでのデータアクセス

さまざまな API で、これまで ActionScript では使用できなかったデータに低レベルでアクセスできます。ダウンロード中のデータの場合、URLLoader によって実装される URLStream クラスを使用すると、データをダウンロード中に生のバイナリデータとしてデータにアクセスできます。ByteArray クラスを使用すると、バイナリデータの読み取り、書き込み、および操作を最適化できます。新しい Sound API では、SoundChannel および SoundMixer クラスからサウンドを詳細に制御できます。セキュリティを処理する新しい API は、SWF ファイルまたはロードされたコンテンツのセキュリティ権限に関する情報を提供し、セキュリティエラーをより適切に処理できるようにします。

テキストの操作

ActionScript 3.0 には、すべてのテキスト関連 API の flash.text パッケージが含まれています。TextLineMetrics クラスは、テキストフィールド内のテキストの行の詳細なメトリックを提供します。これは、ActionScript 2.0 の `TextField.getLineMetrics()` メソッドに取って代わるものです。TextField クラスには、テキストフィールド内のテキスト 1 行または 1 文字に関する情報を提供できる多数の新しい低レベルメソッドが含まれています。たとえば、文字の境界ボックスを表す矩形を返す `getCharBoundaries()`、指定されたポイントにある文字のインデックスを返す `getCharIndexAtPoint()`、段落の最初の文字のインデックスを返す `getFirstCharInParagraph()` などのメソッドがあります。行レベルのメソッドには、テキストの指定された行の文字数を返す `getLineLength()`、および指定された行のテキストを返す `getLineText()` があります。新しい Font クラスを使用すると、SWF ファイルの埋め込みフォントを管理できます。

旧バージョンとの互換性

Flash Player には、以前にパブリッシュされたコンテンツとの完全な後方互換性があります。旧バージョンの Flash Player で実行されていたコンテンツは、Flash Player 9 で実行できます。しかし、Flash Player 9 に ActionScript 3.0 が導入され、Flash Player 9 を実行する新旧コンテンツの相互運用性に次のような互換性の問題が発生しています。

- 1 つの SWF ファイルでは、ActionScript 1.0 または 2.0 のコードと ActionScript 3.0 コードを結合することはできません。
- ActionScript 3.0 コードは、ActionScript 1.0 または 2.0 で記述されている SWF ファイルをロードできますが、SWF ファイルの変数および関数にアクセスすることはできません。
- ActionScript 1.0 または 2.0 で記述されている SWF ファイルは、ActionScript 3.0 で記述されている SWF ファイルをロードすることはできません。つまり、Flash 8 または Flex Builder 1.5 以前のバージョンで作成された SWF ファイルは、ActionScript 3.0 で記述されている SWF ファイルをロードすることができません。

  この規則には唯一例外があり、以前に ActionScript 2.0 SWF ファイルによってそのどのレベルにも何もロードされていなければ、ActionScript 2.0 SWF ファイル自体を ActionScript 3.0 SWF ファイルで置き換えることができることです。ActionScript 2.0 SWF ファイルは、この置き換えを `loadMovieNum()` の呼び出しによって行うことができ、そのときに値 0 を `level` パラメータに渡します。
- 一般に、ActionScript 1.0 または 2.0 で記述されている SWF ファイルは、ActionScript 3.0 で記述されている SWF ファイルと一緒に使用する場合、移行する必要があります。たとえば、ActionScript 2.0 を使用してメディアプレーヤーを作成したとします。このメディアプレーヤーでは、ActionScript 2.0 を使用して作成されたさまざまなコンテンツをロードできます。しかし、ActionScript 3.0 で新しいコンテンツを作成して、このメディアプレーヤーにロードすることはできません。メディアプレーヤーを ActionScript 3.0 に移行する必要があります。

  ただし、ActionScript 3.0 でメディアプレーヤーを作成すれば、ActionScript 2.0 で作成されたコンテンツを簡単にロードできます。

次の表に、新しいコンテンツのロードおよびコードの実行に対する旧バージョンの Flash Player の制限に関する概要を示します。また、異なるバージョンの ActionScript で記述された SWF ファイル間のクロススクリプトの制限に関する概要も示します。

サポートされる機能	実行時環境		
	Flash Player 7	Flash Player 8	Flash Player 9
ロード可能な SWF のパブリッシュ対象	7 以前	8 以前	9 以前
この AVM を含む	AVM1	AVM1	AVM1 および AVM2
ActionScript で記述された SWF の実行	1.0 および 2.0	1.0 および 2.0	1.0、2.0、および 3.0

サポートされる機能*	コンテンツの作成元	
	ActionScript 1.0 および 2.0	ActionScript 3.0
ロードしてコードを実行できるコンテンツの作成元	ActionScript 1.0 および 2.0 のみ	ActionScript 1.0、2.0、および ActionScript 3.0
クロススクリプト可能なコンテンツの作成元	ActionScript 1.0 および 2.0 のみ†	ActionScript 3.0‡

* Flash Player 9 以降で実行するコンテンツ。Flash Player 8 以前で実行するコンテンツは、ActionScript 1.0 および 2.0 のみロード、表示、実行、クロススクリプトが可能。

† LocalConnection を使用する ActionScript 3.0

‡ LocalConnection を使用する ActionScript 1.0 および 2.0

第 2 章

ActionScript の使用について

2

この章は、初めて ActionScript のプログラミングを操作するユーザーが、以降で示される概念と例を理解するのに必要な背景を説明するために作成されています。最初にプログラミングの基本的な概念を述べますが、ActionScript 内でこのような概念をどのように用いるかという文脈を使って説明します。また、ActionScript アプリケーションを整理し、構築する方法の基礎についても扱います。

目次

プログラミングの基礎 26
オブジェクトの操作 28
共通のプログラム要素 39
例 : アニメーションポートフォリオピース 41
ActionScript によるアプリケーションの構築 44
カスタムクラスの作成 50
例 : 基本的なアプリケーションの作成 53
後続の例の実行 60

プログラミングの基礎

ActionScript はプログラミング言語であるため、ActionScript を学習する場合、コンピュータプログラミングの一般的な概念をいくつか理解しておくと役に立ちます。

コンピュータプログラムの内容

最初に、コンピュータプログラムとは何か、またどのような作業を実行するかについての概念を頭に描くことが大切です。コンピュータプログラムには、2 つの重要な側面があります。

- プログラムとは、コンピュータを実行するための一連の命令または手順です。
- 各手順では最終的に一部の情報またはデータの操作が伴います。

一般的な意味では、コンピュータプログラムは単にステップバイステップ形式の命令のリストであり、これをコンピュータに発行して、コンピュータはこれを 1 つずつ実行します。個々の命令はステートメントとして知られています。本書を通じて、ActionScript では各ステートメントはセミコロンで終了します。

実際には、プログラム内の所定の命令は、コンピュータのメモリ内に格納された一部のデータビットを操作するにすぎません。簡単な例として、2 つの数値を加算し、結果をメモリに格納する命令をコンピュータに指定する場合があります。これよりも複雑な例として、画面に矩形が描かれていると想定した場合、画面の他の場所に移動するプログラムを作成できます。コンピュータは矩形に関する特定の情報、すなわち矩形が配置されている x、y 座標や、矩形の幅と高さ、矩形の色などを追跡します。その情報ビットはそれぞれ、コンピュータのメモリ内に保存されます。矩形を別の場所に移動するプログラムは、x 座標を 200 に変更し、y 座標を 150 に変更するのに似た手順になります。つまり、x と y 座標に使用される新しい値を指定することになります。当然ですが、コンピュータはこのデータを使って、それらの数値をコンピュータ画面上に表示されるイメージに実際に変換する何らかの操作を行いますが、操作の対象となる細部について見ると、画面上の矩形の移動プロセスは、実際にはコンピュータのメモリ内のデータビットを変更する作業に過ぎないことを把握するだけで十分です。

変数と定数

プログラミングは主にコンピュータのメモリ内の断片的な情報を変更する作業であるため、個々の情報片をプログラムで表す方法が必要になります。変数はコンピュータのメモリ内の値を表す名前です。値を操作するステートメントを記述する場合、値の代わりに変数名を記述します。コンピュータがプログラムで変数名を見つけると、メモリを検索し、メモリ内で見つかった値を使用します。たとえば、`value1` と `value2` の 2 つの変数があり、それぞれに数値が 1 つずつ格納されている場合、2 つの数値を追加するステートメントは次のように記述できます。

```
value1 + value2
```

コンピュータが実際に手順を実行する場合は、各変数内の値を探し、あわせて追加します。

ActionScript 3.0 では、実際には変数は 3 つの異なるパートから構成されます。

- 変数の名前
- 変数に格納できるデータ型
- コンピュータのメモリに格納される実際の値

ここまで、コンピュータが値のプレースホルダとして名前をどのように使用するかについて説明してきました。データ型も重要です。ActionScript で変数を作成する場合は、ActionScript 内で保持される特定のデータ型を指定します。この時点から、プログラムの命令を使って変数内の指定したデータ型のみ保存できます。また、そのデータ型に関連付けられた特定の文字を使用して値を操作できます。ActionScript では、変数を作成する場合 (これは変数の宣言と呼ばれる)、`var` ステートメントを使用します。

```
var value1:Number;
```

この場合、`value1` という名前の変数の作成をコンピュータに命令しています。コンピュータは Number 型のデータのみを保持します ("Number" は ActionScript 内で定義される固有のデータ型です)。この直後に値を変数に格納することも可能です。

```
var value2:Number = 17;
```

Adobe Flash CS3 Professional では、変数を宣言する方法がもう 1 つあります。ステージにムービークリップシンボル、ボタンシンボル、またはテキストフィールドを配置した場合、プロパティインスペクタでインスタンス名を指定できます。シーンの裏側では、Flash がインスタンス名と同じ名前で変数を作成します。この変数を ActionScript コードに使用すると、同じステージのアイテムを参照できます。したがって、ステージにムービークリップシンボルを配置し、この記号にインスタンス名 `rocketShip` を指定した場合、ActionScript コードで変数 `rocketShip` を使用すると必ずそのムービークリップを操作することになります。

データ型

ActionScript では、作成する変数のデータ型として使用できる多くのデータ型があります。これらの中には、" 単純な " または " 基本的な " データ型と見なすことができるものがあります。

- String : 名前または本の章の文字に似たテキスト値
- Numeric : ActionScript 3.0 では数値データに 3 種類のデータ型を使用します。
  - Number : 小数点付きまたは小数点なしの値を含むすべての数値
  - int : 整数 (小数点のない自然数)
  - uint : " 符号なし " 整数。すなわち負の数にならない自然数
- Boolean : スイッチのオンかオフ、2 つの値が等しいか否かなどの true または false の値

単純なデータ型は、1つの情報を表します。たとえば、単一の数字またはテキストの1文などです。ただし、ActionScript で定義されるデータ型の大半は、まとめてグループ化された値セットを表すため、複雑なデータ型として記述できます。たとえば、データ型 Date の変数は、1つの値、すなわち特定の時点を表します。ただし、その日付の値は実際には、日、月、年、時間、分、秒など複数の値で表され、すべて個別の数値です。このため、日付は単一の値として考えますが (また Date 変数を作成して単一の値として処理できますが)、コンピュータの内部では、まとめて1つの日付を定義する複数値から成るグループと見なされます。

プログラマが定義するデータ型以外にビルトインデータ型の大半は、複雑なデータ型です。複雑なデータ型と認識できるものは次のとおりです。

- MovieClip : ムービークリップシンボル
- TextField : ダイナミックテキストフィールドまたはテキスト入力フィールド
- SimpleButton : ボタンシンボル
- Date : 特定の時点に関する情報 (日付と時間)

データ型と同じ意味で多く使用される用語に、クラスとオブジェクトがあります。クラスは単にデータ型を定義したものです。Example データ型のすべての変数は特性 A、B、C を持つと宣言するような、特定のデータ型のすべてのオブジェクトにテンプレートを作成するのに似ています。一方、オブジェクトは、クラスの実際のインスタンスです。データ型が MovieClip の変数は、MovieClip オブジェクトと記述できます。したがって、次のように同じ内容を異なる方法で表します。

- 変数 `myVariable` のデータ型は Number。
- 変数 `myVariable` は Number インスタンス。
- 変数 `myVariable` は Number オブジェクト。
- 変数 `myVariable` は Number クラスのインスタンス。

オブジェクトの操作

ActionScript は、オブジェクト指向のプログラミング言語として知られています。オブジェクト指向のプログラミングは、プログラミングアプローチの1つであり、実際には、オブジェクトを使用してプログラムでコードを編成する方法に他なりません。

これまでに、コンピュータプログラムはコンピュータが実行する一連の手順または命令として定義してきました。概念上は、コンピュータプログラムを命令の長い1つのリストとして思い描いてもかまわないでしょう。ただし、オブジェクト指向プログラミングでは、プログラム命令は異なるオブジェクト間で分割され、コードは機能グループにまとめられるため、関連するタイプの機能または関連する情報片は、1つのコンテナにグループ化されます。

事実上、Flash でシンボルを操作する場合、実際にはオブジェクトの操作を行っていることになります。矩形の図などのムービークリップシンボルを定義し、そのコピーをステージに配置していると想像してください。そのムービークリップシンボルは、(実際には) ActionScript のオブジェクトでもあり、MovieClip クラスのインスタンスです。

ムービークリップには、変更可能なさまざまな特性があります。たとえば、ムービークリップが選択されている場合、その x 座標、幅、またはアルファ (透明度) の変更やドロップシャドウフィルタの適用など、ざまざまなカラー調整といった、プロパティインスペクタで変更可能な値があります。他の Flash ツールを使用すると、自由変形ツールを使用して矩形を回転するような、さらに多くの変更が可能になります。Flash オーサリング環境でムービークリップシンボルを変更する場合に実行できるこのような機能はすべて、ActionScript では、MovieClip オブジェクトと呼ばれる単一のバンドルにまとめられたデータを変更して実行します。

ActionScript のオブジェクト指向プログラミングでは、クラスに 3 種類の特性を指定できます。

- プロパティ
- メソッド
- イベント

これらの要素はまとめて使用され、プログラムで使用されるデータを管理し、どのアクションがどの順序で実行されるかを決定します。

プロパティ

プロパティは、オブジェクト内でまとめてバンドルされるデータグループの 1 つを表します。Song オブジェクトには、`artist` や `title` という名前のプロパティを指定できます。MovieClip クラスには、`rotation`、`x`、`width`、`alpha` などのプロパティを指定できます。プロパティは個々の変数と同様に操作します。実際に、プロパティはオブジェクト内に格納された "child" 変数と考えてもよいでしょう。

次に示すのは、プロパティを使用する ActionScript コードの例です。このコード行は、`square` という名前の MovieClip を x 座標まで 100 ピクセル移動させます。

```
square.x = 100;
```

このコードでは、`square` MovieClip を `triangle` MovieClip の回転に合わせて回転させるために、rotation プロパティが使用されています。

```
square.rotation = triangle.rotation;
```

このコードは、`square` MovieClip の水平スケールを変更し、実際の使用時の 1.5 倍の幅になるようにしています。

```
square.scaleX = 1.5;
```

共通の構造として、変数 (`square`、`triangle`) をオブジェクト名として使用し、ピリオド (`.`)、およびプロパティの名前 (`x`、`rotation`、`scaleX`) を続けます。ドット演算子として知られるピリオドは、オブジェクトの子要素のいずれかにアクセスしていることを表すために使用します。全体の構造 " 変数名 - ドット - プロパティ名 " は、単一の変数と同様に使用し、コンピュータのメモリ内の単一の値になります。

メソッド

メソッドは、オブジェクトで実行可能なアクションです。たとえば、Flash でタイムラインに複数のキーフレームおよびアニメーションを配置してムービークリップシンボルを作成した場合、そのムービークリップを再生または停止したり、特定のフレームへの再生フレームの移動を指示したりできます。

次のコードは、shortFilm という名前の MovieClip に再生を開始するように指示しています。

```
shortFilm.play();
```

次の行により、shortFilm という名前の MovieClip は再生を停止します (再生ヘッドは、ビデオの休止と同様に、所定の位置で停止します)。

```
shortFilm.stop();
```

次のコードは、MovieClip shortFilm の再生ヘッドをフレーム 1 まで移動させ、再生を停止させます (ビデオの巻き戻しと同じ操作)。

```
shortFilm.gotoAndStop(1);
```

ここまでの説明でおわかりのように、プロパティと同様にメソッドには、オブジェクト名 (変数)、ピリオド、メソッド名、括弧を記述してアクセスします。括弧はメソッドを呼び出している、つまり、対象となっているアクションの実行をオブジェクトに指示していることを示す方法です。アクションの実行に必要な追加情報を渡す方法として、値 (または変数) は括弧内に置かれる場合があります。これらの値はメソッドパラメータと呼ばれます。たとえば、gotoAndStop( ) メソッドは、移動先のフレームを知る必要があるため、括弧内にパラメータが 1 つ必要になります。play( ) や stop( ) などの他のメソッドは名前から判断可能であるため、他の情報は不要です。ただし、この場合も括弧を使って記述されます。

プロパティ (および変数) と異なり、メソッドは値のプレースホルダとして使用されません。ただし、計算を実行し、変数と同様に使用できる結果を返すメソッドもあります。たとえば、Number クラスの toString( ) メソッドは、数値をそのテキスト表現に変換します。

```
var numericData:Number = 9;
var textData:String = numericData.toString();
```

たとえば、画面のテキストフィールドに Number 変数の値を表示する場合は、toString( ) メソッドを使用します。TextField クラスの text プロパティ (画面に表示される実際のテキスト内容を表す) は、String として定義されているため、テキスト値のみ格納できます。このコード行は、変数 numericData の数値をテキストに変換し、その後、画面の TextField オブジェクト calculatorDisplay に表示されるようにします。

```
calculatorDisplay.text = numericData.toString();
```

イベント

ここまで、コンピュータがステップバイステップ方式で実行する一連の命令としての、コンピュータプログラムを説明してきました。コンピュータプログラムによっては、コンピュータが実行するいくつかの手順と、プログラムの終了箇所のみから構成される場合があります。ただし、ActionScript のプログラムはユーザー入力またはその他のイベントの発生を待機しながら、実行を継続するように設計されています。イベントは、コンピュータがどの命令をいつ実行するかを判断するメカニズムです。

基本的には、イベントは ActionScript が認識し応答できることです。ユーザーのボタンのクリックや、キーボードのキー入力など多くのイベントがユーザー操作に関連付けられていますが、他のタイプのイベントもあります。たとえば、ActionScript から外部イメージを読み込む場合、イメージの読み込みの終了時を知らせるイベントが起こります。実際には、Adobe Flash Player は、ActionScript プログラムの実行時に特定のイベントが起こるのを待機しているだけであり、そのようなイベントが起こると、イベントに指定した特定の ActionScript コードを実行します。

基本的なイベント処理

特定のイベントに応答して実行される特定のアクションを指定する技術は、" イベント処理 " と呼ばれます。イベント処理を実行する ActionScript コードを作成する場合、確認が必要な 3 つの重要要素があります。

- イベントソース：イベントはどのオブジェクトに対して発生するか。たとえば、どのボタンをクリックするか、またはどの Loader オブジェクトがイメージを読み込んでいるかなど。イベントソースは、イベントが Flash Player のターゲットとなる (イベントが実際に発生する) オブジェクトなので、イベントターゲットとも呼ばれます。
- イベント：起ころうとしていること、すなわち応答が必要な出来事は何か。これは、多くのオブジェクトが複数のイベントをトリガするため、重要な特定項目です。
- 応答 : イベントが起こったときに、どの手順の実行が必要になるか。

イベントを処理する ActionScript コードを作成する場合は、常にコードに上記の要素が指定され、コードは次の基本構造に従います (太字の要素は、特定の状況で入力するプレースホルダです)。

```
function eventResponse(eventObject:EventType):void
{
  // イベントに対応して実行されるアクションはここに入ります。
}

eventSource.addEventListener(EventType.EVENT_NAME, eventResponse);
```

このコードは 2 つの操作を実行します。まず、関数を定義します。関数はイベントに対応して実行するアクションをユーザーが指定する方法です。次に、ソースオブジェクトの `addEventListener()` メソッドを呼び出します。実際には、イベントの発生時に関数のアクションが実行されるように、指定されたイベントに関数を " サブスクライブ " します。各部分を詳細に検討してみましょう。

関数は、アクションを実行するためのショートカット名に似た単一の名前を使って、アクションをグループ化する方法を提供します。関数はメソッドに似ていますが、必ずしも特定のクラスに関連付けられているわけではありません (実際には、メソッドは特定のクラスに関連付けられた関数として定義することも可能です)。イベント処理の関数を作成する場合は、関数の名前を選択する必要があり (この場合は eventResponse)、また 1 つのパラメータ (この例では eventObject) を指定する必要があります。関数パラメータの指定は、変数の宣言に似ています。このため、パラメータのデータ型を指定する必要があります。各イベントについて定義されている ActionScript クラスがあり、関数パラメータに指定するデータ型は、常に応答の必要な特定のイベントに関連付けられたクラスになります。最後に、開始括弧と終了括弧の間に ({ ... })、イベントの発生時にコンピュータに実行を指示する命令を記述します。

イベント処理関数を作成した後、イベントの発生時に、作成した関数を呼び出す場合に必要になるイベントソースオブジェクト (ボタンなど、イベントの発生原因となるオブジェクト) を指定する必要があります。これには、そのオブジェクトの addEventListener( ) メソッドを呼び出します。イベントが関連付けられるすべてのオブジェクトには、addEventListener( ) メソッドもあります。addEventListener( ) メソッドは 2 つのパラメータを受け取ります。

- 1 つ目は、応答の必要な個々のイベントの名前です。ここでも、各イベントは特定のクラスに関連付けられているため、そのクラスは各イベントに事前定義されている特別な値が割り当てられます。これは、イベント独自の固有名のようなものであり、最初のパラメータに使用します。
- 2 つ目は、イベント応答関数の名前です。関数名をパラメータとして渡す場合は、括弧なしで記述します。

イベント処理プロセスの確認

次に、イベントリスナーを作成するときに発生するプロセスの詳しい手順を説明します。この場合、これは myButton という名前のオブジェクトをクリックしたときに呼び出すリスナー関数を作成する例です。

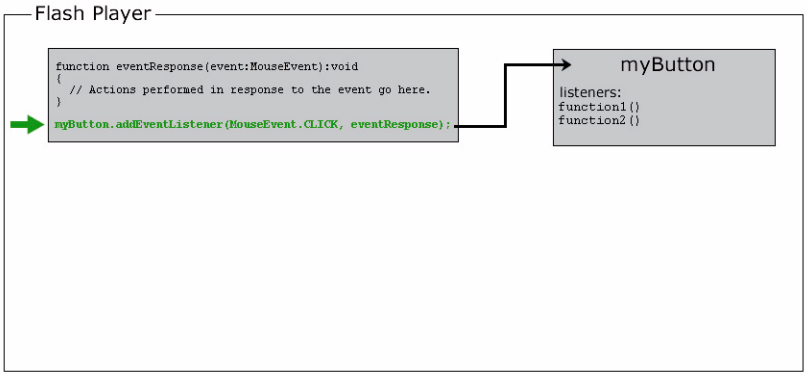
プログラマが作成する実際のコードは次のとおりです。

```
function eventResponse(event:MouseEvent):void
{
   // Actions performed in response to the event go here.
}

myButton.addEventListener(MouseEvent.CLICK, eventResponse);
```

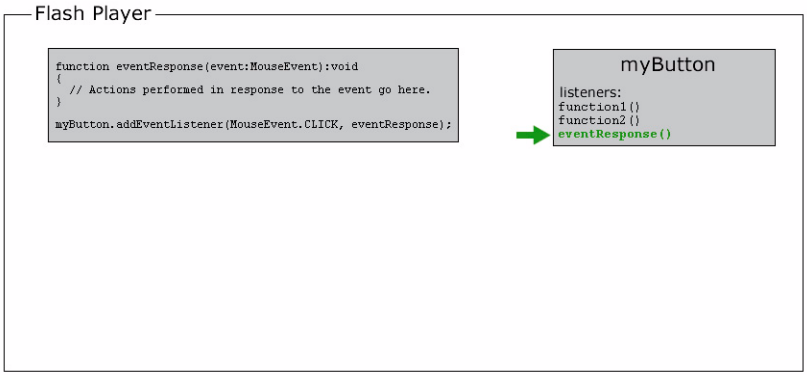
Flash Player で実行したときのこのコードの実際の機能を次に示します。

1. SWF ファイルがロードされると、Flash Player は、`eventResponse()` という名前の関数があることを記録します。

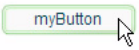
2. 次に、Flash Player はコード (明確には、関数に含まれないコード行) を実行します。この場合は、1 行だけのコードです。イベントソースオブジェクト (`myButton`) で `addEventListener()` メソッドを呼び出し、`eventResponse` 関数をパラメータとして渡します。



a. myButton の内部には、その各イベントを監視する関数のリストがあるので、その addEventListener( ) メソッドが呼び出されると、myButton が eventResponse( ) 関数をそのイベントリスナーのリストに保存します。

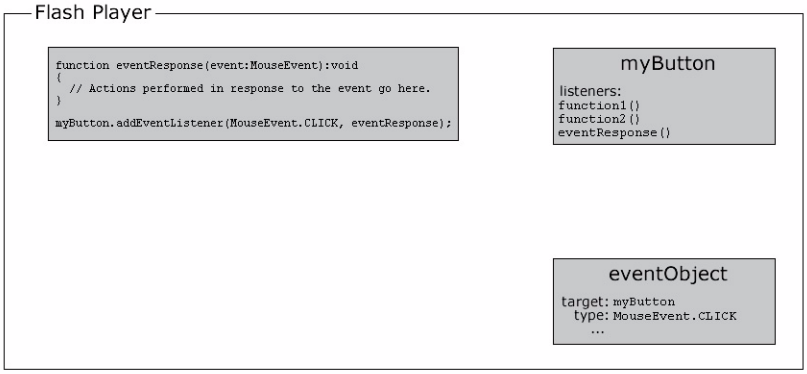


3. ある時点で、ユーザーが myButton オブジェクトをクリックし、その click イベント (コードで MouseEvent.CLICK と識別される) をトリガします。

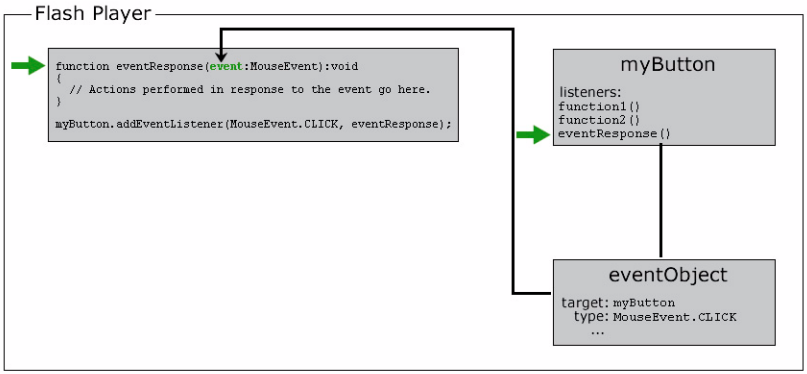


その時点で、以下の処理が行われます。

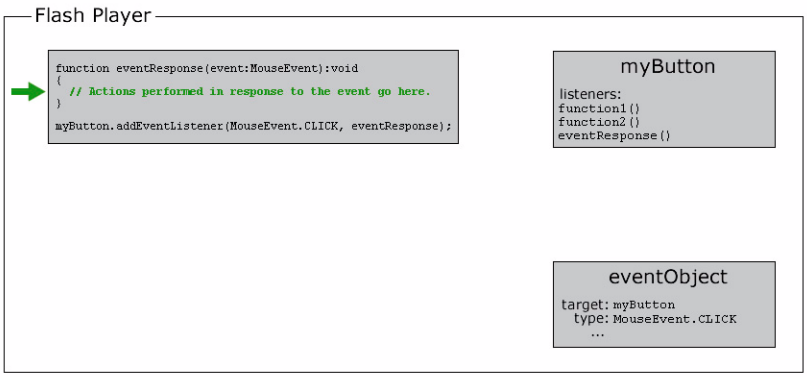
a. Flash Player は、対象のイベントに関連付けられたクラスのインスタンス (この例では MouseEvent) であるオブジェクトを作成します。多くのイベントにとっては、これが Event クラスのインスタンスになります。マウスイベントでは、MouseEvent インスタンスになり、その他のイベントでは、そのイベントに関連付けられたクラスのインスタンスになります。作成されるこのオブジェクトは " イベントオブジェクト " と呼ばれ、ここには、イベントのタイプ、発生場所、その他のイベント固有の情報など、発生したイベントに関する固有の情報が含まれます。



b. 次に、Flash Player は `myButton` によって格納されたイベントリスナのリストを参照します。この関数を 1 つずつ確認し、各関数を呼び出し、イベントオブジェクトをパラメータとして関数に渡します。`eventResponse()` 関数は `myButton` のリスナーの 1 つなので、このプロセスの一環として、Flash Player は `eventResponse()` 関数を呼び出します。



c. eventResponse() 関数が呼び出されると、その関数のコードが実行されるので、指定したアクションが実行されます。



イベント処理の例

ここでは、イベント処理コードの作成時に使用できる共通のイベント要素および可能なバリエーションのいくつかを発想するのに役立つ、いくつかのより具体的なイベント例について説明します。

- ボタンをクリックして、現在のムービークリップの再生を開始。次の例では、playButton はボタンのインスタンス名であり、this は " 現在のオブジェクト " を意味する特別な名前です。

```
this.stop();

function playMovie(event:MouseEvent):void
{
  this.play();
}

playButton.addEventListener(MouseEvent.CLICK, playMovie);
```

- テキストフィールドへの入力の検出。この例では、entryText はテキスト入力フィールド、outputText はダイナミックテキストフィールドです。

```
function updateOutput(event:TextEvent):void
{
  var pressedKey:String = event.text;
  outputText.text = "You typed: " + pressedKey;
}

entryText.addEventListener(TextEvent.TEXT_INPUT, updateOutput);
```

- ボタンをクリックして URL に移動。この場合、linkButton はボタンのインスタンス名です。

```
function gotoAdobeSite(event:MouseEvent):void
{
  var adobeURL:URLRequest = new URLRequest("http://www.adobe.com/");
  navigateToURL(adobeURL);
}

linkButton.addEventListener(MouseEvent.CLICK, gotoAdobeSite);
```

オブジェクトインスタンスの作成

当然ですが、ActionScript でオブジェクトを使用する前に、まずオブジェクトが存在している必要があります。オブジェクト作成の一環として、変数の宣言がありますが、変数の宣言により作成されるのは、コンピュータメモリ内の空き領域のみです。変数に実際の値を割り当てる必要があります。すなわち使用または操作する前に、オブジェクトを作成し、これを変数に格納します。オブジェクト作成のプロセスは、オブジェクトの " インスタンス化 "、すなわち特定のクラスのインスタンス作成として知られます。

オブジェクトのインスタンスを作成する場合、ActionScript をまったく使用しないという簡単な方法もあります。Flash では、ムービークリップシンボル、ボタンシンボル、またはテキストフィールドをステージに配置し、プロパティインスペクタでインスタンス名を割り当てた場合、自動的にそのインスタンス名で変数が宣言され、オブジェクトインスタンスが作成され、そのオブジェクトが変数に格納されます。同様に、Adobe Flex Builder では、アドビ システムズ社が提供する Macromedia® MXML™ でコンポーネントを作成し (MXML タグをコーディングまたはデザインモードのエディタにコンポーネントを配置)、そのコンポーネントに ID を割り当てた場合 (MXML マークアップまたは Flex プロパティビューを使用)、その ID は ActionScript の変数名になり、コンポーネントのインスタンスが作成され、変数に格納されます。

ただし、オブジェクトを常に視覚的に作成する必要がない場合があります。ActionScript のみを使用して、オブジェクトインスタンスを作成する方法もいくつかあります。まず、ActionScript の複数のデータ型を使用した場合、文字式、すなわち ActionScript コードに直接記述される値を使用して、インスタンスを作成できます。次に例を示します。

- Literal 数値 (数字を直接入力):

```
var someNumber:Number = 17.239;
var someNegativeInteger:int = -53;
var someUint:uint = 22;
```

- Literal String 値 (テキストを二重引用符で囲む):

```
var firstName:String = "George";
var soliloquy:String = "To be or not to be, that is the question...";
```

- Literal Boolean 値 (リテラル値 true または false を使用):

```
var niceWeather:Boolean = true;
var playingOutside:Boolean = false;
```

■ Literal XML 値 (XML を直接入力):

```
var employee:XML = <employee>
     <firstName>Harold</firstName>
     <lastName>Webster</lastName>
  </employee>;
```

ActionScript は、Array、RegExp、Object、Function の各データ型にも文字式を定義します。これらのクラスの詳細については、211 ページの「配列の操作」、273 ページの「正規表現の使用」、および 91 ページの「Object データ型」を参照してください。

他のデータ型については、オブジェクトインスタンスを作成する場合、次のように new 演算子をクラス名と一緒に使用します。

```
var raceCar:MovieClip = new MovieClip();
var birthday:Date = new Date(2006, 7, 9);
```

new 演算子を使用したオブジェクトの作成は、多くの場合 " クラスのコンストラクタの呼び出し " と呼ばれます。コンストラクタは、クラスのインスタンスの作成プロセスの一環として呼び出される特別なメソッドです。インスタンスをこの方法で作成する場合、クラス名の後に括弧を配置し、場合によってはパラメータ値を指定します。この 2 つの作業はメソッドを呼び出す場合にも行います。

メモ

文字式を使用したインスタンスの作成が可能になるデータの場合でも、オブジェクトインスタンスの作成には new 演算子を使用できます。たとえば、次の 2 行のコードは、完全に同じ操作を行います。

```
var someNumber:Number = 6.33;
```

var someNumber:Number = new Number(6.33);

new *ClassName*( ) を使用したオブジェクトの作成方法に慣れておく必要があります。視覚的に表現されない (したがって Flash ステージやデザインモードの Flex Builder の MXML エディタにアイテムを配置して作成できない) ActionScript データ型のインスタンスの作成が必要になる場合、このために実行できる方法は、ActionScript で new 演算子を使用してオブジェクトを直接作成する方法のみです。

特に Flash では、new 演算子は Library で定義され、ステージに配置されていないムービークリップシンボルのインスタンスの作成にも使用できます。詳細については、475 ページの「ActionScript での MovieClip オブジェクトの作成」を参照してください。

共通のプログラム要素

変数の宣言、オブジェクトインスタンスの作成、そのプロパティとメソッドを使用したオブジェクトの操作以外に、ActionScript プログラムの作成に使用する他の構築ブロックがいくつかあります。

演算子

演算子は、計算の実行に使用される特殊記号です (words と呼ばれる場合もあります)。演算子はほとんどの場合、算術演算に使用されますが、値同士の比較にも使用されます。一般則として、演算子は複数の値を使用することもでき、単一の結果に " 演算 " します。たとえば、次のようになります。

- 加算演算子 (+) は、2 つの値を加算し、1 つの数字を返します。

  ```
  var sum:Number = 23 + 32;
  ```

- 乗算演算子 (*) は 1 つの値を別の値と乗算し、1 つの数字を返します。

  ```
  var energy:Number = mass * speedOfLight * speedOfLight;
  ```

- 等価演算子 (==) は、2 つの値を比較し、等価であるかどうかを確認し、1 つの true-or-false (Boolean) 値を返します。

  ```
  if (dayOfWeek == "Wednesday")
  {
    takeOutTrash();
  }
  ```

  ここで示すように、等価演算子とその他の " 比較 " 演算子は、ほとんどの場合 if ステートメントで使用され、特定の命令を実行すべきかどうかが決定されます。

  詳細および演算子の例については、102 ページの「演算子」を参照してください。

コメント

ActionScript を作成する場合、コードの特定の行の動作、または特定の選択を行った理由を説明するなど、注釈を自由に記述することが多くあります。コードコメントは、コード内でコンピュータが無視するテキストを記述するのに使用できるツールです。ActionScript では 2 種類のコメントが使用されます。

- 1 行コメント : 1 行コメントは 2 つのスラッシュを行内の任意の場所に配置して指定します。スラッシュ以降の行末までのすべての内容は、コンピュータにより無視されます。

  ```
  // これはコメントです。コンピュータに無視されます。
  var age:Number = 10; // デフォルトでは、10 才に設定します。
  ```

- 複数行コメント : 複数行コメントは開始コメントマーカー (/*)、コメント内容、終了コメントマーカー (*/) から構成されます。開始および終了マーカー間のすべての内容は、コメントが何行に渡っていても、コンピュータに無視されます。

```
/*
これは実際に長い説明になる場合があります。特定の関数が
何に使用されるか、あるいはコードセクションの説明になると思われます。

いずれの場合も、これらの行はすべてコンピュータで無視されます。
*/
```

コメントの共通の使い方としては、コードの 1 行または複数行を一時的に " 無効にする " 方法があります。たとえば、何かを行う方法を別の方法でテストする場合、または特定の ActionScript コードが予想どおりに動作しない理由を考える場合などです。

フロー制御

プログラムでは、特定のアクションを繰り返す、特定のアクションのみを実行し、他のアクションを実行しない、あるいは特定の条件に応じて別のアクションを実行する場合が数多くあります。フロー制御は、実行されるアクションに対する制御です。ActionScript では、複数の種類のフロー制御要素を利用できます。

- 関数 : 関数はショートカットに似ており、一連のアクションを 1 つの名前でグループ化する方法です。また計算の実行に使用できます。関数はイベント処理に特に重要ですが、一連の命令をグループ化するための汎用ツールとしても使用されます。関数の詳細については、115 ページの「関数」を参照してください。
- ループ : ループ構造を使用すると、コンピュータが指定した回数、または特定の条件が変化するまで実行する命令セットを指定できます。ループは多くの場合、複数の関連性のあるアイテムの操作に使用され、コンピュータがループを操作するたびに値が変化する変数を使用します。ループの詳細については、112 ページの「ループ」を参照してください。
- 条件ステートメント : 条件ステートメントは、特定の状況の下でのみ実行される特定の命令を指定する方法、または異なる条件に対して別の命令セットを指定する方法を提供します。条件ステートメントの最も一般的なタイプに、if ステートメントがあります。if ステートメントは、その括弧内の値または式を確認します。値が true であれば、括弧内のコード行は実行され、それ以外の場合は無視されます。次に実際の使用例を示します。

```
if (age < 20)
{
  // show special teenager-targeted content
}
```

if ステートメントと対になる else ステートメントを使用すると、条件が true 以外の場合に実行される代替命令を指定できます。

```
if (username == "admin")
{
```

```
    // 特別オプションの表示など、管理者専用の操作を実行
}
else
{
    // 管理者以外の操作を実行
}
```

条件ステートメントの詳細については、109 ページの「条件演算」を参照してください。

例 : アニメーションポートフォリオピース

この例は、ActionScript のピースをつなげて、完全な (または ActionScript が多用されている) アプリケーションを作成する方法を示す最初の例です。アニメーションポートフォリオピースは、既存の線状アニメーション (クライアントのために作成されたピースなど) に小さなインタラクティブ要素を追加して、そのアニメーションをオンラインポートフォリオに組み込む方法を示す 1 つの例です。アニメーションに追加するインタラクティブビヘイビアには、ユーザーがクリックできる 2 つのボタンが含まれます。1 つはアニメーションを開始するボタンであり、もう 1 つは別の URL (ポートフォリオメニューや作者のホームページなど) に移動するボタンです。

このピースを作成するプロセスは、以下のメインセクションに分けることができます。

1. ActionScript とインタラクティブ要素を追加するための FLA ファイルを準備します。
2. ボタンを作成して追加します。
3. ActionScript コードを作成します。
4. アプリケーションのテスト

インタラクティブ機能を追加する準備

アニメーションにインタラクティブ要素を追加する前に、FLA ファイルに新しいコンテンツを追加する場所を作成しておくと便利です。これには、ボタンを配置する実際の場所をステージに作成しておく作業と、さまざまなアイテムを切り離しておくための FLA ファイル内の " スペース " の作成が含まれます。

インタラクティブ要素を追加するために FLA をセットアップするには、次のようにします。

1. インタラクティブ機能を追加する線状アニメーションがまだ存在しない場合は、シングルモーショントゥイーンやシェイプトゥイーンなどの単純なアニメーションを含む新しい FLA ファイルを作成します。アニメーションがある場合は、プロジェクトでショーケースするアニメーションを含んでいる FLA ファイルを開き、新しい名前で保存して、新しい作業ファイルを作成します。

2. 2 つのボタン (アニメーションを開始するボタンと、作者のポートフォリオやホームページにリンクするボタン) を表示する画面上の位置を決めます。必要な場合は、ステージ上にこの新しいコンテンツのためのスペースを作ります。アニメーションにスプラッシュ画面がない場合は、最初のフレームにスプラッシュ画面を作成することもできます (その場合、アニメーションがフレーム 2 以降から始まるようにアニメーションをシフトできます)。
3. タイムラインの他のレイヤーの上に新しいレイヤーを追加して、buttons という名前に変更します。このレイヤーが、ボタンを追加するレイヤーになります。
4. buttons レイヤーの上に新しいレイヤーを追加して、actions という名前を付けます。これが、アプリケーションに ActionScript コードを追加する場所になります。

ボタンの作成と追加

次に、インタラクティブアプリケーションの中心となるボタンを実際に作成して、配置する必要があります。

ボタンを作成して、FLA に追加するには :

1. 描画ツールを使用して、buttons レイヤーの最初のボタン (“play” ボタン) の外観を作成します。たとえば、横長の楕円形の上にテキストを付けます。
2. 選択ツールを使用して、1つのボタンのすべてのグラフィックパーツを選択します。
3. メインメニューから、[変更]-[シンボルに変換] を選択します。
4. ダイアログボックスでシンボルのタイプとして [ボタン] を選択し、シンボルに名前を付けて、[OK] をクリックします。
5. ボタンを選択すると、プロパティインスペクタはボタンに `playButton` というインスタンス名を付けます。
6. ステップ 1 ～ 5 を繰り返して、作者のホームページに移動するためのボタンを作成します。このボタンに `homeButton` という名前を付けます。

コードの作成

このアプリケーションの ActionScript コードは 3 つの機能セットに分けることができますが、すべて同じ場所に入力されます。このコードでは、次の 3 つのことを行う必要があります。

- SWF ファイルが読み込まれたら (再生ヘッドがフレーム 1 に入ったら)、すぐに再生ヘッドを停止します。
- イベントを監視し、ユーザーが再生ボタンをクリックしたときに SWF ファイルの再生を開始します。
- イベントを監視し、ユーザーが作者ホームページボタンをクリックしたときにブラウザを適切な URL に移動します。

フレーム 1 に入ったときに再生ヘッドを停止するコードを作成するには :

1. actions レイヤーのフレーム 1 のキーフレームを選択します。
2. メインメニューから [ウィンドウ]-[アクション] を選択して、[アクション] パネルを開きます。
3. [スクリプト] ペインに次のコードを入力します。

```
stop();
```

再生ボタンがクリックされたときにアニメーションを開始するコードを作成するには :

1. 前のステップで入力したコードの末尾に、空白行を 2 行追加します。
2. スクリプトの末尾に、次のコードを入力します。

```
function startMovie(event:MouseEvent):void
{
  this.play();
}
```

このコードでは、startMovie( ) という名前の関数を定義します。startMovie( ) が呼び出されると、メインタイムラインが再生を開始します。

3. 前のステップで追加したコードの次の行に、次のコード行を入力します。

```
playButton.addEventListener(MouseEvent.CLICK, startMovie);
```

このコード行は、startMovie( ) 関数を playButton の click イベントのリスナーとして登録します。言い換えると、playButton という名前のボタンがクリックされるたびに startMovie( ) 関数が呼び出されるようにします。

ホームページボタンがクリックされたときにブラウザを URL に移動するコードを作成するには :

1. 前のステップで入力したコードの末尾に、空白行を 2 行追加します。
2. スクリプトの末尾に、次のコードを入力します。

```
function gotoAuthorPage(event:MouseEvent):void
{
  var targetURL:URLRequest = new URLRequest("http://example.com/");
  navigateToURL(targetURL);
}
```

このコードは、gotoAuthorPage( ) という名前の関数を定義します。この関数は、まず http://example.com/ という URL を表す URLRequest インスタンスを作成し、その URL を navigateToURL( ) 関数に渡します。これによって、ユーザーのブラウザでその URL が開きます。

3. 前のステップで追加したコードの次の行に、次のコード行を入力します。

```
homeButton.addEventListener(MouseEvent.CLICK, gotoAuthorPage);
```

このコード行は、gotoAuthorPage( ) 関数を homeButton' の click イベントのリスナーとして登録します。言い換えると、homeButton という名前のボタンがクリックされるたびに gotoAuthorPage( ) 関数が呼び出されるようにします。

アプリケーションのテスト

この時点で、アプリケーションは完全に機能するはずです。テストして、確認してみましょう。

アプリケーションをテストするには :

1. メインメニューから、[コントロール]-[ムービープレビュー] を選択します。Flash で SWF ファイルが作成され、Flash Player ウィンドウで開かれます。
2. 両方のボタンが期待通りに機能するか確認してみます。
3. ボタンが機能しない場合は、以下の点を確認してください。
   - 両方のボタンのインスタンス名が同じになっていませんか。
   - `addEventListener()` メソッド呼び出しで使用される名前がボタンのインスタンス名と同じになっていますか。
   - `addEventListener()` メソッド呼び出しで使用されるイベント名は正しいですか。
   - それぞれの関数に対して正しいパラメータが指定されていますか (どちらにも、データ型 MouseEvent のパラメータが 1 つ指定されている必要があります)。

   このような間違いがあると、[ムービープレビュー] コマンドを選択したときやボタンをクリックしたときにエラーメッセージが表示されます。[コンパイルエラー] パネルでコンパイルエラー (初めて [ムービープレビュー] を選択したときに発生するエラー) がないかどうかを確認し、[出力] パネルでランタイムエラー (ユーザーがボタンをクリックしたときなど、SWF の再生中に発生するエラー) がないかどうかを確認してください。

ActionScript によるアプリケーションの構築

アプリケーションを構築するための ActionScript の作成プロセスは、構文や使用するクラス名を確認する以外の作業が要求されます。本書内のほとんどの情報は、それらの 2 つのトピック (構文と ActionScript クラスの使用) を対象としていますが、ActionScript の作成に使用できるプログラム名や、ActionScript コードをアプリケーション内に配置しインクルードする方法、ActionScript アプリケーションの開発時に従うべき手順などの情報も把握する必要があります。

コードの構成オプション

ActionScript 3.0 のコードを使用すると、簡単なグラフィックアニメーションから複雑なクライアント / サーバートランザクション処理システムまであらゆるものを強化できます。構築中のアプリケーションの種類に応じて、ActionScript をプロジェクトにインクルードするさまざまな方法から 1 つまたは複数を選択できます。

Flash タイムラインのフレームへのコードの保存

Flash オーサリング環境では、タイムラインのフレームに ActionScript コードを追加できます。このコードは、ムービーの再生の間、再生ヘッドがそのフレームに移動すると実行されます。

フレームに ActionScript コードを配置すると、Flash オーサリングツールに組み込まれているアプリケーションにビヘイビアを簡単に追加できます。メインタイムラインのフレームまたは MovieClip シンボルのタイムラインのフレームに、コードを追加できます。しかし、こうした柔軟性を持たせると問題がでてきます。大規模なアプリケーションを構築する場合、どのフレームにどのスクリプトが入っているかを見逃しがちになります。これによって、時間の経過に伴いアプリケーションを維持するのがより困難になります。

多くの開発者は、Flash オーサリングツールで、タイムラインの先頭フレーム内のみまたは Flash ドキュメントの特定のレイヤー上のみにコードを配置して、ActionScript コードの構成を簡略化します。これにより、Flash FLA ファイルのコードが見つけやすくなり、メンテナンスも簡単になります。ただし、別の Flash プロジェクトで同じコードを使用するには、コードをコピーして新しいファイルにペーストする必要があります。

作成した ActionScript コードを今後別の Flash プロジェクトで使用できるようにする場合は、外部 ActionScript ファイル (拡張子 .as のテキストファイル) にコードを保存します。

ActionScript ファイルへのコードの保存

プロジェクトに重要な ActionScript コードが含まれている場合、別の ActionScript ソースファイル (.as 拡張子のテキストファイル) でコードを編集するのが最適です。ActionScript ファイルは、アプリケーションで予定される用途に応じて次のいずれかの方法で構築できます。

- 構造化されていない ActionScript コード : タイムラインスクリプト、MXML ファイルなどに直接入力されたのと同様に記述された、ステートメントまたは関数定義を含む ActionScript コードの行。

  この方法で記述された ActionScript は、ActionScript の `include` ステートメント、Flex MXML の `<mx:Script>` タグを使用してアクセスできます。ActionScript の `include` ステートメントは、外部 ActionScript ファイルの内容をスクリプトの所定範囲内の特定の場所に、直接入力する場合と同様に挿入します。Flex MXML 言語では、`<mx:Script>` タグを使用すると外部 ActionScript ファイルを特定するソース属性の、アプリケーションの該当箇所への読み込みを指定することができます。たとえば、次のタグは "Box.as" という外部 ActionScript ファイルをロードします。

  ```
  <mx:Script source="Box.as" />
  ```

- ActionScript クラスの定義 : メソッドとプロパティの定義を含む ActionScript クラスの定義。

  クラスを定義する場合、組み込まれている ActionScript クラスのいずれかを操作する場合とまったく同様に、クラスのインスタンスを作成し、そのプロパティ、メソッド、イベントを使用することで、クラスで ActionScript コードにアクセスできます。これには 2 段階の操作が必要です。

- `import` ステートメントを使用して、ActionScript コンパイラが検索箇所を見つけられるように、クラスの完全名を指定します。たとえば、ActionScript で MovieClip クラスを使用する場合、まず完全な名前を使用し、そのクラスをパッケージとクラスを含めて読み込む必要があります。

  ```
  import flash.display.MovieClip;
  ```

  または、MovieClip クラスを格納したパッケージを読み込むこともできます。この操作は、パッケージ内の各クラスに `import` ステートメントを個別に記述するのと同じです。

  ```
  import flash.display.*;
  ```

  コードでクラスを参照する場合、クラスの読み込みが要求されない唯一の例外として、トップレベルのクラスがあります。これはパッケージで定義されていないためです。

メモ	Flash では、タイムラインのフレームにアタッチされたスクリプトについては、(flash.* パッケージに含まれる) 組み込みクラスは自動的に読み込まれます。ただし、カスタムクラスを作成している場合、Flash オーサリングコンポーネント (fl.* パッケージ) を使用している場合、または Flex で作業している場合は、クラスを明示的に読み込み、そのクラスのインスタンスを作成するコードを記述する必要があります。

- クラス名を固有に参照するコードを記述します (通常はデータ型がそのクラスの変数を宣言するか、クラスのインスタンスを作成し変数に格納する操作になります)。ActionScript コード内の別のクラス名を参照して、コンパイラにそのクラスの定義をロードするように指示します。たとえば、Box という外部クラスがあるとすると、このステートメントによって Box クラスの新しいインスタンスが作成されます。

  ```
  var smallBox:Box = new Box(10,20);
  ```

  コンパイラが Box クラスの参照を最初に見つけると、読み込まれたソースコード内で Box クラスの定義を探します。

適切なツールの選択

プロジェクトのニーズと使用できるリソースに応じて、ActionScript コードの作成および編集に複数のツールのいずれか (または他のツールと組み合わせた複数のツール) を使用することができます。

Flash オーサリングツール

グラフィックとアニメーションの作成機能以外に、Adobe Flash CS3 Professional には、FLA ファイルまたは ActionScript 専用の外部ファイル内の要素にアタッチされた ActionScript コードを操作するためのツールがあります。Flash オーサリングツールは、重要なアニメーションまたはビデオを含むプロジェクト、またはグラフィックアセットの大半を自分で作成するプロジェクト、特にユーザー操作または ActionScript を要求する機能が最小限に抑えられたプロジェクトに適しています。Flash オーサリングツールを使用した ActionScript プロジェクトの開発を選択するもう 1 つの理由に、同じアプリケーションでビジュアルアセットを作成しコードを記述する方法を好む場合があります。またユーザーインターフェイスのビルトインコンポーネントを使用する必要があり、SWF のサイズの縮小化または簡単なビジュアルスキニングが、プロジェクトで優先的に考慮される場合にも、Flash オーサリングの使用が適しています。

Adobe Flash CS3 Professional には、ActionScript コードを記述するための 2 つのツールが含まれています。

- [アクション] パネル : FLA ファイルを操作する場合に選択可能なこのパネルは、タイムラインのフレームにアタッチされた ActionScript コードを作成する場合に使用します。
- [スクリプト] ウィンドウ : [スクリプト] ウィンドウは、ActionScript コードファイル (.as) の操作専用のテキストエディタです。

Flex Builder

Adobe Flex Builder は、Flex フレームワークを含むプロジェクトを作成するためのプレミアツールです。ビジュアルレイアウトおよび MXML の編集ツール以外に、Flex Builder にはフル機能を備えた ActionScript エディタも含まれているため、Flex または ActionScript 専用のプロジェクトを作成する場合にも使用できます。Flex アプリケーションからもたらされる多くのメリットの中には、ビルトインユーザーインターフェイス制御、柔軟な動的レイアウト制御、および外部データソースを操作し、外部データをユーザーインターフェイス要素に結合するためのビルトインメカニズムなどの豊富な機能が含まれています。ただし、このような機能を引き出すためにはコードの追加が必要になるので、Flex アプリケーションの SWF ファイルのサイズは大きくなり、Flash アプリケーションのように簡単に、完全な再スキニングが実行できない場合があります。

Flex Builder は、フル機能を備えたデータ駆動型の高度なインターネットアプリケーションを Flex で作成する場合、また ActionScript コードの編集、MXML コードの編集、アプリケーションの視覚的なレイアウトをすべて単一のツール内で実行する場合に使用します。

サードパーティ ActionScript エディタ

ActionScript (.as) ファイルは簡単なテキストファイルとして保存されているため、プレーンテキストファイルの編集機能を持つすべてのプログラムを ActionScript ファイルの作成に使用できます。アドビ システムズ社の ActionScript 製品以外に、ActionScript 専用の機能を備えた複数のサードパーティテキスト編集プログラムが作成されています。MXML ファイルまたは ActionScript クラスは、どのテキストエディタプログラムを使用しても作成できます。この場合、Flex フレームワーククラスおよび Flex コンパイラを含む Flex SDK を使用したファイルに基づき、SWF アプリケーション (Flex または ActionScript 専用アプリケーション) を作成することができます。また、多くの開発者は、ActionScript クラスを作成するためのサードパーティ ActionScript エディタを、グラフィックコンテンツを作成するための Flash オーサリングツールと組み合わせて使用しています。

サードパーティ ActionScript エディタの使用は、次の場合に選択できます。

- Flash を使ったビジュアル要素のデザインと同時に、別のプログラムで ActionScript コードの作成を希望する場合。
- ActionScript 以外のプログラミング用のアプリケーションを使用し (他のプログラミング言語による HTML ページの作成やアプリケーションの構築など)、ActionScript のコーディングにも同じアプリケーションを使用する必要がある場合。
- ActionScript 専用のプロジェクトまたは Flex プロジェクトを、Flash または Flex Builder を購入せずに Flex SDK を使用して作成する場合。

ActionScript 専用のサポートを行う主なコードエディタには、次のようなものがあります。

- Adobe Dreamweaver® CS3
- ASDT
- FDT
- FlashDevelop
- PrimalScript
- SE|PY
- XCode (ActionScript テンプレートとコードヒントファイルを含む)

ActionScript の開発プロセス

ActionScript プロジェクトの大小に関わらず、アプリケーションの設計および開発のためのプロセスを活用すると、より効率的かつ効果的に作業を進めることができます。次の手順では、ActionScript 3.0 を使用するアプリケーションを構築するための基本的な開発プロセスを説明します。

1. アプリケーションを設計します。

   アプリケーションの作成を開始する前に、何らかの方法でアプリケーションを記述する必要があります。

2. ActionScript 3.0 コードを構成します。

   ActionScript コードは、Flash、Flex Builder、Dreamweaver、テキストエディタのいずれかを使用して作成できます。

3. Flash または Flex アプリケーションファイルを作成して、コードを実行します。

   Flash オーサリングツールを使用する場合は、新しい FLA ファイルを作成し、パブリッシュ設定を行います。次に、アプリケーションにユーザーインターフェイスコンポーネントを追加して、ActionScript コードを参照します。Flex 開発環境で新しいアプリケーションファイルを作成するには、MXML を使用してアプリケーションを定義し、ユーザーインターフェイスコンポーネントを追加して、ActionScript コードを参照します。

4. ActionScript アプリケーションをパブリッシュし、テストします。

   Flash オーサリング環境または Flex 開発環境内からアプリケーションを実行して、意図したことがすべて実行されていることを確認します。

必ずしも次の手順を順番に実行する必要はありません。また現在の手順を完全に終了してから別の操作を行う必要もありません。たとえば、アプリケーションのある画面の設計を行い (手順 1)、ActionScript コードを作成し (手順 2) テストする (手順 4) 前にグラフィック、ボタンなどを作成 (手順 3) することも可能です。または画面の一部を設計し、同時にボタンまたはインターフェイスの 1 つの要素を追加して、それぞれの ActionScript を作成し、組み込みと同時にテストすることも可能です。開発プロセスの上記の 4 ステージは覚えておくと役に立ちますが、実際の開発では、通常はステージ間を必要に合わせて前後に戻ったり進んだりするのが効果的です。

カスタムクラスの作成

プロジェクトで使用するクラスを作成するプロセスは、困難に見えるかもしれません。しかし、クラス作成では、クラスの設計作業、すなわちクラスに含まれるメソッド、プロパティ、イベントの特定の方が困難です。

クラスの設計戦略

オブジェクト指向型の設計の原則は複雑なものです。この分野の学術的な研究と専門的な実務に、これまでの知識がすべて注がれてきました。ここではさらに、設計に取り組みやすくするためのアプローチをいくつか提言します。

1. このクラスのインスタンスがアプリケーションで果たす役割を考えてみましょう。一般、オブジェクトは次の 3 つの役割のいずれかを果たします。
   - 値オブジェクト : このオブジェクトは、主にデータのコンテナとして機能します。すなわち複数のプロパティとこれよりも数の少ないメソッドを含む可能性があります (あるいはメソッドをまったく含まない場合があります)。このオブジェクトは、一般には、ミュージックプレイヤーアプリケーションの Song クラス (実世界の 1 曲の歌を表す) や、Playlist クラス (概念上の歌のグループを表す) など、明確に定義されたアイテムをコードで表現したものです。
   - 表示オブジェクト : 実際に画面上に表示されるオブジェクト。表示オブジェクトの例として、ドロップダウンリストやステータス読み出しなどのユーザーインターフェイス要素や、ビデオゲームの創造物などのグラフィカル要素などがあります。
   - アプリケーション構造 : このオブジェクトは、ロジック内のサポート、またはアプリケーションで実行される処理といった幅広い役割を担います。アプリケーション構造の例として、バイオロジーシミュレーションの特定の計算を実行するオブジェクトや、ミュージックプレイヤーアプリケーションでダイヤル制御とボリューム読み出しとの値を同期させる役割を持つオブジェクト、ビデオゲームでルールを管理するオブジェクト、描画アプリケーションで保存された写真を読み込むオブジェクトがあります。
2. クラスで必要とされる個々の機能を決定します。異なる種類の機能がクラスのメソッドになることが頻繁にあります。
3. クラスが値オブジェクトとして機能する意図を持つ場合、インスタンスに含まれるデータを決定します。このようなアイテムは、適切なプロパティ候補になります。
4. クラスはプロジェクト用に特別に設計されるため、アプリケーションで必要とされる機能を提供することが最も重要な目的になります。次の質問の答えを考えてみると役に立つ場合があります。
   - どのような情報分野がアプリケーションで保存、追跡、操作されるか。これを決定すると、必要な値オブジェクトとプロパティを特定しやすくなります。
   - 最初に読み込まれるアプリケーションの種類や、クリックされる特定のボタン、ムービーが再生を停止するタイミングなど、どのアクションセットを実行する必要があるか。このような情報は、適切なメソッド (または個々の値の変更のみが " アクション " である場合はプロパティの) 候補になります。

- 特定のアクションについて、クラスがそのアクションを実行するため、どのような種類の情報を知っておく必要があるか。このような種類の情報は、メソッドのパラメータになります。
- アプリケーションが作業を進める過程で、アプリケーションの別の部分が知る必要のある、どのようなことがクラスで変化するか。これは適切なイベント候補になります。

5. 一部の追加機能が欠け、後で追加できる点を除き、必要なオブジェクトに似たオブジェクトが存在する場合、サブクラス (すべて独自の機能を定義するのではなく、既存のクラスの機能に基づき構築されるクラス) の作成を検討してください。たとえば、画面上に視覚的なオブジェクトとして表示されるクラスを作成する場合、既存の表示オブジェクト (Sprite や MovieClip など) のいずれかの動作をクラスの基本として使用できます。その場合、MovieClip (または Sprite) は、基本クラスになり、カスタムクラスではそのクラスが拡張されます。サブクラスの作成の詳細については、152 ページの「継承」を参照してください。

クラスのコードの作成

クラスの設計プランを決定した後、またはどのような情報を追跡し続ける必要があるか、あるいは最低でも、どのようなアクションを実行する必要があるかについて考案した後は、実際のクラス作成の構文はまったく簡単なものになります。

次にカスタム ActionScript クラスを作成する最小限の手順を示します。

1. Flex Builder や Flash などの ActionScript 専用のプログラム、Dreamweaver など汎用プログラミングツール、プレーンテキストドキュメントを操作できる任意のプログラムのいずれかで新しいテキストドキュメントを開きます。
2. クラスの名前を定義する `class` ステートメントを入力します。これには、単語 `public class` を入力してから、クラス名を入力し、その後にクラスの内容 (メソッドとプロパティの定義) を開始括弧と終了括弧の間に入力します。次に実際の使用例を示します。

   ```
   public class MyClass
   {
   }
   ```

   単語 `public` は、そのクラスが他のコードからアクセスできることを示します。他の例については、135 ページの「アクセス制御名前空間の属性」を参照してください。
3. パッケージの検索に使用されるパッケージ名を指定する、package　ステートメントを入力します。この構文は単語 `package` の後に完全なパッケージ名を続け、開始括弧と終了括弧で終了する形になります (括弧は `class` ステートメントブロックを囲みます)。たとえば、前の手順のコードは次のように変更できます。

   ```
   package mypackage
   {
      public class MyClass
      {
      }
   }
   ```

4. クラス本文内で var ステートメントを使用して、クラスの各プロパティを定義します。この構文は変数の宣言に使用する構文と同じものになります (public モディファイアが追加)。たとえば、クラス定義の開始括弧と終了括弧の間に次の行を追加すると、textVariable、numericVariable、dateVariable という名前のプロパティが生成されます。

```
public var textVariable:String = "some default value";
public var numericVariable:Number = 17;
public var dateVariable:Date;
```

5. 関数の定義に使用されるのと同じ構文を使用して、クラスにメソッドを定義します。次に実際の使用例を示します。

   - myMethod() メソッドを作成するには、次のように入力します。

     ```
     public function myMethod(param1:String, param2:Number):void
     {
        // パラメータの処理をする
     }
     ```

   - コンストラクタ (クラスのインスタンスの作成プロセスの一環として呼び出される特別なメソッド) を作成する場合、そのクラス名と名前が正確に一致するメソッドを作成します。

     ```
     public function MyClass()
     {
        // プロパティの初期値の設定
        // またはオブジェクトの設定のための処理を実行する
        textVariable = "Hello there!";
        dateVariable = new Date(2001, 5, 11);
     }
     ```

     クラスにコンストラクタメソッドをインクルードしない場合、コンパイラは自動的にクラスに空のコンストラクタ (パラメータもステートメントも指定されないコンストラクタ) を作成します。

さらにいくつかのクラス要素を定義できます。これらの要素を詳しく説明します。

- アクセサは、メソッドとプロパティを特別に組み合わせたものです。クラスを定義するコードを作成する場合、複数のアクションを実行できるように、メソッドに似たアクセサを記述します (プロパティを定義する場合に実行するような、値の読み取りや割り当ては行いません)。ただし、クラスのインスタンスを作成する場合、プロパティと同様にアクセサを処理します。すなわち名前を使用して値を読み取り、あるいは値を割り当てます。詳細については、143 ページの「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、特定の構文を使用して定義されません。構文を使用する代わりに、EventDispatcher クラスの機能を使用して、イベントのリスナを追跡し、イベントをリスナに通知する方法で、クラスにイベントを定義します。カスタムクラスのイベントの作成の詳細については、299 ページ、第 10 章の「イベントの処理」を参照してください。

クラスの構成に関する提案

旧バージョンの ActionScript と異なり、ActionScript 3.0 にはファイルあたりに使用するクラスを 1 つに制限する 1 ファイル、1 クラスの制限はありません。ActionScript 3.0 を使用して、1 つの .as ファイルに複数のクラスのソースコードを保存できます。1 つのソースファイルに複数のクラスを保存する方が便利な場合もありますが、一般には次の 2 つの理由からこの方法はお勧めしません。

- 複数のクラスが 1 つの大きなファイルに保存されている場合、個々のクラスを再利用するのが困難になります。
- ファイル名とクラス名が対応していない場合、特定のクラスのソースコードを見つけるのが困難です。

このような理由から、個々のクラスのソースコードは常に個人用のファイルに保存し、ファイルにはクラスと同じ名前を付けることをお勧めしています。

例 : 基本的なアプリケーションの作成

Flash、Flex Builder、Dreamweaver、または任意のテキストエディタを使用して、.as 拡張子を持つ外部 ActionScript ソースファイルを作成できます。

ActionScript 3.0 は、Flash オーサリングツールや Flex Builder などのさまざまなアプリケーション開発環境で使用できます。

このセクションでは、Flash オーサリングツールまたは Flex Builder 2 ツールを使用した簡単な ActionScript 3.0 アプリケーションの作成と拡張をステップ方式で学習します。作成するアプリケーションは、ActionScript 3.0 の外部クラスファイルを Flash および Flex アプリケーションで使用するための単純な見本を提示します。そのパターンは、本書の他のすべてのアプリケーション例に適用できます。

ActionScript アプリケーションの設計

アプリケーションの作成を始める前に、どのようなアプリケーションにするかを考えておく必要があります。

設計は、アプリケーションの名前と目的を示す短い説明などの簡単なもので表しても、多数の UML (Unified Modeling Language) 図を含む要件書一式のような複雑なもので表しても構いません。本書では、ソフトウェアの設計分野を詳細に扱いませんが、ActionScript アプリケーションの開発においてアプリケーション設計は欠かすことのできない手順であることに注意してください。

ActionScript アプリケーションの最初の例は、単純な設計を使用した、標準的な "Hello World" アプリケーションです。

- アプリケーションの名前は HelloWorld です。
- "Hello, World!" という文字列が含まれたテキストフィールドが 1 つ表示されます。

- このフィールドを簡単に再利用できるように、Greeter という名前のオブジェクト指向クラスが 1 つ使用されます。このクラスは Flash ドキュメントまたは Flex アプリケーション内で使用できます。
- 基本になるアプリケーションを作成した後、ユーザーにユーザー名を入力させ、アプリケーションにその名前をユーザーリストと照合させる新しい機能を追加します。

この簡潔な定義を使用して、アプリケーションの作成を開始します。

HelloWorld プロジェクトと Greeter クラスの作成

Hello World アプリケーションの設計説明では、再利用しやすいコードを使用するよう指定されています。この目標を考慮し、アプリケーションは Greeter という名前のオブジェクト指向クラスを 1 つ使用します。このクラスは、Flex Builder または Flash オーサリングツールで作成したアプリケーション内から使用されます。

Flash オーサリングツールで Greeter クラスを作成するには :

1. Flash オーサリングツールで [ファイル]-[新規] を選択します。
2. [新規ドキュメント] ダイアログボックスで ActionScript ファイルを選択し、[OK] をクリックします。

   新しい ActionScript 編集ウィンドウが表示されます。
3. [ファイル]-[保存] を選択します。アプリケーションを保存するフォルダを選択し、ActionScript ファイルに Greeter.as という名前を付けて [OK] をクリックします。

   54 ページの「Greeter クラスへのコードの追加」に進みます。

Greeter クラスへのコードの追加

Greeter クラスはオブジェクト `Greeter` を定義し、これは HelloWorld アプリケーションで使用できます。

Greeter クラスにコードを追加するには :

1. 新しいファイルに次のコードを入力します。

```
package
{
   public class Greeter
   {
      public function sayHello():String
      {
         var greeting:String;
         greeting = "Hello World!";
         return greeting;
      }
   }
}
```

Greeter クラスには、指定されたユーザー名に対して "Hello" という文字列を返す 1 つの `sayHello()` メソッドが格納されます。

2. [ファイル]-[保存] を選択して、この ActionScript ファイルを保存します。

これで、Flash または Flex アプリケーションで Greeter クラスを使用することができます。

ActionScript コードを使用するアプリケーションの作成

作成した Greeter クラスは、必要なものをすべて備えたアプリケーション機能を定義しますが、アプリケーション全部を表すわけではありません。Greeter クラスを使用するには、Flash ドキュメントまたは Flex アプリケーションを作成する必要があります。

HelloWorld アプリケーションでは、Greeter クラスの新しいインスタンスが作成されます。次に示すのは、アプリケーションに Greeter クラスをアタッチする方法です。

Flash オーサリングツールを使用して ActionScript アプリケーションを作成するには :

1. [ファイル]-[新規] を選択します。
2. [新規ドキュメント] ダイアログボックスで [Flash ドキュメント] を選択し、[OK] をクリックします。

   新しい Flash ウィンドウが表示されます。
3. [ファイル]-[保存] を選択します。Greeter.as クラスファイルと同じフォルダを選択し、Flash ドキュメントに HelloWorld.fla という名前を付けて [OK] をクリックします。
4. [Flash ツール] パレットで [テキストツール] を選択した後、ステージ上をドラッグして、幅約 300 ピクセル、高さ約 100 ピクセルの新しいテキストフィールドを定義します。
5. ステージ上のテキストフィールドが選択された状態のままで、[プロパティ] ウィンドウにテキストフィールドのインスタンス名として `mainText` と入力します。
6. メインタイムラインの最初のフレームをクリックします。
7. [アクション] パネルに次のスクリプトを入力します。

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("Bob");
```

8. ファイルを保存します。

56 ページの「ActionScript アプリケーションのパブリッシュとテスト」に進みます。

ActionScript アプリケーションのパブリッシュとテスト

アプリケーション開発は反復プロセスです。コードを記述し、コンパイルして、正しくコンパイルされるまでコードを編集します。コンパイルされたアプリケーションを実行しテストして、意図した設計が実現されているかどうかを確認します。実現されていない場合は、実現するまでコードを編集します。Flash および Flex Builder の開発環境には、アプリケーションをパブリッシュ、テスト、およびデバッグする多数の方法が用意されています。

次に各環境で HelloWorld アプリケーションをテストする基本的な手順を示します。

Flash オーサリングツールを使用して ActionScript アプリケーションをパブリッシュおよびテストするには :

1. アプリケーションをパブリッシュして、コンパイルエラーがないか確認します。Flash オーサリングツールで [コントロール]-[ムービープレビュー] を選択して、ActionScript コードをコンパイルして HelloWorld アプリケーションを実行します。
2. アプリケーションをテストしたときに、[出力] ウィンドウにエラーや警告が表示された場合は、HelloWorld.fla または HelloWorld.as ファイルでエラーの原因を修正してから、アプリケーションのテストを再び試みます。
3. コンパイルエラーがなければ、Flash Player ウィンドウに Hello World アプリケーションが表示されます。ウィンドウには "Hello, Bob" というテキストが表示されます。

ActionScript 3.0 を使用する、簡単ですが完全なオブジェクト指向アプリケーションが作成できました。56 ページの「HelloWorld アプリケーションの拡張」に進みます。

HelloWorld アプリケーションの拡張

ここでは、アプリケーションをもう少し興味深いものにしてみます。そのために、アプリケーションにユーザー名を要求させ、それを定義済みの名前リストと検証させます。

まず、Greeter クラスを更新して新しい機能を追加します。次に、Flex または Flash アプリケーションを更新して新しい機能を使用します。

Greeter.as ファイルを更新するには :

1. "Greeter.as" ファイルを開きます。

2. ファイルの内容を次のように変更します。新しい行および変更された行は太字で表示されています。

```
package
{
   public class Greeter
   {
      /**
       * Defines the names that should receive a proper greeting.
       */
      public static var validNames:Array = ["Sammy", "Frank", "Dean"];

      /**
       * Builds a greeting string using the given name.
       */
      public function sayHello(userName:String = ""):String
      {
         var greeting:String;
         if (userName == "")
         {
            greeting = "Hello. Please type your user name, and then press the Enter key.";
         }
         else if (validName(userName))
         {
            greeting = "Hello, " + userName + ".";
         }
         else
         {
            greeting = "Sorry, " + userName + ", you are not on the list.";
         }
         return greeting;
      }

      /**
       * Checks whether a name is in the validNames list.
       */
      public static function validName(inputName:String = ""):Boolean
      {
         if (validNames.indexOf(inputName) > -1)
         {
            return true;
         }
         else
         {
            return false;
         }
      }
   }
}
```

Greeter クラスは、いくつかの新しい機能が追加されました。

- validNames 配列は、有効なユーザー名をリスト表示します。この配列は、Greeter クラスのロード時に 3 つの名前のリストに初期化されます。
- sayHello() メソッドは、ユーザー名を受け付け、何らかの条件に基づいてあいさつを変更できるようになります。userName が空の文字列 ("") である場合、greeting プロパティはユーザーに名前の入力を要求する設定になります。ユーザー名が有効であれば、あいさつは "Hello, *userName*" になります。最後に、2 つの条件のいずれかに適合しない場合、greeting 変数は、"Sorry, userName, you are not on the list." に設定されます。
- validName() メソッドは、inputName が validNames 配列内で見つかった場合は true を、見つからない場合は false を返します。ステートメント validNames.indexOf(inputName) は、validNames 配列内で inputName 文字列と同じ文字列をそれぞれ検査します。Array.indexOf() メソッドは、配列内のオブジェクトの最初のインスタンスのインデックス位置、またはオブジェクトが配列に見つからない場合は値 -1 を返します。

次に、この ActionScript クラスを参照する Flash ファイルまたは Flex ファイルを編集します。

Flash オーサリングツールを使用してアプリケーションを変更するには :

1. HelloWorld.fla ファイルを開きます。
2. フレーム 1 のスクリプトを変更して、空の文字列 ("") が Greeter クラスの sayHello() メソッドに渡されるようにします。

```
var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");
```

3. [ツール] パレットの [テキスト] ツールを選択して、ステージ上で既存の mainText テキストフィールドのすぐ下に 2 つの新しいテキストフィールドを並べて作成します。
4. 最初のテキストフィールドに、ラベルとなるテキストとして User Name: と入力します。
5. もう 1 つの新しいテキストフィールドを選択して、プロパティインスペクタでテキストフィールドのタイプとして InputText を選択します。インスタンス名として textIn と入力します。
6. メインタイムラインの最初のフレームをクリックします。
7. [アクション] パネルで、既存のスクリプトの末尾に以下の行を追加します。

```
mainText.border = true;
textIn.border = true;

textIn.addEventListener(KeyboardEvent.KEY_UP, keyPressed);

function keyPressed(event:Event):void
{
   if (event.keyCode == Keyboard.ENTER)
   {
      mainText.text = myGreeter.sayHello(textIn.text);
   }
}
```

この新しいコードでは、以下の機能が追加されます。

- 最初の 2 行は、単純に 2 つのテキストフィールドの境界線を定義します。
- textIn フィールドなどの入力テキストフィールドには、ディスパッチできるイベントのセットがあります。addEventListener() メソッドによって、特定のタイプのイベントが発生したときに実行される関数を定義することができます。この例で該当するイベントは、キーボードの Enter キーが押されることです。
- keyPressed() カスタム関数は、myGreeter オブジェクトの sayHello() メソッドを呼び出し、textIn テキストフィールドのテキストをパラメータとして渡します。このメソッドは、渡された値に基づいて挨拶の文字列を返します。返された文字列は、mainText テキストフィールドの text プロパティに割り当てられます。

フレーム 1 の完全なスクリプトは次のようになります。

```
mainText.border = true;
textIn.border = true;

var myGreeter:Greeter = new Greeter();
mainText.text = myGreeter.sayHello("");

textIn.addEventListener(KeyboardEvent.KEY_UP, keyPressed);

function keyPressed(event:Event):void
{
  if (event.keyCode == Keyboard.ENTER)
  {
    mainText.text = myGreeter.sayHello(textIn.text);
  }
}
```

8. ファイルを保存します。
9. [コントロール]-[ムービープレビュー] を選択して、アプリケーションを実行します。

   アプリケーションを実行すると、ユーザー名の入力を求められます。有効なユーザー名 (Sammy、Frank、または Dean) を入力すると、“hello” という確認メッセージが表示されます。

後続の例の実行

"Hello World" ActionScript 3.0 アプリケーションを開発し、実行した場合、本書に記載した他のコード例を実行するために必要な基本的な知識を備えておく必要があります。

章内のコード例のテスト

このマニュアルをお読みになる際は、さまざまなトピックを説明するために使用されているコード例を試すことができます。このようなテストには、プログラムの特定の時点で変数の値を表示したり、画面上のコンテンツを表示または操作することが含まれます。視覚的なコンテンツやインタラクションのテストに必要な要素は、該当するコードの前または内部に記述されています。そのため、記述されている要素を含んだドキュメントを作成するだけで、コードをテストすることができます。プログラムの特定時点での変数の値を確認する場合は、複数の方法があります。1 つの方法は、Flex Builder および Flash に組み込まれているものなど、デバッガを使用する方法です。ただし、単純なテストでは、確認が可能な場所に変数の値を出力する方が簡単かもしれません。

コードのテストと変数値の確認のために使用できる Flash ドキュメントを作成するには、以下のステップが役立ちます。

章内の例をテストする際に使用する Flash ドキュメントを作成するには :

1. 新しい Flash ドキュメントを作成し、ハードディスクに保存します。
2. テスト値をステージ上のテキストフィールドに表示するには、テキストツールをアクティブにして、ステージ上に新しい Dynamic テキストフィールドを作成します。[線] タイプが [複数行] に設定された、十分な幅や高さ、および境界線のあるテキストフィールドが最も適しています。プロパティインスペクタで、テキストフィールドにインスタンス名を付けます ("outputText" など)。値をテキストフィールドに書き込むには、`appendText()` メソッドを呼び出すコードをコード例に追加します (後述)。
3. または、`trace()` 関数呼び出しをコードに追加して (後述)、例の結果を表示することもできます。
4. このサンプルをテストするには、コードリストを [アクション] パネルにコピーします。必要な場合は、`trace()` 関数呼び出しを追加するか、その `appendText()` メソッドを使用してテキストフィールドに値を追加します。
5. メインメニューから [コントロール]-[ムービープレビュー] を選択し、SWF ファイルを作成して結果を表示します。

このアプローチは変数の値を確認するためのものなので、コード例を試しているときに変数の値を簡単に確認できる方法が 2 つあります。値をステージ上のテキストフィールドに書き込む方法と、`trace()` 関数を使用して値を [出力] パネルに書き出す方法です。

- trace( ) 関数 : ActionScript の trace( ) 関数は、渡されたパラメータの値 (変数またはリテラル式) を [出力] パネルに書き込みます。本マニュアルのコード例の多くには既に trace( ) 関数呼び出しが含まれているので、そのような例については、コードをドキュメントにコピーするだけでプロジェクトをテストできます。まだ trace( ) が含まれていないコードリストの変数の値をテストするには、コードに trace( ) 呼び出しを追加して、パラメータとして変数を渡します。たとえば、章の中に次のようなコードがあった場合、

  ```
  var albumName:String = "Three for the money";
  ```

  コードを [アクション] パネルにコピーしてから、次のような trace( ) 関数の呼び出しを追加すると、コードの結果をテストできます。

  ```
  var albumName:String = "Three for the money";
  trace("albumName =", albumName);
  ```

  プログラムを実行すると、次のような行が表示されます。

  ```
  albumName = Three for the money
  ```

  それぞれの trace( ) 関数呼び出しは複数のパラメータを取ることができ、すべてが 1 行にまとめられます。それぞれの trace( ) 関数呼び出しの末尾に改行が追加されるので、trace( ) 関数ごとに個別の行に出力されます。

- ステージ上のテキストフィールド : trace( ) 関数を使用しない場合は、テキストツールを使用してステージに Dynamic テキストフィールドを追加し、そのテキストフィールドに値を書き出すと、コードの結果を確認できます。TextField クラスの appendText( ) メソッドを使用すると、テキストフィールドの内容の末尾に String 値を追加できます。ActionScript を使用してテキストフィールドにアクセスするには、プロパティインスペクタでインスタンス名を付ける必要があります。たとえば、テキストフィールドのインスタンス名が outputText である場合、次のコードを使用して変数 albumName の値を確認できます。

  ```
  var albumName:String = "Three for the money";
  outputText.appendText("albumName = ");
  outputText.appendText(albumName);
  ```

  このコードは、outputText という名前のテキストフィールドに次のテキストを書き込みます。

  ```
  albumName = Three for the money
  ```

  この例に示されているように、appendText( ) メソッドは前の内容と同じ行にテキストを追加するので、複数の appendText( ) 呼び出しを使用して複数の値を同じテキスト行に追加できます。テキストを強制的に次の行に送るには、改行文字 ("\n") を追加します。

  ```
  outputText.appendText("\n"); // adds a line break to the text field
  ```

  trace( ) 関数とは異なり、appendText( ) メソッドは 1 つの値だけをパラメータとして受け入れます。その値は文字列 (String インスタンスまたは文字列リテラル) でなければなりません。文字列以外の変数の値を出力するには、値を String に変換する必要があります。このための最も簡単な方法は、オブジェクトの toString( ) メソッドを呼び出すことです。

  ```
  var albumYear:int = 1999;
  outputText.appendText("albumYear = ");
  outputText.appendText(albumYear.toString());
  ```

章の終わりの例の使用

この章と同様に、本マニュアルのほとんどの章には、その章で説明した多くの概念を結びつける重要な章の終わりの例が含まれます。ただし、この章の Hello World の例とは異なり、これらの例は、順を追ったチュートリアル形式では説明していません。各例の該当する ActionScript 3.0 コードを強調表示して説明していますが、特定の開発環境で例を実行するための手順については説明されていません。ただし、本書で配布される例題ファイルには、選択した開発環境で例を簡単にコンパイルおよび実行するのに必要なすべてのファイルが収められる予定です。

第 3 章

ActionScript 言語とシンタックス

3

ActionScript 3.0 は、コア ActionScript 言語と Adobe Flash Player アプリケーションプログラミングインターフェイス (API) の両方で構成されます。コア言語は、ECMAScript (ECMA-262), Edition 4 言語仕様案を実装する ActionScript の一部です。Flash Player API では、プログラムによって Flash Player にアクセスできます。

本章では、コア ActionScript 言語とシンタックスについて簡単に説明します。この章を読むと、データ型と変数の操作方法、適切なシンタックスの使用方法、およびプログラム内でのデータフローの制御方法についての基礎知識を習得できます。

目次

言語の概要 64
オブジェクトとクラス 65
パッケージと名前空間 66
変数 78
データ型 82
シンタックス 97
演算子 102
条件演算 109
ループ 112
関数 115

言語の概要

オブジェクトは、ActionScript 3.0 言語の中心部に位置する基本的要素です。宣言する変数、記述する関数、作成するクラスインスタンスはすべてオブジェクトです。ActionScript 3.0 プログラムは、タスクを実行し、イベントに応答し、相互に通信するオブジェクトのグループと考えることができます。

Java または C++ のオブジェクト指向プログラミング (OOP) に詳しいプログラマは、オブジェクトをメンバー変数またはプロパティに格納されたデータ、またはメソッドを介してアクセスできる動作の 2 種類のメンバーを含むモジュールと見なす場合があります。ActionScript 3.0 が準拠している ECMAScript 第 4 版案では、よく似ていますが少し異なる方法でオブジェクトが定義されています。ECMAScript の草案では、オブジェクトは単にプロパティの集まりです。プロパティは、データだけではなく関数や他のオブジェクトも保持できるコンテナです。このようにオブジェクトに関連付けられている関数をメソッドと呼びます。

ECMAScript の草案の定義は Java または C++ のプログラマには少し奇妙に思えるかもしれませんが、実際には ActionScript 3.0 クラスを使用するオブジェクト型の定義は、Java や C++ でクラスを定義する方法に非常によく似ています。オブジェクトの 2 つの定義の違いは ActionScript オブジェクトモデルおよびその他の高度なテクニックについて説明する場合には重要ですが、ほとんどの場合、" プロパティ " という用語は、メソッドとは対照的に、クラスのメンバー変数を意味します。たとえば、『ActionScript 3.0 コンポーネントリファレンスガイド』では、" プロパティ " は変数または getter/setter プロパティを示すために使用されています。草案では、クラスの一部である関数を意味する用語 " メソッド " が使用されています。

ActionScript のクラスと Java または C++ のクラスには微妙な違いがあります。それは、ActionScript ではクラスは単なる抽象エンティティではないということです。ActionScript のクラスは、クラスのプロパティおよびメソッドを格納するクラスオブジェクトによって表されます。これにより、クラスまたはパッケージの最上位のインクルードステートメントまたは実行可能なコードなど、Java および C++ プログラマには異質なものに見える手法が可能になります。

ActionScript のクラスと Java または C++ のクラスのもう 1 つの違いは、ActionScript の各クラスには " プロトタイプオブジェクト " と呼ばれるものがあることです。旧バージョンの ActionScript では、プロトタイプオブジェクトは " プロトタイプチェーン " に関連付けられてクラスの継承階層全体の基盤として機能しましたが、ActionScript 3.0 では、プロトタイプオブジェクトは継承システムにおいて小さな役割を果たすだけです。ただし、クラスのすべてのインスタンスでプロパティとその値を共有する場合、プロトタイプオブジェクトは静的プロパティおよびメソッドの代わりとして引き続き役立ちます。

これまで、熟練した ActionScript プログラマなら、特別なビルトイン言語エレメントを使用してプロトタイプチェーンを直接操作できました。現在では、ActionScript はクラスベースのプログラミングインターフェイスのさらに成熟した実装を提供しています。`__proto__` や `__resolve` などの特別な言語エレメントの多くは ActionScript に含まれていません。また、Flash Player のパフォーマンスを大幅に向上させる内部継承メカニズムが最適化され、継承メカニズムに直接アクセスできなくなりました。

オブジェクトとクラス

ActionScript 3.0 では、各オブジェクトはクラスにより定義されます。クラスは、オブジェクト型のテンプレートまたは設計図と考えることができます。クラス定義には、データ値を保持する変数と定数、およびクラスにバインドされているビヘイビアをカプセル化する関数であるメソッドを含めることができます。プロパティに格納される値は、プリミティブ値または他のオブジェクトです。プリミティブ値とは、数値、ストリング、またはブール値です。

ActionScript には、コア言語の一部であるビルトインクラスが多数あります。Number、Boolean、String などのビルトインクラスは、ActionScript で使用可能なプリミティブ値を表します。Array、Math、XML などのその他のクラスは、ECMAScript 規格に定められている、より複雑なオブジェクトを定義します。

ビルトインおよびユーザー定義のすべてのクラスは、Object クラスから派生します。旧バージョンの ActionScript に慣れているプログラマは、他のクラスはすべて Object クラスから派生しますが、Object データ型はデフォルトのデータ型ではなくなったことに注意する必要があります。
ActionScript 2.0 では、型注釈が存在しないため、変数が Object 型であることを意味しているので、次の 2 行のコードが同等でした。

```
var someObj:Object;
var someObj;
```

ただし、ActionScript 3.0 は、型指定されていない変数の概念を導入しています。この変数は、次の 2 つの方法で指定することができます。

```
var someObj:*;
var someObj;
```

型指定されていない変数は、Object 型の変数と同じではありません。主な違いは、型指定されていない変数は特別な値 `undefined` を保持できますが、Object 型の変数はその値を保持できないことです。

`class` キーワードを使用して独自のクラスを定義することができます。クラスプロパティは 3 通りの方法で宣言できます。定数を定義するには `const` キーワード、変数を定義するには `var` キーワード、getter および setter プロパティを定義するにはメソッド宣言で `get` および `set` 属性を使用します。メソッドを宣言するには、`function` キーワードを使用します。

クラスのインスタンスを作成するには、`new` 演算子を使用します。次の例では、`myBirthday` と呼ばれる Date クラスのインスタンスを作成します。

```
var myBirthday:Date = new Date();
```

パッケージと名前空間

パッケージと名前空間は関連する概念です。パッケージを使用すると、コードを共有でき、名前のコンフリクトを最小限に抑えられるようにクラス定義をバンドルできます。名前空間を使用すると、プロパティ名やメソッド名などの識別子の可視性を制御でき、コードがパッケージ内部にあるのか外部にあるのかに関係なく適用できます。パッケージを使用してクラスファイルを構成でき、名前空間を使用して個々のプロパティおよびメソッドの可視性を制御できます。

パッケージ

ActionScript 3.0 では、パッケージは名前空間で実装されますが、名前空間と同義ではありません。パッケージを宣言すると、コンパイル時に必ず既知である名前空間が暗黙的に作成されます。名前空間は、明示的に作成された場合はコンパイル時に既知である必要はありません。

次の例では、package ディレクティブを使用して、クラスを 1 つ含む単純なパッケージを作成します。

```
package samples
{
  public class SampleCode
  {
    public var sampleGreeting:String;
    public function sampleFunction()
    {
      trace(sampleGreeting + " from sampleFunction()");
    }
  }
}
```

この例のクラスの名前は SampleCode です。クラスは samples パッケージ内にあるので、コンパイラはコンパイル時にクラス名を自動的に修飾し、完全修飾名の samples.SampleCode にします。また、コンパイラはプロパティやメソッドの名前も修飾するので、sampleGreeting および sampleFunction() は、それぞれ samples.SampleCode.sampleGreeting および samples.SampleCode.sampleFunction() になります。

多くの開発者、特に Java 開発者は、パッケージの最上位にクラスだけを配置することを選択します。しかし、ActionScript 3.0 では、パッケージの最上位でクラスだけではなく、変数、関数、ステートメントもサポートされています。この機能の高度な使用方法として、パッケージ内のすべてのクラスで使用できるようにパッケージの最上位で名前空間を定義できます。ただし、パッケージの最上位で指定できるアクセス指定子は public と internal の 2 つだけです。ネストされたクラスをプライベート宣言できる Java と異なり、ActionScript 3.0 はネストされたクラスも、プライベートクラスもサポートしていません。

しかし、その他の多くの点で、ActionScript 3.0 のパッケージは Java のパッケージと似ています。前の例を見ればわかるとおり、完全修飾パッケージ参照は、Java と同じように、ドット演算子 (.) を使用して表します。パッケージを使用すると、他のプログラマも使用できるように、直感的な階層構造にコードを構成できます。これにより、独自のパッケージを作成して他のプログラマと共有したり、他のプログラマが作成したパッケージを利用したりでき、コードの共有が容易になります。

パッケージを使用することで、使用する識別子名が一意になり、他の識別子名と競合しなくなります。実際、これがパッケージの最も重要な利点であるという人もいます。たとえば、コードを共有しようとしている 2 人のプログラマがそれぞれ SampleCode というクラスを作成したとします。パッケージを使用しなければ、名前の競合が発生し、いずれかのクラスの名前を変更するしかなくなります。ただし、パッケージを使用すると、一意の名前を持つパッケージ内のクラスのいずれか、またはできれば両方を配置することによって、名前の競合は簡単に回避されます。

パッケージ名に埋め込みドットを入れて、ネストされたパッケージを作成することもできます。これにより、パッケージの階層構造を作成できます。その良い例は、Flash Player API によって提供される flash.xml パッケージです。flash.xml パッケージは、flash パッケージ内にネストされています。

flash.xml パッケージには、旧バージョンの ActionScript で使用されていた古い XML パーサーが含まれています。古い XML パーサーが flash.xml パッケージに置かれる理由の 1 つは、古い XML クラスの名前が、ActionScript 3.0 で使用可能な ECMAScript for XML 仕様の機能を実装する新しい XML クラスの名前と競合するためです。

古い XML クラスをパッケージに移動するのは最初の手順として適していますが、古い XML クラスを使用するほとんどの場合で、flash.xml パッケージが読み込まれます。この場合も、必ず古い XML クラスの完全修飾名 (flash.xml.XML) を使用しなければ、同じ名前の競合が発生します。この問題を回避するために、古い XML クラスは、次の例に示すように XMLDocument という名前になりました。

```
package flash.xml
{
  class XMLDocument {}
  class XMLNode {}
  class XMLSocket {}
}
```

Flash Player API のほとんどは flash パッケージにあります。たとえば、flash.display パッケージには表示リスト API が含まれ、flash.events パッケージには新しいイベントモデルが含まれています。

パッケージの作成

ActionScript 3.0 では、パッケージ、クラス、およびソースファイルを整理する方法に大きな柔軟性を持たせています。旧バージョンの ActionScript では、ソースファイルごとにクラスが 1 つだけ許可され、ソースファイルの名前がクラスの名前と一致する必要がありました。ActionScript 3.0 では、1 つのソースファイルに複数のクラスを含めることができますが、ファイルの外部にあるコードで使用できるのは各ファイルのクラス 1 つだけです。つまり、各ファイルのクラスを 1 つだけパッケージ宣言内で宣言できます。その他のクラスはパッケージ定義の外部で定義する必要があります。これにより、追加されるクラスはソースファイル外部にあるコードに対して表示されなくなります。パッケージ定義内で定義されたクラスの名前は、ソースファイルの名前に一致している必要があります。

ActionScript 3.0 では、パッケージをより柔軟に宣言することができます。旧バージョンの ActionScript では、パッケージはソースファイルを配置するディレクトリを表すだけでした。また、`package` ステートメントでパッケージを宣言するのではなく、クラス宣言内の完全修飾クラス名の一部としてパッケージ名を含めていました。ActionScript 3.0 でもパッケージはディレクトリを表しますが、パッケージに含めることができるのはクラスだけではありません。ActionScript 3.0 では、`package` ステートメントを使用してパッケージを宣言します。つまり、パッケージの最上位で変数、関数、および名前空間を宣言することもできます。パッケージの最上位に実行可能ステートメントを含めることもできます。パッケージの最上位で変数、関数、または名前空間を宣言した場合は、そのレベルで使用できる属性は `public` および `internal` だけです。また、この宣言がクラス、変数、関数、または名前空間のいずれであるかに関係なく、`public` 属性を使用できるのはファイルごとにパッケージレベルの宣言 1 つだけです。

パッケージは、コードを構成したり、名前のコンフリクトを防いだりするのに便利です。パッケージの概念および関連のないクラス継承の概念を混同しないようにしてください。同じパッケージ内にある 2 つのクラスには共通する名前空間がありますが、この 2 つのクラスはその他で関連しているとは限りません。同様に、ネストされたパッケージは、親パッケージと意味的な関係がない場合があります。

パッケージの読み込み

パッケージ内部にあるクラスを使用する場合、パッケージまたは使用するクラスを読み込む必要があります。これは、クラスの読み込みがオプションであった ActionScript 2.0 とは異なります。

たとえば、この章で前述した SampleCode クラスの例を考えてみます。クラスが samples という名前のパッケージにある場合、SampleCode クラスを使用する前に次の import ステートメントのいずれかを使用する必要があります。

```
import samples.*;
```

または

```
import samples.SampleCode;
```

一般的に、import ステートメントは可能な限り明確にします。samples パッケージから SampleCode クラスだけを使用する場合、属するパッケージ全体ではなく SampleCode クラスだけを読み込む必要があります。パッケージ全体の読み込みにより、予期しない名前の競合が発生する場合があります。

クラスパス内にパッケージまたはクラスを定義するソースコードを配置する必要もあります。クラスパスは、読み込むパッケージおよびクラスをコンパイラが検索する場所を決定するローカルディレクトリパスのユーザー定義リストです。クラスパスは " ビルドパス " または " ソースパス " と呼ばれることもあります。

クラスまたはパッケージを適切に読み込むと、クラスの完全修飾名 (samples.SampleCode) またはクラス名のみ (SampleCode) のいずれかを使用できます。

完全修飾名は同じ名前のクラス、メソッド、またはプロパティが存在してコードがあいまいになるときに役立ちますが、すべての識別子に使用すると管理が困難になる可能性があります。たとえば、SampleCode クラスのインスタンスをインスタンス化するときに完全修飾名を使用すると、コードが冗長になります。

```
var mySample:samples.SampleCode = new samples.SampleCode();
```

ネストされたパッケージのレベルが高くなるにつれて、コードの可読性が低下します。あいまいな識別子が問題ではないことが確実な場合、単純な識別子を使用してコードを読みやすくすることができます。たとえば、クラス識別子のみを使用する場合、SampleCode クラスの新しいインスタンスのインスタンス化はかなり簡単になります。

```
var mySample:SampleCode = new SampleCode();
```

最初に適切なパッケージまたはクラスを読み込まずに識別子の名前を使用しようとすると、コンパイラがクラス定義を検出できない場合があります。また、パッケージまたはクラスを読み込んだ場合は、読み込んだ名前と競合する名前を定義しようとすると、エラーが生成されます。

パッケージの作成時には、そのパッケージのすべてのメンバーのデフォルトのアクセス制御子は internal です。この場合、デフォルトではパッケージメンバーは、そのパッケージの他のメンバーに対してのみ表示されます。パッケージ外部のコードでクラスを使用できるようにする場合、public でクラスを宣言する必要があります。たとえば、次のパッケージには、SampleCode と CodeFormatter の 2 つのクラスが含まれています。

```
// SampleCode.as ファイル
package samples
{
  public class SampleCode {}
}

// CodeFormatter.as ファイル
package samples
{
  class CodeFormatter {}
}
```

SampleCode クラスは、public クラスとして宣言されるので、パッケージの外部に表示されます。しかし、CodeFormatter クラスは、samples パッケージ内にのみ表示されます。サンプルパッケージの外部にある CodeFormatter クラスにアクセスしようとすると、次の例に示すように、エラーが発生する場合があります。

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // OK、public クラス
var myFormatter:CodeFormatter = new CodeFormatter(); // エラー
```

パッケージ外部で両方のクラスを使用する場合、両方のクラスを public として宣言する必要があります。パッケージ宣言に public 属性を適用することはできません。

完全修飾名は、パッケージを使用するときに発生する可能性がある名前のコンフリクトを解決する際に役立ちます。たとえば、同じ識別子でクラスを定義する 2 つのパッケージを読み込む場合などです。たとえば、SampleCode という名前のクラスも持つ次のパッケージがあるとします。

```
package langref.samples
{
  public class SampleCode {}
}
```

次に示すように両方のクラスを読み込むと、SampleCode クラスの参照時に名前の競合が発生することがあります。

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // 名前の競合
```

コンパイラには、どちらの SampleCode クラスを使用するのかを知る方法はありません。この競合を解決するには、次のように各クラスの完全修飾名を使用する必要があります。

```
var sample1:samples.SampleCode = new samples.SampleCode();
var sample2:langref.samples.SampleCode = new langref.samples.SampleCode();
```

メモ	C++ を使用していたプログラマは、import ステートメントと #include を混同しがちです。C++ コンパイラは一度に 1 つのファイルを処理するため、C++ では #include ディレクティブが必要で、このディレクティブはヘッダファイルが明示的に含まれていない限り、他のファイル内でクラス定義を探しません。ActionScript 3.0 には include ディレクティブがありますが、クラスやパッケージを読み込むように設計されていません。ActionScript 3.0 でクラスまたはパッケージを読み込むには、import ステートメントを使用して、パッケージを含むソースファイルをクラスパスに配置する必要があります。

名前空間

名前空間では、作成したプロパティおよびメソッドの可視性を制御することができます。public、private、protected、および internal アクセス制御指定子は、ビルトイン名前空間のようなものです。これらのあらかじめ定義されているアクセス制御指定子が要件を満たさない場合は、独自の名前空間を作成することができます。

ActionScript 実装のシンタックスおよび詳細は XML とは少し異なりますが、XML 名前空間を使い慣れていれば、ここでの説明は新しいものではないかもしれません。これまで名前空間を操作したことがない場合は、概念自体は単純ですが、その実装には特定の用語を習得する必要があります。

名前空間の操作を理解するには、プロパティまたはメソッドの名前に識別子と名前空間の 2 つの部分があることを認識することが役立ちます。識別子は名前のようなものと考えることができます。たとえば、次のクラス定義の識別子は sampleGreeting と sampleFunction() です。

```
class SampleCode
{
  var sampleGreeting:String;
  function sampleFunction () {
    trace(sampleGreeting + " from sampleFunction()");
  }
}
```

定義の前に名前空間属性がい場合、名前は常にデフォルトの internal 名前空間によって修飾されます。つまり、名前は同じパッケージの呼び出し元だけに表示されます。コンパイラが strict モードに設定されている場合は、名前空間属性を変更せずに internal 名前空間がすべての識別子に適用されることを示す警告がコンパイラから出されます。識別子をどこでも使用できるようにするためには、識別子名の前に public 属性を付ける必要があります。前のコード例では、sampleGreeting と sampleFunction() の両方とも internal の名前空間値を持ちます。

名前空間を使用するときは、次の 3 つの手順に従います。最初に、namespace キーワードを使用して名前空間を定義する必要があります。たとえば、次のコードは、version1 名前空間を定義します。

```
namespace version1;
```

次に、プロパティまたはメソッドの宣言でアクセス制御指定子の代わりに使用して、名前空間を適用します。次の例では、myFunction() という関数を version1 名前空間に配置します。

```
version1 function myFunction() {}
```

最後に、名前空間を適用すると、use ディレクティブを使用するか、識別子の名前を名前空間で修飾して参照できるようになります。次の例では、use ディレクティブから myFunction() 関数を参照します。

```
use namespace version1;
myFunction();
```

次の例に示すように、修飾名を使用して myFunction() 関数を参照することもできます。

```
version1::myFunction();
```

名前空間の定義

名前空間には、名前空間名とも呼ばれる URI (Uniform Resource Identifier) という値が含まれます。URI によって、名前空間定義を一意にすることができます。

次のいずれかの方法で名前空間定義を宣言して、名前空間を作成します。XML 名前空間を定義するように、明示的な URI で名前空間を定義することも、URI を省略することもできます。次の例では、URI を使用して名前空間を定義する方法を示します。

```
namespace flash_proxy = “http://www.adobe.com/flash/proxy”;
```

URI は、その名前空間の一意の識別ストリングとして機能します。次の例のように URI を省略した場合、コンパイラは URI の代わりに一意の内部 ID ストリングを作成します。この内部 ID ストリングにはアクセスできません。

```
namespace flash_proxy;
```

名前空間を定義すると、URI の有無に関わらず、その名前空間は同じスコープ内で再定義することはできません。同じスコープ内で以前に定義された名前空間を定義しようとすると、コンパイルエラーが発生します。

名前空間がパッケージまたはクラス内で定義された場合は、適切なアクセス制御指定子が使用されていなければ、名前空間はそのパッケージまたはクラスの外部にあるコードに対して表示されません。たとえば、次のコードは flash.utils パッケージ内で定義された `flash_proxy` 名前空間を示します。次の例で、アクセス制御指定子が存在しないことは、`flash_proxy` 名前空間が flash.utils パッケージ内のコードだけに表示され、パッケージ外のコードからは参照できないことを意味します。

```
package flash.utils
{
  namespace flash_proxy;
}
```

次のコードは、`public` 属性を使用して、`flash_proxy` 名前空間がパッケージ外のコードから参照できるようにします。

```
package flash.utils
{
  public namespace flash_proxy;
}
```

名前空間の適用

名前空間を適用するとは、名前空間に定義を配置することです。名前空間に配置できる定義には、関数、変数、および定数があります。カスタム名前空間にクラスを配置することはできません。

たとえば、public アクセス制御名前空間で宣言された関数があるとします。関数定義に public 属性を使用すると、関数はパブリックの名前空間に配置され、すべてのコードで利用できるようになります。名前空間を定義した後は、public 属性を使用する場合と同様に定義した名前空間を使用でき、カスタム名前空間を参照できるコードでその定義を使用できるようになります。たとえば、次の例に示すように、名前空間 example1 を定義すると、example1 を属性として使用して myFunction() というメソッドを追加できます。

```
namespace example1;
class someClass
{
  example1 myFunction() {}
}
```

名前空間 example1 を属性として使用して myFunction() メソッドを宣言することは、メソッドが example1 名前空間に属することを意味します。

名前空間を適用する際には、次の点に注意してください。

- 適用できる名前空間は宣言ごとに 1 つだけです。
- 名前空間属性を一度に複数の定義に適用することはできません。つまり、10 個の異なる関数に名前空間を適用する場合、10 個の異なる関数それぞれに名前空間を属性として追加する必要があります。
- 名前空間とアクセス制御指定子は相互に排他的であるため、名前空間を適用すると、アクセス制御指定子を指定することはできません。つまり、名前空間を適用すると、public、private、protected、または internal として関数またはプロパティを宣言することはできません。

名前空間の参照

public、private、protected、internal などのアクセス制御名前空間で宣言されたメソッドまたはプロパティを使用するとき、名前空間を明示的に参照する必要はありません。このような特別な名前空間へのアクセスはコンテキストによって制御されるからです。たとえば、private 名前空間に配置された定義は、自動的に同じクラス内のコードで利用できるようになります。しかし、定義した名前空間では、こうした状況依存性は存在しません。カスタム名前空間に配置したメソッドまたはプロパティを使用するには、名前空間を参照する必要があります。

use namespace ディレクティブで名前空間を参照できます。また、名前修飾子 (::) を使用して名前空間で名前を修飾できます。use namespace ディレクティブで名前空間を参照すると、修飾されていない識別子に名前空間を適用できるように、名前空間が開きます。たとえば、example1 名前空間を定義している場合、use namespace example1 を使用して、その名前空間内の名前にアクセスすることができます。

```
use namespace example1;
myFunction();
```

一度に複数の名前空間を開くことができます。use namespace で名前空間を開くと、名前空間はそれ自体が開かれたコードブロック全体で開かれたままになります。名前空間を明示的に閉じることはできません。

しかし、複数の名前空間を開くと、名前のコンフリクトが起こりやすくなります。名前空間を開かない場合は、メソッドまたはプロパティの名前を名前空間と名前修飾子で修飾すると、use namespace ディレクティブを使用する必要はありません。たとえば、次のコードは、example1 名前空間で名前 myFunction() を修飾する方法を示します。

```
example1::myFunction();
```

名前空間の使用

Flash Player API の一部である flash.utils.Proxy クラスで名前のコンフリクトを防ぐために使用する名前空間の実際例を参照できます。ActionScript 2.0 の Object.__resolve プロパティに代わる Proxy クラスを使用すると、エラーが発生する前に、未定義のプロパティまたはメソッドへの参照を取得できます。Proxy クラスのすべてのメソッドは、名前の競合を避けるため flash_proxy 名前空間に置かれます。

flash_proxy 名前空間の使用方法についての理解を深めるために、Proxy クラスの使用方法を理解する必要があります。Proxy クラスの機能は、Proxy クラスを継承するクラスでのみ使用できます。つまり、オブジェクト上で Proxy クラスのメソッドを使用する場合、そのオブジェクトのクラス定義は Proxy クラスを拡張する必要があります。たとえば、未定義のメソッドへの呼び出しの試みを取得する場合、Proxy クラスを拡張し、Proxy クラスの callProperty() メソッドをオーバーライドします。

名前空間の実装には通常、名前空間の定義、適用、および参照の 3 つの手順を実行します。しかし、Proxy クラスのメソッドを明示的に呼び出していないため、flash_proxy 名前空間は定義および適用されるだけで、参照されません。Flash Player API では、flash_proxy 名前空間を定義し、Proxy クラスに適用します。コードは、flash_proxy 名前空間を Proxy クラスを拡張するクラスに適用するだけです。

flash_proxy 名前空間は、次のような方法で flash.utils パッケージで定義されます。

```
package flash.utils
{
  public namespace flash_proxy;
}
```

次の Proxy クラスからの抜粋に示すように、名前空間が Proxy クラスのメソッドに適用されます。

```
public class Proxy
{
  flash_proxy function callProperty(name:*, ... rest):*
  flash_proxy function deleteProperty(name:*):Boolean
  ...
}
```

次のコードに示すように、まず Proxy クラスと flash_proxy 名前空間の両方を読み込む必要があります。次に、Proxy クラスを拡張するようにクラスを宣言する必要があります。strict モードでコンパイルしている場合は、dynamic 属性を追加する必要もあります。callProperty() メソッドをオーバーライドするときは、flash_proxy 名前空間を使用する必要があります。

```
package
{
  import flash.utils.Proxy;
  import flash.utils.flash_proxy;

  dynamic class MyProxy extends Proxy
  {
    flash_proxy override function callProperty(name:*, ...rest):*
    {
      trace("method call intercepted: " + name);
    }
  }
}
```

MyProxy クラスのインスタンスを作成し、次の例で呼び出される testing() メソッドなどの未定義のメソッドを呼び出すと、Proxy オブジェクトがメソッドの呼び出しを取得し、オーバーライドされた callProperty() メソッド内のステートメントを実行します (この場合は、単純な trace() ステートメント)。

```
var mySample:MyProxy = new MyProxy();
mySample.testing(); // 取得されたメソッドの呼び出し :
```

flash_proxy 名前空間内に Proxy クラスのメソッドを配置すると、2 つの利点があります。1 つ目は、別の名前空間があると、Proxy クラスを拡張するクラスのパブリックインターフェイスを整理できます。Proxy クラスには、オーバーライドできるメソッドが 10 個以上ありますが、これらのメソッドは直接呼び出せるように設計されていません。これらすべてをパブリック名前空間に配置すると、混乱の元になります。2 つ目は、Proxy サブクラスに Proxy クラスのメソッドと一致する名前のインスタンスメソッドが含まれている場合、flash_proxy 名前空間を使用すると、名前のコンフリクトを回避することができます。たとえば、独自のメソッド callProperty() に名前を付けるとします。この callProperty() メソッドは別の名前空間にあるため、次のコードは有効です。

```
dynamic class MyProxy extends Proxy
{
  public function callProperty() {}
  flash_proxy override function callProperty(name:*, ...rest):*
  {
    trace("method call intercepted: " + name);
  }
}
```

名前空間は、4 つのアクセス制御指定子 (public、private、internal、および protected) では実行できない方法でメソッドまたはプロパティにアクセスする場合にも便利です。たとえば、複数のパッケージに分散するいくつかのユーティリティメソッドがあるとします。これらのメソッドをパブリックにしないで、すべてのパッケージで利用できるようにするには、新しい名前空間を作成して、独自のアクセス制御指定子として使用します。

次の例では、ユーザー定義の名前空間を使用して、異なるパッケージにある 2 つの関数をグループ化します。これらを同じ名前空間にグループ化することにより、両方の関数を 1 つの use namespace ステートメントを介してクラスまたはパッケージに表示することができます。

この例では、4 つのファイルを使用してその手法を示します。ファイルはすべてクラスパス内にある必要があります。1 つ目のファイル "myInternal.as" を使用して myInternal 名前空間を定義します。このファイルは example パッケージにあるため、example という名前のフォルダに配置する必要があります。名前空間は public とマークされているので、他のパッケージに読み込むことができます。

```
// "example" フォルダ内の "myInternal.as"
package example
{
  public namespace myInternal = "http://www.adobe.com/2006/actionscript/
  examples";
}
```

2 つ目と 3 つ目のファイル "Utility.as" と "Helper.as" は、他のパッケージで使用できるメソッドを含むクラスを定義します。Utility クラスは example.alpha パッケージ内にあります。つまり、"example" フォルダのサブフォルダである alpha という名前のフォルダにファイルを配置する必要があります。Helper クラスは example.beta パッケージ内にあります。つまり、"example" フォルダのサブフォルダでもある beta という名前のフォルダにファイルを配置する必要があります。これらのパッケージ example.alpha と example.beta の両方とも名前空間を使用する前に読み込む必要があります。

```
// "example/alpha" フォルダ内の "Utility.as"
package example.alpha
{
  import example.myInternal;

  public class Utility
  {
    private static var _taskCounter:int = 0;

    public static function someTask()
    {
      _taskCounter++;
    }

    myInternal static function get taskCounter():int
    {
      return _taskCounter;
    }
  }
}
```

```
// "example/beta" フォルダ内の "Helper.as"
package example.beta
{
  import example.myInternal;

  public class Helper
  {
    private static var _timeStamp:Date;

    public static function someTask()
    {
      _timeStamp = new Date();
    }

    myInternal static function get lastCalled():Date
    {
      return _timeStamp;
    }
  }
}
```

4 つ目のファイル "NamespaceUseCase.as" は、メインアプリケーションクラスで、"example" フォルダと兄弟関係になければなりません。Adobe Flash CS3 Professional では、このクラスは FLA の document クラスとして使用されます。NamespaceUseCase クラスも myInternal 名前空間を読み込み、この名前空間を使用して、他のパッケージにある 2 つの静的メソッドを呼び出します。この例では、コードを簡略化するためにのみ静的メソッドを使用します。静的メソッドおよびインスタンスメソッドは両方とも myInternal 名前空間に配置することができます。

```
// NamespaceUseCase.as
package
{
  import flash.display.MovieClip;
  import example.myInternal;       // 名前空間を読み込む
  import example.alpha.Utility;    // Utility クラスを読み込む
  import example.beta.Helper;      // Helper クラスを読み込む

  public class NamespaceUseCase extends MovieClip
  {
    public function NamespaceUseCase()
    {
      use namespace myInternal;

      Utility.someTask();
      Utility.someTask();
      trace(Utility.taskCounter); // 2

      Helper.someTask();
      trace(Helper.lastCalled);   // [ 最後に someTask() が呼び出された回数 ]
    }
  }
}
```

変数

変数を使用すると、プログラムで使用する値を格納できます。変数を宣言するには、変数名を使用した var ステートメントを使用する必要があります。ActionScript 2.0 では、型注釈を使用する場合にのみ var ステートメントを使用する必要があります。ActionScript 3.0 では、常に var ステートメントを使用する必要があります。たとえば、ActionScript の次の行は、i という名前の変数を宣言します。

```
var i;
```

変数を宣言するときに var ステートメントを省略すると、strict モードではコンパイルエラーが発生し、standard モードではランタイムエラーが発生します。たとえば、次のコード行は、変数 i があらかじめ定義されていない場合はエラーになります。

```
i; // i があらかじめ定義されていない場合はエラー
```

変数をデータ型に関連付ける場合は、変数を宣言するときに行います。変数の型を指定しないで変数を宣言することができますが、strict モードではコンパイラ警告が生成されます。変数の型を指定するには、変数の名前にコロン (:) と変数の型を追加します。たとえば、次のコードは int 型の変数 i を宣言します。

```
var i:int;
```

代入演算子 (=) を使用して、変数に値を割り当てることができます。たとえば、次のコードは、変数 i を宣言し、その変数に値 20 を割り当てます。

```
var i:int;
i = 20;
```

次の例に示すように、変数を宣言するときに同時に変数を割り当てる方が便利です。

```
var i:int = 20;
```

変数を宣言するときに変数に値を割り当てる手法は、整数やストリングなどのプリミティブな値を割り当てる場合だけではなく、配列を作成する場合やクラスのインスタンスをインスタンス化する場合にもよく使用されています。次の例は、コードを1行使用して宣言され、値が割り当てられる配列を示します。

```
var numArray:Array = ["zero", "one", "two"];
```

new 演算子を使用して、クラスのインスタンスを作成することができます。次の例では、CustomClass というクラスのインスタンスを作成し、新しく作成されたクラスインスタンスへの参照を customItem という変数に割り当てます。

```
var customItem:CustomClass = new CustomClass();
```

複数の変数を宣言する場合は、カンマ演算子 (,) を使用して変数を区切り、1行のコードですべての変数を宣言することができます。たとえば、次のコードは、コード1行で 3 つの変数を宣言します。

```
var a:int, b:int, c:int;
```

同じコード行で、各変数に値を割り当てることもできます。たとえば、次のコードは、3 つの変数 (a、b、および c) を宣言し、各変数に値を割り当てます。

```
var a:int = 10, b:int = 20, c:int = 30;
```

カンマ演算子を使用して変数宣言を 1 つのステートメントにグループ化できますが、コードの可読性が低下する可能性があります。

変数のスコープについて

変数の " スコープ " とは、レキシカル参照によって変数にアクセスできるコードの範囲です。" グローバル " 変数は、コードのすべての範囲で定義される変数で、" ローカル " 変数は、コードの一部分でのみ定義される変数です。ActionScript 3.0 では、変数には、常にその変数が宣言された関数またはクラスのスコープが割り当てられます。グローバル変数は、関数またはクラスの定義の外部で定義する変数です。たとえば、次のコードは、関数の外部で宣言して、グローバル変数 strGlobal を作成します。この例では、グローバル変数は関数定義の内部および外部の両方で利用できることを示します。

```
var strGlobal:String = "Global";
function scopeTest()
{
  trace(strGlobal); // グローバル
}
scopeTest();
trace(strGlobal); // グローバル
```

ローカル変数を宣言するには、関数定義内で変数を宣言します。ローカル変数を定義できるコードの最小範囲が関数定義です。関数内で宣言されたローカル変数は、その関数内でのみ存在します。たとえば、変数 str2 を関数 localScope() の中で宣言すると、この変数は関数の外部では利用できません。

```
function localScope()
{
  var strLocal:String = "local";
}
localScope();
trace(strLocal); // strLocal がグローバルに定義されていないことによるエラー
```

ローカル変数に使用した変数名がグローバル変数として宣言されている場合、ローカル変数がスコープ内にある間は、ローカル定義がグローバル定義を非表示に (シャドウ) します。ただし、グローバル変数は関数の外でも存在しています。たとえば、次のコードは、グローバルストリング変数 str1 を作成した後、同じ名前のローカル変数を scopeTest() 関数内に作成します。関数内の trace ステートメントはこの変数のローカル値を出力しますが、関数の外にある trace ステートメントはこの変数のグローバル値を出力します。

```
var str1:String = "Global";
function scopeTest ()
{
  var str1:String = "Local";
  trace(str1); // ローカル
}
scopeTest();
trace(str1); // グローバル
```

C++ や Java の変数とは異なり、ActionScript の変数にはブロックレベルのスコープはありません。コードブロックは、左中括弧 ({) と右中括弧 (}) で囲まれたステートメントのグループです。C++ や Java などのプログラミング言語では、コードブロック内で宣言された変数は、コードブロック外では利用できません。このスコープ制限はブロックレベルのスコープと呼ばれますが、ActionScript にはありません。コードブロック内で変数を宣言すると、変数はそのコードブロック内だけではなく、コードブロックが属する関数の他の部分でも利用できます。たとえば、次の関数にはさまざまなブロックスコープで定義された変数が含まれています。すべての変数を関数全体で利用できます。

```
function blockTest (testArray:Array)
{
  var numElements:int = testArray.length;
  if (numElements > 0)
  {
    var elemStr:String = "Element #";
    for (var i:int = 0; i < numElements; i++)
    {
      var valueStr:String = i + ": " + testArray[i];
      trace(elemStr + valueStr);
    }
    trace(elemStr, valueStr, i);   // すべて定義済みのまま
  }
  trace(elemStr, valueStr, i); // numElements > 0 の場合、すべて定義済み
}

blockTest(["Earth", "Moon", "Sun"]);
```

ブロックレベルのスコープが存在しないので、関数が終了する前に宣言されている限り、変数を宣言する前に読み込んだり、変数に書き込んだりできます。これは、" ホイスト " と呼ばれる手法によるもので、コンパイラによりすべての変数宣言が関数の最上位に移動されます。たとえば、num 変数が宣言される前に num 変数の初期 trace() 関数が実行される場合でも、次のコードはコンパイルされます。

```
trace(num); // NaN
var num:Number = 10;
trace(num); // 10
```

しかし、コンパイラは代入ステートメントをホイストしません。このため、num の初期の trace() は数値データ型の変数のデフォルト値である NaN (非数) になります。つまり、次の例に示すように、変数が宣言される前でも変数に値を割り当てることができます。

```
num = 5;
trace(num); // 5
var num:Number = 10;
trace(num); // 10
```

デフォルト値

" デフォルト値 " とは、値を設定する前に変数に格納されている値です。変数に初めて値を設定する場合は、変数を " 初期化 " します。変数を宣言して、値を設定しない場合、その変数は初期化されません。初期化されていない変数の値は、そのデータ型によって異なります。次の表は、変数のデフォルト値をデータ型に整理して示します。

データ型	デフォルト値
Boolean	`false`
int	0
Number	`NaN`
Object	`null`
String	`null`
uint	0
宣言されていない (型注釈 `*` と同じ)	`undefined`
ユーザー定義クラスを含むその他すべてのクラス	`null`

Number 型の変数の場合、デフォルト値は `NaN` (非数) です。これは、IEEE-754 規格で定義されている特別な値で、数値を表さない値です。

変数を宣言して、そのデータ型を宣言しない場合、デフォルトのデータ型 `*` が適用されます。これは、変数の型指定がないことを示します。型指定されていない変数を値で初期化しない場合も、デフォルト値は `undefined` です。

Boolean、Number、int、および uint 以外のデータ型の場合、初期化されていない変数のデフォルト値は `null` です。これは、Flash Player API で定義されたすべてのクラスおよび作成したカスタムクラスに適用されます。

値 `null` は、Boolean、Number、int、および uint 型の変数では有効な値ではありません。値 `null` をこれらの変数に割り当てようとすると、この値はそのデータ型のデフォルト値に変換されます。Object 型の変数の場合、値 `null` を割り当てることができます。値 `undefined` を Object 型の変数に割り当てようとすると、この値は `null` に変換されます。

Number 型の変数では、変数が数値でない場合にブール値 `true` を返し、そうでない場合に `false` を返す `isNaN()` という名前の特殊なトップレベル関数があります。

データ型

" データ型 " は値のセットを定義します。たとえば、Boolean データ型は、厳密に 2 つの値 `true` と `false` のセットです。ActionScript 3.0 では、Boolean データ型に加えて、String、Number、Array などのよく使用されるデータ型が定義されています。独自のデータ型を定義するには、クラスまたはインターフェイスを使用して値のカスタムセットを定義します。ActionScript 3.0 のすべての値は、プリミティブ値または複合値にかかわらず、オブジェクトです。

" プリミティブ値 " とは、Boolean、int、Number、String、および uint のいずれかのデータ型に属する値を言います。ActionScript では、プリミティブ値はメモリとスピードを最適化できるように特別に格納されるので、通常プリミティブ値は複合値より速く操作できます。

メモ	技術的に説明すると、ActionScript ではプリミティブ値はイミュータブルオブジェクトとして内部的に格納されます。イミュータブルオブジェクトとして格納されるということは、参照渡しが値渡しと同じように効果的であることを示しています。参照は通常、値自体よりかなり小さいので、これによりメモリ使用量が削減し、実行速度が向上します。

" 複合値 " とはプリミティブ値ではない値です。複合値のセットを定義するデータ型には、Array、Date、Error、Function、RegExp、XML、XMLList などがあります。

多くのプログラム言語では、プリミティブ値とそのラッパーオブジェクトは区別されています。たとえば、Java には int プリミティブとそれをラップする java.lang.Integer クラスがあります。Java のプリミティブはオブジェクトではありませんが、そのラッパーはオブジェクトです。このため、処理によってはプリミティブを使用する方が便利な場合とラッパーオブジェクトを使用する方が適切な場合とがあります。ActionScript 3.0 では、プリミティブ値とそのラッパーオブジェクトは実際には区別できません。プリミティブ値を含むすべての値はオブジェクトです。Flash Player では、これらのプリミティブ型は、オブジェクトのように動作しながら、オブジェクトを作成するための通常のオーバーヘッドが不要な特殊なケースとして扱われます。これは、次の 2 つのコード行が同等であることを意味します。

```
var someInt:int = 3;
var someInt:int = new int(3);
```

上記に示したすべてのプリミティブデータ型と複合データ型は、ActionScript 3.0 コアクラスで定義されています。コアクラスを使用すると、`new` 演算子を使用する代わりにリテラル値を使用してオブジェクトを作成できます。たとえば、次のようにリテラル値または Array クラスのコンストラクタを使用して、配列を作成することができます。

```
var someArray:Array = [1, 2, 3]; // リテラル値
var someArray:Array = new Array(1,2,3); // Array コンストラクタ
```

型チェック

型チェックは、コンパイル時または実行時のいずれかの時点で行われます。C++ や Java のような型指定を静的に行う言語では、コンパイル時に型チェックを行います。Smalltalk や Python のような型指定を動的に行う言語では、実行時に型チェックを行います。ActionScript 3.0 は型指定を動的に行う言語なので、実行時に型チェックが行われますが、"strict モード " と呼ばれる特別なコンパイラモードでのコンパイル時型チェックもサポートされています。strict モードでは、型チェックはコンパイル時と実行時の両方で行われますが、standard モードでは、型チェックは実行時にのみ行われます。

型指定を動的に行う言語では、コードを非常に柔軟に構成できる反面、型指定エラーが実行時に明らかになるという難点があります。型指定を静的に行う言語では、コンパイル時に型指定エラーが報告されますが、コンパイル時に型情報が分かっている必要があるという難点があります。

コンパイル時の型チェック

コンパイル時の型チェックは、大規模なプロジェクトでよく利用されます。プロジェクトの規模が拡大するにつれて、通常はデータ型の柔軟性よりできるだけ早い時点で型指定エラーを把握する方が重要になるためです。これは、デフォルトで Adobe Flash CS3 と Adobe Flex Builder 2 の ActionScript コンパイラが strict モードで実行するように設定されているからです。

コンパイル時に型チェックを行うためには、コンパイラがコード内の変数または式のデータ型情報を分かっている必要があります。変数のデータ型を明示的に宣言するには、変数名の接尾辞としてコロン演算子 (`:`) とその後ろにデータ型を追加します。データ型をパラメータに関連付けるには、コロン演算子の後ろにデータ型を使用します。たとえば、次のコードは `xParam` パラメータにデータ型の情報を追加し、変数 `myParam` を明示的なデータ型で宣言します。

```
function runtimeTest(xParam:String)
{
  trace(xParam);
}
var myParam:String = “hello”;
runtimeTest(myParam);
```

strict モードでは、ActionScript コンパイラは、型の不一致をコンパイルエラーとしてレポートします。たとえば、次のコードは、Object 型の関数パラメータ xParam を宣言した後で、このパラメータに String 型と Number 型の値を割り当てようとします。これは、strict モードでコンパイルエラーを発生します。

```
function dynamicTest(xParam:Object)
{
  if (xParam is String)
  {
    var myStr:String = xParam; // strict モードでのコンパイルエラー
    trace("String: " + myStr);
  }
  else if (xParam is Number)
  {
    var myNum:Number = xParam; // strict モードでのコンパイルエラー
    trace("Number: " + myNum);
  }
}
```

ただし、strict モードの場合でも、型指定されていない代入ステートメントの右側をそのままにして、コンパイル時の型チェックを行わないように選択できます。変数または式に型指定なしとマークするには、型注釈を省略するか、特別なアスタリスク (*) 型注釈を使用します。たとえば、上記の例で、型注釈を使用しないように xParam パラメータを変更して、strict モードでコードをコンパイルします。

```
function dynamicTest(xParam)
{
  if (xParam is String)
  {
    var myStr:String = xParam;
    trace("String: " + myStr);
  }
  else if (xParam is Number)
  {
    var myNum:Number = xParam;
    trace("Number: " + myNum);
  }
}
dynamicTest(100)
dynamicTest("one hundred");
```

実行時の型チェック

ActionScript 3.0 では、実行時の型チェックは、strict モードと standard モードのいずれでコンパイルされるかに関係なく行われます。配列を必要とする関数に値 3 がパラメータとして渡されるとします。strict モードでは、値 3 は Array データ型と互換性がないため、コンパイラはエラーを生成します。strict モードを無効にし、standard モードで実行すると、コンパイラは型の不一致について報告しませんが、Flash Player による実行時の型チェックではライタイムエラーになります。

次の例では、Array 引数を必要とするにもかかわらず、値 3 が渡される `typeTest()` という名前の関数を示します。この場合、値 3 はパラメータで宣言されたデータ型 (Array) のメンバーではないので、standard モードでランタイムエラーが発生します。

```
function typeTest(xParam:Array)
{
  trace(xParam);
}
var myNum:Number = 3;
typeTest(myNum);
// ActionScript 3.0 standard モードでのランタイムエラー
```

strict モードで操作しているときでも、実行時の型指定エラーが発生する場合もあります。これは strict モードを使用しているときに、型指定されていない変数を使用してコンパイル時の型チェックをしないようにしている場合に発生します。型指定されていない変数を使用すると、型チェックは省略されるのではなく、実行時まで保留されます。たとえば、上記の例の `myNum` 変数に宣言したデータ型が存在しない場合、コンパイラは型の不一致を検出できません。ただし、Flash Player は、代入ステートメントの結果として 3 に設定された実行時の値 `myNum` を Array データ型に設定された `xParam` の型と比較するので、ランタイムエラーを生成します。

```
function typeTest(xParam:Array)
{
  trace(xParam);
}
var myNum = 3;
typeTest(myNum);
// ActionScript 3.0 のランタイムエラー
```

実行時に型チェックを行うと、コンパイル時に型チェックを行うより継承を柔軟に使用することができます。standard モードでは、型チェックを実行時まで保留することで、" アップキャスト " する場合でもサブクラスのプロパティを参照できます。アップキャストは、基本クラスを使用してクラスインスタンスの型を宣言し、クラスのインスタンス化にサブクラスを使用するときに行われます。たとえば、拡張可能な ClassBase という名前のクラスを作成することができます (`final` 属性を持つクラスは拡張できません)。

```
class ClassBase
{
}
```

続いて、次のように someString という名前の 1 つのプロパティを持つ ClassExtender という名前の ClassBase のサブクラスを作成することができます。

```
class ClassExtender extends ClassBase
{
   var someString:String;
}
```

両方のクラスを使用して、ClassBase データ型を使用して宣言されているが、ClassExtender コンストラクタを使用してインスタンス化されているクラスインスタンスを作成することができます。基本クラスにはサブクラスにないプロパティやメソッドは含まれないので、アップキャストは安全な操作と考えられています。

```
var myClass:ClassBase = new ClassExtender();
```

ただし、サブクラスには、その基本クラスに含まれないプロパティまたはメソッドが含まれます。たとえば、ClassExtender クラスには someString プロパティが含まれますが、このプロパティは ClassBase クラスには含まれません。ActionScript 3.0 の standard モードでは、次の例に示すように、myClass インスタンスを使用してコンパイル時エラーを生成せずにこのプロパティを参照することができます。

```
var myClass:ClassBase = new ClassExtender();
myClass.someString = "hello";
// ActionScript 3.0 standard モードでのエラーなし
```

is 演算子

ActionScript 3.0 で新しく導入された is 演算子を使用すると、変数または式が特定のデータ型のメンバーであるかどうかをテストすることができます。旧バージョンの ActionScript では instanceof 演算子に同じ機能がありましたが、ActionScript 3.0 では instanceof 演算子はデータ型のメンバーシップのテストに使用しません。手動による型チェックでは、instanceof 演算子の代わりに is 演算子を使用する必要があります。これは、式 x instanceof y は、x のプロトタイプチェーンに y があるかどうかをチェックするだけであるため、また ActionScript 3.0 では、プロトタイプチェーンで継承階層全体を確認できないためです。

is 演算子は、適切な継承階層を調べます。また、オブジェクトが特定のクラスのインスタンスであるかどうかだけではなく、特定のインターフェイスを実装するクラスのインスタンスであるかどうかも調べるために使用することができます。次の例では、Sprite クラス mySprite のインスタンスを作成し、is 演算子を使用して mySprite が Sprite クラスおよび DisplayObject クラスのインスタンスであるかどうか、および IEventDispatcher インターフェイスを実装しているかどうかをテストします。

```
var mySprite:Sprite = new Sprite();
trace(mySprite is Sprite);                  // true
trace(mySprite is DisplayObject);           // true
trace(mySprite is IEventDispatcher);        // true
```

is 演算子は、継承階層を調べて、mySprite が Sprite クラスおよび DisplayObject クラスと互換性があることを適切に報告します。ただし、Sprite クラスは DisplayObject クラスのサブクラスです。is 演算子は、mySprite が IEventDispatcher インターフェイスを実装するクラスを継承するかどうかも調べます。Sprite クラスは、IEventDispatcher インターフェイスを実装する EventDispatcher クラスを継承するため、is 演算子は、mySprite が同じインターフェイスを実装することを正しく報告します。

次の例は、前の例と同じテストを示していますが、is 演算子の代わりに instanceof 演算子を使用しています。instanceof 演算子は mySprite が Sprite または DisplayObject のインスタンスであることを正しく識別しますが、mySprite が IEventDispatcher インターフェイスを実装しているかどうかをテストするのに使用すると false を返します。

```
trace(mySprite instanceof Sprite);           // true
trace(mySprite instanceof DisplayObject);    // true
trace(mySprite instanceof IEventDispatcher); // false
```

as 演算子

ActionScript 3.0 で新しく追加された as 演算子を使用しても、式が指定されたデータ型のメンバーであるかどうかをチェックすることができます。しかし、is 演算子とは異なり、as 演算子はブール値を返しません。as 演算子は、true の代わりに式の値、false の代わりに null を返します。次の例は、Sprite インスタンスが DisplayObject、IEventDispatcher、および Number データ型のメンバーであるかどうかをチェックする場合に、is 演算子の代わりに as 演算子を使用した結果を示します。

```
var mySprite:Sprite = new Sprite();
trace(mySprite as Sprite); // [object Sprite]
trace(mySprite as DisplayObject); // [object Sprite]
trace(mySprite as IEventDispatcher); // [object Sprite]
trace(mySprite as Number);   // null
```

as 演算子を使用する場合、右側のオペランドはデータ型である必要があります。右側のオペランドにデータ型ではなく式を使用しようとすると、エラーになります。

ダイナミッククラス

" ダイナミック " クラスは、プロパティおよびメソッドを追加したり変更したりすることで実行時に変更可能なオブジェクトを定義します。String クラスなどの動的ではないクラスは、"sealed" クラスです。実行時に、sealed クラスにプロパティまたはメソッドを追加することはできません。

ダイナミッククラスを作成するには、クラスを宣言するときに dynamic 属性を使用します。たとえば、次のコードは Protean という名前のダイナミッククラスを作成します。

```
dynamic class Protean
{
  private var privateGreeting:String = "hi";
  public var publicGreeting:String = "hello";
  function Protean()
```

```
  {
    trace("Protean instance created");
  }
}
```

引き続き Protean クラスのインスタンスをインスタンス化する場合、クラス定義外のインスタンスにプロパティまたはメソッドを追加することができます。たとえば、次のコードは、Protean クラスのインスタンスを作成し、プロパティ aString とプロパティ aNumber をそのインスタンスに追加します。

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
trace(myProtean.aString, myProtean.aNumber); // テスト 3
```

ダイナミッククラスのインスタンスに追加したプロパティは、実行時エンティティです。したがって、型チェックは実行時に行われます。この方法で追加されたプロパティに型注釈を追加することはできません。

また、myProtean インスタンスにメソッドを追加するには、関数を定義し、その関数を myProtean インスタンスのプロパティに関連付けます。次のコードは、trace ステートメントを traceProtean() というメソッドに移動します。

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
myProtean.traceProtean = function ()
{
  trace(this.aString, this.aNumber);
};
myProtean.traceProtean(); // テスト 3
```

しかし、この方法で作成されたメソッドは、Protean クラスのプライベートプロパティまたはメソッドにアクセスできません。また、Protean クラスのパブリックプロパティまたはメソッドへの参照も、this キーワードまたはクラス名のいずれかで修飾する必要があります。次の例は、Protean クラスのプライベート変数およびパブリック変数にアクセスしようとする traceProtean() メソッドを示します。

```
myProtean.traceProtean = function ()
{
  trace(myProtean.privateGreeting); // undefined
  trace(myProtean.publicGreeting); // hello
};
myProtean.traceProtean();
```

データ型の記述

プリミティブデータ型には、Boolean、int、Null、Number、String、uint、および void があります。ActionScript コアクラスは、複合データ型 Object、Array、Date、Error、Function、RegExp、XML、および XMLList も定義します。

Boolean データ型

Boolean データ型は、`true` と `false` の 2 つの値で構成されます。Boolean 型の変数に有効な値は、この 2 つだけです。宣言されているが、初期化されていない Boolean 変数のデフォルト値は `false` です。

int データ型

int データ型は、32 ビット整数として内部に保存され、-2,147,483,648 (-2^{31}) ～ 2,147,483,647 ($2^{31} - 1$) (両端を含む) の整数のセットで構成されます。旧バージョンの ActionScript には、整数と浮動小数点数の両方に使用する Number データ型しかありませんでした。ActionScript 3.0 では、32 ビット符号付および符号なし整数の低レベルマシン型にアクセスすることができるようになりました。変数が浮動小数点数を使用しない場合、Number データ型ではなく int データ型を使用する方が速く、効率的です。

最小 int 値と最大 int 値の範囲外の整数値の場合は、Number データ型を使用します。Number データ型では、-9,007,199,254,740,992 ～ +9,007,199,254,740,992 の値 (53 ビット整数値) を処理できます。int データ型である変数のデフォルト値は 0 です。

Null データ型

Null データ型は、1 つの値 `null` だけで構成されます。これは、String データ型および Object クラスを含む複合データ型を定義するすべてのクラスのデフォルト値です。Boolean、Number、int、uint などの他のプリミティブデータ型には、値 `null` は含まれません。Flash Player では、Boolean、Number、int、または uint 型の変数に値 `null` を割り当てようとすると、`null` は該当するデフォルト値に変換されます。このデータ型を型注釈として使用することはできません。

Number データ型

ActionScript 3.0 では、Number データ型は、整数、符号なし整数、および浮動小数点数を表すことができます。ただし、パフォーマンスを最大化するために、32 ビット `int` および `uint` 型より大きい整数値に対してのみ、Number データ型を使用する必要があります。また、Number データ型には、浮動小数点数を格納することができます。浮動小数点数を格納するには、数値に小数点を含めます。小数点を省略すると、数値は整数として格納されます。

Number データ型では、2 進数浮動小数点計算のための IEEE 規格 (IEEE-754) で指定されている 64 ビット倍精度フォーマットを使用します。この規格では、64 ビットを使用して浮動小数点数を格納する方法が指定されています。1 ビットを使用して数値が正か負かを指定します。指数には 11 ビットが使用され、2 を底として格納されます。残りの 52 ビットは、指数で指定される累乗した数値である " 仮数部 " (" 仮数 " とも呼ばれます) を格納するために使用されます。

Number データ型では、ビットの一部を使用して指数を格納することで、仮数のビットすべてを使用する場合より大きな浮動小数点数を格納できます。たとえば、Number データ型で 64 ビットすべてを使用して仮数を格納した場合は、2^{65} - 1 の数値を格納できます。Number データ型では、11 ビットを使用して指数を格納することで、仮数を 2^{1023} 乗できます。

Number 型が表すことができる最大値と最小値は、`Number.MAX_VALUE` および `Number.MIN_VALUE` と呼ばれる Number クラスの静的プロパティに格納されます。

```
Number.MAX_VALUE == 1.79769313486231e+308
Number.MIN_VALUE == 4.940656458412467e-324
```

この数値の範囲が膨大である一方で、この範囲の精度が犠牲になっています。Number データ型では、仮数の格納に 52 ビットを使用します。その結果、分数 1/3 など、正確に表すためには 53 ビット以上必要な数値は近似値にしかなりません。アプリケーションで小数値を持つ絶対精度が必要な場合、2 進浮動小数点数計算ではなく、10 進浮動小数点計算を実装するソフトウェアを使用する必要があります。

Number データ型で整数値を格納する場合、仮数の 52 ビットだけが使用されます。Number データ型は、これらの 52 ビットと特殊な非表示のビットを使用して、-9,007,199,254,740,992 (-2^{53}) ～ 9,007,199,254,740,992 (2^{53}) の整数を表します。

Flash Player では、`Number` 型の変数のデフォルト値としてだけではなく、数値を返す必要があるが返さない演算の結果としても、`NaN` 値を使用します。たとえば、負の数値の平方根を計算しようとすると、その結果は `NaN` になります。その他の特殊な Number の値には " 正の無限大 " と " 負の無限大 " があります。

メモ	`0` による除算の結果は、除数も `0` の場合、`NaN` だけです。`0` による除算の結果は、被除数が正の場合に " 正の無限大 "、被除数が負の場合に " 負の無限大 " となります。

String データ型

String データ型は、16 ビット文字のシーケンスを表します。ストリングは、UTF-16 形式を使用して、Unicode 文字として内部的に格納されます。ストリングは、Java プログラミング言語の場合と同様に不変値です。String 値を操作すると、ストリングの新しいインスタンスが返されます。String データ型で宣言される変数のデフォルト値は `null` です。両方とも文字列が存在しないことを表しますが、値 `null` は、空のストリング (`""`) と同じではありません。

uint データ型

uint データ型は、32 ビット符号なし整数として内部に保存され、0 ～ 4,294,967,295 (2^{32}- 1) (両端を含む) の整数のセットで構成されます。負以外の整数が必要な特別な場合に、uint データ型を使用します。たとえば、ピクセルカラー値を表すためには uint データ型を使用する必要があります。これは、int データ型にはカラー値の処理には適していない内部符号ビットがあるためです。最大 uint 値より大きな整数値には、53 ビット整数値を処理できる Number データ型を使用します。uint データ型の変数のデフォルト値は 0 です。

void データ型

void データ型は、1 つの値 `undefined` だけで構成されます。旧バージョンの ActionScript では、`undefined` は Object クラスのインスタンスのデフォルト値でした。ActionScript 3.0 では、Object インスタンスのデフォルト値は `null` です。値 `undefined` を Object クラスのインスタンスに割り当てようとすると、Flash Player ではこの値は `null` に変換されます。型指定されていない変数には、`undefined` という値のみを割り当てることができます。型指定されていない変数は、型注釈がないか、型注釈にアスタリスク記号 (*) が使用されている変数です。`void` は、戻り値の型注釈としてのみ使用することができます。

Object データ型

Object データ型は Object クラスによって定義されます。Object クラスは、ActionScript のすべてのクラス定義の基本クラスです。ActionScript 3.0 の Object データ型は、次の 3 つの点で旧バージョンとは異なります。1 つ目は、Object データ型は、型注釈のない変数に割り当てられるデフォルトのデータ型ではなくなりました。2 つ目は、Object データ型には、Object インスタンスのデフォルト値であった値 `undefined` が含まれなくなりました。3 つ目は、ActionScript 3.0 では、Object クラスのインスタンスのデフォルト値が `null` になった点です。

旧バージョンの ActionScript では、型注釈のない変数には自動的に Object データ型が割り当てられていました。ActionScript 3.0 ではこれは行われず、真に型指定されていない変数の概念が導入されました。型注釈のない変数は、型が指定されていないと見なされます。コード内で変数の型を指定しないままにしておくことを明確に示す場合、型注釈に新しくアスタリスク記号 (*) を使用することができます。これは、型注釈を省略するのと同じことを示します。次の例は、いずれも型指定されていない変数 x を宣言する 2 つの同等のステートメントを示します。

```
var x
var x:*
```

型指定されていない変数だけが値 `undefined` を保持することができます。値 `undefined` をデータ型が指定されている変数に割り当てようとすると、Flash Player は値 `undefined` をそのデータ型のデフォルト値に変換します。Object データ型のインスタンスでは、デフォルト値は `null` です。これは、Object インスタンスに `undefined` を割り当てようとすると、Flash Player が値 `undefined` を `null` に変換することを意味します。

型変換

型変換は、値が別のデータ型の値に変換されることです。型変換は、" 暗黙的 " または " 明示的 " に行うことができます。暗黙的な変換は、" 強制型変換 " とも呼ばれ、実行時に Flash Player によって行われることがあります。たとえば、値 2 が Boolean データ型の変数に割り当てられると、Flash Player は値 2 をブール値 true に変換してから、変数に割り当てます。明示的な変換は、" キャスト " とも呼ばれ、コードであるデータ型の変数を別のデータ型に属している場合と同様に処理するようにコンパイラに指示されると実行されます。プリミティブ値が使用されている場合、キャストによって実際にあるデータ型の値が別のデータ型に変換されます。オブジェクトを異なる型にキャストするには、オブジェクト名を括弧で囲み、その前に新しい型の名前を置きます。たとえば、次のコードはブール値を取り、整数にキャストします。

```
var myBoolean:Boolean = true;
var myINT:int = int(myBoolean);
trace(myINT); // 1
```

暗黙的な変換

暗黙的な変換は、次のような状況で実行時に行われます。

- 代入ステートメント内
- 値が関数パラメータとして渡されるとき
- 関数から値が返されるとき
- 加算 (+) 演算子などの特定の演算子を使用した式内

ユーザー定義型の場合、変換される値が変換先のクラスまたは変換先のクラスから派生するクラスのインスタンスである場合に、暗黙的な変換は成功します。暗黙的な変換が成功しなかった場合は、エラーが発生します。たとえば、次のコードには、成功する暗黙的な変換と失敗する暗黙的な変換が含まれています。

```
class A {}
class B extends A {}

var objA:A = new A();
var objB:B = new B();
var arr:Array = new Array();

objA = objB; // 変換に成功した場合
objB = arr; // 変換に失敗した場合
```

プリミティブ型の場合、暗黙的な変換は、明示的な変換関数によって呼び出される同じ内部変換アルゴリズムを呼び出して処理されます。次のセクションで、これらのプリミティブ型の変換について詳細に説明します。

明示的な変換

strict モードでコンパイルする場合、明示的な変換、つまりキャストを使用すると便利です。型の不一致によるコンパイル時エラーを生成したくない場合があるためです。これは、実行時に強制的に値が適切に変換されることがわかっている場合などです。たとえば、フォームから受信したデータを操作するとき、強制的にストリング値を数値に変換することができます。次のコードは、コードが standard モードで正しく実行されている場合であっても、コンパイル時エラーを発生します。

```
var quantityField:String = "3";
var quantity:int = quantityField; // strict モードでのコンパイル時エラー
```

strict モードを引き続き使用して、ストリングを整数に変換する場合、次のように明示的な変換を使用することができます。

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // 明示的な変換に成功した場合
```

int、uint、および Number へのキャスト

任意のデータ型を 3 つの数値型 int、uint、および Number 型のいずれかにキャストできます。何らかの理由で Flash Player で数値を変換できない場合、int および uint データ型ではデフォルト値 0 が割り当てられ、Number データ型ではデフォルト値 `NaN` が割り当てられます。ブール値を数値に変換すると、`true` は値 1、`false` は値 0 になります。

```
var myBoolean:Boolean = true;
var myUINT:uint = uint(myBoolean);
var myINT:int = int(myBoolean);
var myNum:Number = Number(myBoolean);
trace(myUINT, myINT, myNum); // 1 1 1
myBoolean = false;
myUINT = uint(myBoolean);
myINT = int(myBoolean);
myNum = Number(myBoolean);
trace(myUINT, myINT, myNum); // 0 0 0
```

数字のみを含むストリング値は、数値型のいずれかに変換することができます。数値型では、負の数のように見えるストリングまたは 16 進数値 (たとえば、`0x1A`) を表すストリングも変換することができます。変換プロセスでは、ストリング値の先頭および末尾の空白は無視されます。`Number()` を使用して浮動小数点数のように見えるストリングをキャストすることもできます。小数点が含まれていると、`uint()` および `int()` は、小数点とその後の数字が切り捨てられた整数を返します。たとえば、次のストリング値は数値にキャストすることができます。

```
trace(uint("5"));      // 5
trace(uint("-5"));     // 4294967291。MAX_VALUE に達すると反対の端に戻る
trace(uint(" 27 "));   // 27
trace(uint("3.7"));    // 3
trace(int("3.7"));     // 3
trace(int("0x1A"));    // 26
trace(Number("3.7")); // 3.7
```

数値以外の文字を含むストリング値は、int() または uint() でキャストすると 0 を返し、Number() でキャストすると NaN を返します。変換プロセスでは、先頭および末尾の空白は無視されますが、ストリング値に 2 つの数値を区切る空白がある場合は、0 または NaN が返されます。

```
trace(uint("5a"));    // 0
trace(uint("ten"));   // 0
trace(uint("17 63")); // 0
```

ActionScript 3.0 では、Number() 関数は 8 進数をサポートしていません。ActionScript 2.0 の Number() 関数に先頭が 0 のストリングを指定した場合、数値は 8 進数として解釈され、10 進数に変換されます。これは、ActionScript 3.0 の Number() 関数には当てはまりません。この関数では、先頭の 0 は無視されます。たとえば、次のコードは、異なるバージョンの ActionScript を使用してコンパイルすると異なる出力を生成します。

```
trace(Number("044"));
// ActionScript 3.0 44
// ActionScript 2.0 36
```

数値型の値が別の数値型の変数に割り当てられる場合、キャストは必要ありません。strict モードでも、数値型は別の数値型に暗黙的に変換されます。これは、場合によっては、型の範囲を超えると予期しない値になることを示します。次の例では、予期しない値が生成される場合もありますが、すべて strict モードでコンパイルされます。

```
var myUInt:uint = -3; // int/Number 型の値を uint 変数に代入する
trace(myUInt); // 4294967293

var myNum:Number = sampleUINT; // int/uint 型の値を Number 変数に代入する
trace(myNum) // 4294967293

var myInt:int = uint.MAX_VALUE + 1; // Number 型の値を uint 変数に代入する
trace(myInt); // 0

myInt = int.MAX_VALUE + 1; // uint/Number 型の値を int 変数に代入する
trace(myInt); // -2147483648
```

次の表に、別のデータ型から Number、int、uint データ型にキャストした結果の概要を示します。

データ型または値	Number、int、uint 型への変換結果
Boolean	値が true の場合は 1、それ以外の場合は 0
Date	Date オブジェクトの内部表現で、1970 年 1 月 1 日午前 0 時 (世界時) からのミリ秒
null	0
Object	インスタンスが null で Number 型に変換される場合は NaN、それ以外の場合は 0
ストリング	Flash Player でストリングを数値に変換できる場合は数値、それ以外の場合は Number 型に変換されると NaN、int 型または uint 型に変換されると 0
undefined	Number 型に変換される場合は NaN、int 型または uint 型に変換される場合は 0

Boolean へのキャスト

任意の数値データ型 (uint、int、および Number) から Boolean へキャストすると、数値が 0 の場合は `false` が、それ以外の場合は `true` が返されます。Number データ型では、値 `NaN` のときも `false` になります。次の例では、数値 -1、0、および 1 をキャストした場合の結果を示します。

```
var myNum:Number;
for (myNum = -1; myNum<2; myNum++)
{
  trace("Boolean(" + myNum +") is " + Boolean(myNum));
}
```

この例による出力では、3 つの数値の内 0 だけが値 `false` を返すことを示します。

```
Boolean(-1) is true
Boolean(0) is false
Boolean(1) is true
```

ストリングが `null` または空のストリング (`""`) の場合、String 値から Boolean 値へキャストすると、`false` が返されます。それ以外の場合は、`true` を返します。

```
var str1:String;       // 初期化されていないストリングは null になる
trace(Boolean(str1));  // false

var str2:String = "";  // 空のストリング
trace(Boolean(str2));  // false

var str3:String = " "; // 空白文字のみ
trace(Boolean(str3));  // true
```

Object クラスのインスタンスから Boolean にキャストすると、インスタンスが `null` の場合は `false` が返され、それ以外の場合は `true` が返されます。

```
var myObj:Object;      // 初期化されていないオブジェクトは null になる
trace(Boolean(myObj)); // false

myObj = new Object();  // インスタンス化
trace(Boolean(myObj)); // true
```

Boolean 型の変数は、strict モードで任意のデータ型の値をキャストしないで Boolean 型変数に割り当てることができます。すべてのデータ型から Boolean データ型への暗黙的な強制型変換は、strict モードでも行われます。つまり、ほとんどのデータ型とは異なり、strict モードのエラーを防ぐために Boolean 型へのキャストは必要ありません。次の例では、すべて strict モードでコンパイルされ、実行時に意図したとおりに動作します。

```
var myObj:Object = new Object();  // インスタンス化
var bool:Boolean = myObj;
trace(bool); // true
bool = "random string";
trace(bool); // true
bool = new Array();
trace(bool); // true
bool = NaN;
trace(bool); // false
```

次の表に、他のデータ型から Boolean データ型へのキャストの結果を要約して示します。

データ型または値	Boolean 型への変換結果
String	値が `null` または空のストリング (`""`) の場合は `false`、それ以外の場合は `true`
`null`	`false`
Number、int、uint	値が `NaN` または 0 の場合は `false`、それ以外の場合は `true`
Object	インスタンスが `null` の場合は `false`、それ以外の場合は `true`

String へのキャスト

任意の数値データ型から String データ型へキャストすると、数値のストリング表現が返されます。Boolean 型から String 型にキャストすると、値が `true` の場合はストリング `"true"`、値が `false` の場合はストリング `"false"` が返されます。

Object クラスのインスタンスから String データ型にキャストすると、インスタンスが `null` の場合はストリング `"null"` が返されます。それ以外の場合は、Object クラスから String 型にキャストすると、ストリング `"[object Object]"` が返されます。

Array クラスのインスタンスから String 型にキャストすると、すべての配列エレメントのカンマ区切りリストから構成されるストリングが返されます。たとえば、次の String データ型へのキャストは、配列の 3 つのエレメントすべてから成るストリングを 1 つ返します。

```
var myArray:Array = ["primary", "secondary", "tertiary"];
trace(String(myArray)); // primary、secondary、tertiary
```

Date クラスのインスタンスから String 型にキャストすると、インスタンスに含まれる日付のストリング表現が返されます。たとえば、次の例は Date クラスインスタンスのストリング表現を返します。この出力は、太平洋標準時の場合の結果を示します。

```
var myDate:Date = new Date(2005,6,1);
trace(String(myDate)); // Fri Jul 1 00:00:00 GMT-0700 2005
```

次の表に、別のデータ型から String データ型にキャストした結果の概要を示します。

データ型または値	String 型への変換結果
Array	すべての配列エレメントで構成されるストリング
Boolean	`"true"` または `"false"`
Date	Date オブジェクトのストリング表現
`null`	`"null"`
Number、int、uint	数値のストリング表現
Object	インスタンスが null の場合は `"null"`、それ以外の場合は `"[object Object]"`

シンタックス

言語のシンタックスは、実行可能なコードを記述する際に遵守する必要がある一連の規則を定義します。

大文字と小文字の区別

ActionScript 3.0 は、大文字と小文字を区別する言語です。スペルが同じで大文字か小文字かだけが異なる識別子は、異なる識別子と見なされます。たとえば、次のコードは 2 つの異なる変数を生成します。

```
var num1:int;
var Num1:int;
```

ドットシンタックス

ドット演算子 (.) は、オブジェクトのプロパティまたはメソッドにアクセスする方法を提供します。ドットシンタックスを使用すると、インスタンス名の後にドット演算子とプロパティまたはメソッドの名前を付けたものを使ってクラスのプロパティまたはメソッドを参照できます。たとえば、次のようなクラス定義があるとします。

```
class DotExample
{
  public var prop1:String;
  public function method1():void {}
}
```

ドットシンタックスを使用して、次のコードで作成されたインスタンス名を使用する prop1 プロパティと method1() メソッドにアクセスできます。

```
var myDotEx:DotExample = new DotExample();
myDotEx.prop1 = “hello”;
myDotEx.method1();
```

パッケージを定義するときに、ドットシンタックスを使用することができます。ドット演算子を使用してネストされたパッケージを参照します。たとえば、EventDispatcher クラスは、flash というパッケージ内でネストされている events というパッケージにあります。次の式を使用して、events パッケージを参照することができます。

```
flash.events
```

この式を使用して、EventDispatcher クラスを参照することもできます。

```
flash.events.EventDispatcher
```

スラッシュシンタックス

ActionScript 3.0 では、スラッシュシンタックスはサポートされていません。スラッシュシンタックスは、ActionScript の以前のバージョンで、ムービークリップまたは変数のパスを指定するために使用されていました。

リテラル

" リテラル " は、コードに直接表示される値です。次の例は、すべてのリテラルを示します。

```
17
"hello"
-3
9.4
null
undefined
true
false
```

リテラルをグループ化して、複合リテラルを形成することもできます。配列リテラルは、角括弧 ([ ]) で囲まれ、カンマを使用して配列エレメントが区切られます。

配列リテラルを使用して配列を初期化することができます。次の例では、配列リテラルを使用して初期化される 2 つの配列を示します。new ステートメントを使用して、複合リテラルを Array クラスのコンストラクタにパラメータとして渡すことができますが、ActionScript コアクラスの Object、Array、String、Number、int、uint、XML、XMLList、および Boolean のインスタンスをインスタンス化するときに、リテラル値を直接割り当てることもできます。

```
// new ステートメントを使用する
var myStrings:Array = new Array(["alpha", "beta", "gamma"]);
var myNums:Array = new Array([1,2,3,5,8]);

// 直接リテラルを割り当てる
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1,2,3,5,8];
```

リテラルは汎用オブジェクトの初期化にも使用できます。汎用オブジェクトは Object クラスのインスタンスです。オブジェクトリテラルは、中括弧 ({ }) で囲まれ、カンマを使ってオブジェクトプロパティが区切られます。各プロパティはコロン (:) で宣言されます。コロンにより、プロパティ名とプロパティ値が区切られます。

new ステートメントを使用して汎用オブジェクトを作成し、オブジェクトリテラルをパラメータとして Object クラスコンストラクタに渡すか、宣言するインスタンスにオブジェクトリテラルを直接割り当てることができます。次の例では、新しい汎用オブジェクトを作成し、3 つのプロパティ (propA、propB、および propC) を使ってオブジェクトを初期化します。プロパティの値はそれぞれ 1、2、および 3 に設定されます。

```
// new ステートメントを使用する
var myObject:Object = new Object({propA:1, propB:2, propC:3});

// 直接リテラルを割り当てる
var myObject:Object = {propA:1, propB:2, propC:3};
```

詳細については、192 ページの「ストリングの基礎」、274 ページの「正規表現の基礎」、および 338 ページの「XML 変数の初期化」を参照してください。

セミコロン

セミコロン (;) を使用してステートメントを終了することができます。セミコロンを省略した場合は、コンパイラはコードの各行が 1 つのステートメントであると見なします。セミコロンを使用してステートメントの終了を示すことに慣れているプログラマが多いので、セミコロンを使用してステートメントを終了するとコードが読みやすくなります。

セミコロンを使用してステートメントを終了すると、1 行に複数のステートメントを配置することができますが、こうするとコードが読みにくくなることがあります。

括弧

ActionScript 3.0 では、括弧 (( )) に 3 つの使用方法があります。1 つ目は、括弧を使用して、式内の演算の順序を変更します。括弧内にグループ化された演算は、常に最初に実行されます。たとえば、括弧は次のコードの演算の順序を変更するために使用されます。

```
trace(2 + 3 * 4);    // 14
trace( (2 + 3) * 4); // 20
```

2 つ目は、次の例に示すように、括弧にカンマ演算子 (,) を使用して、一連の式を評価し、最後に実行された式の結果を返します。

```
var a:int = 2;
var b:int = 3;
trace((a++, b++, a+b)); // 7
```

3 つ目は、次の例に示すように、括弧を使用して 1 つまたは複数のパラメータを関数またはメソッドに渡します。この例では、trace( ) 関数にストリング値を渡します。

```
trace("hello"); // hello
```

コメント

ActionScript 3.0 コードは、単一行コメントと複数行コメントの 2 種類のコメントをサポートします。このコメントメカニズムは、C++ や Java のコメントメカニズムに似ています。コンパイラは、コメントとしてマークされたテキストを無視します。

単一行コメントは、2 つのスラッシュ (//) で始まり、その行の最後までです。たとえば、次のコードには単一行コメントが含まれています。

```
var someNumber:Number = 3; // 単一行コメント
```

複数行コメントは、スラッシュとアスタリスク (/*) で始まり、アスタリスクとスラッシュ (*/) で終わります。

```
/* これは複数行のコードにまたがる
複数行コメント */
```

キーワードと予約語

" 予約語 " とは、単語が ActionScript で使用するために予約されているので、ユーザーがコードで識別子として使用できない単語です。予約語には、コンパイラによってプログラム名前空間から削除される " レキシカルキーワード " が含まれます。レキシカルキーワードを識別子として使用すると、コンパイラはエラーを報告します。次の表は、ActionScript 3.0 のレキシカルキーワードの一覧です。

as	break	case	catch
class	const	continue	default
delete	do	else	extends
false	finally	for	function
if	implements	import	in
instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof
use	var	void	while
with			

" シンタックスキーワード " と呼ばれるいくつかのキーワードがあります。このキーワードは識別子として使用できますが、コンテキストによっては特別な意味になります。次の表に、ActionScript 3.0 シンタックスのキーワードを示します。

each	get	set	namespace
include	dynamic	final	native
override	static		

さらに、" 将来の予約語 " と呼ばれる識別子もいくつかあります。これらの識別子は ActionScript 3.0 では予約されていませんが、その一部は、ActionScript 3.0 を組み込んだソフトウェアでキーワードとして扱われることがあります。これらの識別子の多くはコードで使用可能ですが、使用しないことをお勧めします。以降のバージョンの言語でキーワードとして表示される可能性があるからです。

abstract	boolean	byte	cast
char	debugger	double	enum
export	float	goto	intrinsic
long	prototype	short	synchronized
throws	to	transient	type
virtual	volatile		

定数

ActionScript 3.0 では、定数を作成するために使用する `const` ステートメントがサポートされています。定数は、変更できない固定値を持つプロパティです。定数には値を 1 回だけ割り当てることができますが、定数の宣言のきわめて近くで割り当てる必要があります。たとえば、定数がクラスのメンバーとして宣言されると、宣言の一部としてのみ、またはクラスコンストラクタ内でのみ、その定数に値を割り当てることができます。

次のコードは、2 つの定数を宣言します。1 つ目の定数 `MINIMUM` には、宣言ステートメントの一部として値が割り当てられます。2 つ目の定数 `MAXIMUM` には、コンストラクタ内で値が割り当てられます。

```
class A
{
  public const MINIMUM:int = 0;
  public const MAXIMUM:int;

  public function A()
  {
    MAXIMUM = 10;
  }
}

var a:A = new A();
trace(a.MINIMUM); // 0
trace(a.MAXIMUM); // 10
```

他の方法で定数に初期値を割り当てようとすると、エラーが発生します。たとえば、クラス外部にある MAXIMUM の初期値を設定しようとすると、ランタイムエラーが発生します。

```
class A
{
  public const MINIMUM:int = 0;
  public const MAXIMUM:int;
}

var a:A = new A();
a["MAXIMUM"] = 10; // ランタイムエラー
```

Flash Player API では、さまざまな定数が定義されています。表記規則により、ActionScript の定数には大文字だけを使用し、単語はアンダースコア文字 (_) で区切ります。たとえば、MouseEvent クラス定義では、マウス入力に関連するイベントを表す定数にこの命名規則を使用します。

```
package flash.events
{
  public class MouseEvent extends Event
  {
    public static const CLICK:String               = "click";
    public static const DOUBLE_CLICK:String        = "doubleClick";
    public static const MOUSE_DOWN:String          = "mouseDown";
    public static const MOUSE_MOVE:String          = "mouseMove";
    ...
  }
}
```

演算子

演算子とは、1 つまたは複数のオペランドを取り、1 つの値を返す特殊な関数です。" オペランド " とは、演算子が入力として使用する値で、通常はリテラル、変数、または式を取ります。たとえば、次のコードでは、加算 (+) および乗算 (*) 演算子をリテラルオペランド (2、3、および 4) と共に使用して、値を返します。この値は、続いて代入 (=) 演算子によって使用され、戻り値 14 を変数 sumNumber に代入します。

```
var sumNumber:uint = 2 + 3 * 4; // uint = 14
```

演算子には、単項、二項、および三項があります。単項演算子は、オペランドを 1 つだけ取ります。たとえば、インクリメント (++) 演算子は、オペランドを 1 つだけ取るため、単項演算子です。二項演算子は、オペランドを 2 つ取ります。たとえば、除算 (/) 演算子はオペランドを 2 つ取ります。三項演算子はオペランドを 3 つ取ります。たとえば、条件 (?:) 演算子は 3 つのオペランドを取ります。

一部の演算子は " オーバーロード " されます。つまり、渡されたオペランドの種類または数に応じて動作が異なります。加算 (+) 演算子は、オペランドのデータ型によって動作が異なるオーバーロード演算子の例です。オペランドが両方とも数値の場合、加算演算子は値の合計値を返します。オペランドが両方ともストリングの場合、加算演算子は 2 つのオペランドの連結を返します。次のコード例は、演算子の動作がオペランドによってどのように異なるかを示します。

```
trace(5 + 5);     // 10
trace("5" + "5"); // 55
```

演算子は、指定されたオペランドの数によっても動作が異なります。除算 (-) 演算子は、単項および二項演算子です。オペランドが 1 つだけ指定されると、除算演算子はオペランドを否定し、その結果を返します。オペランドが 2 つ指定されると、除算演算子は 2 つのオペランドの差を返します。次の例では、最初に単項演算子、次に二項演算子として使用される除算演算子を示します。

```
trace(-3);  // -3
trace(7-2); // 5
```

演算子の優先順位と結合性

演算子の優先順位と結合性により、演算子を処理する順序が決まります。算術プログラミングに慣れている開発者にとって、コンパイラが乗算 (*) 演算子を加算 (+) 演算子より先に処理するのは自然なことですが、コンパイラは、どの演算子を最初に処理するかについて明示的な指示を必要とします。このような指示を、総称して " 演算子の優先順位 " と呼びます。ActionScript では、括弧 (( )) を使用して変更できる、演算子のデフォルトの優先順位が定義されています。たとえば、次のコードは、前の例のデフォルトの優先順位を変更して、乗算演算子の前に加算演算子を処理するようにコンパイラに強制します。

```
var sumNumber:uint = (2 + 3) * 4; // uint == 20
```

同じ優先順位の複数の演算子が同じ式にある状況があります。このような状況では、コンパイラは " 結合性 " のルールを使用して、最初に処理する演算子を決定します。代入演算子以外のすべての二項演算子は、" 左結合 " となります。つまり、左にある演算子が右にある演算子よりも先に処理されます。代入演算子と条件 (?:) 演算子は " 右結合 " となります。つまり、右にある演算子が左にある演算子よりも先に処理されます。

たとえば、" より小さい " (<) および " より大きい " (>) 演算子について考えてみます。これらの演算子の優先順位は同じです。両方の演算子を同じ式で使用すると、左の演算子が最初に処理されます。これは、両方の演算子が左結合であるためです。つまり、次の 2 つのステートメントでは同じ出力が得られます。

```
trace(3 > 2 < 1);   // false
trace((3 > 2) < 1); // false
```

" より大きい " 演算子が最初に処理され、その結果は値 true になります。これはオペランド 3 がオペランド 2 より大きいためです。次に、値 true がオペランド 1 と共に " より小さい " 演算子に渡されます。次のコードは、この中間状態を表します。

```
trace((true) < 1);
```

より小さい演算子は、値 true を数値 1 に変換し、この数値を 2 番目のオペランド 1 と比較して、値 false を返します (つまり、値 1 は 1 より小さくない)。

```
trace(1 < 1); // false
```

デフォルトの左結合を括弧演算子を使用して変更できます。演算子とオペランドを括弧で囲んで、" より小さい " 演算子を最初に処理するようにコンパイラに指示できます。次の例では、括弧演算子を使用して、前の例と同じ数値で異なる出力を生成します。

```
trace(3 > (2 < 1)); // true
```

" より小さい " 演算子が最初に処理され、その結果は値 `false` になります。これは、オペランド 2 がオペランド 1 より小さくないためです。次に、値 `false` がオペランド 3 と共に " より大きい " 演算子に渡されます。次のコードは、この中間状態を表します。

```
trace(3 > (false));
```

より大きい演算子は、値 `false` を数値 0 に変換し、この数値を別のオペランド 3 と比較して、値 `true` を返します (つまり、値 3 は 0 より大きい)。

```
trace(3 > 0); // true
```

次の表は、ActionScript 3.0 の演算子を優先順位の高いものから順に示します。同じ行に示されている演算子は優先順位が同じです。各行に示されている演算子はその下に表示される行の演算子より優先順位が高くなります。

グループ	演算子
基本	`[] {x:y} () f(x) new x.y x[y] <><\/> @ :: ..`
後置	`x++ x--`
単項	`++x --x + - ~ ! delete typeof void`
乗法	`* / %`
加算	`+ -`
ビット単位シフト	`<< >> >>>`
関係	`< > <= >= as in instanceof is`
等価	`== != === !==`
ビット単位の論理積 (AND)	`&`
ビット単位の排他的論理和 (XOR)	`^`
ビット単位の論理和 (OR)	`|`
論理積 (AND)	`&&`
論理和 (OR)	`||`
条件	`?:`
代入	`= *= /= %= += -= <<= >>= >>>= &= ^= |=`
カンマ	`,`

基本演算子

基本演算子は、Array リテラルおよび Object リテラルの作成、式のグループ化、関数の呼び出し、クラスインスタンスのインスタンス化、およびプロパティへのアクセスに使用する演算子です。

次の表の基本演算子の優先順位はすべて同じです。E4X 仕様に規定されている演算子には、(E4X) と表示されます。

演算子	実行される演算
`[]`	配列の初期化
`{x:y}`	オブジェクトの初期化
`()`	式のグループ化
`f(x)`	関数の呼び出し
`new`	コンストラクタの呼び出し
`x.y x[y]`	プロパティへのアクセス
`<></>`	XMLList オブジェクトの初期化 (E4X)
`@`	属性へのアクセス (E4X)
`::`	名前の修飾 (E4X)
`..`	子孫 XML エレメントへのアクセス (E4X)

後置演算子

後置演算子は、1 つの演算子を取り、演算子の値をインクリメントまたはデクリメントします。これらの演算子は単項演算子ですが、優先順位が高いことと、その特別なビヘイビアにより、他の単項演算子からは区別して分類されます。後置演算子をより大きい式の一部として使用する場合、後置演算子が処理される前に、式の値が返されます。たとえば、次のコードは、値がインクリメントされる前に、式の値 xNum++ が返される方法を示します。

```
var xNum:Number = 0;
trace(xNum++); // 0
trace(xNum);   // 1
```

次の表の後置演算子の優先順位はすべて同じです。

演算子	実行される演算
`++`	インクリメント (後置方式)
`--`	デクリメント (後置方式)

単項演算子

単項演算子は1つのオペランドを取ります。このグループのインクリメント (++) 演算子とデクリメント (--) 演算子は、" 前置演算子 " であり、式ではオペランドの前に表示されます。前置演算子が後置演算子と異なるのは、インクリメント演算またはデクリメント演算が、式全体の値が返される前に完了することです。たとえば、次のコードは、値がインクリメントされてから、式の値 ++xNum が返される方法を示します。

```
var xNum:Number = 0;
trace(++xNum); // 1
trace(xNum);   // 1
```

次の表の単項演算子の優先順位はすべて同じです。

演算子	実行される演算
++	インクリメント (前置方式)
--	デクリメント (後置方式)
+	単項プラス
-	単項マイナス
!	論理否定 (NOT)
~	ビット単位の論理否定 (NOT)
delete	プロパティの削除
typeof	タイプ情報を返す
void	未定義の値を返す

乗法演算子

乗法演算子は、2 つのオペランドを取り、乗算、除算、または剰余計算を実行します。

次の表の乗法演算子の優先順位はすべて同じです。

演算子	実行される演算
*	乗算
/	除算
%	剰余

加算演算子

加算演算子は 2 つのオペランドを取り、加算または減算を実行します。次の表の加算演算子の優先順位はすべて同じです。

演算子	実行される演算
+	加算
-	減算

ビット単位シフト演算子

ビット単位シフト演算子は 2 つのオペランドを取り、2 番目のオペランドで指定された範囲で最初のオペランドのビットをシフトします。次の表のビット単位シフト演算子の優先順位はすべて同じです。

演算子	実行される演算
<<	ビット単位の左シフト
>>	ビット単位の右シフト
>>>	ビット単位の符号なし右シフト

関係演算子

関係演算子は 2 つのオペランドを取り、値を比較してブール値を返します。次の表の関係演算子の優先順位はすべて同じです。

演算子	実行される演算
<	より小さい
>	より大きい
<=	より小さいか等しい
>=	より大きいか等しい
as	データ型のチェック
in	オブジェクトプロパティのチェック
instanceof	プロトタイプチェーンのチェック
is	データ型のチェック

等価演算子

等価演算子は 2 つのオペランドを取り、値を比較してブール値を返します。次の表の等価演算子の優先順位はすべて同じです。

演算子	実行される演算
`==`	等価
`!=`	不等価
`===`	厳密な等価
`!==`	厳密な不等価

ビット論理演算子

ビット論理演算子は 2 つのオペランドを取り、ビットレベルの論理演算を実行します。ビット論理演算子の優先順位は異なりますが、高いものから順に次の表に示します。

演算子	実行される演算
`&`	ビット単位の論理積 (AND)
`^`	ビット単位の排他的論理和 (XOR)
`|`	ビット単位の論理和 (OR)

論理演算子

論理演算子は 2 つのオペランドを取り、ブール値の結果を返します。論理演算子の優先順位は異なりますが、高いものから順に次の表に示します。

演算子	実行される演算
`&&`	論理積 (AND)
`||`	論理和 (OR)

条件演算子

条件演算子は三項演算子であり、3 つのオペランドを取ります。条件演算子は `if..else` 条件ステートメントを適用する簡易的な方法です。

演算子	実行される演算
`?:`	三項演算

代入演算子

代入演算子は 2 つのオペランドを取り、他のオペランドの値に基づいて 1 つのオペランドに値を割り当てます。次の表の代入演算子の優先順位はすべて同じです。

演算子	実行される演算
=	代入
*=	乗算後代入
/=	除算後代入
%=	剰余を代入
+=	加算後代入
-=	減算後代入
<<=	ビット単位での左シフト後代入
>>=	ビット単位での右シフト後代入
>>>=	ビット単位での符号なし右シフト後代入
&=	ビット単位の論理積 (AND) を代入
^=	ビット単位の排他的論理和 (XOR) を代入
|=	ビット単位の排他的論理和 (OR) を代入

条件演算

ActionScript 3.0 は、プログラムフローの制御に使用できる 3 つの基本的な条件ステートメントを提供します。

if..else

`if..else` 条件ステートメントを使用して、条件をテストし、その条件が存在する場合はコードブロックを実行し、その条件が存在しない場合は代わりのコードブロックを実行します。たとえば、次のコードは x の値が 20 を超えるかどうかをテストし、超える場合は `trace()` 関数を生成し、超えない場合は異なる `trace()` 関数を生成します。

```
if (x > 20)
{
  trace("x is > 20");
}
else
{
  trace("x is <= 20");
}
```

代わりのコードブロックを実行しない場合は、`else` ステートメントなしで `if` ステートメントを使用することができます。

if..else if

if..else if 条件ステートメントを使用して複数の条件をテストできます。たとえば、次のコードは、x の値が 20 を超えるかどうかをテストすると共に、x が負の値かどうかをテストします。

```
if (x > 20)
{
  trace("x is > 20");
}
else if (x < 0)
{
  trace("x is negative");
}
```

if または else ステートメントに続くステートメントが 1 つだけの場合は、ステートメントを中括弧で囲む必要はありません。たとえば、次のコードは中括弧を使用していません。

```
if (x > 0)
  trace("x is positive”);
else if (x < 0)
  trace("x is negative");
else
  trace("x is 0");
```

ただし、後で中括弧のない条件ステートメントがステートメントに追加された場合に予期しない動作を引き起こす可能性があるため、常に中括弧を使用することをお勧めします。たとえば、次のコードで positiveNums の値は、条件が true と評価されるかどうかに関係なく 1 増加します。

```
var x:int;
var positiveNums:int = 0;

if (x > 0)
  trace("x is positive");
  positiveNums++;

trace(positiveNums); // 1
```

switch

switch ステートメントは、同じ条件式に依存する複数の実行パスが存在する場合に便利です。一連の if..else if ステートメントに似た機能ですが、多少読みやすくなっています。switch ステートメントは、ブール値の条件をテストするのではなく、式を評価し、その結果を使用して実行するコードブロックを決定します。コードブロックは、case ステートメントで始まり、break ステートメントで終わります。たとえば、次の switch ステートメントは、Date.getDay() メソッドによって返された日数に基づいて曜日を出力します。

```
var someDate:Date = new Date();
var dayNum:uint = someDate.getDay();
switch(dayNum)
{
  case 0:
    trace("Sunday");
    break;
  case 1:
    trace("Monday");
    break;
  case 2:
    trace("Tuesday");
    break;
  case 3:
    trace("Wednesday");
    break;
  case 4:
    trace("Thursday");
    break;
  case 5:
    trace("Friday");
    break;
  case 6:
    trace("Saturday");
    break;
  default:
    trace("Out of range");
    break;
}
```

ループ

ループステートメントを使用すると、一連の値または変数を使用して特定のコードブロックを繰り返し実行できます。コードブロックは常に中括弧 ({}) で囲むことをお勧めします。コードブロックに含まれるステートメントが 1 つのみ場合は中括弧を省略できますが、条件の場合と同じ理由から省略することはお勧めしません。後で追加したステートメントがコードブロックから誤って除外される可能性が高くなるからです。コードブロックに含めるステートメントを後で追加するときに、必要な中括弧の追加を忘れた場合、そのステートメントはループの一部としては実行されません。

for

for ループを使用して、特定の値範囲で変数を繰り返し処理することができます。for ステートメントには、3 つの式を指定する必要があります。3 つの式とは、初期値に設定する変数、ループがいつ終了するかを指定する条件ステートメント、および各ループで変数の値を変更する式です。たとえば、次のコードは 5 回ループします。変数 i の値は 0 で始まり、4 で終了し、出力は数値 0 ～ 4 の数値を取り、それぞれが独自の行に出力されます。

```
var i:int;
for (i = 0; i < 5; i++)
{
  trace(i);
}
```

for..in

for..in ループは、オブジェクトのプロパティまたは配列のエレメントの繰り返し処理を行います。たとえば、for..in ループを使用して、汎用オブジェクトのプロパティを繰り返し処理することができます (オブジェクトプロパティは特定の順番で保持されないので、プロパティは一見ランダムな順番のように表示されます)。

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj)
{
  trace(i + ": " + myObj[i]);
}
// 出力 :
// x: 20
// y: 30
```

配列のエレメントの繰り返し処理を実行することもできます。

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray)
{
  trace(myArray[i]);
}
// 出力 :
// one
// two
// three
```

クラスがダイナミッククラスでない限り、オブジェクトがユーザー定義クラスのインスタンスである場合、そのプロパティの繰り返し処理を実行することはできません。ダイナミッククラスのインスタンスの場合であっても、動的に追加されるプロパティによってしか繰り返し処理を実行できません。

for each..in

for each..in ループは、XML または XMLList オブジェクトでタグ付けできるコレクションのアイテム、オブジェクトプロパティで保持される値、または配列のエレメントを繰り返し処理します。たとえば、次のコードの抜粋に示すように、for each..in ループを使用して汎用オブジェクトを繰り返し処理できますが、for..in ループとは異なり、for each..in ループのイテレータ変数にはプロパティの名前の代わりにプロパティによって保持される値が含まれます。

```
var myObj:Object = {x:20, y:30};
for each (var num in myObj)
{
  trace(num);
}
// 出力 :
// 20
// 30
```

次の例に示すように、XML または XMLList オブジェクトの繰り返し処理を実行することができます。

```
var myXML:XML = <users>
                   <fname>Jane</fname>
                   <fname>Susan</fname>
                   <fname>John</fname>
                </users>;

for each (var item in myXML.fname)
{
  trace(item);
}
/* 出力
Jane
Susan
John
*/
```

以下の例が示すように、配列のエレメントの繰り返し処理を実行することもできます。

```
var myArray:Array = ["one", "two", "three"];
for each (var item in myArray)
{
  trace(item);
}
// 出力 :
// one
// two
// three
```

オブジェクトが sealed クラスのインスタンスである場合、そのプロパティの繰り返し処理を実行することはできません。ダイナミッククラスのインスタンスの場合であっても、クラス定義の一部として定義されるプロパティである固定プロパティの繰り返し処理を実行することはできません。

while

while ループは、条件が true である限り繰り返される if ステートメントに似ています。たとえば、次のコードは for ループの例と同じ出力を生成します。

```
var i:int = 0;
while (i < 5)
{
  trace(i);
  i++;
}
```

for ループの代わりに while ループを使用することの 1 つの短所は、while ループを使用する方が無限ループが記述されやすいことです。カウンタ変数をインクリメントする式を省略した場合、for ループのサンプルコードはコンパイルされませんが、while ループのサンプルコードはコンパイルされます。i をインクリメントする式がないと、ループは無限ループになります。

do..while

do..while ループは、コードブロックが実行された後に条件がチェックされるので、少なくとも 1 回はコードブロックが実行される while ループです。次のコードは、条件を満たさない場合でも出力を生成する do..while ループの簡単な例を示します。

```
var i:int = 5;
do
{
  trace(i);
  i++;
} while (i < 5);
// 出力 : 5
```

関数

" 関数 " は、特定のタスクを実行し、プログラム内で再利用できるコードブロックです。ActionScript 3.0 には、" メソッド " と " 関数クロージャ " の 2 種類の関数があります。関数をメソッドと呼ぶか関数クロージャと呼ぶかは、関数が定義されたコンテキストによって決まります。関数をクラス定義の一部として定義した場合、またはオブジェクトのインスタンスに関連付けた場合は、メソッドと呼びます。関数がその他の方法で定義された場合は、関数クロージャと呼びます。

ActionScript では、関数は非常に重要です。ActionScript 1.0 では、たとえば、`class` キーワードが存在しなかったので、" クラス " はコンストラクタ関数で定義されました。その後、`class` キーワードが追加されましたが、ActionScript をフルに活用するには、関数について理解しておくことが重要です。しかし、これは、ActionScript の関数が C++ や Java などの言語の関数と同じように動作することを期待するプログラマにとっては難しい場合があります。経験豊富なプログラマにとっては基本的な関数の定義や呼び出しは問題がありませんが、ActionScript の関数の高度な機能の中には説明が必要なものもあります。

基本的な関数の概念

このセクションでは、基本的な関数の定義と呼び出し方法について説明します。

関数の呼び出し

関数を呼び出すには、識別子の後に括弧 (`()`) を使用します。関数に渡す関数パラメータを括弧で囲みます。たとえば、Flash Player API の最上位関数である `trace()` 関数は、本マニュアルのさまざまな場所で使用されます。

```
trace("Use trace to help debug your script");
```

関数をパラメータなしで呼び出す場合、空の括弧のペアを使用する必要があります。たとえば、パラメータを取らない `Math.random()` メソッドは、乱数を生成します。

```
var randomNum:Number = Math.random();
```

独自の関数の定義

ActionScript 3.0 の関数を定義するには、function ステートメントを使用する方法と関数式を使用する方法の 2 つがあります。どちらの方法を選択するかは、プログラミングスタイルをより静的にするか動的にするかによって決まります。静的な、つまり strict モードのプログラミングの方を好む場合は、function ステートメントで関数を定義します。そうすることに特定の必要性がある場合は、関数式で関数を定義します。関数式は、動的、つまり standard モードのプログラミングでより頻繁に使用されます。

function ステートメント

function ステートメントは、strict モードで関数を定義するのに適しています。function ステートメントは、function キーワードで始まり、次のアイテムが続きます。

- 関数の名前
- 括弧で囲まれたカンマ区切りリストで指定されたパラメータ
- 関数の本体、つまり関数を呼び出したときに実行される中括弧で囲まれた ActionScript コード

たとえば、次のコードは、パラメータを定義する関数を作成し、ストリング "hello" をパラメータ値として使用して関数を呼び出します。

```
function traceParameter(aParam:String)
{
  trace(aParam);
}

traceParameter("hello"); // hello
```

関数式

関数を宣言する 2 つ目の方法は、代入ステートメントに関数式を使用することです。関数式は、関数リテラルまたは匿名関数とも呼ばれます。これは、旧バージョンの ActionScript で広く使用されている、より冗長になる方法です。

関数式を持つ代入ステートメントは、var キーワードで始まり、次のアイテムが続きます。

- 関数の名前
- コロン演算子 (:)
- データ型を示すための Function クラス
- 代入演算子 (=)
- function キーワード
- 括弧で囲まれたカンマ区切りリストで指定されたパラメータ
- 関数の本体、つまり関数を呼び出したときに実行される中括弧で囲まれた ActionScript コード

たとえば、次のコードは関数式を使用して traceParameter 関数を宣言します。

```
var traceParameter:Function = function (aParam:String)
{
  trace(aParam);
};
traceParameter("hello"); // hello
```

function ステートメントとは異なり、関数の名前を指定していないことに注意してください。関数式が function ステートメントと異なるもう 1 つの特徴は、関数式はステートメントではなく式であることです。つまり、function ステートメントのように、関数式はそれだけでは成立しません。関数式は、通常は代入ステートメントなど、ステートメントの一部としてのみ使用することができます。次の例は、配列エレメントに代入された関数式を示します。

```
var traceArray:Array = new Array();
traceArray[0] = function (aParam:String)
{
  trace(aParam);
};
traceArray[0]("hello");
```

ステートメントと式の選択

原則として、式を使用する必要がある場合を除いて、function ステートメントを使用します。function ステートメントは、関数式より簡潔で、strict モードと standard モードで一貫した使いやすさを提供します。

function ステートメントは、関数式を含む代入ステートメントより読みやすくなります。function ステートメントを使用すると、コードが簡潔になり、`var` と `function` キーワードを両方使用する必要がある関数式よりわかりやすくなります。

function ステートメントは、2 つのコンパイラモードで一貫した使いやすさを提供します。つまり、strict モードおよび standard モードの両方でドットシンタックスを使用し、function ステートメントを使用して宣言されたメソッドを呼び出すことができます。これは、関数式を使用して宣言されたメソッドには必ずしも当てはまりません。たとえば、次のコードは、2 つのメソッドで Example というクラスを定義します。関数式で宣言される `methodExpression()` と function ステートメントで宣言される `methodStatement()` です。strict モードでは、ドットシンタックスを使用して、`methodExpression()` メソッドを呼び出すことはできません。

```
class Example
{
  var methodExpression = function() {}
  function methodStatement() {}
}

var myEx:Example = new Example();
myEx.methodExpression(); // strict モードではエラー、standard モードでは OK
myEx.methodStatement(); // strict モードと standard モードで OK
```

関数式は、実行時、つまり動的なビヘイビアを中心にしたプログラミングに適しています。strict モードを使用し、関数式で宣言されるメソッドを呼び出す必要がある場合は、次の 2 つの方法のいずれかを使用することができます。1 つ目は、ドット (`.`) 演算子ではなく、角括弧 (`[]`) を使用してメソッドを呼び出す方法です。次のメソッドの呼び出しは strict モードと standard モードの両方で成功します。

```
myExample["methodLiteral"]();
```

2 つ目は、クラス全体をダイナミッククラスとして宣言することができます。この場合、ドット演算子を使用してメソッドを呼び出すことができますが、そのクラスのすべてのインスタンスで strict モードの一部の機能が犠牲になるという短所があります。たとえば、ダイナミッククラスのインスタンスの未定義のプロパティにアクセスしようとした場合、コンパイラはエラーを生成しません。

関数式が便利な場合があります。関数式の一般的な使用法は、1 回だけ使用された後で破棄される関数に使用することです。また、一般的な使用法ではありませんが、関数をプロトタイププロパティに関連付けるために使用します。詳細については、166 ページの「プロトタイプオブジェクト」を参照してください。

function ステートメントと関数式には、どちらを使用するかを選択する際に考慮する必要がある微妙な違いが 2 つあります。1 つ目の違いは、関数式は、メモリ管理およびガベージコレクションに関してオブジェクトとして単独で存在しません。つまり、配列エレメントやオブジェクトプロパティなどの別のオブジェクトに関数式を割り当てると、コード内にその関数式への唯一の参照が作成されます。関数式が関連付けられている配列またはオブジェクトがスコープ外に移動するか、使用できなくなった場合、関数式にアクセスできなくなります。配列またはオブジェクトが削除されると、関数式が使用するメモリがガベージコレクションの対象となります。つまり、メモリは他の目的で再要求され、再利用される対象となります。

次の例では、関数式の場合、関数式が割り当てられているプロパティが削除されると、関数が利用できなくなることを示します。クラス Test は動的です。つまり、関数式を保持する `functionExp` というプロパティを追加できます。`functionExp()` 関数は、ドット演算子を使用して呼び出すことができますが、`functionExp` プロパティが削除されると、関数にアクセスできなくなります。

```
dynamic class Test {}
var myTest:Test = new Test();

// 関数式
myTest.functionExp = function () { trace("Function expression") };
myTest.functionExp();   // 関数式
delete myTest.functionExp;
myTest.functionExp();   // エラー
```

その一方で、関数が最初に function ステートメントで定義された場合、関数はそのオブジェクトとして存在し、関連付けられているプロパティを削除しても存在します。`delete` 演算子はオブジェクトのプロパティでのみ動作するので、関数 `stateFunc()` 自体を削除する呼び出しであっても動作しません。

```
dynamic class Test {}
var myTest:Test = new Test();

// function ステートメント
function stateFunc() { trace("Function statement") }
myTest.statement = stateFunc;
myTest.statement(); // function ステートメント
delete myTest.statement;
delete stateFunc;   // 無効
stateFunc();        // function ステートメント
myTest.statement(); // エラー
```

function ステートメントと関数式の 2 つ目の違いは、function ステートメントは、関数ステートメントの前に現れるステートメント内を含む、定義されたスコープ全体で存在することです。対照的に、関数式はそれ以降のステートメントに対してのみ定義されます。たとえば、次のコードは、定義される前に scopeTest() 関数を正常に呼び出します。

```
statementTest(); // statementTest

function statementTest():void
{
  trace("statementTest");
}
```

関数式は定義されるまで使用できないので、次のコードはランタイムエラーを発生します。

```
expressionTest(); // ランタイムエラー

var expressionTest:Function = function ()
{
  trace("expressionTest");
}
```

関数からの戻り値

関数から値を返すには、return ステートメントの後に、返す式またはリテラル値を使用します。たとえば、次のコードは、パラメータを表す式を返します。

```
function doubleNum(baseNum:int):int
{
  return (baseNum * 2);
}
```

return ステートメントは関数を終了させるので、次のように return ステートメントの下にあるステートメントは実行されないことに注意してください。

```
function doubleNum(baseNum:int):int {
  return (baseNum * 2);
  trace("after return"); // この trace ステートメントは実行されない
}
```

strict モードでは、戻り値の型を指定するように選択した場合、適切な型の値を返す必要があります。たとえば、次のコードは有効な値を返さないので、strict モードではエラーが発生します。

```
function doubleNum(baseNum:int):int
{
  trace("after return");
}
```

ネストされた関数

関数をネスト、つまり別の関数内で関数を宣言することができます。ネストされた関数は、関数への参照が外部コードに渡される場合を除いて、その親関数内でのみ使用できます。たとえば、次のコードは getNameAndVersion() 関数内に 2 つのネストされた関数を宣言します。

```
function getNameAndVersion():String
{
  function getVersion():String
  {
    return "9";
  }
  function getProductName():String
  {
    return "Flash Player";
  }
  return (getProductName() + " " + getVersion());
}
trace(getNameAndVersion()); // Flash Player 9
```

ネストされた関数が外部コードに渡される場合、関数クロージャとして渡されます。つまり、関数が定義されるとき、スコープ内にある定義を保持します。詳細については、127 ページの「関数クロージャ」を参照してください。

関数パラメータ

ActionScript 3.0 は、ActionScript を初めて使用するプログラマには斬新な関数パラメータの機能を備えています。値または参照によってパラメータを渡す方法は、ほとんどのプログラマが使い慣れている方法ですが、arguments オブジェクトと ... (rest) パラメータは、多くのプログラマにとって初めての方法と思われます。

引数の値渡しと参照渡し

多くのプログラミング言語では、値渡しと参照渡しによるパラメータの受け渡しの違いを理解しておくことが重要です。この違いは、コードの設計方法に影響します。

値渡しとは、関数内で使用するためにパラメータの値がローカル変数にコピーされることです。参照渡しとは、実際の値ではなく、パラメータへの参照のみが渡されることです。実際の値がコピーされるのではなく、パラメータとして渡される、変数への参照が作成され、関数内で使用するためにローカル変数に割り当てられます。関数外の変数への参照では、ローカル変数は元の変数の値を変更する機能を提供します。

ActionScript 3.0 では、値はすべてオブジェクトとして格納されているため、すべてのパラメータは参照渡しです。しかし、Boolean、Number、int、uint、String などのプリミティブデータ型に属するオブジェクトには、値渡しのように動作する特別な演算子があります。たとえば、次のコードは、xParam と yParam という 2 つの int 型パラメータを定義する passPrimitives() という関数を作成します。この 2 つのパラメータは、passPrimitives() 関数の本体内で宣言されるローカル変数に似ています。関数がパラメータ xValue および yValue で呼び出されると、パラメータ xParam および yParam は、xValue と yValue で表される int オブジェクトへの参照で初期化されます。パラメータはプリミティブなので、値渡しのように動作します。xParam と yParam は、初期状態では xValue と yValue オブジェクトへの参照のみが含まれますが、関数本体内の変数の変更によりメモリ内に新しい値のコピーが生成されます。

```
function passPrimitives(xParam:int, yParam:int):void
{
  xParam++;
  yParam++;
  trace(xParam, yParam);
}

var xValue:int = 10;
var yValue:int = 15;
trace(xValue, yValue);          // 10 15
passPrimitives(xValue, yValue); // 11 16
trace(xValue, yValue);          // 10 15
```

passPrimitives() 関数内では、xParam および yParam の値はインクリメントされますが、最後の trace ステートメントに示されているように、これは xValue および yValue の値に影響を与えません。パラメータが変数 xValue および yValue と同じ名前である場合でも同じです。これは、関数内の xValue および yValue は関数の外部にある同じ名前の変数とは別に存在するメモリ内の新しい位置を参照するためです。

プリミティブデータ型に属さないその他すべてのオブジェクトは、常に参照によって渡され、元の変数の値を変更する機能を提供します。たとえば、次のコードは、2 つのプロパティ x および y を持つ objVar というオブジェクトを作成します。このオブジェクトは、passByRef() 関数にパラメータとして渡されます。オブジェクトはプリミティブ型ではないため、参照渡しで渡されるだけではなく、参照のままでもあります。つまり、関数内のパラメータを変更すると、関数の外側にあるオブジェクトのプロパティも影響を受けます。

```
function passByRef(objParam:Object):void
{
  objParam.x++;
  objParam.y++;
  trace(objParam.x, objParam.y);
}
var objVar:Object = {x:10, y:15};
trace(objVar.x, objVar.y); // 10 15
passByRef(objVar);         // 11 16
trace(objVar.x, objVar.y); // 11 16
```

objParam パラメータは、グローバル変数 objVar と同じオブジェクトを参照します。この例の trace ステートメントからもわかるように、objParam オブジェクトの x および y プロパティを変更すると、objVar オブジェクトにも反映されます。

デフォルトのパラメータ値

ActionScript 3.0 では、関数のデフォルトのパラメータ値を宣言する機能が新しく追加されました。デフォルトのパラメータ値を使用した関数の呼び出しでデフォルト値のパラメータが省略されると、そのパラメータの関数定義で指定された値が使用されます。デフォルト値のパラメータはすべてパラメータリストの末尾に配置する必要があります。デフォルト値として割り当てられた値はコンパイル時定数である必要があります。パラメータのデフォルト値が存在すると、そのパラメータは " オプションパラメータ " になります。デフォルト値を持たないパラメータは、"必須パラメータ " と見なされます。

たとえば、次のコードは、3 つのパラメータで関数を作成します。このうち、2 つのパラメータにはデフォルト値があります。パラメータ 1 つだけで関数を呼び出す場合、そのパラメータのデフォルト値が使用されます。

```
function defaultValues(x:int, y:int = 3, z:int = 5):void
{
  trace(x, y, z);
}
defaultValues(1); // 1 3 5
```

arguments オブジェクト

パラメータが関数に渡されると、arguments オブジェクトを使用して関数に渡されたパラメータについての情報にアクセスできます。arguments オブジェクトには、次のようないくつかの重要な側面があります。

- arguments オブジェクトは、関数に渡されるすべてのパラメータを含む配列です。
- arguments.length プロパティは、関数に渡されるパラメータの数を報告します。
- arguments.callee プロパティを使用すると、関数自体を参照することができます。これは関数式の再帰呼び出しに便利です。

メモ

パラメータが arguments という名前の場合、または ... (rest) パラメータを使用する場合、arguments オブジェクトは利用できません。

ActionScript 3.0 では、関数呼び出しで関数定義内で定義されているパラメータより多いパラメータを指定できますが、パラメータ数が必須パラメータ数より少ない場合は strict モードでコンパイルエラーが生成されます。arguments オブジェクトの配列を使用すると、関数に渡されるパラメータが関数定義内で定義されたかどうかに関わらず、このパラメータにアクセスできます。次の例では、arguments 配列と arguments.length プロパティを使用して、traceArgArray( ) 関数に渡されるすべてのパラメータをトレースします。

```
function traceArgArray(x:int):void
{
  for (var i:uint = 0; i < arguments.length; i++)
  {
    trace(arguments[i]);
  }
}

traceArgArray(1, 2, 3);

// 出力 :
// 1
// 2
// 3
```

arguments.callee プロパティは、再帰を作成する場合に匿名関数でよく使用されます。このプロパティを使用すると、コードに柔軟性を持たせることができます。開発過程で再帰関数の名前が変更された場合でも、関数名ではなく arguments.callee を使用していれば、関数本体内の再帰呼び出しの変更について考慮する必要はありません。arguments.callee プロパティは、次の関数式で使用して再帰を有効にします。

```
var factorial:Function = function (x:uint)
{
  if(x == 0)
  {
    return 1;
  }
  else
  {
    return (x * arguments.callee(x - 1));
  }
}

trace(factorial(5)); // 120
```

関数宣言に ... (rest) パラメータを使用する場合は、arguments オブジェクトは利用できません。代わりに、宣言したパラメータ名を使用してパラメータにアクセスする必要があります。

ストリング "arguments" をパラメータ名として使用しないようにする必要があります。このストリングは arguments オブジェクトをシャドウするためです。たとえば、関数 traceArgArray() が arguments パラメータが追加されるように記述し直されると、関数本体内の arguments への参照は、arguments オブジェクトではなく、このパラメータを参照します。次のコードは出力を作成しません。

```
function traceArgArray(x:int, arguments:int):void
{
  for (var i:uint = 0; i < arguments.length; i++)
  {
    trace(arguments[i]);
  }
}

traceArgArray(1, 2, 3);

// 出力なし
```

旧バージョンの ActionScript の arguments オブジェクトには、caller という名前のプロパティも含まれていました。これは、現在の関数を呼び出した関数への参照です。ActionScript 3.0 には caller プロパティはありませんが、呼び出し関数の参照が必要な場合は、それ自体を参照する特別なパラメータを渡すように呼び出し関数を変更することができます。

...(rest) パラメータ

ActionScript 3.0 は、...(rest) パラメータと呼ばれる新しいパラメータ宣言を導入しています。このパラメータを使用すると、任意の数のカンマ区切りのパラメータを受け入れる配列パラメータを指定できます。パラメータには、予約語ではない名前を指定することができます。このパラメータ宣言は、指定される最後のパラメータである必要があります。このパラメータを使用すると、arguments オブジェクトは使用できなくなります。... (rest) パラメータには arguments 配列および arguments.length プロパティと同じ機能がありますが、arguments.callee のような機能はありません。... (rest) パラメータを使用する前に、arguments.callee を使用する必要はありません。

次の例は、arguments オブジェクトの代わりに ... (rest) パラメータを使用して、traceArgArray() 関数を書き直したものです。

```
function traceArgArray(... args):void
{
  for (var i:uint = 0; i < args.length; i++)
  {
    trace(args[i]);
  }
}
```

```
traceArgArray(1, 2, 3);

// 出力 :
// 1
// 2
// 3
```

... (rest) パラメータは、最後にパラメータとしてリストする限り、他のパラメータと共に使用することもできます。次の例は、最初のパラメータ x を int 型にし、2 つ目のパラメータとして ...(rest) パラメータを使用するように、traceArgArray() 関数を変更します。最初のパラメータは ...(rest) パラメータで作成された配列の一部でなくなっているので、出力は最初の値をスキップします。

```
function traceArgArray(x: int, ... args)
{
  for (var i:uint = 0; i < args.length; i++)
  {
    trace(args[i]);
  }
}

traceArgArray(1, 2, 3);

// 出力 :
// 2
// 3
```

オブジェクトとしての関数

ActionScript 3.0 の関数はオブジェクトです。関数を作成する場合、パラメータとして別の関数に渡すことができるオブジェクトを作成するだけではなく、プロパティとメソッドも関連付けることができます。

パラメータとして別の関数に渡される関数は、値渡しではなく、参照渡しによって渡されます。関数をパラメータとして渡す場合、識別子のみを使用し、メソッドを呼び出すために使用する括弧は使用しません。たとえば、次のコードは、addEventListener() メソッドへのパラメータとして clickListener() という関数を渡します。

```
addEventListener(MouseEvent.CLICK, clickListener);
```

Array.sort() メソッドは、関数を受け入れるパラメータも定義します。Array.sort() 関数の引数として使用されるカスタムソート関数の例については、218 ページの「配列のソート」を参照してください。

ActionScript を初めて使用するプログラマには奇妙に思えるかもしれませんが、他のオブジェクトと同様に関数にプロパティおよびメソッドを含めることができます。実際には、どの関数にも、その関数用に定義されたパラメータの数を格納する length という読み取り専用プロパティがあります。これは、関数に渡されたパラメータの数を報告する arguments.length プロパティとは異なります。ActionScript では、関数に渡されたパラメータの数がその関数用に定義されたパラメータの数を上回ってもかまいません。次の例では、strict モードでは渡されたパラメータの数と定義されたパラメータの数が完全に一致する必要があるので、standard モードでのみコンパイルされています。この例は 2 つのプロパティの違いを示します。

```
function traceLength(x:uint, y:uint):void
{
  trace("arguments received: " + arguments.length);
  trace("arguments expected: " + traceLength.length);
}

traceLength(3, 5, 7, 11);
/* 出力 :
受け取ったパラメータ : 4
必要なパラメータ : 2 */
```

独自の関数プロパティを定義するには、関数本体の外側で定義します。関数プロパティは、関数に関連する変数の状態を保存できる準静的なプロパティになります。たとえば、特定の関数が呼び出される回数を追跡するとします。ゲームを記述していて、ユーザーが特定のコマンドを使用する回数を追跡する場合に、こうした機能は便利ですが、この場合静的クラスプロパティを使用することもできます。次のコードは、関数宣言の外部に関数プロパティを作成し、関数が呼び出されるたびにプロパティをインクリメントします。

```
someFunction.counter = 0;

function someFunction():void
{
  someFunction.counter++;
}

someFunction();
someFunction();
trace(someFunction.counter); // 2
```

関数のスコープ

関数のスコープは、プログラム内の関数を呼び出すことができる場所だけではなく、関数がアクセスできる定義も決定します。変数識別子に適用される同じスコープの規則が関数識別子にも適用されます。グローバルスコープで宣言された関数は、コード全体で使用することができます。たとえば、ActionScript 3.0 には、isNaN()、parseInt() などのコード内のどこでも使用できるグローバル関数があります。別の関数内で宣言された、ネストされた関数は、関数が宣言されている関数内のどの場所でも使用できます。

スコープチェーン

関数が実行を開始するたびに、多数のオブジェクトおよびプロパティが作成されます。最初に、" アクティベーションオブジェクト " という特別なオブジェクトが作成され、そこには関数本体で宣言されるパラメータとローカル変数または関数が格納されます。アクティベーションオブジェクトは内部メカニズムであるため、そこに直接アクセスすることはできません。次に、" スコープチェーン " が作成され、そこには Flash Player によって識別子宣言の有無がチェックされるオブジェクトが列挙されたリストが含まれます。実行されるどの関数にも、内部プロパティに格納されるスコープチェーンがあります。ネストされた関数の場合は、スコープチェーンはそのアクティベーションオブジェクトから始まり、その後に親関数のアクティベーションオブジェクトが続きます。このように、スコープチェーンはグローバルオブジェクトに達するまで続きます。グローバルオブジェクトは、ActionScript プログラムが起動すると作成され、すべてのグローバル変数および関数を含みます。

関数クロージャ

" 関数クロージャ " は、関数の静的なスナップショットとその " レキシカル環境 " を含むオブジェクトです。関数のレキシカル環境には、関数のスコープチェーン内のすべての変数、プロパティ、メソッド、およびオブジェクトがその値と共に含まれます。関数クロージャは、オブジェクトまたはクラスとは別に、関数が実行されるたびに作成されます。関数クロージャはその関数クロージャが定義されたスコープを保持することから、関数がパラメータまたは戻り値として別のスコープに渡されると興味深い結果が生まれます。

たとえば、次のコードは 2 つの関数を作成します。作成される関数の foo() は、矩形の面積を計算する rectArea() という名前のネストされた関数を返し、bar() は、foo() を呼び出し、myProduct という名前の変数に返された関数クロージャを格納します。bar() 関数が独自のローカル変数 x (値 2 を持つ) を定義する場合でも、関数クロージャ myProduct() が呼び出されると、関数 foo() で定義された変数 x (値 40 を持つ) を保持します。このため、bar() 関数は値 8 ではなく、値 160 を返します。

```
function foo():Function
{
   var x:int = 40;
   function rectArea(y:int):int // 定義された関数クロージャ
   {
      return x * y
```

```
    }
    return rectArea;
}
function bar():void
{
    var x:int = 2;
    var y:int = 4;
    var myProduct:Function = foo();
    trace(myProduct(4)); // 呼び出された関数クロージャ
}
bar(); // 160
```

メソッドは、そのメソッドが作成されたレキシカル環境に関する情報も維持するように動作します。この特性が最も顕著なのは、メソッドがバインドメソッドを作成するそのインスタンスから抽出されるときです。関数クロージャとバインドメソッドの主な違いは、バインドメソッドの this キーワードの値は常に最初に関連付けられたインスタンスを参照しますが、関数クロージャでは this キーワードの値が変更可能という点です。詳細については、144 ページの「バインドメソッド」を参照してください。

第 4 章

ActionScript のオブジェクト指向プログラミング

4

この章では、オブジェクト指向プログラミング (OOP) をサポートする ActionScript のエレメントについて説明します。この章では、オブジェクト設計、抽象化、カプセル化、継承、ポリモーフィズムなどの OOP の一般的な原則については説明しません。ここでは、ActionScript 3.0 を使用してこれらの原則を適用する方法を中心に解説します。

ActionScript はスクリプト言語であることから、ActionScript 3.0 では OOP サポートを選択できます。これにより、プログラマはさまざまなスコープや複雑さのプロジェクトに対して最適な手法を選択できる柔軟性が得られます。小さなタスクの場合は、ActionScript と手続き型のプログラミングパラダイムを使用するだけで済みます。大規模なプロジェクトの場合は、OOP の原則を適用すると、コードを理解、保守、拡張しやすくすることができます。

目次

オブジェクト指向プログラミングの基礎 130
クラス. 132
インターフェイス 148
継承 152
高度なトピック 161
例 : GeometricShapes 170

オブジェクト指向プログラミングの基礎

オブジェクト指向プログラミングの概要

オブジェクト指向プログラミング (OOP) は、情報 (データ値) と機能が含まれる個別のエレメントであるオブジェクトとしてコードをグループ化することによって、プログラム内のコードを構成する方法です。オブジェクト指向のアプローチを使用してプログラムを構成すると、特定の情報 (アルバムタイトル、トラックタイトル、アーチスト名などの音楽情報など) を、その情報と関連付けられた共通の機能やアクション (" プレイリストにトラックを追加 " や " このアーチストのすべての曲を再生 " など) と共にグループ化できます。これらの項目が結合されて、1 つの項目、つまりオブジェクト ("Album" または "MusicTrack" など) になります。これらの値と機能を結合できることの利点として、複数の変数ではなく 1 つの変数だけを追跡すればよいこと、関連する機能をあわせて構成できること、現実により近い方法でプログラムを構成できることなどがあります。

一般的なオブジェクト指向プログラミングタスク

実際には、オブジェクト指向プログラミングは 2 つの部分で構成されています。1 つは、プログラム設計の戦略と手法です (多くの場合、" オブジェクト指向設計 " と呼ばれる)。これは幅広いトピックであり、この章では説明しません。OOP のもう 1 つの部分は実際のプログラミング構造です。この構造を特定のプログラミング言語で使用し、オブジェクト指向のアプローチを用いてプログラミングを作成できます。この章では、OOP の次の一般的なタスクについて説明します。

- クラスの定義
- プロパティ、メソッド、および get/set アクセサ (アクセサメソッド) の作成
- クラス、プロパティ、メソッド、アクセサに対するアクセスの制御
- 静的プロパティとメソッドの作成
- 列挙に類似した構造の作成
- インターフェイスの定義と使用
- クラスエレメントのオーバーライドを含む継承の使用

重要な概念と用語

次の参照リストには、この章で使用されている重要な用語が含まれます。

- 属性 : クラス定義でクラスエレメントに割り当てられる特性 (プロパティやメソッドなど)。一般に、属性は、プロパティまたはメソッドに対するプログラミングの他の部分のコードからのアクセスを可能にするかどうかを定義する場合に使用します。たとえば、`private` と `public` は属性です。プライベートメソッドは、クラス内部のコードだけから呼び出すことができますが、パブリックメソッドはプログラム内部のどのコードからも呼び出すことができます。
- クラス ： 特定の型のオブジェクトの構造とビヘイビア (そのデータ型のオブジェクトのテンプレートや設計図など) の定義。
- クラス階層 : 複数の関連するクラスの構造で、他のクラスから機能を継承するクラスを指定します。
- コンストラクタ ： クラスで定義できる特別なメソッドで、クラスのインスタンスが作成されたときに呼び出されます。コンストラクタは、通常はデフォルト値の指定に使用されますが、それ以外の場合は、オブジェクトのセットアップ処理に使用されます。
- データ型 : 特定の変数が格納できる情報のタイプ。一般に、" データ型 " の意味は " クラス " と同じです。
- ドット演算子 : ピリオド記号 (`.`)。ActionScript やその他の多くのプログラム言語で使用され、名前がオブジェクトの子エレメント (プロパティやメソッドなど) を参照することを示します。たとえば、式 `myObject.myProperty` では、ドット演算子は、用語 `myProperty` が、`myObject` と呼ばれるオブジェクトのエレメントであるいずれかの値を参照することを示します。
- 列挙 : 便宜上 1 つのクラスのプロパティとしてグループ化される、関連する定数値のセット。
- 継承 : 1 つのクラス定義に別のクラス定義のすべての機能を (通常は、それ自体の機能に加えて) 含めることができる OOP メカニズム。
- インスタンス : プログラムで作成される実際のオブジェクト。
- 名前空間 ： 基本的にはカスタム属性であり、他のコードにアクセスできるコードをより詳細に制御できます。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章のコードリストは主にデータ型の定義と操作に関するものなので、サンプルをテストするには、定義するクラスのインスタンスを作成し、そのインスタンスのプロパティまたはメソッドを操作した後、そのインスタンスのプロパティの値を確認する必要があります。これらの値を確認するには、値をステージ上のテキストフィールドインスタンスに書き込むか、`trace()` 関数を使用して値を [出力] パネルに出力します。これらの手法の詳細については、60 ページの「章内のコード例のテスト」を参照してください。

クラス

クラスは、オブジェクトの抽象表現です。クラスには、オブジェクトが保持できるデータの型およびオブジェクトが表すことができるビヘイビアに関する情報が格納されます。相互にやり取りする 2、3 個のオブジェクトのみを含む小さなスクリプトを記述する場合には、このような抽象化の利点はわかりにくいかもしれません。しかし、プログラムのスコープが拡大し、管理しなければならないオブジェクトの数が増加すると、クラスを使用することで、オブジェクトの作成方法と、オブジェクトの相互のやり取りの方法を制御しやすくなります。

ActionScript 1.0 では、Function オブジェクトを使用してクラスに似たコンストラクトを作成できました。ActionScript 2.0 では、`class`、`extends` などのキーワードによるクラスのサポートが正式に追加されました。ActionScript 3.0 では、ActionScript 2.0 で導入されたキーワードを引き続きサポートするだけでなく、`protected` および `internal` 属性によるアクセス制御の強化や、`final` および `override` キーワードによる継承の高度な制御などの新しい機能も追加されています。

Java、C++、C# などのプログラミング言語でクラスを作成した経験があれば、ActionScript でも同じように作成できます。ActionScript では、`class`、`extends`、`public` など、多くの同じキーワード名と属性名を共用しています。次のセクションでは、これらについて説明します。

メモ	この章では、" プロパティ " という用語は、変数、定数、メソッドを含む、オブジェクトまたはクラスのメンバーを示します。また、" クラス " と " 静的 " という用語は同じ意味で使用されることが頻繁にありますが、この章ではこの 2 つの用語は区別します。たとえば、" クラスプロパティ " という語句は、静的メンバーだけではなくクラスのすべてのメンバーを指します。

クラス定義

ActionScript 3.0 のクラス定義には、ActionScript 2.0 のクラス定義に使用したシンタックスと似たシンタックスを使用します。クラス定義の正しいシンタックスは、`class` キーワードの後にクラス名が必要です。クラス名の後には、中括弧 (`{}`) で囲まれたクラスの本体が続きます。たとえば、次のコードは、`visible` という名前の変数を 1 つ含む Shape という名前のクラスを作成します。

```
public class Shape
{
  var visible:Boolean = true;
}
```

シンタックスが大きく変更された点の 1 つに、パッケージ内のクラス定義があります。ActionScript 2.0 では、クラスがパッケージ内部にある場合は、クラス宣言にパッケージ名を含める必要があります。ActionScript 3.0 では package ステートメントが導入されていますが、パッケージ名は、クラス宣言ではなくパッケージ宣言に含める必要があります。たとえば、次のクラス宣言は、flash.display パッケージに含まれる BitmapData クラスを、ActionScript 2.0 および ActionScript 3.0 で定義する方法を示します。

```
// ActionScript 2.0
class flash.display.BitmapData {}

// ActionScript 3.0
package flash.display
{
  public class BitmapData {}
}
```

クラス属性

ActionScript 3.0 では、次の 4 つの属性のいずれかを使用して、クラス定義を変更できます。

属性	定義
ダイナミック	実行時にインスタンスにプロパティを追加できます。
final	別のクラスによって拡張することはできません。
internal (デフォルト)	現在のパッケージ内の参照に対して表示されます。
public	すべての場所にある参照に対して表示されます。

internal を除くこれらの属性では、属性を明示的に含めて関連付けられたビヘイビアを取得します。たとえば、クラスを定義するときに dynamic 属性を含めなかった場合、実行時にクラスインスタンスにプロパティを追加することはできません。次のコードで示すように、クラス定義の開始部分に属性を配置することで、明示的に属性を割り当てます。

```
dynamic class Shape {}
```

このリストには、abstract という名前の属性は含まれていません。ActionScript 3.0 では抽象クラスをサポートしていないためです。また、リストには private および protected という属性も含まれていません。これらの属性は、クラス定義内でのみ意味を持ち、クラス自体には適用できません。パッケージの外部でクラスをパブリックに表示しない場合は、パッケージ内にクラスを配置し、internal 属性でクラスを宣言します。または、internal と public 属性をいずれも省略することができます。この場合は、コンパイラによって internal 属性が自動的に追加されます。クラスが定義されたソースファイル外でクラスを表示しない場合は、ソースファイルの下部の、パッケージ定義の閉じ中括弧の下にクラスを配置します。

クラス本体

中括弧で囲まれたクラス本体は、クラスの変数、定数、およびメソッドを定義するために使用します。次の例は、Flash Player API の Accessibility クラスの宣言を示します。

```
public final class Accessibility
{
  public static function get active():Boolean;
  public static function updateProperties():void;
}
```

クラス本体内で名前空間を定義することもできます。次の例は、クラス本体内で名前空間を定義し、そのクラスのメソッドの属性として使用する方法を示します。

```
public class SampleClass
{
  public namespace sampleNamespace;
  sampleNamespace function doSomething():void;
}
```

ActionScript 3.0 では、クラス本体内に、定義だけでなくステートメントを含めることもできます。クラス本体内にあるが、メソッド定義の外にあるステートメントは、クラス定義が最初に検出され、関連付けられたクラスオブジェクトが作成されるときに 1 度のみ実行されます。次の例には、外部関数である `hello()` と、クラスが定義されたときに確認メッセージを出力する `trace` ステートメントの呼び出しが含まれています。

```
function hello():String
{
  trace("hola");
}
class SampleClass
{
  hello();
  trace("class created");
}
// クラス作成時に出力
hola
class created
```

旧バージョンの ActionScript とは異なり、ActionScript 3.0 では、同じクラス本体内で同じ名前を持つ静的プロパティとインスタンスプロパティを定義することができます。たとえば、次のコードは、`message` という名前の静的変数と同じ名前のインスタンス変数を宣言します。

```
class StaticTest
{
  static var message:String = "static variable";
  var message:String = "instance variable";
}
// スクリプト内で
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // 出力 : 静的変数
trace(myST.message);       // 出力 : インスタンス変数
```

クラスプロパティの属性

ActionScript オブジェクトモデルの説明では、" プロパティ " という用語は、変数、定数、メソッドなど、クラスのメンバーになり得るものすべてを意味します。しかし、『ActionScript 3.0 コンポーネントリファレンスガイド』では、" プロパティ " は狭い意味で使用され、変数であるか、getter または setter メソッドにより定義されるクラスメンバーのみを意味します。ActionScript 3.0 には、クラスのプロパティと共に使用できる一連の属性があります。次の表は、この属性の一覧です。

属性	定義
`internal` (デフォルト)	同じパッケージ内の参照に対して表示されます。
`private`	同じクラス内の参照に対して表示されます。
`protected`	同じクラスおよび派生クラス内の参照に対して表示されます。
`public`	すべての場所にある参照に対して表示されます。
静的	プロパティが、クラスのインスタンスではなくクラスに属することを指定します。
`UserDefinedNamespace`	ユーザーが定義するカスタム名前空間の名前です。

アクセス制御名前空間の属性

ActionScript 3.0 には、クラス内で定義されたプロパティへのアクセスを制御する 4 つの特別な属性、`public`、`private`、`protected`、および `internal` があります。

`public` 属性は、スクリプト内の任意の場所でプロパティを表示します。たとえば、パッケージ外部のコードでメソッドを使用するには、`public` 属性でメソッドを宣言する必要があります。これは、宣言されたキーワードが `var`、`const`、`function` のいずれであるかにかかわらず、あらゆるプロパティに当てはまります。

`private` 属性は、プロパティが定義されたクラス内の呼び出し元に対してのみプロパティを表示します。このビヘイビアは、サブクラスがスーパークラスのプライベートプロパティにアクセスできる ActionScript 2.0 の `private` 属性とは異なります。ランタイムアクセスに関しても、ビヘイビアが大きく変更されています。ActionScript 2.0 では、`private` キーワードによりアクセスが禁止されるのはコンパイル時のみで、実行時にはアクセスが禁止されません。ActionScript 3.0 では、これは当てはまりません。`private` とマークされたプロパティは、コンパイル時も実行時も使用することができません。

たとえば、次のコードは、プライベート変数を 1 つ持つ PrivateExample という単純なクラスを作成し、クラス外部からこのプライベート変数にアクセスしようとします。ActionScript 2.0 では、コンパイル時アクセスは禁止されていましたが、コンパイル時ではなく実行時にプロパティルックアップを実行するプロパティアクセス演算子 ([]) を使用することでアクセスすることができました。

```
class PrivateExample
{
  private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);    // strict モードではコンパイル時エラー
trace(myExample["privVar"]); // ActionScript 2.0 ではアクセスを許可するが、
  ActionScript 3.0 ではこれはラインタイムエラー。
```

ActionScript 3.0 では、ドット演算子 (myExample.privVar) を使用してプライベートプロパティにアクセスしようとすると、strict モードを使用している場合にコンパイル時エラーになります。それ以外の場合は、プロパティアクセス演算子 (myExample["privVar"]) を使用するときと同じように、エラーは実行時に報告されます。

次の表に、sealed (dynamic ではない) クラスに属するプライベートプロパティにアクセスしようとした結果の概要を示します。

	strict モード	standard モード
ドット演算子 (.)	コンパイル時エラー	ランタイムエラー
角括弧演算子 ([])	ランタイムエラー	ランタイムエラー

dynamic 属性で宣言されたクラスでは、プライベート変数にアクセスしようとしても、ランタイムエラーにはなりません。変数は表示されないだけなので、Flash Player は戻り値として undefined を返します。しかし、strict モードでドット演算子を使用すると、コンパイル時エラーが発生します。次の例は、PrivateExample クラスがダイナミッククラスとして宣言されている点を除き、前の例と同じです。

```
dynamic class PrivateExample
{
  private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);    // strict モードではコンパイル時エラー
trace(myExample["privVar"]); // 出力 : undefined
```

通常、ダイナミッククラスは、クラス外部のコードがプライベートプロパティにアクセスしようとすると、エラーを生成するのではなく、戻り値として undefined を返します。次の表は、ドット演算子を使用して strict モードでプライベートプロパティにアクセスしようとしたときにのみエラーが生成されることを示します。

	strict モード	standard モード
ドット演算子 (.)	コンパイル時エラー	undefined
角括弧演算子 ([])	undefined	undefined

ActionScript 3.0 で新しく導入された protected 属性は、クラス内またはサブクラス内の呼び出し元に対してプロパティを表示します。つまり、protected プロパティは、そのクラス内または継承階層内でそのクラスの下の任意の場所にあるクラスで使用することができます。これは、サブクラスが同じパッケージ内にあるか別のパッケージ内にあるかに関係なく、同じです。

ActionScript 2.0 を使い慣れていれば、この機能が ActionScript 2.0 の private 属性に似ていることがわかります。ActionScript 3.0 の protected 属性は Java の protected 属性にも似ていますが、Java の属性では同じパッケージ内の呼び出し元にアクセスできる点で異なります。protected 属性は、サブクラスで必要なメソッドまたは変数を継承チェーン外部のコードに対して非表示にするときに便利です。

ActionScript 3.0 で新しく導入された internal 属性は、パッケージ内にある呼び出し元に対してプロパティを表示します。これはパッケージ内のコードのデフォルト属性であり、次のいずれの属性も持たないあらゆるプロパティに適用されます。

- public
- private
- protected
- ユーザー定義の名前空間

internal 属性は、Java のデフォルトのアクセス制御に似ています。ただし、Java ではこのレベルのアクセスに明示的な名前はなく、他のアクセス修飾子が省略された場合にのみこのデフォルトのアクセス制御が使用されます。internal 属性は ActionScript 3.0 で使用可能で、パッケージ内の呼び出し元に対してのみプロパティを表示する意図を明示的に表すオプションがあります。

static 属性

static 属性は、var、const、または function キーワードで宣言されたプロパティと共に使用でき、クラスのインスタンスではなくクラスにプロパティを関連付けることができます。クラス外にあるコードは、インスタンス名ではなくクラス名を使用して静的プロパティを呼び出す必要があります。

静的プロパティはサブクラスに継承されませんが、サブクラスのスコープチェーンの一部です。つまり、サブクラスの本体内では、静的変数またはメソッドは、定義されたクラスを参照しなくても使用できます。詳細については、158 ページの「継承されない静的プロパティ」を参照してください。

ユーザー定義の名前空間属性

事前に定義されているアクセス制御属性の代わりに、属性として使用するカスタム名前空間を作成できます。1 つの定義に使用できる名前空間属性は 1 つだけです。また、名前空間属性をアクセス制御属性 (public、private、protected、internal) のいずれかと組み合わせて使用することはできません。名前空間の使用の詳細については、71 ページの「名前空間」を参照してください。

変数

変数は、var または const のいずれかのキーワードで宣言できます。var キーワードで宣言した変数は、スクリプトの実行中に複数回その値を変更できます。const キーワードで宣言した変数は定数と呼ばれ、1 回だけ値を割り当てることができます。初期化された定数に新しい値を割り当てようとすると、エラーが発生します。詳細については、101 ページの「定数」を参照してください。

静的変数

静的変数は、static キーワードと、var または const ステートメントのいずれかの組み合わせを使用して宣言します。クラスのインスタンスではなく、クラスに関連付けられた静的変数は、オブジェクトのクラス全体に適用される情報の格納および共有に役立ちます。たとえば、静的変数は、クラスがインスタンス化された回数を記録する場合や可能なクラスインスタンスの最大数を格納しておく場合に適しています。

次の例では、クラスのインスタンス化の回数を追跡する totalCount 変数およびインスタンス化の最大回数を格納する MAX_NUM 定数を作成します。totalCount と MAX_NUM 変数には、特定のインスタンスではなくクラス全体に適用される値が含まれているので、この 2 つの変数は静的です。

```
class StaticVars
{
  public static var totalCount:int = 0;
  public static const MAX_NUM:uint = 16;
}
```

StaticVars クラスとそのサブクラスの外部にあるコードは、StaticVars クラス自体を通じた場合にのみ、totalCount プロパティと MAX_NUM プロパティを参照できます。たとえば、次のコードは正常に動作します。

```
trace(StaticVars.totalCount); // 出力 : 0
trace(StaticVars.MAX_NUM); // 出力 : 16
```

クラスのインスタンスから静的変数にアクセスできないため、次のコードはエラーを返します。

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // エラー
trace(myStaticVars.MAX_NUM); // エラー
```

static および const の 2 つのキーワードで宣言された変数は、StaticVars クラスが MAX_NUM に対する場合と同様に、定数の宣言時に初期化する必要があります。コンストラクタまたはインスタンスメソッド内の MAX_NUM に値を割り当てることはできません。次のコードは静的定数を初期化する有効な方法でないため、エラーが生成されます。

```
// !! この方法で静的定数を初期化するとエラー
class StaticVars2
{
  public static const UNIQUESORT:uint;
  function initializeStatic():void
  {
    UNIQUESORT = 16;
  }
}
```

インスタンス変数

インスタンス変数には、var および const キーワードを使用し、static キーワードを使用せずに宣言されたプロパティが含まれます。クラス全体ではなくクラスインスタンスに関連付けられたインスタンス変数は、インスタンス固有の値の格納に便利です。たとえば、Array クラスには、この Array クラスの特定のインスタンスが保持する配列エレメントの数を格納する length という名前のインスタンスプロパティがあります。

インスタンス変数は、var または const として宣言されたかどうかにかかわらず、サブクラスではオーバーライドできません。ただし、getter および setter メソッドをオーバーライドすることにより、変数をオーバーライドするのに似た機能を実現できます。詳細については、143 ページの「get および set アクセサメソッド」を参照してください。

メソッド

メソッドは、クラス定義に含まれる関数です。クラスのインスタンスが作成されると、メソッドはそのインスタンスにバインドされます。メソッドは、クラス外で宣言された関数とは異なり、その関連付けられているインスタンスと別個に使用することはできません。

メソッドは、function キーワードを使用して定義します。次のような関数ステートメントを使用できます。

```
public function sampleFunction():String {}
```

または、関数式を割り当てる変数を次のように使用することもできます。

```
public var sampleFunction:Function = function () {}
```

多くの場合、関数式ではなく関数ステートメントを使用するのは、次のような理由のためです。

- 関数ステートメントは、関数式より簡潔で読みやすくなります。
- 関数ステートメントでは、override および final キーワードを使用できます。詳細については、156 ページの「メソッドのオーバーライド」を参照してください。

- 関数ステートメントでは、識別子 (関数の名前) とメソッド本体内のコードがより強力に結合されます。変数の値は代入ステートメントで変更できるので、変数とその関数式の結合をいつでも切り離すことができます。var ではなく const で変数を宣言するとこの問題を回避できますが、コードが読みにくくなり、override および final キーワードが使用できなくなるので、最適な方法とは言えません。

関数式を使用する必要があるのは、関数をプロトタイプオブジェクトに関連付ける場合などです。詳細については、166 ページの「プロトタイプオブジェクト」を参照してください。

コンストラクタメソッド

コンストラクタメソッドは、単にコンストラクタと呼ばれることもあり、定義されたクラスと同じ名前を共有する関数です。コンストラクタメソッドに含まれるコードは、クラスのインスタンスが new キーワードで作成されるときに実行されます。たとえば、次のコードは、status という名前のプロパティを 1 つ含む単純なクラス Example を定義します。status 変数の初期値は、コンストラクタ関数内で設定します。

```
class Example
{
  public var status:String;
  public function Example()
  {
    status = "initialized";
  }
}

var myExample:Example = new Example();
trace(myExample.status); // 出力 : initialized
```

コンストラクタメソッドにはパブリックだけを指定できますが、public 属性を使用するかどうかは任意です。コンストラクタに、private、protected、internal やその他のアクセス制御指定子を使用することはできません。また、ユーザー定義の名前空間をコンストラクタメソッドで使用することもできません。

コンストラクタは、super( ) ステートメントを使用して直接のスーパークラスのコンストラクタを明示的に呼び出すことができます。スーパークラスのコンストラクタを明示的に呼び出さない場合は、コンパイラによってコンストラクタ本体の最初のステートメントの前に呼び出しが自動的に挿入されます。スーパークラスへの参照として super 接頭辞を使用し、スーパークラスのメソッドを呼び出すこともできます。同じコンストラクタ本体で super( ) と super を使用する場合は、必ず最初に super( ) を呼び出します。呼び出さないと、super 参照は意図したとおりに動作しません。super( ) コンストラクタも、throw または return ステートメントの前に呼び出す必要があります。

次に、super() コンストラクタを呼び出す前に super 参照を使用しようとした場合の例を示します。新しいクラス ExampleEx は Example クラスを拡張します。ExampleEx コンストラクタは、super() を呼び出す前に、スーパークラスで定義された status 変数にアクセスしようとします。ExampleEx コンストラクタ内の trace() ステートメントは、値 null を生成します。これは、super() コンストラクタが実行されるまで status 変数を使用できないからです。

```
class ExampleEx extends Example
{
  public function ExampleEx()
  {
    trace(super.status);
    super();
  }
}

var mySample:ExampleEx = new ExampleEx(); // 出力 : null
```

コンストラクタ内で return ステートメントを使用することはできますが、値を返すことはできません。つまり、return ステートメントに式または値を関連付けることはできません。したがって、コンストラクタメソッドは値を返すことができず、戻り値の型を指定できません。

クラスでコンストラクタメソッドを定義しなかった場合、コンパイラが自動的に空のコンストラクタを作成します。クラスが別のクラスを拡張する場合、コンパイラには、そのコンパイラが生成するコンストラクタ内の super() 呼び出しが含まれます。

静的メソッド

静的メソッドは、クラスメソッドとも呼ばれ、static キーワードで宣言されたメソッドです。静的メソッドは、クラスのインスタンスではなくクラスに関連付けられ、個々のインスタンスの状態以外のものに影響を与える機能のカプセル化に役立ちます。静的メソッドはクラス全体に関連付けられるため、クラスのインスタンスではなくクラスからのみアクセスできます。

静的メソッドは、クラスインスタンスの状態に影響を与えるだけではない機能のカプセル化に役立ちます。つまり、メソッドにクラスインスタンスの値に直接影響しない機能がある場合、そのメソッドは静的です。たとえば、Date クラスには、ストリングを取得して数値に変換する parse() という静的メソッドがあります。このメソッドは、クラスの個々のインスタンスに影響を与えないため静的です。parse() メソッドは、日付値を表すストリングを取得し、そのストリングを解析して、Date オブジェクトの内部表現と互換性がある形式で数値を返します。このメソッドは、Date クラスのインスタンスに適用しても意味がないため、インスタンスメソッドではありません。

静的な parse() メソッドと、getMonth() などの Date クラスのインスタンスメソッドを比較してみます。getMonth() メソッドは、Date インスタンスの特定のコンポーネントである月を取得することで、インスタンスの値に対して直接実行されるため、インスタンスメソッドです。

静的メソッドは個々のインスタンスにバインドされていないので、静的メソッドの本体内で this または super キーワードを使用できません。this 参照および super 参照は、インスタンスメソッドのコンテキスト内でのみ有効です。

他のクラスベースのプログラミング言語とは異なり、ActionScript 3.0 では静的メソッドは継承されません。詳細については、158 ページの「継承されない静的プロパティ」を参照してください。

インスタンスメソッド

インスタンスメソッドは、static キーワードを使用せずに宣言されたメソッドです。インスタンスメソッドは、クラス全体ではなくクラスのインスタンスに関連付けられ、クラスの個々のインスタンスに影響を与える機能の実装に役立ちます。たとえば、Array クラスには、Array インスタンスに直接実行される sort() という名前のインスタンスメソッドが含まれています。

インスタンスメソッドの本体内では、静的変数およびインスタンス変数はスコープ内にあります。つまり、同じクラス内で定義された変数は、単純な識別子を使用して参照できます。たとえば、次の CustomArray クラスは Array クラスを拡張します。CustomArray クラスは、クラスインスタンスの総数を追跡する静的変数 arrayCountTotal、インスタンスが作成された順序を追跡するインスタンス変数 arrayNumber、およびこれらの変数の値を返すインスタンスメソッド getPosition() を定義します。

```
public class CustomArray extends Array
{
  public static var arrayCountTotal:int = 0;
  public var arrayNumber:int;

  public function CustomArray()
  {
    arrayNumber = ++arrayCountTotal;
  }

  public function getArrayPosition():String
  {
     return ("Array " + arrayNumber + " of " + arrayCountTotal);
  }
}
```

クラス外部にあるコードは、CustomArray.arrayCountTotal を使用して、クラスオブジェクトから arrayCountTotal 静的変数を参照する必要がありますが、getPosition() メソッドの本体内にあるコードは、arrayCountTotal 静的変数を直接参照できます。これは、スーパークラスの静的変数でも同じです。ActionScript 3.0 では静的プロパティは継承されませんが、スーパークラスの静的プロパティはスコープ内です。たとえば、Array クラスにはいくつかの静的変数があり、そのうちの 1 つは DESCENDING という定数です。Array サブクラス内にあるコードは、単純な識別子を使用して静的定数 DESCENDING を参照できます。

```
public class CustomArray extends Array
{
  public function testStatic():void
```

```
  {
    trace(DESCENDING); // 出力 : 2
  }
}
```

インスタンスメソッドの本体内の this 参照の値は、メソッドが関連付けられているインスタンスへの参照です。次のコードは、this 参照がメソッドを含むインスタンスを参照していることを示します。

```
class ThisTest
{
  function thisValue():ThisTest
  {
    return this;
  }
}

var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // 出力 : true
```

インスタンスメソッドの継承は、override と final のキーワードを使用して制御できます。override 属性を使用して継承されたメソッドを定義できます。また、final 属性を使用すると、サブクラスによってメソッドがオーバーライドされないようにできます。詳細については、156 ページの「メソッドのオーバーライド」を参照してください。

get および set アクセサメソッド

getter および setter とも呼ばれる get および set アクセサ関数を使用すると、作成したクラスの使いやすいプログラミングインターフェイスを提供すると同時に、情報を非表示およびカプセル化するというプログラミング原則に従うことができます。get および set 関数を使用して、クラスプロパティをクラスに対してプライベートに保持できますが、クラスのユーザーは、クラスメソッドを呼び出すのではなくクラス変数にアクセスしている場合と同じように、これらのプロパティにアクセスできます。

この方法の利点は、getPropertyName()、setPropertyName() など、従来の冗長な名前のアクセサ関数を使用する必要がないことです。また、getter および setter には、読み取りおよび書き込みアクセスを可能にする各プロパティに対して 2 つの公開関数を持つ必要がないという利点もあります。

次の例の GetSet クラスには、privateProperty という名前のプライベート変数へのアクセスを提供する publicAccess() という名前の get および set アクセサ関数が含まれています。

```
class GetSet
{
  private var privateProperty:String;

  public function get publicAccess():String
  {
    return privateProperty;
  }

  public function set publicAccess(setValue:String):void
```

```
    {
        privateProperty = setValue;
    }
}
```

privateProperty プロパティに直接アクセスしようとすると、エラーが発生します。

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); // エラーが発生する
```

GetSet クラスを使用している場合は、publicAccess という名前のプロパティのように見える、privateProperty という名前のプライベートプロパティに対して動作する get と set のアクセサ関数の組み合わせを使用します。次の例では、GetSet クラスをインスタンス化し、publicAccess という名前のパブリックアクセサを使用して privateProperty の値を設定します。

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // 出力 : null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // 出力 : hello
```

getter および setter 関数でも、スーパークラスから継承されるプロパティをオーバーライドできますが、通常のクラスメンバー変数を使用したときにはオーバーライドできません。var キーワードで宣言されたクラスメンバー変数は、サブクラスではオーバーライドできません。一方、getter および setter 関数を使用して作成されたプロパティにはこの制限はありません。スーパークラスから継承した getter 関数と setter 関数に override 属性を使用できます。

バインドメソッド

バインドメソッドは、メソッドクロージャとも呼ばれ、単にインスタンスから抽出されるメソッドです。バインドメソッドの例には、関数にパラメータとして渡され、関数から値として返されるメソッドがあります。ActionScript 3.0 で新しく導入されたバインドメソッドは、インスタンスから抽出されたときでもレキシカル環境が保持されるメソッドクロージャに似ています。バインドメソッドとメソッドクロージャの主な違いは、バインドメソッドの this 参照は、バインドメソッドを実装するインスタンスにリンクされたまま、つまりバインドされたままであるという点です。これは、バインドメソッドの this 参照が、常にこのメソッドを実装する元のオブジェクトを指していることを意味します。関数クロージャの場合、this 参照は汎用です。すなわち、この関数が呼び出された時点で関連付けられているオブジェクトが何であっても、そのオブジェクトを参照します。

this キーワードを使用する場合は、バインドメソッドを理解していることが重要です。this キーワードを使用すれば、メソッドの親オブジェクトを参照できます。ほとんどの ActionScript プログラマは、this キーワードが常にメソッドの定義を含むオブジェクトまたはクラスを参照すると考えますが、メソッドのバインディングがないと必ずしもそうなりません。旧バージョンの ActionScript では、たとえば、this 参照はメソッドを実装するインスタンスを常に参照するわけではありませんでした。ActionScript 2.0 でインスタンスからメソッドを抽出すると、this 参照が元のインスタンスにバインドされないだけでなく、インスタンスのクラスのメンバー変数およびメソッドも使用できません。ActionScript 3.0 では、メソッドをパラメータとして渡すとバインドメソッドが自動的に作成されるため、これは問題にはなりません。バインドメソッドにより、this キーワードは常に、メソッドが定義されたオブジェクトまたはクラスを参照します。

次のコードは、ThisTest というクラスを定義します。このクラスには、バインドメソッドを定義するメソッド foo()、およびバインドメソッドを返すメソッド bar() が含まれます。クラス外部にあるコードは、ThisTest クラスのインスタンスを作成し、bar() メソッドを呼び出して、myFunc という変数に戻り値を格納します。

```
class ThisTest
{
  private var num:Number = 3;
  function foo():void // 定義されたバインドメソッド
  {
    trace("foo's this: " + this);
    trace("num: " + num);
  }
  function bar():Function
  {
    return foo; // 返されたバインドメソッド
  }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // 出力 : [object global]
myFunc();
/* 出力 :
foo の this: [object ThisTest]
出力 : num: 3 */
```

コードの最後の 2 行は、バインドメソッド foo() の this 参照が、その直前の行の this 参照がグローバルオブジェクトを指していても、依然として ThisTest クラスのインスタンスを指すことを示しています。さらに、myFunc 変数に格納されているバインドメソッドは、引き続き ThisTest クラスのメンバー変数にアクセスすることができます。この同じコードを ActionScript 2.0 で実行すると、this 参照が一致し、num 変数は undefined になります。

addEventListener() メソッドでは関数またはメソッドをパラメータとして渡す必要があるため、バインドメソッドの追加が最もわかりやすいのはイベントハンドラを使用する場合です。詳細については、314 ページの「クラスメソッドとして定義されたリスナー関数」を参照してください。

クラスによる列挙

列挙は、値の小さなセットをカプセル化するために作成するカスタムデータ型です。ActionScript 3.0 は、C++ の enum キーワードや Java の Enumeration インターフェイスとは異なり、特定の列挙機能をサポートしません。ただし、クラスと静的定数を使用して列挙を作成できます。たとえば、次のコードに示すように、Flash Player API の PrintJob クラスは、PrintJobOrientation という列挙を使用して "landscape" および "portrait" で構成される値のセットを格納します。

```
public final class PrintJobOrientation
{
  public static const LANDSCAPE:String = "landscape";
  public static const PORTRAIT:String = "portrait";
}
```

慣例では、列挙クラスはクラスを拡張する必要がないので、final 属性で宣言されます。このクラスは静的メンバーのみで構成されます。つまり、クラスのインスタンスを作成しません。代わりに、次の抜粋されたコードに示すように、クラスオブジェクトを通じて直接列挙値にアクセスします。

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
  if (pj.orientation == PrintJobOrientation.PORTRAIT)
  {
    ...
  }
  ...
}
```

Flash Player API 内のすべての列挙クラスには、String、int、または uint のいずれかの型の変数のみが含まれます。リテラルのストリング値または数値ではなく列挙を使用する利点は、表記ミスを見つけやすいことです。列挙の名前を間違って入力した場合、ActionScript コンパイラはエラーを生成します。リテラル値を使用した場合、単語を間違って入力するか数値を間違って使用してもコンパイラに認識されません。前の例では、次のコードの抜粋に示すように、列挙定数の名前が間違っている場合にコンパイラはエラーを生成します。

```
  if (pj.orientation == PrintJobOrientation.PORTRAI) // コンパイルエラー
```

しかし、次のようにリテラルストリング値にスペルミスがある場合、コンパイラはエラーを生成しません。

```
  if (pj.orientation == "portrai") // コンパイルエラーなし
```

列挙を作成するもう1つの方法でも、列挙の静的プロパティで別のクラスを作成することが必要になります。ただし、この方法は、静的プロパティにストリング値または整数値ではなくクラスのインスタンスが含まれる点で異なります。たとえば、次のコードは、曜日に対する列挙クラスを作成します。

```
public final class Day
{
  public static const MONDAY:Day = new Day();
  public static const TUESDAY:Day = new Day();
  public static const WEDNESDAY:Day = new Day();
  public static const THURSDAY:Day = new Day();
  public static const FRIDAY:Day = new Day();
  public static const SATURDAY:Day = new Day();
  public static const SUNDAY:Day = new Day();
}
```

Flash Player API ではこの方法は使用されていませんが、この方法により型チェックの向上を希望する多くの開発者によって採用されています。たとえば、列挙値を返すメソッドは、その戻り値を列挙データ型に制限することができます。次のコードは、曜日を返す関数だけでなく、列挙型を型注釈として使用する関数呼び出しも示します。

```
function getDay():Day
{
  var date:Date = new Date();
  var retDay:Day;
  switch (date.day)
  {
    case 0:
      retDay = Day.MONDAY;
      break;
    case 1:
      retDay = Day.TUESDAY;
      break;
    case 2:
      retDay = Day.WEDNESDAY;
      break;
    case 3:
      retDay = Day.THURSDAY;
      break;
    case 4:
      retDay = Day.FRIDAY;
      break;
    case 5:
      retDay = Day.SATURDAY;
      break;
    case 6:
      retDay = Day.SUNDAY;
      break;
  }
  return retDay;
}
```

```
var dayOfWeek:Day = getDay();
```

Day クラスを拡張することにより、整数を各曜日に関連付け、曜日のストリング表現を返す toString() メソッドを提供することもできます。必要に応じて、この方法で Day クラスの機能拡張を実習として試してみることができます。

埋め込みアセットクラス

ActionScript 3.0 では、埋め込みアセットを表現するために、埋め込みアセットクラスと呼ばれる特別なクラスを使用します。" 埋め込みアセット " は、コンパイル時に SWF ファイルに含まれるサウンド、イメージ、フォントのようなアセットです。アセットを動的にロードするのではなく埋め込むと、実行時に利用可能になることが保証されますが、SWF ファイルのサイズは大きくなってしまいます。

Flash での埋め込みアセットクラスの使用

アセットを埋め込むには、まずアセットを FLA ファイルのライブラリに保存します。次に、アセットの linkage プロパティを使用して、アセットの埋め込みアセットクラスに名前を付けます。その名前のクラスがクラスパスに見つからない場合、クラスは自動的に生成されます。これにより、埋め込みアセットクラスのインスタンスを作成し、そのクラスによって定義または継承された任意のプロパティとメソッドを使用できるようになります。たとえば、次のコードを使用して、PianoMusic という名前の埋め込みアセットクラスにリンクされた埋め込みサウンドを再生できます。

```
var piano:PianoMusic = new PianoMusic();
var sndChannel:SoundChannel = piano.play();
```

インターフェイス

インターフェイスは、関連しないオブジェクトの相互の通信を可能にするメソッド宣言の集合です。たとえば、Flash Player API では、クラスでイベントオブジェクトを処理するために使用できるメソッド宣言を含む IEventDispatcher インターフェイスを定義します。IEventDispatcher インターフェイスは、オブジェクトが相互にイベントオブジェクトを渡し合うための標準的な手段を確立します。次のコードは、IEventDispatcher インターフェイスの定義を示します。

```
public interface IEventDispatcher
{
  function addEventListener(type:String, listener:Function,
       useCapture:Boolean=false, priority:int=0,
       useWeakReference:Boolean = false):void;
  function removeEventListener(type:String, listener:Function,
       useCapture:Boolean=false):void;
  function dispatchEvent(event:Event):Boolean;
  function hasEventListener(type:String):Boolean;
  function willTrigger(type:String):Boolean;
}
```

インターフェイスは、メソッドのインターフェイスとその実装との違いに基づいています。メソッドのインターフェイスには、メソッドの名前、すべてのパラメータ、および戻り値の型など、そのメソッドを呼び出すために必要なすべての情報が含まれます。メソッドの実装には、インターフェイス情報だけではなく、メソッドのビヘイビアを実行する実行可能ステートメントも含まれます。インターフェイス定義には、メソッドインターフェイスのみが含まれ、インターフェイスを実装するクラスはメソッドの実装を定義します。

Flash Player API では、EventDispatcher クラスは、IEventDispatcher インターフェイスのメソッドすべてを定義し、各メソッドにメソッドの本体を追加して、IEventDispatcher インターフェイスを実装します。次のコードは、EventDispatcher クラスの定義からの抜粋です。

```
public class EventDispatcher implements IEventDispatcher
{
   function dispatchEvent(event:Event):Boolean
   {
      /* 実装ステートメント */
   }

   ...
}
```

IEventDispatcher インターフェイスは、プロトコルとして機能します。EventDispatcher インスタンスは、このプロトコルを使用してイベントオブジェクトを処理し、IEventDispatcher インターフェイスを実装する他のオブジェクトに渡します。

インターフェイスは、クラスのようにデータ型を定義すると説明することもできます。したがって、インターフェイスはクラスのように型注釈として使用できます。インターフェイスは、データ型として is および as 演算子などのデータ型を必要とする演算子と使用することもできます。しかし、クラスとは異なり、インターフェイスをインスタンス化することはできません。この違いから、多くのプログラマは、インターフェイスを抽象データ型、クラスを具象データ型と見なしています。

インターフェイスの定義

インターフェイス定義の構造は、クラス定義の構造に似ていますが、インターフェイスにはメソッド本体のないメソッドしか含めることができません。インターフェイスに変数や定数を含めることはできませんが、getter および setter は含めることができます。インターフェイスを定義するには、`interface` キーワードを使用します。たとえば、次の IExternalizable インターフェイスは、Flash Player API の flash.utils パッケージに含まれています。IExternalizable インターフェイスは、オブジェクトを直列化する、つまりオブジェクトをデバイスでの保存用またはネットワーク上の伝達用に適した形式に変換するためのプロトコルを定義します。

```
public interface IExternalizable
{
   function writeExternal(output:IDataOutput):void;
   function readExternal(input:IDataInput):void;
}
```

IExternalizable インターフェイスは、public アクセス制御修飾子で宣言されます。インターフェイス定義は、public および internal アクセス制御指定子でのみ変更できます。インターフェイス定義内のメソッド宣言には、アクセス制御指定子を使用できません。

Flash Player API では、インターフェイス名は大文字の I で始まるという表記規則に従いますが、インターフェイス名には有効な任意の識別子を使用できます。インターフェイス定義は、通常パッケージの最上位に配置されます。クラス定義内または別のインターフェイス定義内に、インターフェイス定義を配置することはできません。

インターフェイスは、他のインターフェイスを拡張できます。たとえば、次のインターフェイス、IExample は、IExternalizable インターフェイスを拡張します。

```
public interface IExample extends IExternalizable
{
  function extra():void;
}
```

IExample インターフェイスを実装するクラスはいずれも、extra() メソッドの実装だけでなく、IExternalizable インターフェイスから継承される writeExternal() および readExternal() メソッドの実装も含む必要があります。

クラス内でのインターフェイスの実装

クラスは、インターフェイスを実装できる唯一の ActionScript 3.0 言語エレメントです。クラス宣言内で implements キーワードを使用して、1 つまたは複数のインターフェイスを実装します。次の例では、IAlpha および IBeta の 2 つのインターフェイスと、これらの両方を実装する Alpha クラスを定義します。

```
interface IAlpha
{
  function foo(str:String):String;
}

interface IBeta
{
  function bar():void;
}

class Alpha implements IAlpha, IBeta
{
  public function foo(param:String):String {}
  public function bar():void {}
}
```

インターフェイスを実装するクラスでは、実装されたメソッドは以下を行う必要があります。

- public アクセス制御識別子を使用する。
- インターフェイスメソッドと同じ名前を使用する。
- 同じ数のパラメータを含む。各パラメータのデータ型は、インターフェイスメソッドパラメータのデータ型と一致する必要があります。
- 同じ戻り値の型を使用する。

実装するメソッドのパラメータには、ある程度自由に名前を付けることができます。実装されるメソッドとインターフェイスメソッドのパラメータ数および各パラメータのデータ型は一致する必要がありますが、パラメータ名を一致させる必要はありません。たとえば、前の例では Alpha.foo() メソッドのパラメータの名前は param ですが、

```
public function foo(param:String):String {}
```

IAlpha.foo() インターフェイスメソッドでのパラメータの名前は str です。

```
function foo(str:String):String;
```

デフォルトのパラメータ値には若干の柔軟性があります。インターフェイス定義は、デフォルトのパラメータ値を備えた関数宣言を含むことができます。このような関数宣言を実装するメソッドは、インターフェイス定義に指定されている値と同じデータ型のメンバーであるデフォルトのパラメータ値を持つ必要がありますが、実際の値が一致する必要はありません。たとえば、次のコードは、デフォルトのパラメータ値 3 を持つメソッドを含むインターフェイスを定義します。

```
interface IGamma
{
  function doSomething(param:int = 3):void;
}
```

次のクラス定義は、Igamma インターフェイスを実装しますが、異なるデフォルトパラメータ値を使用します。

```
class Gamma implements IGamma
{
  public function doSomething(param:int = 4):void {}
}
```

この柔軟性は、インターフェイスを実装する規則がデータ型の互換性を確保するように特別に設計されているからです。このために、同じパラメータ名およびデフォルトのパラメータ値を要求する必要はありません。

継承

継承は、コード再利用の一形式です。プログラマは継承を使用して、既存のクラスに基づく新しいクラスを開発することができます。既存のクラスは " 基本クラス " または " スーパークラス " と呼ばれますが、新しいクラスは、通常は " サブクラス " と呼ばれます。継承の主な利点は、既存のコードをそのまま維持しながら、基本クラスのコードを再利用できることです。さらに、継承では他のクラスが基本クラスとやり取りする方法を変更する必要がありません。入念にテストされたか、既に使用されている既存クラスを変更するのではなく、継承を使用することでそのクラスを追加プロパティまたはメソッドで拡張できる統合モジュールとして使用できます。したがって、extends キーワードを使用して、クラスが別のクラスを継承することを示すことができます。

継承を使用すると、コード内で " ポリモーフィズム " を利用することもできます。ポリモーフィズムを使用すると、異なるデータ型に適用された場合に異なる動作を実行するメソッドに単一のメソッド名を使用できます。単純な例として、2 つのサブクラス Circle と Square を持つ Shape という名前の基本クラスがあります。Shape クラスは、形状の面積を返す area() というメソッドを定義します。ポリモーフィズムが実装されている場合、Circle 型および Square 型のオブジェクトで area() メソッドを呼び出して、正しい計算を行うことができます。継承では、サブクラスが基本クラスからメソッドを継承し、再定義できるようにする、つまりオーバーライドを許可することで、ポリモーフィズムが有効になります。次の例では、area() メソッドを、Circle クラスと Square クラスによって再定義します。

```
class Shape
{
  public function area():Number
  {
    return NaN;
  }
}

class Circle extends Shape
{
  private var radius:Number = 1;
  override public function area():Number
  {
    return (Math.PI * (radius * radius));
  }
}

class Square extends Shape
{
  private var side:Number = 1;
  override public function area():Number
  {
    return (side * side);
  }
}
```

```
var cir:Circle = new Circle();
trace(cir.area()); // 出力 : 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // 出力 : 1
```

各クラスはデータ型を定義するため、継承を使用すると、基本クラスと基本クラスを拡張するクラスの特別な関係が作成されます。サブクラスは、基本クラスのすべてのプロパティを保有します。これは、サブクラスのインスタンスは常に基本クラスのインスタンスの代わりに使用できることを意味します。たとえば、メソッドが Shape 型のパラメータを定義する場合、Circle 型は Shape 型を拡張するので Circle 型のパラメータを渡すことができます。次に例を示します。

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

インスタンスプロパティと継承

インスタンスプロパティは、function、var、または const キーワードのいずれで定義されているかにかかわらず、プロパティが基本クラスの private 属性で宣言されていない限り、すべてのサブクラスに継承されます。たとえば、Flash Player API の Event クラスには、すべてのイベントオブジェクトに共通するプロパティを継承する多数のサブクラスがあります。

一部の型のイベントでは、Event クラスにイベントを定義するために必要なすべてのプロパティが含まれます。これらの型のイベントでは、Event クラスで定義されたもの以外のインスタンスプロパティは必要ありません。このようなイベントの例として、データが正常にロードされたときに発生する complete イベント、ネットワーク接続が確立されたときに発生する connect イベントなどがあります。

次の例は、サブクラスに継承されるプロパティおよびメソッドの一部を示す Event クラスからの抜粋です。プロパティが継承されるので、あらゆるサブクラスのインスタンスは、これらのプロパティにアクセスできます。

```
public class Event
{
  public function get type():String;
  public function get bubbles():Boolean;
  ...

  public function stopPropagation():void {}
  public function stopImmediatePropagation():void {}
  public function preventDefault():void {}
  public function isDefaultPrevented():Boolean {}
  ...
}
```

その他の型のイベントは、Event クラスで使用できない固有のプロパティを必要とします。これらのイベントは、Event クラスで定義されたプロパティに新しいプロパティを追加できるように、Event クラスのサブクラスを使用して定義されます。このようなサブクラスには、mouseMove イベント、click イベントなど、マウス操作またはマウスクリックに関連付けられているイベントに固有のプロパティを追加する MouseEvent クラスなどがあります。次の例は、MouseEvent クラスからの抜粋であり、サブクラスには存在するが基本クラスには存在しないプロパティの定義を示しています。

```
public class MouseEvent extends Event
{
  public static const CLICK:String      = "click";
  public static const MOUSE_MOVE:String = "mouseMove";
  ...

  public function get stageX():Number {}
  public function get stageY():Number {}
  ...
}
```

アクセス制御指定子と継承

プロパティが public キーワードで宣言された場合、このプロパティは任意の場所のコードから参照できます。つまり、public キーワードは、private、protected、および internal キーワードとは異なり、プロパティの継承に制限を加えません。

プロパティが private キーワードで宣言された場合、このプロパティは定義されたクラス内でのみ参照でき、サブクラスに継承されません。この動作は、旧バージョンの ActionScript とは異なります。旧バージョンでは、private キーワードは ActionScript 3.0 の protected キーワードと同じように動作しました。

protected キーワードは、定義されたクラス内だけでなくすべてのサブクラスからもプロパティを参照できることを示します。Java の protected キーワードとは異なり、ActionScript 3.0 の protected キーワードでは同じパッケージ内の他のすべてのクラスからプロパティを参照できるわけではありません。ActionScript 3.0 では、サブクラスのみが protected キーワードで宣言されたプロパティにアクセスできます。また、protected 指定されたプロパティは、サブクラスが基本クラスと同じパッケージ内にあるか、または異なるパッケージ内にあるかにかかわらず、サブクラスから参照できます。

プロパティが定義されているパッケージからのプロパティへの参照を制限するには、internal キーワードを使用するか、またはアクセス制御指定子を一切使用しないようにします。internal アクセス制御指定子は、アクセス制御指定子が指定されていないときに適用されるデフォルトのアクセス制御指定子です。internal とマークされたプロパティは、同じパッケージ内に存在するサブクラスによってのみ継承されます。

次の例を使用して、各アクセス制御指定子がパッケージ境界を越えてどのように継承に影響するかを確認できます。次のコードは、AccessControl というメインアプリケーションクラスと Base および Extender というその他のクラス 2 つを定義します。Base クラスは foo パッケージ内にあり、Base クラスのサブクラスである Extender クラスは bar パッケージ内にあります。AccessControl クラスは Extender クラスのみを読み込み、Base クラスで定義されている str という変数にアクセスしようとする Extender のインスタンスを作成します。str 変数は public として宣言され、この結果、次の抜粋に示すようにコードがコンパイルされて実行されます。

```
// "foo" フォルダ内の "Base.as"
package foo
{
  public class Base
  {
    public var str:String = "hello"; // この行の public を変更
  }
}

// "bar" というフォルダ内の Extender.as
package bar
{
  import foo.Base;
  public class Extender extends Base
  {
    public function getString():String {
      return str;
    }
  }
}

// "ProtectedExample.as" というファイル内のメインアプリケーションクラス
  import flash.display.MovieClip;
  import bar.Extender;
  public class AccessControl extends MovieClip
  {
    public function AccessControl()
    {
      var myExt:Extender = new Extender();
      trace(myExt.testString);  // str が public でない場合はエラー
      trace(myExt.getString()); // str が private または internal の場合はエラー
    }
  }
}
```

その他のアクセス制御指定子が前の例のコンパイルおよび実行に与える影響を確認するには、AccessControl クラスから次の行を削除またはコメントアウトした後、str 変数のアクセス制御指定子を private、protected、または internal に変更します。

```
trace(myExt.testString);  // str が public でない場合はエラー
```

変数のオーバーライド禁止

var または const キーワードで宣言されたプロパティは継承されますが、オーバーライドすることはできません。プロパティをオーバーライドすると、サブクラスでプロパティが再定義されます。オーバーライドできるプロパティの型はメソッド、すなわち function キーワードで宣言されるプロパティのみです。インスタンス変数をオーバーライドすることはできませんが、インスタンス変数の getter および setter メソッドを作成し、これらのメソッドをオーバーライドすることで、同じ機能を実現できます。詳細については、158 ページの「getter および setter のオーバーライド」を参照してください。

メソッドのオーバーライド

メソッドをオーバーライドすると、継承されたメソッドの動作が再定義されます。静的メソッドは継承されず、オーバーライドすることはできません。ただし、次の 2 つの条件が満たされていれば、インスタンスメソッドはサブクラスに継承され、オーバーライドできます。

- インスタンスメソッドが基本クラス内で final キーワードで宣言されていない。インスタンスメソッドで使用された場合、final キーワードは、サブクラスによってメソッドがオーバーライドされないように、プログラマが意図的に指定したことを示します。
- インスタンスメソッドが基本クラス内で private アクセス制御指定子で宣言されていない。基本クラス内でメソッドが private としてマークされている場合、基本クラスのメソッドはサブクラスから参照できないので、サブクラス内で同じ名前のメソッドを定義するときに override キーワードを使用する必要はありません。

上記の条件を満たすインスタンスメソッドをオーバーライドするには、サブクラス内のメソッドの定義は override キーワードを使用し、次の方法でメソッドのスーパークラスバージョンと一致する必要があります。

- オーバーライドメソッドには、基本クラスのメソッドと同じレベルのアクセス制御が必要です。internal とマークされたメソッドには、アクセス制御指定子を持たないメソッドと同じレベルのアクセス制御が必要です。
- オーバーライドメソッドには、基本クラスのメソッドと同じ数のパラメータが必要です。
- オーバーライドメソッドのパラメータには、基本クラスのメソッドのパラメータと同じデータ型注釈が必要です。
- オーバーライドメソッドには、基本クラスのメソッドと同じ戻り値の型が必要です。

ただし、オーバーライドメソッドのパラメータの名前は、パラメータの数および各パラメータのデータ型が一致していれば、基本クラスのパラメータの名前と一致する必要はありません。

super ステートメント

メソッドをオーバーライドするとき、多くのプログラマが、動作を完全に置き換えるのではなく、オーバーライドするスーパークラスメソッドの動作に追加したいと考えるでしょう。これには、サブクラス内のメソッドがそれ自体のスーパークラスバージョンを呼び出すメカニズムが必要です。super ステートメントは、直接のスーパークラスへの参照が含まれるそのようなメカニズムを提供します。次の例では、thanks() というメソッドを含む Base クラスと、thanks() メソッドをオーバーライドする Extender という Base クラスのサブクラスを定義します。Extender.thanks() メソッドは、super ステートメントを使用して Base.thanks() を呼び出します。

```
package {
  import flash.display.MovieClip;
  public class SuperExample extends MovieClip
  {
    public function SuperExample()
    {
      var myExt:Extender = new Extender()
      trace(myExt.thanks()); // 出力 : Mahalo nui loa
    }
  }
}

class Base {
  public function thanks():String
  {
    return "Mahalo";
  }
}

class Extender extends Base
{
  override public function thanks():String
  {
    return super.thanks() + " nui loa";
  }
}
```

getter および setter のオーバーライド

スーパークラスで定義された変数をオーバーライドすることはできませんが、getter および setter をオーバーライドすることができます。たとえば、次のコードでは、Flash Player API の MovieClip クラスで定義された currentLabel という名前の getter をオーバーライドします。

```
package
{
  import flash.display.MovieClip;
  public class OverrideExample extends MovieClip
  {
    public function OverrideExample()
    {
      trace(currentLabel)
    }
    override public function get currentLabel():String
    {
      var str:String = "Override: ";
      str += super.currentLabel;
      return str;
    }
  }
}
```

OverrideExample クラスコンストラクタの trace() ステートメントの出力は、Override: null です。これは、継承された currentLabel プロパティをオーバーライドできたことを示します。

継承されない静的プロパティ

静的プロパティはサブクラスに継承されません。つまり、サブクラスのインスタンスから静的プロパティにアクセスすることはできません。静的プロパティは、そのプロパティが定義されたクラスオブジェクトからのみアクセス可能です。たとえば、次のコードは、Base という基本クラスと、Extender という Base クラスを拡張するサブクラスを定義します。静的変数 test は Base クラスで定義されています。次の抜粋で記述されているコードは、strict モードではコンパイルされず、standard モードではランタイムエラーが生成されます。

```
package {
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
      trace(myExt.test);  // エラー
    }
  }
}
```

```
class Base {
  public static var test:String = "static";
}

class Extender extends Base { }
```

静的変数 test には、次のコードに示すように、クラスオブジェクトからアクセスする必要があります。

```
Base.test;
```

ただし、静的プロパティと同じ名前を使用してインスタンスプロパティを定義することができます。このインスタンスプロパティは、静的プロパティと同じクラス内またはサブクラス内で定義できます。たとえば、前述の例の Base クラスでは、test というインスタンスプロパティを定義できます。次のコードはコンパイルおよび実行されます。これは、インスタンスプロパティが Extender クラスに継承されるからです。test インスタンス変数の定義が Extender クラスにコピーされるのではなく、移動された場合も、コードはコンパイルされ、実行されます。

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
      trace(myExt.test);  // 出力 : instance
    }
  }
}

class Base
{
  public static var test:String = "static";
  public var test:String = "instance";
}

class Extender extends Base {}
```

静的プロパティとスコープチェーン

静的プロパティは継承されませんが、そのプロパティが定義されたクラスおよびそのクラスのサブクラスのスコープチェーン内にあります。このため、静的プロパティは、定義されたクラスおよびサブクラスのスコープ内にあると言います。つまり、静的プロパティが定義されたクラスおよびそのサブクラスの本体内から直接静的プロパティにアクセスできます。

次の例では、前述の例で定義したクラスを変更して、Base クラスで定義された静的変数 test が Extender クラスのスコープ内にあることを示します。つまり、Extender クラスは、test を定義したクラスの名前を接頭辞として変数に付けなくても静的変数 test にアクセスできます。

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
    }
  }
}

class Base {
  public static var test:String = "static";
}

class Extender extends Base
{
  public function Extender()
  {
    trace(test); // 出力 : static
  }

}
```

インスタンスプロパティが、同じクラスまたはスーパークラス内で静的プロパティと同じ名前を使用するように定義されている場合は、インスタンスプロパティはスコープチェーン内で優先順位が高くなります。インスタンスプロパティは、静的プロパティを " シャドウする " と言います。これは、静的プロパティの値ではなくインスタンスプロパティの値が使用されることを意味します。たとえば、次のコードは、Extender クラスが test という名前のインスタンス変数を定義するときに、trace() ステートメントが、静的変数の値ではなくインスタンス変数の値を使用することを示しています。

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
    }
  }
}

class Base
```

```
{
  public static var test:String = "static";
}

class Extender extends Base
{
  public var test:String = "instance";
  public function Extender()
  {
    trace(test); // 出力 : instance
  }

}
```

高度なトピック

このセクションでは、最初に ActionScript と OOP の歴史について簡潔に説明します。次に、ActionScript 3.0 オブジェクトモデル、およびこれを使って新しい ActionScript 仮想マシン (AVM2) を、古い ActionScript 仮想マシン (AVM1) を使用する旧バージョンの Flash Player よりもはるかに高速に実行可能にする方法について解説します。

ActionScript OOP サポートの歴史

ActionScript 3.0 は旧バージョンの ActionScript をベースに構築されているため、ActionScript オブジェクトモデルがどのように進化してきたかを理解しておくと役立ちます。ActionScript は、Flash オーサリングツールの初期バージョン用の単純なスクリプトメカニズムとして生まれました。その後、プログラマは ActionScript を使用してより複雑なアプリケーションを作成し始めました。プログラマのニーズに応えるために、その後の各リリースには複雑なアプリケーションの作成を容易にする言語機能が追加されてきました。

ActionScript 1.0

ActionScript 1.0 は、Flash Player 6 以前で使用されていた言語のバージョンです。この開発初期段階でも、ActionScript オブジェクトモデルは基本的なデータ型としてのオブジェクトの概念をベースにしていました。ActionScript オブジェクトは、プロパティのグループを持つ複合データ型です。オブジェクトモデルについて説明するとき、" プロパティ " という用語には、変数、関数、メソッドなど、オブジェクトに関連付けられるあらゆるものが含まれます。

第 1 世代の ActionScript では、class キーワードによるクラスの定義はサポートされませんが、プロトタイプオブジェクトと呼ばれる特殊な種類のオブジェクトを使用してクラスを定義することができます。class キーワードを使用して具象オブジェクトにインスタンス化する抽象クラスの定義を作成する、Java や C++ などのクラスベース言語の場合と異なり、ActionScript 1.0 のようなプロトタイプベース言語では、既存オブジェクトを他のオブジェクトのモデル (またはプロトタイプ) として使用します。クラスベース言語のオブジェクトはテンプレートになるクラスを示す場合がありますが、プロトタイプベース言語のオブジェクトはテンプレートになる別のオブジェクト、つまりプロトタイプを示します。

ActionScript 1.0 でクラスを作成するには、クラスのコンストラクタ関数を定義します。ActionScript の関数は、単に抽象的な定義ではなく実際のオブジェクトです。作成したコンストラクタ関数は、そのクラスのインスタンスのプロトタイプ的なオブジェクトになります。次のコードでは、Shape という名前のクラスを作成し、デフォルトで true に設定される visible というプロパティを 1 つ定義します。

```
// 基本クラス
function Shape() {}
// visible というプロパティを作成する。
Shape.prototype.visible = true;
```

このコンストラクタ関数は、new 演算子でインスタンス化できる Shape クラスを次のように定義します。

```
myShape = new Shape();
```

Shape() コンストラクタ関数オブジェクトが Shape クラスのインスタンスのプロトタイプとして機能するのと同様に、これは Shape のサブクラスのプロトタイプ、すなわち Shape クラスを拡張する他のクラスとしても機能します。

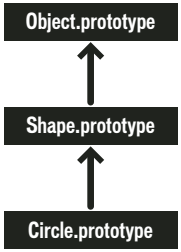
Shape クラスのサブクラスであるクラスを作成するには、次の 2 つの手順を実行します。最初に、次のようにクラスのコンストラクタ関数を定義して、クラスを作成します。

```
// 子クラス
function Circle(id, radius)
{
    this.id = id;
    this.radius = radius;
}
```

次に、new 演算子を使用して、Shape クラスが Circle クラスのプロトタイプであると宣言します。デフォルトでは、作成したクラスはそのプロトタイプとして Object クラスを使用します。つまり、現時点では、Circle.prototype には汎用オブジェクト (Object クラスのインスタンス) が含まれています。Circle のプロトタイプに Object ではなく Shape を指定するには、汎用オブジェクトではなく Shape オブジェクトが含まれるように、次のコードを使用して Circle.prototype の値を変更します。

```
// Circle を Shape のサブクラスにする
Circle.prototype = new Shape();
```

これで、Shape クラスと Circle クラスは、" プロトタイプチェーン " と呼ばれる継承関係内で相互にリンクされました。次の図は、プロトタイプチェーン内の関係を示しています。



各プロトタイプチェーンの最後にある基本クラスは Object クラスです。Object クラスには、ActionScript 1.0 で作成されたすべてのオブジェクトの基本プロトタイプオブジェクトを指す `Object.prototype` という静的プロパティが含まれます。このプロトタイプチェーンの次のオブジェクトは Shape オブジェクトです。これは、`Shape.prototype` プロパティは明示的に設定されていないので、依然として汎用オブジェクト (Object クラスのインスタンス) を保持しているからです。このプロトタイプチェーンの最後のリンクは Circle クラスで、それ自体のプロトタイプである Shape クラスにリンクされています。`Circle.prototype` プロパティは Shape オブジェクトを保持します。

次の例に示すように、Circle クラスのインスタンスを作成すると、インスタンスは Circle クラスのプロトタイプチェーンを継承します。

```
// Circle クラスのインスタンスを作成する
myCircle = new Circle();
```

Shape クラスのメンバーとして、`visible` という名前のプロパティを作成しました。例では、`visible` プロパティは、`myCircle` オブジェクトの一部としてではなく Shape オブジェクトのメンバーとしてのみ存在しますが、次のコード行では `true` が出力されます。

```
trace(myCircle.visible); // 出力 : true
```

Flash Player では、プロトタイプチェーン内を移動することにより、`myCircle` オブジェクトが `visible` プロパティを継承することを確認できます。このコードを実行すると、Flash Player は最初に `myCircle` オブジェクトのプロパティから `visible` というプロパティを検索しますが、このプロパティは見つかりません。Flash Player は次に `Circle.prototype` オブジェクトを検索しますが、`visible` というプロパティは見つかりません。Flash Player はプロトタイプチェーン内を引き続き検索し、最後に `Shape.prototype` オブジェクトで定義された `visible` プロパティを見つけ、そのプロパティの値を出力します。

簡潔にするために、このセクションではプロトタイプチェーンの詳細および複雑さに関する説明の多くを省略し、代わりに ActionScript 3.0 オブジェクトモデルの理解に役立つ情報を提供します。

ActionScript 2.0

ActionScript 2.0 では、Java や C++ などのクラスベース言語の経験がある方には使い慣れた方法でクラスを定義できる class、extends、public、および private などの新しいキーワードが導入されました。ActionScript 1.0 と ActionScript 2.0 では基になる継承メカニズムが変更されていないことを理解しておくことが重要です。ActionScript 2.0 では、クラスを定義するための新しいシンタックスが追加されただけです。プロトタイプチェーンは、両方の言語バージョンで同じように機能します。

次の抜粋に示すように ActionScript 2.0 で導入された新しいシンタックスでは、より直観的な方法でクラスを定義できます。

```
// 基本クラス
class Shape
{
    var visible:Boolean = true;
}
```

ActionScript 2.0 では、コンパイル時の型チェックに使用するための型注釈も導入されました。これにより、前述の例の visible プロパティにはブール値だけが含まれることを宣言できます。また、新しい extends キーワードにより、サブクラスを作成するプロセスが簡略化されます。次の例では、ActionScript 1.0 では 2 つの手順が必要なプロセスが、extends キーワードにより 1 つの手順で実行されています。

```
// 子クラス
class Circle extends Shape
{
  var id:Number;
  var radius:Number;
  function Circle(id, radius)
  {
    this.id = id;
    this.radius = radius;
  }
}
```

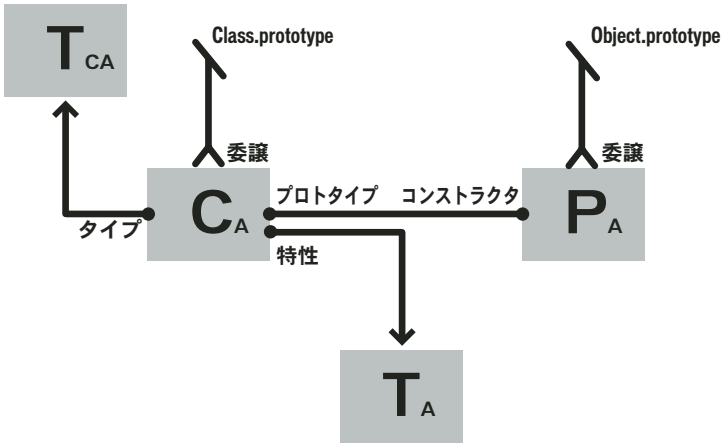
コンストラクタは、クラス定義の一部として宣言されるようになりました。また、クラスプロパティ id および radius は明示的に宣言する必要があります。

ActionScript 2.0 ではインターフェイス定義のサポートも追加され、オブジェクト間の通信用に正式に定義されたプロトコルでオブジェクト指向プログラムをさらに改良できるようになりました。

ActionScript 3.0 のクラスオブジェクト

通常 Java や C++ に関連付けられる一般的なオブジェクト指向プログラミングパラダイムでは、クラスを使用してオブジェクトの型を定義します。このパラダイムを採り入れたプログラミング言語も、クラスにより定義されるデータ型のインスタンスを作成するためにクラスを使用する傾向にあります。ActionScript では、この両方の目的でクラスを使用しますが、プロトタイプベース言語であることから興味深い特徴が付け加えられています。ActionScript ではクラス定義ごとに、動作と状態の両方を共有できる特別なクラスオブジェクトを作成します。とは言え、コーディングする上でこの違いが実質的な影響があると感じる ActionScript プログラマはほとんどいないでしょう。ActionScript 3.0 は、この特別なオブジェクトを使用しなくても、さらには理解していなくても、高度なオブジェクト指向 ActionScript アプリケーションを作成できるように設計されています。このセクションでは、クラスオブジェクトを活用したいと考える上級者のプログラマを対象に、この問題を詳細に説明します。

次の図は、`class A {}` ステートメントで定義された A という名前の単純なクラスを表すクラスオブジェクトの構造を示します。



図中の四角形はオブジェクトを表します。図中の各オブジェクトには、クラス A に属していることを表す添え字 A が付いています。クラスオブジェクト (C_A) には、その他の重要なオブジェクトへのいくつかの参照が含まれます。インスタンス特性オブジェクト (T_A) は、クラス定義内で定義されたインスタンスプロパティを格納します。クラス特性オブジェクト (T_{CA}) はクラスの内部型を表し、そのクラスによって定義された静的プロパティを格納します (添え字 C は " クラス " を意味します)。プロパティオブジェクト (P_A) は、常に `constructor` プロパティを通じて関連付けられた元のクラスオブジェクトを参照します。

特性オブジェクト

ActionScript 3.0 で新しく導入された特性オブジェクトは、パフォーマンスを考慮して実装されました。旧バージョンの ActionScript では、名前のルックアップは、Flash Player がプロトタイプチェーン内を移動するため時間のかかるプロセスでした。ActionScript 3.0 では、継承されたプロパティが、スーパークラスからサブクラスの特性オブジェクトにコピーされます。このため、名前のルックアップが効率的になり、所要時間も短縮されています。

特性オブジェクトはプログラマコードに直接アクセスできませんが、パフォーマンスが向上しメモリ使用量が削減することからオブジェクトの存在がわかります。特性オブジェクトは、AVM2 にクラスのレイアウトと内容に関する詳細情報を提供します。この情報を使って、AVM2 は多くのマシン命令を直接生成することができます。このため、時間のかかる名前のルックアップを行わずにプロパティにアクセスしたり、メソッドを直接呼び出したりすることができ、実行時間を大幅に短縮できます。

特性オブジェクトのおかげで、オブジェクトのメモリフットプリントは、旧バージョンの ActionScript の同様のオブジェクトと比較してかなり小さくすることができます。たとえば、クラスが sealed の場合 (つまり、クラスが dynamic と宣言されていない場合)、クラスのインスタンスは動的に追加するプロパティにハッシュテーブルを必要とせず、このクラスで定義された固定プロパティの特性オブジェクトおよびスロットへのポインタを保持するだけです。その結果、ActionScript 2.0 で 100 バイトのメモリが必要だったオブジェクトが、ActionScript 3.0 では 20 バイトで済みます。

メモ	特性オブジェクトは、内部実装の詳細です。ActionScript の今後のバージョンで変更されない、またはなくならないとは限りません。

プロトタイプオブジェクト

ActionScript のクラスオブジェクトには、クラスのプロトタイプオブジェクトを参照する `prototype` プロパティがあります。プロトタイプオブジェクトは、ActionScript のプロトタイプベース言語としてのルーツのレガシーです。詳細については、161 ページの「ActionScript 1.0」を参照してください。

`prototype` プロパティは読み取り専用で、別のオブジェクトを指すように変更することはできません。これは、旧バージョンの ActionScript のクラスの `prototype` プロパティとは異なります。旧バージョンのプロパティでは、別のクラスを指すようにプロトタイプを再割り当てできました。`prototype` プロパティは読み取り専用ですが、参照されるプロトタイプオブジェクトは読み取り専用ではありません。つまり、プロトタイプオブジェクトに新しいプロパティを追加することができます。プロトタイプオブジェクトに追加されたプロパティは、クラスのすべてのインスタンス間で共有されます。

旧バージョンの ActionScript では唯一の継承メカニズムであったプロトタイプチェーンは、ActionScript 3.0 では二次的な役割のみを果たします。主要継承メカニズムである固定プロパティの継承は、特性オブジェクトによって内部的に処理されます。固定プロパティは、クラス定義の一部として定義される変数またはメソッドです。固定プロパティの継承は、`class`、`extends`、`override` などのキーワードで関連付けられる継承メカニズムなので、クラス継承とも呼ばれます。

プロトタイプチェーンは、固定プロパティの継承より動的な代替継承メカニズムになります。プロパティは、クラス定義の一部としてだけでなく、実行時にクラスオブジェクトの prototype プロパティからもクラスのプロトタイプオブジェクトに追加することができます。ただし、コンパイラを strict モードに設定した場合は、クラスを dynamic キーワードで宣言しない限り、プロトタイプオブジェクトに追加されたプロパティにアクセスできない場合があります。

いくつものプロパティがプロトタイプオブジェクトに関連付けられているクラスの好例として、Object クラスがあります。Object クラスの toString() と valueOf() メソッドは、実際には Object クラスのプロトタイプオブジェクトのプロパティに割り当てられた関数です。次の例は、これらのメソッドの宣言が理論的にどのように見えるかを示します。ただし、実装の詳細により実際の実装はやや異なります。

```
public dynamic class Object
{
  prototype.toString = function()
  {
    // ステートメント
  };
  prototype.valueOf = function()
  {
    // ステートメント
  };
}
```

前述したとおり、プロパティは、クラス定義外部でクラスのプロトタイプオブジェクトに関連付けることができます。たとえば、toString() メソッドは、次のように Object クラスの定義外部で定義することもできます。

```
Object.prototype.toString = function()
{
  // ステートメント
};
```

ただし、固定プロパティの継承とは異なり、プロトタイプの継承では、サブクラスでメソッドを再定義する場合、override キーワードは必要ありません。たとえば、Object クラスのサブクラスで valueOf() メソッドを再定義する場合、次の 3 つのオプションがあります。1 つ目は、クラス定義内部のサブクラスのプロトタイプオブジェクトで、valueOf() メソッドを定義します。次のコードは、Foo という Object のサブクラスを作成し、Foo のプロトタイプオブジェクトで、クラス定義の一部として valueOf() メソッドを再定義します。すべてのクラスは Object から継承されるので、extends キーワードを使用する必要はありません。

```
dynamic class Foo
{
  prototype.valueOf = function()
  {
    return "Instance of Foo";
  };
}
```

2 番目に、クラス定義外部の Foo のプロトタイプオブジェクトで valueOf() メソッドを次のコードのように再定義できます。

```
Foo.prototype.valueOf = function()
{
  return "Instance of Foo";
};
```

3 番目に、Foo クラスの一部として、valueOf() という名前の固定プロパティを定義できます。この方法は、固定プロパティの継承とプロトタイプの継承が混在するという点で他の 2 つとは異なります。valueOf() を再定義する Foo のサブクラスでは、override キーワードを使用する必要があります。次のコードは、Foo で固定プロパティとして定義された valueOf() を示します。

```
class Foo
{
  function valueOf():String
  {
    return "Instance of Foo";
  }
}
```

AS3 名前空間

固定プロパティの継承およびプロトタイプの継承という 2 つの別個の継承メカニズムが混在することで、コアクラスのプロパティおよびメソッドに関して留意すべき互換性の問題が生じます。ECMAScript Edition 4 言語仕様案との互換性からは、プロトタイプの継承を使用する必要があります。つまり、コアクラスのプロパティおよびメソッドは、そのクラスのプロトタイプオブジェクトで定義されます。一方、Flash Player API との互換性では、固定プロパティの継承を使用することが要求されます。つまり、コアクラスのプロパティおよびメソッドは、const、var、および function のキーワードを使用してクラス定義で定義されます。さらに、プロトタイプではなく固定プロパティの継承を使用すると、ランタイムパフォーマンスの大幅な向上につながります。

ActionScript 3.0 では、この問題を解決するために、コアクラスにプロトタイプの継承と固定プロパティの継承の両方を使用しています。各コアクラスに、2 セットのプロパティおよびメソッドが含まれます。1 セットは、ECMAScript 仕様との互換性のためにプロトタイプオブジェクトで定義され、残りの 1 セットは、Flash Player API との互換性のために固定プロパティおよび AS3 名前空間で定義されます。

AS3 名前空間は、この 2 セットのプロパティおよびメソッド間の選択のために便利なメカニズムを提供します。AS3 名前空間を使用しない場合、コアクラスのインスタンスは、コアクラスのプロトタイプオブジェクトに定義されているプロパティおよびメソッドを継承します。AS3 名前空間を使用することに決めた場合は、固定プロパティがプロトタイププロパティよりも常に優先されるため、コアクラスのインスタンスは AS3 バージョンを継承します。つまり、固定プロパティが利用可能な場合には、同じ名前のプロトタイププロパティではなく、その固定プロパティが必ず使用されます。

AS3 名前空間で修飾することによって、AS3 名前空間バージョンのプロパティまたはメソッドを選択的に使用することができます。たとえば、次のコードでは、AS3 バージョンの Array.pop() メソッドを使用しています。

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // 出力 : 1,2
```

代わりに、use namespace ディレクティブを使用して、コードブロック内のすべての定義に対して AS3 名前空間を開くことができます。たとえば、次のコードでは、use namespace ディレクティブを使用して pop() メソッドと push() メソッドの両方に対して AS3 名前空間を開いています。

```
use namespace AS3;

var nums:Array = new Array(1, 2, 3);
nums.pop();
nums.push(5);
trace(nums) // 出力 : 1,2,5
```

また、ActionScript 3.0 では、プロパティセットごとにコンパイラオプションを備えており、AS3 名前空間をプログラム全体に適用できます。-as3 コンパイラオプションは AS3 名前空間を表し、-es コンパイラオプション (es は ECMAScript の略) はプロトタイプ継承オプションを表します。プログラム全体に対して AS3 名前空間を開くには、-as3 コンパイラオプションを true に、-es コンパイラオプションを false に設定します。プロトタイプバージョンを使用するには、両方のコンパイラオプションを反対の値に設定します。Adobe Flex Builder 2 および Adobe Flash CS3 Professional のデフォルトのコンパイラ設定は、-as3 = true および -es = false です。

いずれかのコアクラスを拡張してメソッドのいずれかをオーバーライドする計画がある場合には、オーバーライドメソッドの宣言方法に AS3 名前空間がどのように影響する可能性があるかを理解しておく必要があります。AS3 名前空間を使用する場合は、コアクラスメソッドのいずれのメソッドオーバーライドも、override 属性で AS3 名前空間を使用する必要があります。AS3 名前空間を使用せずにコアクラスメソッドをサブクラスで再定義する場合は、AS3 名前空間も override キーワードも使用しないでください。

例 : GeometricShapes

サンプルアプリケーション GeometricShapes では、ActionScript 3.0 を使用して次のようにオブジェクト志向の概念および機能をいろいろと適用できることがわかります。

- クラスの定義
- クラスの拡張
- ポリモーフィズムと `override` キーワード
- インターフェイスの定義、拡張、実装

さらに、クラスインスタンスを作成する " ファクトリメソッド " も含まれ、インターフェイスのインスタンスとして戻り値を宣言し、戻されたオブジェクトを汎用的な方法で使用する方法が示されます。

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。GeometricShapes アプリケーションのファイルは、"Samples/GeometricShapes" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
GeometricShapes.mxml または GeometricShapes.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/geometricshapes/IGeometricShape.as	すべての GeometricShapes アプリケーションクラスによって実装されるメソッドを定義する基本インターフェイス。
com/example/programmingas3/geometricshapes/IPolygon.as	複数の辺を持つ GeometricShapes アプリケーションクラスによって実装されるメソッドを定義するインターフェイス。
com/example/programmingas3/geometricshapes/RegularPolygon.as	シェイプの中心から対称の位置に等しい長さの辺を有する一種の幾何学図形。
com/example/programmingas3/geometricshapes/Circle.as	円を定義する一種の幾何学図形。
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	正三角形を定義する RegularPolygon のサブクラス。
com/example/programmingas3/geometricshapes/Square.as	正四角形を定義する RegularPolygon のサブクラス。
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	指定された種類およびサイズでシェイプを作成するためのファクトリメソッドを含むクラス。

GeometricShapes クラスの定義

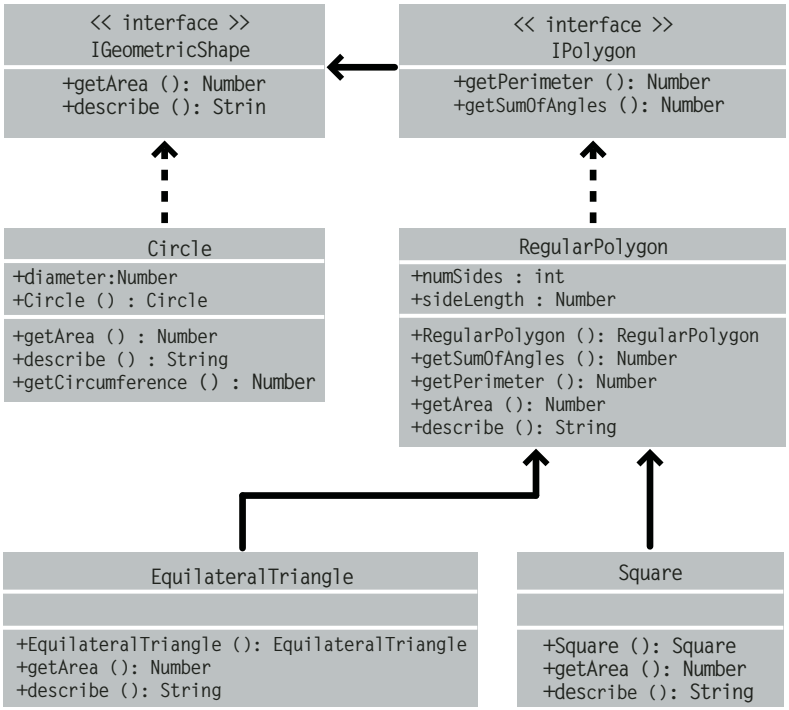
GeometricShapes アプリケーションを使用するユーザーは、幾何学図形の種類とサイズを指定することができます。それに対しアプリケーションは、シェイプの説明、その面積、および周辺の長さで応答します。

アプリケーションユーザーインターフェイスは、シェイプの種類を選択、サイズを設定、説明を表示するための小数のコントロールを含み、ごく普通です。このアプリケーションで最も興味深い部分は、内部にありそれはクラスおよびインターフェイス自体の構造に関係します。

このアプリケーションは幾何学図形を扱いますが、それを図では表示しません。次の章の例で再利用する、クラスおよびインターフェイスの小規模なライブラリを提供します (158 ページの「getter および setter のオーバーライド」を参照してください)。SpriteArranger の例ではシェイプが図で表示され、GeometricShapes アプリケーションによってここで提供されるクラスフレームワークに基づいて、ユーザーはシェイプを操作することができます。

次の図に、この例で幾何学図形を定義するクラスとインターフェイスを UML (Unified Modeling Language) 表記で示します。

GeometricShapes Example クラス



インターフェイスでの一般的な動作の定義

この GeometricShapes アプリケーションでは、円、四角形、および等辺三角形の 3 種類のシェイプを処理します。GeometricShapes クラス構造は、非常に単純なインターフェイスである、3 種類すべてのシェイプに共通するメソッドをリストする IGeometricShape で開始します。

```
package com.example.programmingas3.geometricshapes
{
  public interface IGeometricShape
  {
    function getArea():Number;
    function describe():String;
  }
}
```

インターフェイスでは、シェイプの領域を計算して返す getArea() メソッドと、シェイプのプロパティのテキスト説明を構成する describe() メソッドの 2 つのメソッドを定義します。

各シェイプの周囲の長さを知りたいこともあります。ただし、円の周囲は円周と呼ばれ、その計算方法は固有であるため、三角形または正方形とは動作が異なります。それでもなお、三角形、四角形、およびその他の多角形の間には十分な類似性が存在します。従って、これらに新しいインターフェイスクラス IPolygon を定義することは意味があります。IPolygon インターフェイスも、次に示すように、どちらかと言えば単純です。

```
package com.example.programmingas3.geometricshapes
{
  public interface IPolygon extends IGeometricShape
  {
    function getPerimeter():Number;
    function getSumOfAngles():Number;
  }
}
```

このインターフェイスは、すべての辺の長さの合計を測定する getPerimeter() メソッドと、すべての内角の和を計算する getSumOfAngles() メソッドの、すべての多角形に対して共通の 2 つのメソッドを定義します。

IPolygon インターフェイスは、IGeometricShape インターフェイスを拡張します。これは、IPolygon インターフェイスを実装するクラスはいずれも、4 つのすべてのメソッド (IGeometricShape インターフェイスから 2 つ、IPolygon インターフェイスから 2 つ) を宣言する必要があることを意味します。

シェイプクラスの定義

各種類のシェイプに共通するメソッドに関する考えがまとまったら、シェイプクラス自体を定義することができます。実装する必要のあるメソッドの数で見た場合、最も単純なシェイプはここに示すように、Circle クラスです。

```
package com.example.programmingas3.geometricshapes
{
  public class Circle implements IGeometricShape
  {
    public var diameter:Number;

    public function Circle(diam:Number = 100):void
    {
      this.diameter = diam;
    }

    public function getArea():Number
    {
      // The formula is Pi * radius * radius.
      var radius:Number = diameter / 2;
      return Math.PI * radius * radius;
    }

    public function getCircumference():Number
    {
      // The formula is Pi * diameter.
      return Math.PI * diameter;
    }

    public function describe():String
    {
      var desc:String = "This shape is a Circle.\n";
      desc += "Its diameter is " + diameter + " pixels.\n";
      desc += "Its area is " + getArea() + ".\n";
      desc += "Its circumference is " + getCircumference() + ".\n";
      return desc;
    }
  }
}
```

Circle クラスは、IGeometricShape インターフェイスを実装するので、`getArea()` メソッドと `describe()` メソッドの両方に対するコードを提供する必要があります。さらに、Circle クラスに固有の `getCircumference()` メソッドを定義します。Circle クラスでは、他の多角形クラスでは見られない `diameter` プロパティの宣言もあります。

残りの 2 つの種類のシェイプの正方形および等辺三角形には、他の点で共通することがあります。つまり、等しい長さの辺を有する点で、両方について共通の式を使用して周辺と内角の総和を計算することができます。実際、これらの共通の式は、今後定義する必要がある他の正多角形にも適用されます。

RegularPolygon クラスは、Square クラスと EquilateralTriangle クラスの両方のスーパークラスです。スーパークラスを使用すると、共通メソッドを 1 か所で定義することできるため、サブクラスごとに別々に共通メソッドを定義する必要がありません。RegularPolygon クラスのコードを次に示します。

```
package com.example.programmingas3.geometricshapes
{
  public class RegularPolygon implements IPolygon
  {
    public var numSides:int;
    public var sideLength:Number;

    public function RegularPolygon(len:Number = 100, sides:int = 3):void
    {
      this.sideLength = len;
      this.numSides = sides;
    }

    public function getArea():Number
    {
      // This method should be overridden in subclasses.
      return 0;
    }

    public function getPerimeter():Number
    {
      return sideLength * numSides;
    }

    public function getSumOfAngles():Number
    {
      if (numSides >= 3)
      {
        return ((numSides - 2) * 180);
      }
      else
      {
        return 0;
      }
    }

    public function describe():String
    {
      var desc:String = "Each side is " + sideLength + " pixels long.\n";
      desc += "Its area is " + getArea() + " pixels square.\n";
      desc += "Its perimeter is " + getPerimeter() + " pixels long.\n";
      desc += "The sum of all interior angles in this shape is " +
  getSumOfAngles() + " degrees.\n";
      return desc;
    }
  }
}
```

最初に、RegularPolygon クラスは、すべての正多角形に共通する 2 つのプロパティを宣言します。つまり、各辺の長さの sideLength プロパティと辺の数の numSides プロパティです。

RegularPolygon クラスは、IPolygon インターフェイスを実装し、IPolygon インターフェイスメソッドのすべての 4 つを宣言します。これらのうちの 2 つ (getPerimeter() メソッドと getSumOfAngles() メソッド) を、共通の式を使用して実装します。

getArea() メソッドの式はシェイプによって異なるので、基本クラスバージョンのメソッドは、サブクラスメソッドで継承可能な共通の論理を含むことができません。代わりに、面積を計算しなかったことを示すには、単にデフォルト値 0 を返します。各シェイプの面積を正しく計算するには、RegularPolygon クラスのサブクラスは getArea() メソッド自体をオーバーライドする必要があります。

EquilateralTriangle クラスの次のコードは、getArea() メソッドをオーバーライドする方法を示しています。

```
package com.example.programmingas3.geometricshapes
{
  public class EquilateralTriangle extends RegularPolygon
  {
    public function EquilateralTriangle(len:Number = 100):void
    {
      super(len, 3);
    }

    public override function getArea():Number
    {
        // The formula is ((sideLength squared) * (square root of 3)) / 4.
        return ( (this.sideLength * this.sideLength) * Math.sqrt(3) ) / 4;
    }

    public override function describe():String
    {
         /* starts with the name of the shape, then delegates the rest
         of the description work to the RegularPolygon superclass */
        var desc:String = "This shape is an equilateral Triangle.\n";
        desc += super.describe();
        return desc;
    }
  }
}
```

override キーワードは、EquilateralTriangle.getArea() メソッドが、RegularPolygon スーパークラスからの getArea() メソッドを意図的にオーバーライドすることを示します。EquilateralTriangle.getArea() メソッドが呼び出されると、前記のコードの式を使用して面積が計算されます。RegularPolygon.getArea() メソッドのコードが実行されることは決してありません。

対照的に、EquilateralTriangle クラスは独自のバージョンの getPerimeter() メソッドを定義しません。EquilateralTriangle.getPerimeter() メソッドが呼び出されると、継承チェーンに進み、RegularPolygon サブクラスの getPerimeter() メソッドのコードが実行されます。

EquilateralTriangle() コンストラクタは super() ステートメントを使用して、そのスーパークラスの RegularPolygon() コンストラクタを明示的に呼び出します。両方のコンストラクタのパラメータセットが同じ場合は、EquilateralTriangle() コンストラクタを完全に省略しておくこともできますが、代わりに RegularPolygon() コンストラクタが実行されます。ただし、RegularPolygon() コンストラクタは追加パラメータの numSides を必要とします。したがって、EquilateralTriangle() コンストラクタは、三角形は 3 辺であることを示すために len 入力パラメータと値 3 を渡す super(len, 3) を呼び出します。

describe() メソッドも super() ステートメントを使用しますが、方法は異なります。すなわち、describe() メソッドの RegularPolygon スーパークラスのバージョンを呼び出します。EquilateralTriangle.describe() メソッドは、最初に desc ストリング変数をシェイプの種類に関する記述に設定します。次に、super.describe() を呼び出すことによって RegularPolygon.describe() メソッドの結果を取得し、その結果を desc ストリングに追加します。

ここでは Square クラスについて詳細は説明しません。しかし、このクラスは、EquilateralTriangle クラスと類似点があり、コンストラクタと getArea() および describe() メソッドの独自の実装を提供します。

ポリモーフィズムとファクトリメソッド

インターフェイスと継承を有効に利用するクラスセットは、いろいろな興味深い方法で使用することができます。たとえば、これまでに説明したすべてのシェイプクラスは、IGeometricShape インターフェイスを実装するか、またはそのようにスーパークラスを拡張します。したがって、IGeometricShape のインスタンスであるように変数を定義する場合は、describe() メソッドを呼び出すために、実際にインスタンスが Circle のインスタンスであるか Square クラスのインスタンスであるかを知っている必要はありません。

次のコードは、この働きを示します。

```
var myShape:IGeometricShape = new Circle(100);
trace(myShape.describe());
```

myShape.describe() が呼び出された場合、変数が IGeometricShape インターフェイスのインスタンスと定義されていても Circle が基礎となるクラスであるので、Circle.describe() メソッドが実行されます。

この例からわかることは、ポリモーフィズムの動作原理です。つまり、正確に同じメソッド呼び出しの場合も、メソッドが呼び出されるオブジェクトのクラスに応じて、実行されるコードは異なります。

GeometricShapes アプリケーションでは、この種類のインターフェイスベースのポリモーフィズムを適用するために、ファクトリメソッドとして知られている簡略化バージョンの設計パターンを使用します。" ファクトリメソッド " という用語は、基礎になるデータ型または内容が状況によって変わる可能性のあるオブジェクトを返す関数を指します。

ここに示す GeometricShapeFactory クラスは、createShape() というファクトリメソッドを次のように定義します。

```
package com.example.programmingas3.geometricshapes
{
  public class GeometricShapeFactory
  {
    public static var currentShape:IGeometricShape;

    public static function createShape(shapeName:String,
                                       len:Number):IGeometricShape
    {
      switch (shapeName)
      {
        case "Triangle":
          return new EquilateralTriangle(len);

        case "Square":
          return new Square(len);

        case "Circle":
          return new Circle(len);
      }
      return null;
    }

    public static function describeShape(shapeType:String,
  shapeSize:Number):String
    {
      GeometricShapeFactory.currentShape =
        GeometricShapeFactory.createShape(shapeType, shapeSize);
      return GeometricShapeFactory.currentShape.describe();
    }
  }
}
```

createShape() ファクトリメソッドを使用すると、シェイプサブクラスのコンストラクタで、作成するインスタンスの詳細を定義できます。一方、IGeometricShape インスタンスとして新しいオブジェクトを返すことにより、さらに一般的な方法でアプリケーションによる処理ができるようになります。

前述の例の describeShape() メソッドは、さらに具体的なオブジェクトへの汎用的な参照を取得するために、アプリケーションでファクトリメソッドをどのように使用することができるかを示しています。次のように、新しく作成された Circle オブジェクトの説明を取得することができます。

```
GeometricShapeFactory.describeShape("Circle", 100);
```

続いて、describeShape() メソッドは同じパラメータ付きで createShape() ファクトリメソッドを呼び出しますが、IGeometricShape オブジェクトとして型指定された currentShape という静的変数には、新しい Circle オブジェクトが格納されています。次に、currentShape オブジェクトで describe() メソッドが呼び出され、その呼び出しは自動的に解決されて Circle.describe() を実行し、円の詳細な説明を返します。

サンプルアプリケーションの機能拡張

インターフェイスおよび継承の実際の威力は、アプリケーションを機能拡張または変更したときに明らかになります。

新しいシェイプとして五角形をこのサンプルアプリケーションに追加するとします。RegularPolygon クラスを拡張して独自のバージョンの `getArea()` メソッドおよび `describe()` メソッドを定義する新しい Pentagon クラスを作成します。次に、アプリケーションのユーザーインターフェイスのコンボボックスに新しい Pentagon オプションを追加します。しかし、これで全部です。Pentagon クラスは、継承によって RegularPolygon クラスから `getPerimeter()` メソッドおよび `getSumOfAngles()` メソッドの機能を自動的に取得します。IGeometricShape インターフェイスを実装するクラスから継承するため、Pentagon インスタンスは IGeometricShape インスタンスとしても扱うことができます。つまり、GeometricShapeFactory クラスのいずれのメソッドも変更する必要がないので、新しい種類のシェイプを必要なときに追加することが非常に簡単になります。

実習として Pentagon クラスを Geometric Shapes の例に追加してみると、アプリケーションに新機能を追加する負荷がインターフェイスおよび継承によってどの程度軽減できるかがわかります。

第 5 章

日付と時刻の操作

5

時間を扱う処理は常に必要なわけではありませんが、多くのソフトウェアアプリケーションにとって重要な要素の 1 つです。ActionScript 3.0 には、カレンダーの日付、時刻、および時間間隔を扱う強力な機能が用意されています。時間に関連したそれらの機能のほとんどは、2 つのメインクラス、すなわち Date クラスと flash.utils パッケージに含まれる新しい Timer クラスにより提供されます。

目次

日付と時刻の基礎 179
カレンダー日付と時刻の管理 180
タイマー間隔の制御 184
例 : 単純なアナログ時計 186

日付と時刻の基礎

日付と時刻の操作の概要

日付と時刻は、ActionScript プログラムで使用される一般的な情報のタイプです。たとえば、現在の曜日を調べたり、ユーザーが多くの画面のうちで特定の画面にどの程度時間を費やしたか測定したりすることが必要になる場合があります。ActionScript では、Date クラスを使用して、日付と時刻の情報を含む単一の時点を表すことができます。Date インスタンスには、年、月、日、曜日、時、分、秒、ミリ秒、およびタイムゾーンを初めとする個別の日付と時刻の単位値が含まれます。ActionScript ではまた、より高度な使用を可能にするために、特定の遅延後、または反復する間隔でアクションを実行するために使用できる Timer クラスが用意されています。

一般的な日付と時刻のタスク

この章では、日付と時刻の情報を操作する次の一般的なタスクについて説明します。

- Date オブジェクトの操作
- 現在の日付と時刻の取得
- 個別の日付と時刻の単位 (日、年、時、分など) へのアクセス
- 日付と時刻の算術演算の実行
- タイムゾーン間の変換
- 反復するアクションの実行
- 設定した時間間隔後のアクションの実行

重要な概念と用語

次の参照リストに、この章で使用する重要な用語を示します。

- UTC 時間 : 世界標準時 - "0 時 " 基準タイムゾーン。他のすべてのタイムゾーンは、UTC 時間に対する相対時間 (前後) として定義されます。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章のコードリストは主に Date オブジェクトに関するものなので、サンプルをテストするには、ステージ上のテキストフィールドに値を書き込むか、`trace()` 関数を使用して [出力] パネルに値を出力することにより、サンプルで使用する変数の値を確認する必要があります。これらのトピックについては、60 ページの「章内のコード例のテスト」を参照してください。

カレンダー日付と時刻の管理

ActionScript 3.0 では、カレンダー日付と時刻の管理に関するすべての機能がトップレベルの Date クラスにまとめられています。Date クラスには、世界標準時 (UTC) または何らかのタイムゾーンに基づくローカル時間において日付と時刻を処理するメソッドおよびプロパティがあります。UTC は時刻の基準となるもので、グリニッジ標準時 (GMT) と基本的には同じです。

Date オブジェクトの作成

Date クラスは、すべてのコアクラスの中で最も多くの用途に使用できるコンストラクタメソッドを備えています。このコンストラクタは次のように 4 つの方法で使用できます。

(1) パラメータを指定しない場合、Date() コンストラクタでは、動作環境のタイムゾーンにおけるローカル時間で現在の日付と時刻を格納した Date オブジェクトを返します。その例を次に示します。

```
var now:Date = new Date();
```

(2) 1 つの数値パラメータを指定した場合、Date() コンストラクタはその値を 1970 年 1 月 1 日からの経過ミリ秒数として解釈し、それに基づいた Date オブジェクトを返します。パラメータは UTC の 1970 年 1 月 1 日を起点とする経過ミリ秒数として解釈されますが、返される Date オブジェクトが示す値は、(UTC 用メソッドを使用して取得および表示しない限り) ローカルのタイムゾーンで表示されます。ミリ秒数を表す 1 個のパラメータを指定して Date オブジェクトを新規作成する際は、ローカル時間と UTC の間のタイムゾーンの時差に注意してください。次のステートメントでは、UTC の 1970 年 1 月 1 日の深夜を表す Date オブジェクトを作成します。

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// 基準日の 1970/1/1 から 1 日後の Date を取得する
var startTime:Date = new Date(millisecondsPerDay);
```

(3) 複数の数値パラメータを Date() コンストラクタに渡すこともできます。その場合、コンストラクタでは各パラメータを順に年、月、日、時、分、秒、ミリ秒の値と解釈し、それらに基づいた Date オブジェクトを返します。パラメータの値は UTC ではなくローカル時間を表すものと見なされます。次のステートメントでは、ローカル時間の 2000 年 1 月 1 日午前 0 時を表す Date オブジェクトを作成します。

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

(4) 1 つのストリングパラメータを Date() コンストラクタに渡した場合、コンストラクタではそのストリングを解析して日付と時刻の要素を抽出しようと試み、それらに基づいた Date オブジェクトを返します。この方法で使用する場合は、Date() コンストラクタを try..catch ブロックで囲み、解析時のエラーをトラップできるようにしておくことをお勧めします。Date() コンストラクタが受け付けるストリング形式は多数あり、その一覧が『ActionScript 3.0 コンポーネントリファレンスガイド』に示されています。次のステートメントでは、新しい Date オブジェクトの初期化にストリング値を使用しています。

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Date() コンストラクタがストリングパラメータの解析に失敗した場合、例外が発生するのではなく、作成した Date オブジェクトに無効な日付値が格納されます。

時間単位の値の取得

Date オブジェクトに格納された各種の時間単位の値は、Date クラスのプロパティとメソッドを使用して取得できます。Date オブジェクトに格納されている時間単位の値を取得するには、次の各プロパティを使用します。

- `fullYear` プロパティ
- `month` プロパティ : 1 月から 12 月を 0 ～ 11 で表す数値形式
- `date` プロパティ : 月の暦日を 1 ～ 31 で表す数値形式
- `day` プロパティ : 日曜日から土曜日を 0 ～ 6 で表す数値形式
- `hours` プロパティ : 時間を 0 ～ 23 で表す数値形式
- `minutes` プロパティ
- `seconds` プロパティ
- `milliseconds` プロパティ

また、Date クラスにはそれぞれの時間単位の値を取得する方法が複数用意されています。たとえば、Date オブジェクトに格納された月の値は次の 4 とおりの方法で取得できます。

- `month` プロパティ
- `getMonth()` メソッド
- `monthUTC` プロパティ
- `getMonthUTC()` メソッド

4 つの方法は効率面ではいずれも同等なので、必要に応じて便利な方法を使用してください。

上記のプロパティはそれぞれ、日付の値全体の一部を構成する要素を表しています。たとえば、milliseconds プロパティの値が 999 を超えることはありません。1000 に到達すると seconds プロパティの値が 1 加算され、milliseconds プロパティの値は 0 になります。

Date オブジェクトの値を 1970 年 1 月 1 日 (UTC) からの経過ミリ秒数として取得する必要がある場合は、`getTime()` メソッドを使用します。また、このメソッドと対照の関係にある `setTime()` メソッドを使用すれば、1970 年 1 月 1 日 (UTC) からの経過ミリ秒数を指定して既存の Date オブジェクトの値を変更できます。

日付と時刻の加算および減算

Date クラスでは、日付と時刻に対して加算および減算を実行できます。日付値は、内部的にはミリ秒換算で維持されているので、Date オブジェクトとの間で加算または減算を行う前に、その他の値をミリ秒に変換する必要があります。

日付と時刻の加算および減算を数多く実行するアプリケーションの場合は、次のように、共通の時刻単位値をミリ秒換算で保持する定数を作成すると便利です。

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

これで、標準の時刻単位を使用して容易に日付の演算を実行できます。次のコードでは、getTime() および setTime() メソッドを使用して、日付値を現在の時刻から 1 時間後に設定します。

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

日付値を設定するもう 1 つの方法は、ミリ秒を表す 1 個のパラメータを使用して新しい Date オブジェクトを作成することです。たとえば、次のコードでは、ある日付に日数を 30 加えた日付を計算します。

```
// 今日を請求書の日付として設定する
var invoiceDate:Date = new Date();

// 30 日を加えて期限を取得する
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

次に、30 日単位の時間を表すために millisecondsPerDay 定数に 30 を掛け、その結果を invoiceDate 値に加えて dueDate 値の設定に使用します。

タイムゾーン間の変換

日付と時刻の演算処理は、あるタイムゾーンを別のタイムゾーンに換算する必要がある場合に便利です。また、その際には、Date オブジェクトのタイムゾーンと UTC との時差 (分) を返す getTimezoneOffset() メソッドも役立ちます。値が分単位で返されるのは、タイムゾーンによっては 1 時間未満の単位で時差が設定されており、隣接するタイムゾーンとの差が 30 分という場合があるためです。

次の例では、タイムゾーンのオフセットを使用してローカル時間の日付を UTC に変換します。まずミリ秒単位のタイムゾーン値を算出し、それに基づいて Date オブジェクトの値を調整するという方法で処理を実行しています。

```
// ローカル時間の Date を作成する
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// タイムゾーンのオフセットを加算または減算して、Date を UTC に変換する
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

タイマー間隔の制御

Flash Adobe Flash CS3 Professional によるアプリケーション開発の場合は、タイムラインを使用してアプリケーションの進行をフレーム単位で確実に制御できますが、ActionScript だけを使用するプロジェクトの場合は、別のタイミング制御メカニズムを使用する必要があります。

ループとタイマーの違い

プログラミング言語によっては、`for` や `do..while` などのループステートメントを使ってタイミングの制御方法を独自に工夫しなければならないことがあります。

一般にループステートメントは、動作環境となるローカルマシンの処理速度に応じてなるべく高速に実行されるため、使用するマシンによって速度に違いが生じます。アプリケーションを一定のタイミングで動作させるには、何らかのカレンダーやクロックによって生成されるタイミングを利用する必要があります。ゲーム、アニメーション、およびリアルタイム制御など、多くのアプリケーションは、どのマシンでも一貫して利用できる、安定した時間ベースのタイミングメカニズムを必要とします。

ActionScript 3.0 の Timer クラスには、こうしたニーズを満たす強力な機能があります。Timer クラスでは ActionScript 3.0 のイベントモデルを使用し、指定された時間間隔が経過するごとにタイマーイベントを送出します。

Timer クラス

タイマー制御関数の扱いについて、ActionScript 3.0 では Timer クラス (flash.utils.Timer) を使用する方法が推奨されています。このクラスを使用すると、所定の時間間隔が経過するたびにイベントを送出することができます。

タイマーを開始するには、最初に Timer クラスのインスタンスを作成し、そのインスタンスに対して、タイマーイベントを生成する頻度と生成終了までの送出回数を指定します。

たとえば次のコードでは、1秒ごとのイベント送出を 60 秒間続けるタイマーインスタンスを作成します。

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

Timer オブジェクトでは、所定の時間間隔が経過するたびに TimerEvent オブジェクトを送出します。TimerEvent オブジェクトのイベント型は `timer` (定数 `TimerEvent.TIMER` によって定義される) です。TimerEvent オブジェクトには、標準の Event オブジェクトと同じプロパティが格納されます。

また、繰り返しの回数が固定されたタイマーインスタンスの場合、最後の時間間隔に到達したときに `timerComplete` イベント (定数 `TimerEvent.TIMER_COMPLETE` によって定義される) も送出されます。

Timer クラスの実際の動作を確認できる短いサンプルアプリケーションを次に示します。

```
package
{
  import flash.display.Sprite;
  import flash.events.TimerEvent;
  import flash.utils.Timer;

  public class ShortTimer extends Sprite
  {
    public function ShortTimer()
    {
      // 5 秒の Timer を新規作成する
      var minuteTimer:Timer = new Timer(1000, 5);

      // 時間間隔イベントおよび完了イベント用のリスナーを指定する
      minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
      minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE,
  onTimerComplete);

      // タイマーのカウントを開始する
      minuteTimer.start();
    }

    public function onTick(event:TimerEvent):void
    {
      // 現在のカウントを表示する
      // このイベントのターゲットは、該当する Timer インスタンス自体である
      trace("tick " + event.target.currentCount);
    }

    public function onTimerComplete(event:TimerEvent):void
    {
      trace("Time's Up!");
    }
  }
}
```

この例では、ShortTimer クラスが作成されたとき、1秒ごとのカウントを 5 秒間続ける Timer インスタンスを作成します。次に、このタイマー用のリスナーを 2 つ指定しています。1つはカウント各回のイベントを受け取るリスナーであり、もう1つは timerComplete イベントを受け取るリスナーです。

その後、タイマーのカウントを開始すると、以後は1秒間隔で onTick() メソッドが繰り返し呼び出されます。

onTick() メソッドでは、単に現在のカウントを表示しています。5 秒が経過すると onTimerComplete() メソッドが呼び出され、カウントが終了したことを表示します。

このサンプルを実行すると、コンソールまたはトレースウィンドウに1秒ごとに次の行が表示されます。

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

flash.utils パッケージに含まれるタイマー制御関数

ActionScript 3.0 には、ActionScript 2.0 と同様のタイマー制御関数が多数用意されています。それらはパッケージレベル関数として flash.utils パッケージに含まれており、ActionScript 2.0 の場合と同じように動作します。

関数	説明
`clearInterval(id:uint):void`	指定した `setInterval()` 呼び出しをキャンセルします。
`clearTimeout(id:uint):void`	指定した `setTimeout()` 呼び出しをキャンセルします。
`getTimer():int`	Adobe Flash Player が初期化された時点からの経過ミリ秒数を返します。
`setInterval(closure:Function, delay:Number, ... arguments):uint`	ミリ秒単位で指定した間隔ごとに関数を実行します。
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	ミリ秒単位で指定した遅延時間の経過後に、指定した関数を実行します。

これらの関数は、後方互換性を維持する目的で ActionScript 3.0 にも残されているものです。新しい ActionScript 3.0 アプリケーションで使用することはお勧めしません。一般に、Timer クラスを使用するほうが容易さの点でも効率の点でも有利です。

例 : 単純なアナログ時計

ここでは、単純なアナログ時計の例を使って、この章で述べた日付と時刻に関する内容のうち次の 2 つについて示します。

- 現在の日付と時刻を取得し、時、分、秒の値を抽出する方法
- Timer を使用してアプリケーションの進行ペースを制御する方法

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。SimpleClock アプリケーションのファイルは、"Samples/SimpleClock" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
SimpleClockApp.mxml または SimpleClockApp.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/simpleclock/SimpleClock.as	メインアプリケーションファイル。
com/example/programmingas3/simpleclock/AnalogClockFace.as	時計の丸い文字盤と時間に合わせた時針、分針、秒針を描画するコード。

SimpleClock クラスの定義

この時計のように単純なアプリケーションであっても、構造をわかりやすく整理しておけば、後で容易に拡張できて便利です。そこで、この SimpleClock アプリケーションの場合は SimpleClock クラスで起動時とタイマー制御の処理を扱い、時刻を実際に表示する処理には別の AnalogClockFace というクラスを使用しています。

次のコードでは、SimpleClock クラスの定義と初期化を行います (Flash バージョンでは、代わって SimpleClock が Sprite クラスを拡張することに注意してください)。

```
public class SimpleClock extends UIComponent
{
   /**
    * 時刻表示コンポーネント
    */
   private var face:AnalogClockFace;

   /**
    * このアプリケーションにとって心臓の鼓動のような役割を持つ Timer
    */
   private var ticker:Timer;
```

このクラスには、2 つの重要なプロパティがあります。

- `face` プロパティ : AnalogClockFace クラスのインスタンス
- `ticker` プロパティ : Timer クラスのインスタンス

SimpleClock クラスではデフォルトのコンストラクタを使用します。`initClock()` メソッドでは実際のセットアップ処理を担当し、文字盤を作成して Timer インスタンスの動作を開始します。

文字盤の作成

次のコードでは、時刻の表示に使用する文字盤を作成します。

```
   /**
    * SimpleClock インスタンスをセットアップする
    */
   public function initClock(faceSize:Number = 200)
   {
      // 文字盤を作成して表示リストに追加する
      face = new AnalogClockFace(Math.max(20, faceSize));
      face.init();
      addChild(face);

      // 時計の初期表示を描画する
      face.draw();
```

`initClock()` メソッドに対して、文字盤のサイズを指定できます。`faceSize` の値を指定しない場合、デフォルト値の 200 ピクセルが使用されます。

次に、このアプリケーションでは文字盤を初期化し、DisplayObject クラスから継承した `addChild()` メソッドを使用して表示リストに文字盤を追加します。さらに、いったん現在の時刻で盤面を表示するために `AnalogClockFace.draw()` メソッドを呼び出します。

タイマーの開始

文字盤を作成した後は、initClock() メソッドでタイマーをセットアップします。

```
// 1 秒ごとにイベントを送出する Timer を作成する
ticker = new Timer(1000);

// Timer イベントを処理する onTick() メソッドを指定する
ticker.addEventListener(TimerEvent.TIMER, onTick);

// 時間の計測を開始する
ticker.start();
```

このメソッドは、1秒 (1000 ミリ秒) ごとにイベントを送出する Timer インスタンスを最初に作成します。第 2 パラメータの repeatCount を Timer() コンストラクタに渡していないので、この Timer はイベントの送出を無期限に繰り返します。

SimpleClock.onTick() メソッドは、timer イベントを受け取って1秒ごとに実行されます。

```
public function onTick(event:TimerEvent):void
{
   // 時計の表示を更新する
   face.draw();
}
```

AnalogClockFace.draw() メソッドは、単に時計の盤面と針を描画します。

現在時刻の表示

AnalogClockFace クラスのコードのほとんどは、時計の盤面を構成する表示要素のセットアップに関するものです。AnalogClockFace が初期化される際には、丸い輪郭を描き、個々の定時目盛りの位置にテキストラベルで数字を配置し、さらに、時針、分針、秒針を表す 3 つの Shape オブジェクトを作成します。

SimpleClock アプリケーションが動作し始めた後は、次のように AnalogClockFace.draw() メソッドが1秒ごとに呼び出されます。

```
/**
 * 表示を描画する際に親コンテナから呼び出される
 */
public override function draw():void
{
   // 現在の日付と時刻をインスタンス変数に格納する
   currentTime = new Date();
   showTime(currentTime);
}
```

このメソッドは、時計の針を描画している間に時刻が変化しないように、現在時刻を変数に保存します。それから、次のように showTime( ) メソッドを呼び出して針を表示します。

```
/**
 * 指定された日付 / 時刻を、アナログ時計のスタイルで表示する
 */
public function showTime(time:Date):void
{
   // 時刻を構成する値を取得する
   var seconds:uint = time.getSeconds();
   var minutes:uint = time.getMinutes();
   var hours:uint = time.getHours();

   // 6 を掛けて角度を求める
   this.secondHand.rotation = 180 + (seconds * 6);
   this.minuteHand.rotation = 180 + (minutes * 6);

   // 30 を掛けて基本の角度を求め、さらに
   // 最大 29.5 度 (59×0.5) までの角度を
   // 分の値に応じて加える
   this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

このメソッドでは、最初に現在時刻の時、分、秒それぞれの値を抽出し、それらに基づいて 3 本の針の角度を計算します。秒針は 60 秒で 1 周するので、1 秒につき 6 度 (360/60) だけ進むことになります。分針も 1 分につきこれと同じ角度だけ進みます。

分針が進むにつれて、時針も少しずつ動きます。時針は 1 時間につき 30 度 (360/12) だけ進みますが、その間も 1 分につき 0.5 度 (30 度 / 60 分) ずつ変化していきます。

第 6 章

ストリングの操作

6

String クラスには、テキストのストリングを操作するメソッドが含まれています。ストリングの扱いは、さまざまなオブジェクトを使用する場合において重要となります。この章で説明する各種のメソッドは、TextField、StaticText、XML、ContextMenu、FileReference などのオブジェクトで使用されるストリングを操作する際に役立ちます。

ストリングは文字の連続によって構成されます。ActionScript 3.0 では ASCII と Unicode の文字がサポートされます。

目次

ストリングの基礎 192
ストリングの作成 194
length プロパティ 195
ストリング内の文字の操作 196
ストリングの比較 197
他のオブジェクトのストリング表現の取得 197
ストリングの連結 198
ストリング内のサブストリングとパターンの検索 198
ストリングの大文字と小文字の変換 203
例 : ASCII アート 204

ストリングの基礎

ストリングの操作の概要

プログラミングの専門用語でストリングは、一連の文字、数値、または他の文字を単一の値に結合したテキスト値です。たとえば、このコード行では String データ型の変数を作成し、その変数にリテラルストリング値を割り当てます。

```
var albumName:String = "Three for the money";
```

この例が示すように、ActionScript では二重引用符または一重引用符でテキストを囲むことにより、ストリング値を表すことができます。次に、ストリングの例をいくつか示します。

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

ActionScript でテキストを操作する場合は常に、ストリング値を操作することになります。ActionScript の String クラスは、テキスト値を操作するために使用できるデータ型です。String インスタンスは、その他の多くの ActionScript クラス内で、プロパティ、メソッドパラメータなどにより頻繁に使用されます。

ストリングの操作に関する一般的なタスク

次に、この章で説明する一般的なストリングに関連するタスクを示します。

- String オブジェクトの作成
- 復帰文字、タブ、非キーボード文字などの特殊文字の操作
- ストリング長の測定
- ストリング内の個別文字の分離
- ストリングの結合
- ストリングの比較
- ストリングの一部の検索、抽出、置換
- ストリングの大文字または小文字への変換

重要な概念と用語

次の参照リストに、この章で使用されている重要な用語を示します。

- ASCII: コンピュータプログラムでテキストの文字および記号を表すための体系。ASCII 体系では、26 文字の英字アルファベットに加えて、限定的な追加文字セットがサポートされます。
- 文字 : テキストデータの最小単位 (単一の文字または記号)。
- 連結 : 複数のストリング値を相互に結合し、あるストリング値を別のストリング値の末尾に追加することにより、新しいストリング値を作成します。
- 空の文字列 : テキスト、空白、その他の文字を含まないストリング。"" として記述されます。空の文字列値は、null 値を持つ String 変数とは異なります。null の String 変数は、String インスタンスが未割り当てなのに対して、空の文字列は文字を含まない値のインスタンスを含む変数です。
- ストリング : テキスト値 (文字シーケンス)。
- ストリングリテラル (“ リテラルストリング ”) : コード内で明示的に記述されるストリング値。二重引用符または一重引用符で囲まれたテキスト値として記述します。
- サブストリング : 別のストリングの一部であるストリング。
- Unicode: コンピュータプログラムでテキストの文字および記号を表すための標準体系。Unicode 体系では、任意の文字記述システムの任意の文字を使用できます。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章のコードリストは主にテキストの操作に関するものなので、サンプルをテストするには、ステージ上のテキストフィールドインスタンスに値を書き込むか、`trace()` 関数を使用して [出力] パネルに値を出力することにより、サンプルで使用する変数の値を確認する必要があります。これらの手法の詳細については、60 ページの「章内のコード例のテスト」を参照してください。

ストリングの作成

String クラスは ActionScript 3.0 でストリング (テキスト) データを表すために使用されます。ActionScript ストリングは ASCII と Unicode 文字の両方をサポートしています。ストリングを作成する最も簡単な方法は、ストリングリテラルを使用することです。ストリングリテラルを宣言するには、垂直の二重引用符 (") または一重引用符 (') を使用します。たとえば、次の 2 つのストリングは同等です。

```
var str1:String = "hello";
var str2:String = 'hello';
```

次のように new 演算子を使用してストリングを宣言することもできます。

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

次の 2 つのストリングは同等です。

```
var str1:String = "hello";
var str2:String = new String("hello");
```

一重引用符 (') の区切り文字で記述されているストリングリテラル内で一重引用符 (') を使用するには、エスケープ文字として円記号 (\) を使用します。同様に、区切り文字が二重引用符 (") で記述されているストリングリテラル内で二重引用符 (") を使用するには、エスケープ文字として円記号 (\) を付けます。次の 2 つのストリングは同等です。

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

次のように、ストリングリテラル内に存在する一重引用符または二重引用符に基づいて一重引用符または二重引用符を使用するように選択することができます。

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

ActionScript では、垂直の一重引用符 (') と右向きまたは左向きの引用符 (‘ または ’) は区別されるので注意してください。この点は二重引用符についても同様です。ストリングリテラルを囲む場合は垂直の引用符を使用する必要があります。別の場所からコピーしたテキストを ActionScript にペーストする際は、正しい文字になっているか確認してください。

次の表に示すとおり、エスケープ文字として円記号 (\) を使用して、ストリングリテラル内の他の文字を定義することができます。

エスケープシーケンス	文字
`\b`	バックスペース文字
`\f`	改ページ
`\n`	改行
`\r`	復帰文字
`\t`	タブ文字
`\u`*nnnn*	16 進数値 nnnn で指定した文字コードの Unicode 文字。例 : `\u263a` はスマイル記号
`\x`*nn*	16 進数値 nn で指定した文字コードの ASCII 文字
`\'`	一重引用符
`\"`	二重引用符
`\\`	単一の円記号

length プロパティ

すべてのストリングには、ストリング内の文字数と等しい `length` プロパティがあります。

```
var str:String = "Adobe";
trace(str.length);            // 出力 : 5
```

次のとおり、空のストリングと null ストリングの length はいずれも 0 です。

```
var str1:String = new String();
trace(str1.length);           // 出力 : 0

str2:String = '';
trace(str2.length);           // 出力 : 0
```

ストリング内の文字の操作

ストリング内のすべての文字は、ストリング内のインデックス位置 (整数) を備えています。先頭の文字を指すインデックス位置は 0 です。たとえば、次のストリング内で y という文字は 0 の位置にあり、w という文字は 5 の位置にあります。

```
"yellow"
```

この例に示すように、charAt() メソッドおよび charCodeAt() メソッドを使用して、ストリング内のさまざまな位置にある個々の文字を調べることができます。

```
var str:String = "hello world!";
for (var:i = 0; i < str.length; i++)
{
  trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

このコードを実行すると、次の出力が生成されます。

```
h - 104
e - 101
l - 108
l - 108
o - 111
  - 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

次の例に示すように、fromCharCode() メソッドを使用して、ストリングを定義する文字コードを使用することもできます。

```
var myStr:String =
  String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
    // myStr を "hello world!" に設定する
```

ストリングの比較

ストリングを比較するために演算子 <、<=、!=、==、=>、および > を使用できます。これらの演算子は、次の例のように if や while などの条件ステートメントで使用できます。

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
  trace("A < a, B < b, C < c, ...");
}
```

これらの演算子をストリングで使用する場合、ActionScript は、次のように文字を左から右へ比較することにより、ストリング内の各文字の文字コード値を考慮します。

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

次の例に示すように、== と != 演算子を使用して、ストリングを相互に比較し、ストリングを他の型のオブジェクトと比較します。

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

他のオブジェクトのストリング表現の取得

任意の種類のオブジェクトのストリング表現を取得することができます。このために、すべてのオブジェクトには toString() メソッドが用意されています。

```
var n:Number = 99.47;
var str:String = n.toString();
  // str == "99.47"
```

String オブジェクトとストリング以外のオブジェクトに対して + 連結演算子を使用する場合、toString() メソッドは必要ありません。連結の詳細については、次のセクションを参照してください。

String() グローバル関数は、指定されたオブジェクトに対し、toString() メソッドを呼び出すオブジェクトが返す値と同じ値を返します。

ストリングの連結

ストリングの連結とは、2 つのストリングを 1 つのストリングに順番に結合することです。たとえば、次のように + 演算子を使用すると 2 つのストリングを連結できます。

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

また、次の例に示すように += 演算子を使用しても同じ結果が得られます。

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

さらに、次のように String クラスの concat() メソッドを使用して連結を実行する方法もあります。

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

+ 演算子 (または += 演算子) を使用して、String オブジェクトとストリング以外のオブジェクトとを連結する場合、ActionScript では、次のようにストリングでないオブジェクトが String オブジェクトに自動変換され、それから式が評価されます。

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

ただし、次のように括弧をグループ化に使用すれば、+ 演算子のコンテキストを提供することができます。

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

ストリング内のサブストリングとパターンの検索

サブストリングとは、ストリング内の一部を構成する連続した文字です。たとえば、ストリング "abc" のサブストリングは、""、"a"、"ab"、"abc"、"b"、"bc"、"c" です。ストリング内のサブストリングは ActionScript のメソッドを使用して抽出できます。

パターンは、ActionScript ではストリングまたは正規表現によって定義されます。たとえば、次の正規表現は、文字 A、B、C の後に数字が続く特定のパターンを定義します (スラッシュは正規表現の区切り記号です)。

```
/ABC\d/
```

ActionScript には、ストリング内のパターンを検索し、一致するパターンが見つかった場合に、置換サブストリングと置き換えるメソッドが用意されています。それらのメソッドについては、次のセクションで説明します。

正規表現を使用すると複雑なパターンを定義できます。詳細については、273 ページ、第 9 章の「正規表現の使用」を参照してください。

文字の位置によるサブストリングの検索

substr() と substring() メソッドは似ています。いずれもストリング内のサブストリングを戻り値として返すメソッドです。また、指定するパラメータは 2 つであり、どちらのメソッドでも、最初のパラメータは、与えられたストリングの開始文字の位置です。しかし、substr() メソッドの場合はサブストリングの " 長さ " を第 2 パラメータで指定するのに対し、substring() メソッドの場合はサブストリングの " 終了位置 " を第 2 パラメータで指定します (終了位置にある文字そのものは戻り値に含まれません)。2 つのメソッドの違いを確認できる例を次に示します。

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // 出力 : Paris, Texas!!!
trace(str.substring(11,15)); // 出力 : Pari
```

slice() メソッドの機能は、substring() メソッドに似ています。パラメータに負でない整数を指定した場合の動作はまったく同じです。しかし、slice() メソッドではパラメータに負の整数を指定することもでき、その場合は、ストリングの末尾を基準として文字の位置を指定したことになります。例を次に示します。

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // 出力 : Pari
trace(str.slice(-3,-1)); // 出力 : !!
trace(str.slice(-3,26)); // 出力 : !!!
trace(str.slice(-3,str.length)); // 出力 : !!!
trace(str.slice(-8,-3)); // 出力 : Texas
```

slice() メソッドのパラメータには、負でない整数と負の整数を組み合わせて指定することもできます。

一致するサブストリングの文字の位置の検索

次の例に示すように、indexOf() と lastIndexOf() メソッドを使用して、ストリング内の一致するサブストリングの位置を特定することができます。

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // 出力 : 10
```

indexOf() メソッドでは大文字と小文字が区別されます。

次のように第 2 パラメータを指定すると、ストリング内で検索を開始する位置を示すことができます。

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // 出力 : 21
```

lastIndexOf() は、ストリング内に一致するサブストリングが最後に出現する位置を検索するメソッドです。

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // 出力 : 30
```

lastIndexOf() メソッドで次のように第 2 パラメータを指定すると、ストリング内の指定したインデックス位置から先頭に向かって (右から左へ) 検索が実行されます。

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // 出力 : 21
```

区切り記号で分割されたサブストリングの配列の作成

split() メソッドを使用して、区切り記号に基づいて分割されたサブストリングの配列を作成できます。たとえば、カンマ区切りやタブ区切りのデータを含んだストリングを複数のストリングに分割できます。

次の例では、アンパサンド (&) を区切り文字としてストリングを分割し、サブストリングの配列を作成します。

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2); // params ==
  ["first=joe","last=cheng"]
```

split() メソッドの第 2 パラメータはオプションで、返される配列の最大サイズを指定します。

区切り文字には、次のように正規表現を指定することもできます。

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

詳細については、273 ページ、第 9 章の「正規表現の使用」および『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

ストリング内のパターンの検索とサブストリングの置換

String クラスには、ストリング内のパターンを扱う次のメソッドがあります。

- match() および search() メソッドでは、パターンに一致するサブストリングを検索できます。
- replace() メソッドでは、パターンに一致するサブストリングを検索し、該当する部分を指定のサブストリングに置換できます。

それらのメソッドについては、次のセクションで説明します。

これらのメソッドで使用するパターンは、ストリングまたは正規表現によって定義できます。正規表現の詳細については、273 ページ、第 9 章の「正規表現の使用」を参照してください。

一致するサブストリングの検索

search( ) メソッドは、この例に示すように、特定のパターンに一致する最初のサブストリングのインデックス位置を返します。

```
var str:String = "The more the merrier.";
// ( この検索は、大文字と小文字を区別する )
trace(str.search("the")); // 出力 : 9
```

検索するパターンは、次の例に示すように、正規表現を使用して定義することもできます。

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

ストリングの先頭にある文字のインデックス位置は 0 なので、この場合は trace( ) メソッドで 0 が出力されます。正規表現の i フラグを設定することで、大文字と小文字を区別せずに検索を実行しています。

search( ) メソッドは、一致するサブストリングを 1 つだけ検索してその開始位置のインデックスを返します。たとえ g (グローバル) フラグを設定しても、複数の一致箇所を検索することはありません。

次の例では、より複雑な正規表現を指定することで、二重引用符に囲まれたストリングを検索します。

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // 出力 : 4

str = "The \"more the merrier.";
trace(str.search(pattern)); // 出力 : -1
// ( 閉じる二重引用符がないので、一致するものがないことを示す )
```

match( ) メソッドの機能も search( ) メソッドと似ており、パターンに一致するサブストリングを検索する点は同じです。ただし、次のように正規表現パターンでグローバルフラグを指定すると、match( ) の場合は一致したサブストリングの配列を返します。

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

results 配列は、次のように設定されます。

```
["bob@example.com","omar@example.org"]
```

正規表現の詳細については、273 ページ、第 9 章の「正規表現の使用」を参照してください。

一致するサブストリングの置換

次の例に示すように、replace() メソッドを使用して、ストリング内の指定されたパターンを検索し、一致するパターンを指定された置換ストリングに置き換えることができます。

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch"));
  //sche sells seaschells by the seaschore.
```

この例では、i (ignoreCase) フラグが正規表現で設定されているので、一致する文字列は大文字と小文字を区別せず、g (global) フラグが設定されているので、複数の一致が置き換えられます。詳細については、273 ページ、第 9 章の「正規表現の使用」を参照してください。

置換ストリングには、次の $ 置換コードを含めることができます。$ 置換コードを記述した位置には、次の表に示す置換テキストが挿入されます。

$ コード	置換テキスト
$$	$
$&	パターンに一致したサブストリング。
$`	ストリングのうち、パターンに一致したサブストリングより前にある部分。このコードで使用する記号は、曲がっていない左側一重引用符 (`) です。垂直の一重引用符 (') や、曲がった左側一重引用符 (‘) ではありません。
$'	ストリングのうち、パターンに一致したサブストリングより後にある部分。このコードで使用する記号は、垂直の一重引用符 (') です。
$n	パターン内のグループ化括弧によってキャプチャされた n 番目のサブストリング。*n* は 1 桁の数字 (1 ～ 9) であり、$*n* の後に 10 進数字は続きません。
$nn	括弧指定されたグループの nn 番目にキャプチャされた一致です。nn は 2 桁の 10 進数字 (01 ～ 99) です。nn 番目のキャプチャが未定義の場合、置換テキストは空のストリングになります。

たとえば、一致する 1 番目と 2 番目のキャプチャグループを表す $2 および $1 置換コードを使用した場合を次に示します。

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

replace() メソッドの第 2 パラメータには関数を指定することもできます。その場合は、一致したテキストが、指定した関数の戻り値によって置換されます。

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));
```

```
function usdToEuro(matchedSubstring:String,
                   capturedMatch1:String,
                   index:int,
                   str:String):String
{
  var usd:String = capturedMatch1;
  usd = usd.replace(",", "");
  var exchangeRate:Number = 0.853690;
  var euro:Number = usd * exchangeRate;
  const euroSymbol:String = String.fromCharCode(8364);
  return euro.toFixed(2) + " " + euroSymbol;
}
```

replace() メソッドの２番目のパラメータとして関数を使用する場合、次の引数が関数に渡されます。

- ストリング内の一致する部分。
- グループ化括弧によってキャプチャされる一致。この方法で渡される引数の数は、括弧内のパターンとの一致数によって異なります。括弧内のパターンとの一致数は、関数コード内の arguments.length - 3 を確認して特定することができます。
- ストリング内で一致部分が始まる場所のインデックス位置。
- ストリング全体。

ストリングの大文字と小文字の変換

次の例に示すように、toLowerCase() メソッドと toUpperCase() メソッドは、ストリング内のアルファベット文字を大文字または小文字にそれぞれ変換します。

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

これらのメソッドを実行した後も、元のストリングの内容は変化しません。元のストリング自体を変換するには、次のコードを使用します。

```
str = str.toUpperCase();
```

これらのメソッドは、a ～ z と A ～ Z だけでなく、次のように拡張文字でも動作します。

```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

例 : ASCII アート

ここでは ASCII アートの例を使って、ActionScript 3.0 の String クラスの操作について次のような機能を示します。

- String クラスの `split()` メソッドを使用して、文字区切りのストリング (タブ区切りテキストファイルのイメージ情報) から値を抽出します。
- `split()`、連結、および `substring()` と `substr()` の使用によるストリング部分の抽出など、何とおりものストリング操作テクニックを使用して、イメージタイトルの各単語の先頭文字を大文字にします。
- `getCharAt()` メソッドを使用して、ストリングから 1 文字を取得します (グレースケールビットマップ値に対応する ASCII 文字を決定します)。
- ストリングの連結を使用して、イメージの ASCII アート表現を一度に 1 文字ずつ構築します。

"ASCII アート " という用語はイメージのテキスト表現を指し、イメージは Courier New 文字のような等幅フォント文字のグリッドで描画されます。次のイメージは、アプリケーションが生成した ASCII アートの例を示しています。

ASCII アートのグラフィックを右側に示してあります。

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。ASCIIArt アプリケーションのファイルは、"Samples/AsciiArt" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
AsciiArtApp.mxml または AsciiArtApp.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	テキストファイルからのイメージメタデータの抽出、イメージのロード、イメージからテキストへの変換プロセスの管理など、アプリケーションのメイン機能を提供するクラス。
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	イメージデータをストリングに変換するための `parseBitmapData()` メソッドを提供するクラス。
com/example/programmingas3/asciiArt/Image.as	ロードしたビットマップイメージを表すクラス。
com/example/programmingas3/asciiArt/ImageInfo.as	タイトル、イメージファイル URL など、ASCII アートイメージのメタデータを表すクラス。
image/	アプリケーションが使用するイメージを含むフォルダ。
txt/ImageData.txt	アプリケーションがロードするイメージに関する情報を含む、タブ区切りのテキストファイル。

タブ区切りの値の抽出

この例では、アプリケーションデータをアプリケーション自体とは別に格納する一般的な慣習に従っています。そのようにしておくと、たとえば、別のイメージが追加またはイメージのタイトルが変更されたときなど、データが変更された場合に SWF ファイルを再作成する必要がありません。この場合、イメージタイトル、実際のイメージファイルの URL、およびイメージの操作に使用する一部の値を含むイメージメタデータは、テキストファイル (プロジェクトの txt/ImageData.txt ファイル) に格納されています。実際のテキストファイルの中身は次のようになっています。

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

ファイルは特定のタブ区切り形式を使用します。最初の行はヘッダ行です。残りの行は、ロードするビットマップごとに次のデータを含みます。

- ビットマップのファイル名
- ビットマップの表示名
- ビットマップの白しきい値および黒しきい値。これらは 16 進値で、この値を超えた場合および下回った場合にピクセルは完全に白または黒と見なされます

アプリケーションが起動するとすぐに、AsciiArtBuilder クラスは、AsciiArtBuilder クラスの `parseImageInfo()` メソッドから次のコードを使用して、表示するイメージの " スタック " を作成するためにテキストファイルの内容をロードして解析します。

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
  var imageInfoRaw:String = lines[i];
  ...
  if (imageInfoRaw.length > 0)
  {
    // 新しいイメージ情報レコードを作成し、イメージ情報の配列に追加する
    var imageInfo:ImageInfo = new ImageInfo();

    // 現在の行を（タブ（\t）文字で区切られた）値に分割し、
    // 個々のプロパティを抽出する
    var imageProperties:Array = imageInfoRaw.split("\t");
    imageInfo.fileName = imageProperties[0];
    imageInfo.title = normalizeTitle(imageProperties[1]);
    imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
    imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
    result.push(imageInfo);
  }
}
```

テキストファイルのすべての内容が単一の String インスタンス `_imageInfoLoader.data` プロパティに含まれます。改行 (`"\n"`) をパラメータとして `split()` メソッドを使用すると、String インスタンスは、要素がテキストファイルの各行になる Array (`lines`) に分割されます。次に、ループを使用して各行の操作を行います (実際の内容ではなくヘッダのみを含む先頭行は除く)。ループ内で、`split()` メソッドをもう一度使用して、単一行の内容を値セット (`imageProperties` という Array オブジェクト) に分割します。各行の値はタブ文字で区切られるため、この場合の `split()` メソッドで使用するパラメータはタブ文字 (`"\t"`) です。

ストリングメソッドの使用によるイメージタイトルの正規化

このアプリケーションの設計に関する決定事項の1つは、すべてのイメージタイトルを標準形式で表示し、そのとき各単語の先頭文字を大文字にします (英語タイトルで先頭大文字にしないのが一般的ないくつかの単語は除く)。適切に整形済みのタイトルがテキストファイルに含まれると見なすのではなく、テキストファイルからの抽出時にタイトルを整形します。

以前のコードリストでは、個々のイメージメタデータ値を抽出する一部として、次のコード行が使用されています。

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

このコードでは、テキストファイルからのイメージのタイトルが、ImageInfo オブジェクトに格納される前に normalizeTitle() メソッドを介して渡されます。

```
private function normalizeTitle(title:String):String
{
  var words:Array = title.split(" ");
  var len:uint = words.length;
  for (var i:uint; i < len; i++)
  {
    words[i] = capitalizeFirstLetter(words[i]);
  }

  return words.join(" ");
}
```

このメソッドは split() メソッドを使用して、タイトルを (空白文字で区切られた) 個々の単語に分割し、capitalizeFirstLetter() メソッドを介して各単語を渡し、次に Array クラスの join() メソッドを使用して、単語をもう一度1つのストリングに結合します。

名前が示すとおり、capitalizeFirstLetter() メソッドは実際に各単語の先頭文字を大文字にする働きがあります。

```
/**
 * 以下に示す単語のいずれかでなければ、その先頭文字を大文字にする
 * 除かれるのは、通常は英語で先頭大文字にしない単語である
 */
private function capitalizeFirstLetter(word:String):String
{
  switch (word)
  {
    case "and":
    case "the":
    case "in":
    case "an":
    case "or":
    case "at":
    case "of":
```

```
        case "a":
            // これらの単語には、何も処理を行わない
            break;
        default:
            // 他の単語では、最初の文字を大文字にする
            var firstLetter:String = word.substr(0, 1);
            firstLetter = firstLetter.toUpperCase();
            var otherLetters:String = word.substring(1);
            word = firstLetter + otherLetters;
    }
    return word;
}
```

英語では、タイトルの各単語の先頭の文字は、“and”、“the”、“in”、“an”、“or”、“at”、“of”、または“a” のいずれかの単語である場合に大文字にされません。これは簡易版の規則です。このロジックを実行するために、最初に switch ステートメントを使用して、先頭を大文字にしない単語のいずれかであるかどうかを確認します。先頭大文字にしない単語の場合は、switch ステートメントからジャンプします。一方、先頭大文字にする単語の場合は、次のように何ステップかでそれを行います。

1. substr(0, 1) を使用して、単語の先頭文字を抽出します。この抽出では、第1パラメータ 0 が指定するストリングの先頭文字、つまりインデックス 0 の文字から開始するサブストリングを抽出します。サブストリングは、第 2 パラメータ 1 が指定する 1文字長になります。
2. toUpperCase() メソッドを使用して、その文字を大文字にします。
3. substring(1) を使用して、元の単語の残りの文字を抽出します。この抽出では、インデックス1 (第 2 文字) から開始して、ストリングの末尾 (substring() メソッドの第 2 パラメータの省略が示す) までのサブストリングを抽出します。
4. firstLetter + otherLetters の文字連結を使用して、新しく大文字にした先頭文字と残りの文字を結合することによって、最終的な単語を作成します。

ASCII アートテキストの生成

BitmapToAsciiConverter クラスは、ビットマップイメージを ASCII テキスト表現に変換する機能を提供します。このプロセスは、次に一部を示してある parseBitmapData() メソッドが実行します。

```
var result:String = "";

// ピクセルの行を上から下までループする
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
   // 各行内で、ピクセルを左から右にループする
   for (var x:uint = 0; x < _data.width; x += horizontalResolution)
   {
      ...

      // 範囲 0 ～ 255 の灰色の値を
      // 範囲 0 ～ 64 の値に変換する ( 使用可能な文字セット内の
      // " 灰色の陰影 " の数値であるため )
      index = Math.floor(grayVal / 4);
      result += palette.charAt(index);
   }
   result += "\n";
}
return result;
```

このコードは、ビットマップイメージの ASCII アート版を作成するために使用する result という名前の String インスタンスを最初に定義します。次に、ソースビットマップイメージについて個々のピクセルのループ処理をします。何とおりかのカラー操作テクニックを使用して (ここでは簡略化のために省略)、個々のピクセルの赤、緑、青のカラー値を単一のグレースケール値 (0 から 255 までの数値) に変換します。続いて例に示してあるように、その値を 4 で割り、0 ～ 63 スケールの値に変換します。値は、変数 index に格納されます。0 ～ 63 スケールを使用するのは、このアプリケーションで使用する利用可能な ASCII 文字のパレットに格納される値の数が 64 のためです。文字パレットは、BitmapToAsciiConverter クラスの String インスタンスとして定義されています。

```
// 文字は最も濃いほうから最も薄いほうの順に並ぶ
// ストリングの位置 ( インデックス ) を相対カラー値に対応させるためである
// (0 = 黒 )
private static const palette:String =
   "@#$%&8BMW*mwqpdbkhaoQOOZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

index 変数は、ビットマップイメージの現在のピクセルに対応するパレット内の ASCII 文字を定義するので、この文字は charAt() メソッドを使用して palette ストリングから取得されます。続いて、連結代入演算子 (+=) を使用して、result String インスタンスに追加されます。さらに、各行のピクセルの最後で result String の最後に改行文字を連結して、文字ピクセルの新しい行を作成するために行の折り返しを強制します。

第 7 章

配列の操作

7

配列を使用すると、1 つのデータ構造の中に複数の値を格納できます。配列にはインデックス配列と結合配列の 2 種類があります。値の格納場所を指定する方法として、インデックス配列では単純な整数のインデックスを使用し、結合配列では任意のキーを使用します。また、配列の個々の要素そのものが配列であるような多次元の配列も作成できます。この章では、さまざまな種類の配列を作成および操作する方法について説明します。

目次

配列の基礎 212
インデックス配列 214
結合配列 222
多次元配列 226
配列のコピー 229
高度なトピック 230
Example: PlayList 235

配列の基礎

配列の操作の概要

プログラミングでは単一のオブジェクトではなく一連のアイテムを操作することが必要になる場合がよくあります。たとえば、ミュージックプレイヤーアプリケーションで、再生を待つ曲のリストを使用することが必要になる場合があります。リストの各曲について個別の変数を作成することは避けたいと考えています。このためには、すべての Song オブジェクトをまとめ、グループとして操作できることが求められます。

配列は、曲のリストなど、一連のアイテムのコンテナとして機能するプログラミング要素です。最も一般的な場合、配列内のすべてのアイテムは同じクラスのインスタンスとして使用されますが、これは ActionScript の要件ではありません。配列内の個別のアイテムを配列の " 要素 " と言います。配列は、変数のファイル用引き出しと見なすことができます。変数は、配列の要素として追加できます。これは、ファイル用引き出しにフォルダを挿入することに似ています。いくつかのファイルを引き出しに収めた後、(引き出し全体を別の場所に移動するように) 配列を単一の変数として操作できます。また、(フォルダを 1 つずつめくって情報を検索するように) 変数をグループとして操作したり、(引き出しを開いて単一のフォルダを選択するように) 個別に変数にアクセスできます。

たとえば、ユーザーが複数の曲を選択してそれをプレイリストに追加できるミュージックプレイヤーアプリケーションを作成する場合を考えてください。ActionScript コードで、パラメータとして単一の配列を受け取る `addSongsToPlaylist()` という名前のメソッドを作成することにします。リストに追加する曲の数にかかわらず (少数または多数の場合、あるいは 1 つだけの場合)、`addSongsToPlaylist()` メソッドを 1 回だけ呼び出して、Song オブジェクトを含む配列を渡すだけで済みます。そして `addSongsToPlaylist()` メソッド内で、配列要素 (曲) を 1 つずつループし、実際にプレイリストに追加できます。

最も一般的なタイプの ActionScript 配列は、" インデックス付き配列 " です。この配列では、各アイテムが番号付きスロット (インデックスと呼ばれています) 内に保存され、この番号をアドレスのように使用してアイテムにアクセスできます。Array クラスは、インデックス付き配列を表すために使用します。インデックス付き配列は、大半のプログラミングニーズを満たします。インデックス付き配列の特殊な用途として、多次元配列があります。これは、インデックス付き配列を要素とするインデックス付き配列であり、要素には他の要素が含まれます。もう 1 つのタイプの配列に " 結合配列 " があります。この配列では、個別の要素を識別するために数値インデックスの代わりにストリングの " キー" を使用します。最後に、ActionScript 3.0 には高度なユーザー向けに、" 辞書 " を表す Dictionary クラスも用意されています。これは、要素を区別するキーとして任意のタイプのオブジェクトを使用できる配列です。

一般的な配列タスク

次に、この章で説明する配列操作のための一般的な作業を示します。

- インデックス付き配列の作成
- 配列要素の追加と削除
- 配列要素のソート
- 配列の一部の抽出
- 結合配列と辞書の操作
- 多次元配列の操作
- 配列要素のコピー
- 配列サブクラスの作成

重要な概念と用語

次の参照リストに、この章で使用する重要な用語を示します。

- 配列 : 複数のオブジェクトをまとめてグループ化するためのコンテナとして役立つオブジェクト。
- 結合配列 : 個別の要素を識別するためにストリングキーを使用する配列。
- 辞書 : アイテムがキーおよび値と呼ばれるペアのオブジェクトから構成される配列。キーは、単一の要素を識別するために数値インデックスの代わりに使用します。
- 要素 : 配列内の単一のアイテム。
- インデックス : インデックス付き配列内の単一の要素を識別するための " 番号アドレス "。
- インデックス付き配列 : 各要素を番号付き要素に保存し、個別の要素を識別するために番号 (インデックス) を使用する標準タイプの配列。
- キー : 結合配列または辞書内で単一要素を識別するためのストリングまたはオブジェクト。
- 多次元配列 : 単一の値ではなく配列であるアイテムを含む配列。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。基本的に、この章のすべてのコードリストには、適切な `trace()` 関数呼び出しが含まれています。この章のコードリストをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   `trace()` 関数の結果が [出力] パネルに表示されます。

サンプルコードリストをテストする方法の詳細については、60 ページの「章内のコード例のテスト」を参照してください。

インデックス配列

インデックス配列では、格納した一連の値に、符号なし整数値によるインデックスを指定してアクセスします。配列内の先頭の要素は常にインデックス 0 で示され、それに続く個々の要素は順に 1 ずつ大きい数のインデックスで示されます。次のコードに示すように、インデックス配列を作成するには Array クラスのコンストラクタを呼び出すか、配列リテラルを使用して配列を初期化します。

```
// Array コンストラクタを使用する
var myArray:Array = new Array();
myArray.push("one");
myArray.push("two");
myArray.push("three");
trace(myArray); // 出力 : one,two,three

// Array リテラルを使用する
var myArray:Array = ["one", "two", "three"];
trace(myArray); // 出力 : one,two,three
```

また、Array クラスのプロパティとメソッドを使用すると、インデックス配列に変更を加えることができます。それらのプロパティとメソッドはほとんどがインデックス配列専用であり、結合配列には使用できません。

インデックス配列のインデックスに使用する数値は、符号なし 32 ビット整数です。インデックス配列の最大サイズは 2^{32}- 1(4,294,967,295) です。この最大サイズより大きい配列を作成しようとすると、ランタイムエラーが発生します。

配列の要素には、あらゆるデータ型の値を格納できます。ActionScript 3.0 では " 型付き配列 " の概念がサポートされていないため、配列内のすべての要素が特定の同じデータ型を持つことを指定する方法はありません。

このセクションでは、Array クラスを使用してインデックス配列を作成および変更する方法について、まず配列の作成から順に説明します。配列に変更を加えるメソッドは、要素の挿入に関するもの、要素の削除に関するもの、配列のソートに関するものの 3 つのグループに大別されます。その他に、既存の配列を読み取り専用として扱うメソッドがあり、そのようなメソッドでは配列に対してクエリを実行するだけです。クエリ関連のメソッドは、既存の配列を変更することなく、新しく作成した配列を返します。このセクションの最後には、Array クラスの拡張に関する話題を取り上げます。

配列の作成

Array コンストラクタ関数には 3 つの使用方法があります。第 1 に、パラメータを指定せずに呼び出した場合、コンストラクタは空の配列を返します。Array クラスの length プロパティを使用すると、配列の要素が 1 つもないことを確認できます。たとえば次のコードは、Array コンストラクタを引数なしで呼び出しています。

```
var names:Array = new Array();
trace(names.length); // 出力 : 0
```

第 2 に、1 個の数値パラメータだけを指定した場合、Array コンストラクタではそのパラメータで指定された長さの配列を作成し、各要素の値を undefined に設定します。指定する引数は、0 ～ 4,294,967,295 の値を持つ符号なし整数である必要があります。たとえば次のコードは、1 個の数値引数を指定して Array コンストラクタを呼び出しています。

```
var names:Array = new Array(3);
trace(names.length); // 出力 : 3
trace(names[0]); // 出力 : undefined
trace(names[1]); // 出力 : undefined
trace(names[2]); // 出力 : undefined
```

第 3 に、パラメータとして要素リストを指定した場合、コンストラクタでは、各パラメータの値を個々の要素に設定した配列を作成します。次のコードでは、Array コンストラクタに対し 3 つの引数を指定しています。

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // 出力 : 3
trace(names[0]); // 出力 : John
trace(names[1]); // 出力 : Jane
trace(names[2]); // 出力 : David
```

また、配列リテラルまたはオブジェクトリテラルから配列を作成することもできます。配列リテラルは、次の例に示すように配列変数に直接割り当てることができます。

```
var names:Array = ["John", "Jane", "David"];
```

配列要素の挿入

配列に要素を挿入する場合は、Array クラスにある -push()、-unshift()、-splice() の 3 つのメソッドを使用できます。push() メソッドでは、配列の末尾に 1 つまたは複数の要素を追加します。したがって、push() メソッドで挿入した最後の要素には最も大きいインデックス番号が付きます。unshift() メソッドでは、配列の先頭に 1 つまたは複数の要素を挿入します。したがって、挿入した要素には必ずインデックス番号 0 が付きます。splice() メソッドでは、配列内の指定されたインデックスに任意の数の要素を挿入します。

これら 3 つのメソッドについて使用例を次に示します。この例では、planets という名前の配列を作成し、惑星の名前を太陽から近い順に並ぶように格納します。まず push() メソッドを呼び出し、最初の要素として Mars を挿入します。次に、配列の先頭に位置する要素 Mercury を unshift() メソッドで挿入します。最後に splice() メソッドで、Venus および Earth を Mercury より後、Mars より前に挿入します。splice() の第 1 パラメータとして渡している整数 1 は、挿入先をインデックス 1 の場所にするという指定です。また、splice() 第 2 パラメータの整数 0 は、既存の要素を 1 つも削除しないという指定です。さらに、splice() の第 3 および第 4 パラメータで指定している Venus と Earth は挿入する要素のリストです。

```
var planets:Array = new Array();
planets.push("Mars");       // 配列の内容 : Mars
planets.unshift("Mercury"); // 配列の内容 : Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets);             // 配列の内容 : Mercury,Venus,Earth,Mars
```

push() および unshift() メソッドはいずれも、変更後の配列の長さを示す符号なし整数値を返します。splice() は要素の挿入に使用した場合、空の配列を返します。これは奇異な動作のようにも見えますが、splice() メソッドの幅広い用途を考えると理にかなっています。splice() メソッドは配列に要素を挿入する場合だけでなく、配列から要素を削除する場合にも使用でき、削除の場合は、元の配列から削除した要素を格納した配列を返します。

配列要素の削除

配列から要素を削除する場合は、Array クラスにある pop()、shift()、splice() の 3 つのメソッドを使用できます。pop() メソッドでは、配列の末尾にある 1 個の要素、つまり最も大きいインデックス番号の付いた要素を削除します。shift() メソッドでは、配列の先頭にある 1 個の要素、つまりインデックス番号が 0 である要素を削除します。splice() メソッドは要素の挿入にも使用できますが、第 1 引数で指定したインデックス番号が示す位置から任意の数の要素を削除する場合にも使用できます。

これら 3 つのメソッドを使用して配列の要素を削除する例を次に示します。この例では、広い海の名前を格納するために oceans という名前の配列を作成していますが、海ではなく湖の名前がいくつか含まれているため、それらを削除します。

まず、splice() メソッドを使用して Aral および Superior の要素を削除し、Atlantic および Indian の要素を挿入します。splice() の第 1 パラメータとして渡している整数 2 は、3 番目の要素 (インデックス 2) から始まる範囲を操作対象にするという指定です。また、第 2 パラメータの整数 2 は、既存の要素を 2 つ削除するという指定です。残りのパラメータで指定している Atlantic と Indian はインデックス 2 の位置に挿入する要素のリストです。

次に、配列の末尾に位置する要素 Huron を pop() メソッドで削除します。最後に shift() メソッドで、配列の先頭に位置する要素 Victoria を削除します。

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian", "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // Aral および Superior を置換する
oceans.pop();   // Huron を削除する
oceans.shift(); // Victoria を削除する
trace(oceans);  // 出力 : Pacific,Arctic,Atlantic,Indian
```

pop() および shift() メソッドはいずれも、削除した要素を返します。配列にはあらゆるデータ型の値を格納できるため、戻り値のデータ型は Object になっています。splice() は、削除した要素を格納した配列を返します。oceans 配列の例を次のように改変すると、splice() の呼び出しで返される戻り値を新しい配列変数に割り当てることができます。

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // 出力 : Aral,Superior
```

配列の要素に対して delete 演算子を使用しているコードもありますが、delete 演算子はその要素の値を undefined に設定するだけであり、配列から要素を削除するわけではありません。たとえば次のコードでは、oceans 配列の 3 番目の要素に対して delete 演算子を使用していますが、配列の要素数はその後も変わらず 5 のままです。

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);        // 出力 : Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]);     // 出力 : undefined
trace(oceans.length); // 出力 : 5
```

配列の要素数を切り詰めるには、length プロパティを使用する方法があります。length プロパティをその配列の現状の長さより小さく設定すると配列を切り詰めることになり、変更後の length の値から 1 を引いた値よりも大きいインデックス番号を持つ要素はすべて削除されます。たとえば、oceans 配列の要素を並べ替え、残したい要素をすべて配列の先頭側に集めたとすると、次のように length プロパティを使用することで末尾側の不要な要素を削除できます。

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // 出力 : Arctic,Pacific
```

配列のソート

配列要素の並び順を変更する場合は、ソートまたは順序の逆転を行う reverse()、sort()、sortOn() の 3 つのメソッドを使用できます。いずれのメソッドでも既存の配列が変化します。reverse() メソッドでは、配列要素の並び順を逆転します。つまり、末尾にあった要素は先頭に、末尾の直前にあった要素は 2 番目に移動します。sort() メソッドでは、さまざまな定義済みの方法で配列要素をソートできる他、独自のソートアルゴリズムを作成して使用することもできます。sortOn() メソッドでは、ソート用キーとして使用可能な複数の共通プロパティを持つオブジェクトのインデックス配列をソートできます。

reverse() メソッドは、パラメータも戻り値もありませんが、既存の配列について現在の状態と、その並び順を逆転した状態とを切り替えるのに使用できます。次の例では、oceans 配列に格納した海の名前を元とは逆の順序に並べ替えます。

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // 出力 : Pacific,Indian,Atlantic,Arctic
```

sort() メソッドはデフォルトのソート順に基づいて配列要素を並べ替えます。このデフォルトのソート順には、次の特性があります。

- 大文字と小文字は区別され、大文字のほうが小文字より先に並びます。たとえば、D は b よりも先に並びます。
- ソート方向は正順です。つまり、文字コードの値の小さい文字 (例 : A) のほうが、文字コードの値の大きい文字 (例 : B) よりも先に並びます。
- 値が同等と見なされる要素は互いに隣接して並びますが、それらの間における並び順は決まっていません。
- ソートはストリングベースで実行されます。つまり、要素は文字列に変換してから比較されます。たとえば、数値の 10 は 3 よりも先に並びます。これは、ストリングの "1" が持つ文字コードのほうがストリングの "3" が持つ文字コードよりも小さいためです。

大文字と小文字を区別しないソート、逆順のソート、またはアルファベット順でなく数値の大きさ順によるソートなどが必要な場合のために、sort() メソッドには、デフォルトのソート順に定められた各種の規則を変更する options パラメータがあります。こうしたオプションは、次に示す Array クラスの静的定数セットによって定義されます。

- Array.CASEINSENSITIVE: このオプションにより、ソートで大文字と小文字が無視されます。これにより、たとえば小文字の b が大文字の D よりも先に並ぶようになります。
- Array.DESCENDING: デフォルトの正順ではなく逆順でソートします。たとえば、B が A よりも先に並ぶようになります。
- Array.UNIQUESORT: 同じ値を持つ複数の要素が見つかった場合にソートを中止するようにします。
- Array.NUMERIC: 数値順でソートします。たとえば、3 が 10 よりも先に並ぶようになります。

これらのオプションの一部を使用した例を次に示します。poets という名前の配列を作成し、いろいろなオプションを使用してソートを実行しています。

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // デフォルトのソート
trace(poets); // 出力 : Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // 出力 : Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // 出力 : cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // 2 つのオプションを併用する
trace(poets); // 出力 : Dante,cummings,Blake,Angelou
```

また、独自のソート関数を作成して sort() メソッドのパラメータに指定することもできます。たとえば、各人のフルネームを要素として格納した人名リストがあるとします。このリストを姓を基準にして並べ替える場合は、各要素を解析して姓の部分を比較するソート関数を独自に作成する必要があります。これを実現するコード例を、次に示します。この例では、カスタム関数を Array.sort() メソッドのパラメータとして使用しています。

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
  var lastName:RegExp = /\b\S+$/;
  var name1 = a.match(lastName);
  var name2 = b.match(lastName);
  if (name1 < name2)
  {
    return -1;
  }
  else if (name1 > name2)
  {
    return 1;
  }
  else
  {
    return 0;
  }
}

trace(names); // 出力 : John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // 出力 : Jane Doe,Mike Jones,John Q. Smith
```

独自のソート関数 orderLastName() では、正規表現を使用して各要素から姓の部分を抽出し、それを比較演算に使用します。names 配列に対する sort() メソッド呼び出しのパラメータとして、この関数の識別子である orderLastName だけを指定しています。ソート関数に a および b の 2 つのパラメータが渡されるのは、一度に 2 つの配列要素が処理の対象となるからです。ソート関数の戻り値は 2 つの要素の並び順を示すものであり、次の意味があります。

- 戻り値が -1 の場合、第 1 パラメータの a を第 2 パラメータの b よりも先に並べることを示します。
- 戻り値が 1 の場合、第 2 パラメータの b を第 1 パラメータの a よりも先に並べることを示します。
- 戻り値が 0 の場合、2 つの要素がソートにおいて同じ順位を持つことを示します。

sortOn() メソッドは、要素がオブジェクトであるインデックス配列のためにあります。すべての要素のオブジェクトには、ソート用キーとなる共通のプロパティが少なくとも 1 つ含まれている必要があります。これらの条件に合わない配列に対して sortOn() メソッドを使用した場合の結果は予測不能です。

次の例は、poets 配列をストリング要素ではなくオブジェクト要素の配列に改変したものです。各オブジェクトには、詩人の姓と生誕年が格納されています。

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

この配列では、born プロパティを基準として sortOn() メソッドによるソートを実行できます。sortOn() メソッドには、fieldName および options の 2 つのパラメータがあります。fieldName にはストリングを指定する必要があります。次の例では、sortOn() にパラメータとして "born" および Array.NUMERIC の 2 つを渡しています。Array.NUMERIC オプションを指定しているのは、アルファベット順ではなく数値順で確実にソートを実行するためです。たとえ各要素に格納されている数値の桁数がすべて同じであっても、必ずこのオプションを指定するようにしておけば、後で桁数の異なる要素が追加されたとしても正常に動作します。

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
  trace(poets[i].name, poets[i].born);
}
/* 出力 :
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

一般に、sort() および sortOn() メソッドを使用すると元の配列が変化します。既存の配列を変更せずにソートを実行する場合は、options パラメータに Array.RETURNINDEXEDARRAY 定数を含めます。このオプションを指定した場合、メソッドの戻り値はソート結果を示す新しい配列となり、元の配列の内容はそのまま維持されます。返される配列はソート後の並び順を示すインデックス番号だけを格納した単純なものであり、元の配列内の要素を格納した配列ではありません。たとえば、poets 配列について元の状態を維持したまま生誕年でのソートを実行するには、options パラメータに Array.RETURNINDEXEDARRAY 定数を含めます。

次の例では、戻り値のインデックス情報を indices という配列に格納し、この indices と元のままの poets を使用して詩人のリストを生誕年順に表示しています。

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
  var index:int = indices[i];
  trace(poets[index].name, poets[index].born);
}
/* 出力 :
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

配列のクエリ

以上で説明した他に、Array クラスには concat()、join()、slice()、toString() の 4 つのメソッドがあります。これらはいずれもクエリによって配列から情報を取得するものであり、配列に変更を加えることはありません。concat() および slice() メソッドは新しい配列を返し、join() および toString() メソッドはストリングを返します。concat() メソッドのパラメータには配列または要素リストを指定します。既存の配列と、パラメータで指定した別の配列または要素とを連結し、新しい配列を作成して返します。slice() メソッドには、startIndex および endIndex という 2 つのパラメータがあります。元の配列から、これらのパラメータで指定したインデックスの範囲を切り出し、その部分のコピーを格納した新しい配列を返します。ただし、返される範囲は startIndex が指す要素から、endIndex が指す要素の直前までであり、endIndex の位置にある要素は含まれません。

次の例では、既存の配列から concat() および slice() を使用して新しい配列を作成します。

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // 出力 : alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // 出力 : alpha,beta,alpha,beta,gamma,delta
```

```
var array4:Array = array3.slice(2,5);
trace(array4); // 出力 : alpha,beta,gamma
```

join() および toString() メソッドでは、配列の内容をストリングとして取得できます。join() メソッドのパラメータを指定しない場合、これら 2 つのメソッドはいずれも、すべての配列要素をカンマで区切った内容のストリングを返します。join() メソッドには、toString() メソッドにない delimiter というパラメータがあり、戻り値のストリングで要素間に使用する区切り文字を指定できます。

次の例では rivers という配列を作成し、join() および toString() を呼び出すことで、配列の内容をストリングとして取得します。toString() メソッドはカンマ区切りの値 (riverCSV) を得るため、また、join() メソッドは + で区切った値を得るために使用しています。

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // 出力 : Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // 出力 : Nile+Amazon+Yangtze+Mississippi
```

join() メソッドを使用する場合には、メインの配列要素に対して指定する区切り文字に関係なく、次に示す例のように、ネストされた配列には常にカンマ区切りの値が返されることに注意する必要があります。

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // 出力 : a+b,c,d+e
```

結合配列

結合配列は " ハッシュ " または " マップ " とも呼ばれ、値の格納場所を数値のインデックスではなく " キー " によって指定する配列です。結合配列に使用する個々のキーは、それぞれに対応する特定の格納値にアクセスするための一意のストリングです。結合配列は Object クラスのインスタンスであり、したがって、各キーはそれぞれ 1 つのプロパティ名に対応します。結合配列は、キー / 値ペアの集合からなる、ソートされていないコレクションです。結合配列内のキーに何らかの決まった順序があるかのような前提でコードを記述してはなりません。

ActionScript 3.0 では、新たに " ディクショナリ " と呼ばれる高機能な結合配列を使用できるようになりました。ディクショナリは flash.utils パッケージに属する Dictionary クラスのインスタンスであり、任意のデータ型 (通常は Object クラスのインスタンス) をキーとして使用できます。つまり、ディクショナリのキーは String 型の値である必要はありません。

このセクションでは、ストリングのキーを使用した結合配列の作成方法、および Dictionary クラスの使用方法について説明します。

ストリングキーを使用した結合配列

ActionScript 3.0 の結合配列を作成するには 2 つの方法があります。1 つは Object コンストラクタを使用する方法で、これには、オブジェクトリテラルを使用して配列を初期化できるというメリットがあります。Object クラスのインスタンス (汎用オブジェクトとも呼ばれる) は、機能的には結合配列と同じです。汎用オブジェクトの個々のプロパティ名が、格納した個々の値にアクセスするためのキーとして機能します。

次の例では、monitorInfo という名前の結合配列を作成します。オブジェクトリテラルを使用して、2 つのキーと値のペアが指定された配列を初期化します。

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// 出力 : Flat Panel 1600 x 1200
```

宣言時に配列を初期化する必要がある場合は、次のように、Object コンストラクタを使って配列を作成できます。

```
var monitorInfo:Object = new Object();
```

オブジェクトリテラルまたは Object クラスのコンストラクタで配列を作成した後は、角括弧演算子 ([]) またはドット演算子 (.) を使用して新しい値を追加できます。次の例では、monitorArray に新しい値を 2 つ追加します。

```
monitorInfo["aspect ratio"] = "16:10"; // この記述は誤り。空白を使用しないこと
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// 出力 : 16:10 16.7 million
```

この例で、aspect ratio というキーの名前には空白文字が含まれています。角括弧演算子ではこのような名前も使用できますが、ドット演算子で使用するとエラーになります。空白文字を含んだキー名を使用することはお勧めしません。

結合配列を作成するもう 1 つの方法は、Array コンストラクタを使用し、それから角括弧演算子 ([]) またはドット演算子 (.) で配列にキー / 値ペアを追加する方法です。結合配列を Array 型として宣言した場合、オブジェクトリテラルで配列を初期化することはできません。次の例では、monitorInfo という名前の結合配列を Array コンストラクタで作成し、type および resolution という 2 つのキーとそれぞれに対応する値を追加します。

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// 出力 :  Flat Panel 1600 x 1200
```

結合配列の作成に Array コンストラクタを使用するメリットはありません。たとえ Array コンストラクタや Array データ型を使用して宣言した結合配列であっても、Array.length プロパティや Array クラスのメソッドはまったく使用できないからです。Array コンストラクタはインデックス配列を作成する場合に限って使用するようお勧めします。

オブジェクトのキーを使用した結合配列

Dictionary クラスを使用すると、ストリングではなくオブジェクトをキーとする結合配列を作成できます。そのような配列は、ディクショナリ、ハッシュ、またはマップと呼ばれることがあります。たとえば、特定のコンテナとの位置関係によって Sprite オブジェクトの位置が決まるようなアプリケーションでは、個々の Sprite オブジェクトをコンテナへマップする場合に Dictionary オブジェクトを使用できます。

次のコードでは、Dictionary オブジェクトのキーとして使用する Sprite クラスのインスタンスを 3 つ作成します。各キーに対しては、GroupA または GroupB いずれかの値を格納します。格納する値には任意のデータ型を使用できますが、この例では GroupA、GroupB のいずれも Object クラスのインスタンスです。次に、プロパティアクセス演算子 ([]) を使用して、各キーに対応する値にアクセスします。

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// キーとして使用するオブジェクト
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// 値として使用するオブジェクト
var groupA:Object = new Object();
var groupB:Object = new Object();

// 新しいキーと値のペアをディクショナリ内に作成する
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
  trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
  trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
  trace("spr3 is in groupB");
}
```

オブジェクトのキーを使用した反復処理

Dictionary オブジェクトの内容に対して反復処理を実行する場合は、for..in ループまたは for each..in ループを使用できます。for..in ループはキーに基づいた反復処理に使用し、for each..in loop ループは各キーに対応する値に基づいた反復処理に使用します。

for..in ループの場合、Dictionary オブジェクト内のオブジェクトキーに直接アクセスできます。また、プロパティアクセス演算子 ([]) を使用すれば Dictionary オブジェクト内の値にアクセスできます。次のコードでは、前の例と同じ groupMap ディクショナリを使用して、for..in ループで Dictionary オブジェクトの内容を反復処理する方法を示します。

```
for (var key:Object in groupMap)
{
  trace(key, groupMap[key]);
}
/* 出力 :
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

for each..in ループの場合、Dictionary オブジェクト内の値に直接アクセスできます。次のコードでもやはり groupMap ディクショナリを使用して、for each..in ループで Dictionary オブジェクトの内容を反復処理する方法を示します。

```
for each (var item:Object in groupMap)
{
  trace(item);
}
/* 出力 :
[object Object]
[object Object]
[object Object]
*/
```

オブジェクトキーとメモリ管理

Adobe Flash Player では、使用されなくなったメモリはガベージコレクションのシステムによって開放されます。オブジェクトはどこからも参照されなくなった時点でガベージコレクションの対象とされ、次回ガベージコレクションが実行されるとき、そのオブジェクトに使用されていたメモリが解放されます。たとえば次のコードでは、新しいオブジェクトを作成し、そのオブジェクトへの参照を myObject という変数に割り当てています。

```
var myObject:Object = new Object();
```

オブジェクトに対して何らかの参照が存在する間は、そのオブジェクトが使用しているメモリがガベージコレクションのシステムによって解放されることはありません。myObject の値が別のオブジェクトを参照するように変更されるか、または null に設定されると、それまで参照先となっていたオブジェクトはガベージコレクションの対象とされます (そのオブジェクトに対する参照が他に存在しない場合)。

この myObject を Dictionary オブジェクトのキーとして使用すると、参照先オブジェクトに対する参照がもう1つ作成されます。たとえば次のコードでは、1つのオブジェクトに対して2つの参照、つまり myObject 変数と、myMap オブジェクトのキーを作成しています。

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

myObject で参照しているオブジェクトをガベージコレクションに対して有効にするには、そのオブジェクトに対する参照をすべて削除する必要があります。この場合、そのためには myObject の値を変更するのに加え、myMap から myObject のキーを削除します。

```
myObject = null;
delete myMap[myObject];
```

別の方法として、Dictionary コンストラクタに対して useWeakReference パラメータを指定し、すべてのディクショナリキーを " 弱参照 " にすることもできます。弱参照はガベージコレクションのシステムで無視されるため、弱参照による参照しか受けていないオブジェクトはガベージコレクションの対象となります。たとえば次のコードでは、myMap 内の myObject キーを削除しなくても、該当するオブジェクトをガベージコレクションの対象とすることができます。

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // オブジェクトをガベージコレクションの対象とする
```

多次元配列

多次元配列には、配列の要素としてさらに配列が格納されています。たとえば、ストリングのインデックス配列として格納される予定リストがあるとします。

```
var tasks:Array = ["wash dishes", "take out trash"];
```

曜日ごとに異なる予定リストを格納する必要がある場合は、各曜日を要素として持つ多次元配列を作成できます。それぞれの要素として、予定リストを表す tasks 配列と同様のインデックス配列を格納します。多次元配列内では、インデックス配列と結合配列を任意に組み合わせて使用できます。以降のセクションでは、2つのインデックス配列を使用する例と、結合配列およびインデックス配列を1つずつ使用する例を示します。また、実習として、考えられる残りの組み合わせについても作成してみてください。

2 つのインデックス配列を使用する例

インデックス配列を 2 つ使用した場合の結果は、テーブルまたはスプレッドシートのような図で表現できます。第 1 の配列に含まれる要素がテーブルの行を表し、第 2 の配列に含まれる要素が列を表します。

たとえば、次の多次元配列には、曜日ごとの予定リストを 2 つのインデックス配列として格納しています。第 1 の配列 masterTaskList は、Array クラスのコンストラクタを使用して作成します。この配列の各要素は曜日に対応し、インデックス 0 が月曜、インデックス 6 が日曜を表します。それらの要素は、テーブルの行と見なすことができます。各曜日の予定リストは、masterTaskList 配列内にある 7 つの要素それぞれに配列リテラルを割り当てることで作成できます。これらの配列リテラルは、テーブルの列を表します。

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

いずれかの予定リストに含まれる個別の予定項目にアクセスするには、角括弧表記法を使用します。最初の角括弧で曜日を指定し、2 番目の角括弧でその日の予定リスト内の項目を指定します。たとえば、水曜日の予定リストから 2 番目の予定を取得するには、水曜日を表すインデックス 2 と、予定リストの 2 番目の項目を表すインデックス 1 を指定します。

```
trace(masterTaskList[2][1]); // 出力 : dentist
```

日曜日の予定リストから 1 番目の予定を取得するには、日曜日を表すインデックス 6 と、予定リストの 1 番目の項目を表すインデックス 0 を指定します。

```
trace(masterTaskList[6][0]); // 出力 : mow lawn
```

結合配列の下位にインデックス配列を組み合わせる例

曜日について結合配列を使用し、予定リストについてインデックス配列を使用すると、個別の配列に対するアクセスが容易になります。結合配列では、特定の曜日を参照するときにドットシンタックスを使用することができますが、結合配列の各要素にアクセスするために、実行時の処理時間が余分に犠牲になります。次の例では、曜日ごとにキー / 値ペアを指定し、予定リストの基礎として結合配列を使用しています。

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

ドットシンタックスを使用すると、角括弧の繰り返しを避けることができるため、コードが読みやすくなります。

```
trace(masterTaskList.Wednesday[1]); // 出力 : dentist
trace(masterTaskList.Sunday[0]);    // 出力 : mow lawn
```

for..in ループを使用して予定リストの反復処理を実行できますが、各キーに対応する値にアクセスするために、ドットシンタックスではなく角括弧表記法を使用する必要があります。masterTaskList は結合配列であるため、次の例に示すように、必ずしも期待する順序で要素を取得する必要はありません。

```
for (var day:String in masterTaskList)
{
  trace(day + ": " + masterTaskList[day])
}
/* output:
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

配列のコピー

Array クラスには、配列のコピーを作成する組み込みメソッドは用意されていません。配列の " 浅いコピー " は、concat() または slice() メソッドをパラメータなしで呼び出すことで作成できます。浅いコピーでは、オブジェクトである要素が元の配列に格納されている場合、コピーした配列内の要素は元の要素に格納されているオブジェクトへの参照となり、参照先オブジェクトはコピーされません。したがって、元の配列とコピーした配列が同一のオブジェクトを参照することになり、オブジェクトが変更されると、両方の配列に対してその変更が反映されます。

" 深いコピー " とは、元の配列内で参照されているオブジェクトをすべてコピーすることです。その場合、元の配列とコピーした配列が同一のオブジェクトを参照する結果にはなりません。深いコピーを実行するには複数行のコードを記述する必要があり、通常、何らかの関数を作成することになります。こうした関数は、グローバルなユーティリティ関数として作成するか、または Array のサブクラスにメソッドとして作成することができます。

次の例では、深いコピーを実行するための clone() という関数を定義しています。採用したアルゴリズムは、Java プログラミングで一般的に使用されているテクニックです。まず元の配列を直列化して ByteArray クラスのインスタンスを作成し、それを新しい配列に読み込むことでコピーを実行しています。次のコードに示されているとおり、この関数はオブジェクトを受け付けるため、インデックス配列と結合配列の両方で使用できます。

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
   var myBA:ByteArray = new ByteArray();
   myBA.writeObject(source);
   myBA.position = 0;
   return(myBA.readObject());
}
```

高度なトピック

Array クラスの拡張

Array クラスはコアクラスとしては珍しく final 指定されていないため、Array のサブクラスを独自に定義できます。このセクションでは、Array のサブクラスを作成する方法について例を示し、その際に発生すると考えられる問題について説明します。

前述したとおり ActionScript の配列には型がありませんが、Array のサブクラスを定義すれば、特定のデータ型の要素だけを格納する配列を作成できるようになります。以下のセクションの例では、Array のサブクラスとして、第 1 パラメータに指定されているデータ型の要素だけを格納する TypedArray というクラスを定義します。TypedArray クラスは、Array クラスの拡張方法の例として示したにすぎず、いくつかの理由で運用目的には適さない場合があります。第 1 に、コンパイル時ではなく、実行時に型チェックが行われます。第 2 に、メソッドは簡単な変更によって例外をスローする可能性がありますが、TypedArray メソッドで不一致が見つかった場合に、その不一致は無視されて例外がスローされます。第 3 に、このクラスでは、配列に任意の型の要素を挿入する配列アクセス演算子の使用を防止できません。第 4 に、コーディングスタイルでは、パフォーマンスの最適化よりも単純さのほうが優先されます。

サブクラスの宣言

定義するクラスが Array のサブクラスであることを示すには、`extends` キーワードを使用します。Array のサブクラスでは、Array と同様に `dynamic` 属性を使用します。これに従わないとサブクラスが正常に機能しません。

次のコードに示す TypedArray クラスの定義には、データ型を保持する定数、コンストラクタメソッド、配列要素の追加に使用できる 4 つのメソッドがあります。ここでは各メソッドのコードを省略していますが、それらの詳細については以降のセクションで説明していきます。

```
public dynamic class TypedArray extends Array
{
  private const dataType:Class;

  public function TypedArray(...args) {}

  AS3 override function concat(...args):Array {}

  AS3 override function push(...args):uint {}

  AS3 override function splice(...args) {}

  AS3 override function unshift(...args):uint {}
}
```

4 つのオーバーライドしたメソッドはすべてが AS3 名前空間を使用し、public 属性は使用していません。この例では、コンパイラオプション -as3 の設定は true、コンパイラオプション -es の設定は false と仮定しているためです。これらの設定は、Adobe Flex Builder 2 および Adobe Flash CS3 Professional のデフォルト設定です。詳細については、168 ページの「AS3 名前空間」を参照してください。

ヒント	上級開発者でプロトタイプ継承の使用を優先したい場合は、TypedArray クラスに 2 つのマイナー変更を加え、コンパイラオプション -es を true に設定してコンパイルします。最初に、すべての出現箇所の override 属性を削除し、AS3 名前空間を public 属性で置換します。次に、出現するすべての 4 つの super の代わりに Array.prototype を使用します。

TypedArray コンストラクタ

このコンストラクタを定義するにあたっては興味深い問題が 1 つあります。それは、任意の長さを持つリストをパラメータとして受け付け、さらに、配列を作成するためにパラメータをスーパーコンストラクタに引き渡す必要があるということです。パラメータのリストを配列として渡せば、スーパーコンストラクタには Array 型のパラメータ 1 個として扱われてしまい、結果として配列の要素数は常に 1 になります。パラメータリストをそのまま引き渡す場合の伝統的な処理方法として Function.apply() メソッドがあります。パラメータ配列をこのメソッドの第 2 パラメータに指定すると、その要素を個々のパラメータに展開して関数を実行させることができます。しかし、Function.apply() メソッドはコンストラクタ内では使用できません。

残る唯一の選択肢は、TypedArray コンストラクタ内に Array コンストラクタと同様のロジックを再現することです。次のコードは、Array クラスのコンストラクタで使用されているアルゴリズムを示しています。これは、Array のサブクラスのコンストラクタでも使用できます。

```
public dynamic class Array
{
  public function Array(...args)
  {
    var n:uint = args.length
    if (n == 1 && (args[0] is Number))
    {
      var dlen:Number = args[0];
      var ulen:uint = dlen;
      if (ulen != dlen)
      {
        throw new RangeError("Array index is not a 32-bit unsigned integer
  ("+dlen+")");
      }
      length = ulen;
    }
    else
    {
      length = n;
      for (var i:int=0; i < n; i++)
```

```
        {
          this[i] = args[i]
        }
      }
    }
}
```

TypedArray コンストラクタは、そのほとんどが Array コンストラクタから流用したコードで構成され、変更点は 4 つだけです。1 つ目は、パラメータリストに Class 型の新しい必須パラメータを含め、配列のデータ型を指定できるようにしたことです。2 つ目は、コンストラクタに渡されるデータ型を dataType 変数に割り当てたことです。3 つ目は、else ステートメントにおいて、length プロパティの値は for ループの実行後に設定し、適切な型のパラメータだけが length に含まれるようにしたことです。4 つ目は、オーバーライドした push() メソッドを for ループの本体の中で使用し、適切なデータ型のパラメータだけを配列に追加するようにしたことです。次の例は、TypedArray コンストラクタ関数を示しています。

```
public dynamic class TypedArray extends Array
{
  private var dataType:Class;
  public function TypedArray(typeParam:Class, ...args)
  {
    dataType = typeParam;
    var n:uint = args.length
    if (n == 1 && (args[0] is Number))
    {
      var dlen:Number = args[0];
      var ulen:uint = dlen
      if (ulen != dlen)
      {
        throw new RangeError("Array index is not a 32-bit unsigned integer
  ("+dlen+")")
      }
      length = ulen;
    }
    else
    {
      for (var i:int=0; i < n; i++)
      {
        // 型チェックは push() により実行される
        this.push(args[i])
      }
      length = this.length;
    }
  }
}
```

TypedArray でオーバーライドしたメソッド

TypedArray クラスでは、Array クラスのメソッドのうち、配列要素の追加に使用できる 4 つのメソッドをオーバーライドしています。いずれのメソッドでも、適切なデータ型以外の要素が追加されるのを防ぐために型チェックを実行してから、スーパークラスにある同じメソッドを呼び出します。

push() メソッドでは、パラメータリストに含まれる個々の要素の型を for..in ループで順にチェックします。不適切な型のパラメータがあれば、splice() メソッドで args 配列から削除します。この for..in ループを終了した後は、args には dataType 型の要素しか含まれていないことになります。それから、次のようにスーパークラスの push() に改変後の args 配列を渡して呼び出します。

```
AS3 override function push(...args):uint
{
   for (var i:* in args)
   {
      if (!(args[i] is dataType))
      {
         args.splice(i,1);
      }
   }
   return (super.push.apply(this, args));
}
```

concat() メソッドでは、型チェックに適合した引数を格納するために、passArgs という名前で一時的な TypedArray を作成します。これは、push() メソッド内の型チェックのコードを再利用するためです。for..in ループで、args 配列内の個々の要素について push() を呼び出します。passArgs の型は TypedArray なので、実行される push() メソッドは TypedArray クラスのものです。それから、次のようにスーパークラスの concat() メソッドを呼び出します。

```
AS3 override function concat(...args):Array
{
   var passArgs:TypedArray = new TypedArray(dataType);
   for (var i:* in args)
   {
      // 型チェックは push() により実行される
      passArgs.push(args[i]);
   }
   return (super.concat.apply(this, passArgs));
}
```

splice() メソッドには任意の数の引数を指定できますが、最初の 2 つは必ず、インデックス番号と削除する要素の数を表す引数になります。このため、オーバーライドした splice() メソッドでは、args のインデックス番号が 2 以降の配列要素に対してのみ型チェックを実行します。このコードでは、for ループ内で splice() を再帰的に呼び出しているように見えますが、実際には再帰ではありません。args の型は TypedArray ではなく Array なので、args.splice() の呼び出しで実行されるのはスーパークラスのメソッドです。この for..in ループを終了した後は、args にはインデックス番号が 2 以上の適切な型の要素しか含まれていません。それから、次のようにスーパークラスの splice() を呼び出します。

```
AS3 override function splice(...args):*
{
  if (args.length > 2)
  {
    for (var i:int=2; i< args.length; i++)
    {
      if (!(args[i] is dataType))
      {
        args.splice(i,1);
      }
    }
  }
  return (super.splice.apply(this, args));
}
```

配列の先頭に要素を挿入する unshift() も、任意の数の引数リストを受け付けるメソッドです。オーバーライドした unshift() メソッドでは、次のコードに示すとおり、push() メソッドと非常によく似たアルゴリズムを使用しています。

```
AS3 override function unshift(...args):uint
{
  for (var i:* in args)
  {
    if (!(args[i] is dataType))
    {
      args.splice(i,1);
    }
  }
  return (super.unshift.apply(this, args));
}
}
```

Example: PlayList

ここでは PlayList の例を使って、配列を操作する手法について、曲のリストを管理する音楽プレイリストアプリケーションのコンテキストで示します。それらの手法は、次のとおりです。

- インデックス配列の作成
- インデックス配列への項目の追加
- 各種ソートオプションを使用したオブジェクト配列のプロパティ別ソート
- 文字で区切られたストリングへの配列の変換

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。PlayList アプリケーションのファイルは、"Samples/PlayList" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
PlayList.mxml または PlayList.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/playlist/Song.as	単一の曲に関する情報を表す値オブジェクト。PlayList クラスが管理する項目は Song インスタンスです。
com/example/programmingas3/playlist/SortProperty.as	利用可能な値が Song クラスのプロパティを表し、このプロパティで Song オブジェクトのリストをソート可能な擬似列挙。

PlayList クラスの概要

PlayList クラスは Song オブジェクトセットを管理するクラスで、曲をプレイリストに追加するための `addSong()` メソッドや、リストの曲をソートするための `sortList()` メソッドなど、さまざまな機能のパブリックメソッドを備えています。このクラスには、読み取り専用アクセサプロパティの `songList` もあり、これによってプレイリストの実際の曲セットにアクセスできます。内部的には、PlayList クラスは、Array プライベート変数を使用して曲を追跡し続けます。

```
public class PlayList
{
  private var _songs:Array;
  private var _currentSort:SortProperty = null;
  private var _needToSort:Boolean = false;
  ...
}
```

PlayList クラスが曲のリストを追跡し続けるために使用する _songs という Array 変数の他に、2 つのプライベート変数 (_needToSort と _currentSort) があり、前者はリストのソートが必要であるかどうかを、後者は指定の時刻にどのプロパティで曲のリストをソートするかを追跡し続けます。

すべてのオブジェクトの場合と同様に、Array インスタンスを宣言するだけでは Array の作成作業は完了しません。Array インスタンスのプロパティまたはメソッドにアクセスする前に、このインスタンスを PlayList クラスのコンストラクタでインスタンス化しておく必要があります。

```
public function PlayList()
{
  this._songs = new Array();
  // 最初のソートを設定する
  this.sortList(SortProperty.TITLE);
}
```

コンストラクタの 1 行目は、_songs 変数をインスタンス化して使用できる状態にします。さらに、sortList() メソッドを呼び出し、最初のソート基準プロパティを設定します。

リストへの曲の追加

ユーザーが新しい曲をアプリケーションに登録すると、データ入力フォームのコードが PlayList クラスの addSong() メソッドを呼び出します。

```
/**
 * 曲をプレイリストに追加する
 */
public function addSong(song:Song):void
{
  this._songs.push(song);
  this._needToSort = true;
}
```

addSong() 内では _songs 配列の push() メソッドが呼び出され、その配列の新しい要素として addSong() に渡された Song オブジェクトが追加されます。push() メソッドでは、以前に適用していたソートとは関係なく、新しい要素を配列の末尾に追加します。つまり、push() メソッドを呼び出した後では、曲のリストはソート順が正しくなっていない可能性が高いため、_needToSort 変数は true に設定されます。理論上は、sortList() メソッドをすぐに呼び出すため、指定の時刻にリストをソートするかどうかを追跡し続けることは不要になります。しかし、実際には、取得する直前まで曲のリストをソートする必要はありません。たとえば、取得前に何曲かリストに追加したときなど、ソート操作を遅らせると、アプリケーションは不要なソートを実行しなくなります。

曲のリストのソート

プレイリストが管理する Song インスタンスは複雑なオブジェクトであるため、そのアプリケーションのユーザーは、曲のタイトルまたは発表の年など、さまざまなプロパティに従ってプレイリストをソートしたいと考えることがあります。PlayList アプリケーションにおいて、曲のリストをソートするタスクは、リストのソート基準としてのプロパティを見分けること、そのプロパティによるソート時に必要なソートオプションを指定すること、実際のソート操作を実行することの 3 つの部分から構成されます。

ソートのプロパティ

Song オブジェクトは、曲のタイトル、アーティスト、発表の年、ファイル名、およびユーザーの選曲ジャンル集など、いくつものプロパティを追跡し続けます。言うまでもなく、ソートに実用的なのは、例に挙げた最初の 3 つのプロパティのみです。開発者の便宜のため、例に含めた SortProperty クラスは、ソートに利用可能なプロパティを表す値を持つ列挙として機能します。

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

SortProperty クラスには、`TITLE`、`ARTIST`、および `YEAR` の 3 つの定数があり、各定数には、ソートに使用する関連 Song クラスプロパティの実際の名前を含むストリングが格納されます。コードの残り全体を通し、ソートプロパティが指定されると必ず列挙メンバーを使用して行われます。たとえば、PlayList コンストラクタでは、次のように、`sortList()` メソッドを呼び出してリストを最初にソートします。

```
// 最初のソートを設定する
this.sortList(SortProperty.TITLE);
```

ソートのプロパティは `SortProperty.TITLE` として指定されているため、曲はタイトルに従ってソートされます。

プロパティによるソートおよびソートオプションの指定

実際には曲のリストのソートは、次に示すように、PlayList クラスが `sortList()` メソッドで実行します。

```
/**
 * 指定されたプロパティに従って曲のリストをソートする
 */
public function sortList(sortProperty:SortProperty):void
{
  ...
  var sortOptions:uint;
  switch (sortProperty)
  {
    case SortProperty.TITLE:
```

```
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.ARTIST:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.YEAR:
            sortOptions = Array.NUMERIC;
            break;
    }

    // データの実際のソートを実行する
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // 現在のソートプロパティを保存する
    this._currentSort = sortProperty;

    // リストがソートされたことを記録する
    this._needToSort = false;
}
```

タイトルやアーティストによってソートする場合はアルファベット順にソートし、年によってソートする場合は数値順にソートします。switch ステートメントを使用して、sortProperty パラメータが指定する値に従った適切なソートオプション (変数 sortOptions に格納) を定義します。ここでも、プロパティの区別には、ハードコードされた値ではなく名前付き列挙メンバーを使用しています。

ソートプロパティおよびソートオプションの決定に続き、_songs 配列を sortOn( ) メソッドの呼び出しで実際にソートし、それらの 2 つの値をパラメータとして渡します。現在のソートプロパティが記録されますが、それは曲のリストが現時点でソートされたということです。

文字で区切られたストリングへの配列要素の結合

配列を使用して曲のリストを PlayList クラスで維持する以外に、この例では Song クラスでも配列を使用して、曲についてのジャンルリストを管理するのに役立てています。次に示す、Song クラスの定義からの抜粋で検討してみます。

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String,
    genres:Array)
{
    ...
    // ジャンルは配列として渡されるが
    // 格納時はセミコロンで区切られたストリングである
    this._genres = genres.join(";");
}
```

新しい Song インスタンスの作成時に、曲のジャンルの指定に使用する genres パラメータが Array インスタンスとして定義されます。そのため、複数のジャンルを一緒のグループにまとめ、コンストラクタに渡せる 1 つの変数とするのに便利です。ただし、内部的には Song クラスは、セミコロンで区切られた String インスタンスとして _genres プライベート変数にジャンルを維持します。Array パラメータは、リテラルストリング値 ";" を区切り文字として指定して join() メソッドを呼び出すと、セミコロンで区切られたストリングに変換されます。

同じように、genres アクセサを使用して、ジャンルを配列として設定また取得できます。

```
public function get genres():Array
{
   // ジャンルはセミコロンで区切られたストリングとして格納される
   // そのため返すには配列に戻す変換が必要である
   return this._genres.split(";");
}
public function set genres(value:Array):void
{
   // ジャンルは配列として渡されるが、
   // 格納時はセミコロンで区切られたストリングである
   this._genres = value.join(";");
}
```

genres set アクセサの動作は、コンストラクタの動作とまったく同じです。配列を受け取って join() メソッドを呼び出し、セミコロン区切りのストリングに変換します。get アクセサは反対の操作を実行します。つまり、_genres 変数の split() メソッドが呼び出されると、指定された区切り文字 (リテラルストリング値 ";" など) を使用して、ストリングを値の配列に分割します。

第 8 章

エラー処理

8

" エラー処理 " とは、アプリケーションのコンパイル時またはコンパイル済みアプリケーションの実行時のいずれかに生成されたエラーに応答する、つまりエラーを解決するロジックをアプリケーションに組み入れることを意味します。アプリケーションによるエラー処理では、応答なしや、いかなるプロセスでも説明なしのエラー異常終了になるのとは対照的に、エラー発生時に応答として何かが起きます。エラー処理を正しく使用すれば、予期しない動作からアプリケーションとユーザーを保護するために役立ちます。

ただし、エラー処理は広範な概念で、コンパイル中または実行時に発生する多くの種類のエラーに対する応答を含みます。この章では、ランタイムエラーを処理する方法、生成される可能性のあるさまざまな種類のエラー、および ActionScript 3.0 における新しいエラー処理システムのメリットを中心に説明します。また、エラー処理に関する独自のアプローチをアプリケーション用に実装する方法についても取り上げます。

目次

エラー処理の基礎 242
エラーの種類 245
ActionScript 3.0 におけるエラー処理 247
デバッガ版の Flash Player の操作 249
アプリケーションの同期エラーの処理 249
独自のエラークラスの作成 255
エラーイベントへの応答とステータス 256
Error クラスの比較 259
例 : CustomErrors アプリケーション 265

エラー処理の基礎

エラー処理の概要

ランタイムエラーは、ActionScript コードで生じた何らかの問題であり、Adobe Flash Player 内での ActionScript コンテンツの実行を中止するものです。ActionScript コードをユーザーにとってスムーズに動作させるには、エラーを処理するように、つまりエラーの修正、回避、または少なくともユーザーへの通知を行うようにアプリケーションでコードを記述する必要があります。この処理を " エラー処理 " と言います。

エラー処理は広範な概念で、コンパイル中または実行時に発生する多くの種類のエラーに対する応答を含みます。多くの場合コンパイル時に発生するエラーはより簡単に識別できます。SWF ファイルの作成処理を完了するには、こうしたエラーを修正する必要があります。この章では、コンパイル時のエラーについては説明しません。コンパイル時のエラーを含まないコードの記述方法の詳細については、63 ページ、第 3 章の「ActionScript 言語とシンタックス」および 129 ページ、第 4 章の「ActionScript のオブジェクト指向プログラミング」を参照してください。この章では、ランタイムエラーに焦点を当てます。

ランタイムエラーが発生するには、誤ったコードが実際に実行されることが必要であるために、その検出はより難しくなります。`if..then..else` ステートメントなど、プログラムのセグメントにコードの分岐がいくつか存在する場合、コードにエラーが含まれないことを確認するために、実際のユーザーが使用する可能性があるあらゆる入力値ですべての条件についてテストする必要があります。

ランタイムエラーは " プログラムエラー " と " 論理エラー " 2 つのカテゴリに分類できます。プログラムエラーは、メソッドパラメータに誤ったデータ型を指定するなど、ActionScript コード内の誤りです。論理エラーは、銀行業務アプリケーションで利率計算に誤った数式を使用するなど、プログラムの論理ロジック (データチェックおよび値操作) の誤りです。この 2 つのタイプのエラーも、アプリケーションの入念なテストにより事前に検出および修正できることが少なくありません。

理想的には、エンドユーザーにリリースする前に、アプリケーションのすべてのエラーを識別して削除することが望まれます。ただし、すべてのエラーを予見または防止することはできません。たとえば、管理下にない特定の Web サイトから ActionScript アプリケーションが情報をロードする場合を考えます。ある時点でその Web サイトが使用可能でなければ、該当する外部データに依存するアプリケーションの一部分が正しく動作しません。エラー処理の最も重要な要素は、こうした不明なケースに備えて安全なエラー処理を行い、ユーザーがアプリケーションの使用を続行できること、または少なくとも処理が不可能である理由を説明するわかりやすいエラーメッセージを表示することが必要です。

ActionScript では、ランタイムエラーは次の 2 つの方法で表されます。

- Error クラス : 多くのエラーが関連エラークラスを持ちます。エラーが発生すると、Flash Player は該当する特定のエラーに関連付けられた特定のエラークラスのインスタンスを作成します。コードでは、エラーオブジェクトに含まれる情報を使用してエラーに対して適切な対処を行うことができます。
- エラーイベント : Flash Player が通常どおりにイベントをトリガしたときにエラーが発生することがあります。こうした場合、Flash Player は代わりにエラーイベントをトリガします。他のイベントの場合と同様に、各エラーイベントにはクラスが関連付けられています。Flash Player は、該当クラスのインスタンスをエラーイベントにサブスクライブされたメソッドに渡します。

特定のメソッドでエラーまたはエラーイベントをトリガできるかどうかについては、ActionScript 3.0 コンポーネントリファレンスガイドのメソッド項目を参照してください。

一般的なエラー処理タスク

次に、コードについて実施する必要がある一般的なエラーに関連するタスクを示します。

- エラーを処理するコードの記述
- エラーのテスト、キャッチ、および再スロー
- 独自のエラークラスの定義
- エラーイベントおよびステータスイベントへの応答

重要な概念と用語

次の参照リストに、この章で使用する重要な用語を示します。

- 非同期 : 即時に結果を得られず、イベントの形式で結果 (またはエラー) が得られるメソッドの呼び出しなどのプログラムコマンド。
- キャッチ : 例外 (ランタイムエラー) が発生し、その例外をコードで認識したとき、「コードが例外を " キャッチ " したと言います。例外がキャッチされると、Flash Player は他の ActionScript コードに例外を通知することを停止します。
- デバッガ版 : ユーザーにランタイムエラーを通知するコードを含む Flash Player の特別版。大半のユーザーが使用する標準版の Flash Player では、ActionScript コードにより処理されないエラーは Flash Player により無視されます。Adobe Flash CS3 Professional および Adobe Flex に含まれる) デバッガ版では、未処理のエラーが発生した場合に警告メッセージが表示されます。
- 例外 : プログラムの実行中に発生し、ランタイム環境 (つまり、Flash Player) が単独で解決できないエラー。
- 再スロー : コードが例外をキャッチした場合、Flash Player は他のオブジェクトに例外を通知しません。他のオブジェクトが例外の通知を受けることが重要である場合、コードで例外を " 再スロー " して、通知処理を再度開始することが必要になる場合があります。

- 同期 : 同じコードブロック内で応答を使用できるように即時に結果を提供する (またはエラーをスローする) メソッドの呼び出しなどのプログラムコマンド。
- スロー : エラーが発生したことを Flash Player (したがって他のオブジェクトおよび ActionScript コード) に通知する処理をエラーの " スロー " と言います。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。基本的に、この章のすべてのコードリストには、適切な `trace()` 関数呼び出しが含まれています。この章のコードリストをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   コードリストの `trace()` 関数の結果が [出力] パネルに表示されます。

後半のコードリストの一部はより複雑で、クラスとして作成されています。これらの例をテストするには :

1. 空の Flash ドキュメントを作成して、コンピュータに保存します。
2. 新しい ActionScript ファイルを作成して、Flash ドキュメントと同じディレクトリに保存します。ファイル名は、コードリスト内のクラスの名前と同じ名前にしてください。たとえば、コードリストで ErrorTest という名前のクラスが定義されている場合は、ErrorTest.as という名前で ActionScript ファイルを保存します。
3. コードリストを ActionScript ファイルにコピーして、ファイルを保存します。
4. Flash ドキュメントで、ステージの何もない部分または作業スペースをクリックして、ドキュメントのプロパティインスペクタを起動します。
5. プロパティインスペクタの [ドキュメントクラス] フィールドに、テキストからコピーした ActionScript クラスの名前を入力します。
6. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   例の結果が、[出力] パネル (例で `trace()` 関数が使用されている場合) またはサンプルコードによって作成されたテキストフィールドに表示されます。

サンプルコードリストをテストする手法の詳細については、60 ページの「章内のコード例のテスト」を参照してください。

エラーの種類

アプリケーションの開発中および実行中に発生するエラーは、さまざまな種類があり、使用する用語が異なります。主要なエラーの種類と用語について簡単に説明します。

- コンパイル時エラー：ActionScript コンパイラによるコンパイル処理の際に発生します。コンパイル時エラーは、コードに文法的な問題があるためアプリケーションを構築できないことを示します。
- ランタイムエラー：コンパイル済みのアプリケーションを実行している際に発生します。ランタイムエラーは、SWF ファイルを Adobe Flash Player 9 で再生中に起きるエラーを表します。多くの場合、発生と同時にランタイムエラーを処理してユーザーに報告し、アプリケーションの実行を続けるために必要な処置をとることができます。致命的なエラー (リモート Web サイトへの接続失敗、必須データのロード失敗など) の場合は、アプリケーションを安全に終了するための手段としてエラー処理を使用できます。
- 同期エラー：関数呼び出しの際に発生するランタイムエラーです。たとえば、特定のメソッドに無効なパラメータを渡して使用しようとすると、Flash Player の例外がスローされる場合です。ほとんどのエラーは同期的に発生 (該当するステートメントを実行した時点で発生) するため、制御フローはすぐに最も適切な catch ステートメントに移ります。

  次の抜粋コードの例では、プログラムがファイルをアップロードしようとする前に browse() メソッドを呼び出していないためランタイムエラーがスローされます。

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
     fileRef.upload("http://www.yourdomain.com/fileupload.cfm");
  }
  catch (error:IllegalOperationError)
  {
     trace(error);
     // エラー #2037 : 関数の呼び出しシーケンスが正しくないか、以前の
     // 呼び出しが失敗しています
  }
  ```

  この場合、ファイルをアップロードしようとする前に browse() メソッドが呼び出されていないことが Flash Player によって検出されたため、ランタイムエラーが同期的にスローされます。

  同期エラー処理の詳細については、249 ページの「アプリケーションの同期エラーの処理」を参照してください。

- 非同期エラー：実行時のさまざまな時点で発生するランタイムエラーで、イベントを生成して、イベントリスナーによってキャッチされるエラーです。非同期操作とは、関数が操作を開始してその完了を待たない操作です。アプリケーションまたはユーザーが何らかの操作を試すのを待つために、エラーイベントリスナーを作成し、その操作が失敗した場合にはイベントリスナーでエラーをキャッチして、そのエラーイベントに応答することができます。イベントリスナーはイベントハンドラ関数を呼び出して、エラーイベントに有効な方法で応答します。たとえば、イベントハンドラは、ユーザーにエラーの解決を促すダイアログボックスを起動できます。

前に示した、ファイルアップロードの同期エラーの例で考えてみます。browse() メソッドを正常に呼び出してからファイルのアップロードを開始したとすると、Flash Player からいくつかのイベントが送出されます。たとえば、アップロードが開始したときには open イベントが送出され、ファイルのアップロードが正常に完了したときには complete イベントが送出されます。イベント処理は非同期のため (つまり、事前に設定済みのわかっている時刻に発生するのではないため)、次のように addEventListener() メソッドを使用して、これら特定のイベントを監視する必要があります。

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
  trace("...select...");
  var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  fileupload.cfm");
  request.method = URLRequestMethod.POST;
  event.target.upload(request.url);
}
function openHandler(event:Event):void
{
  trace("...open...");
}
function completeHandler(event:Event):void
{
  trace("...complete...");
}
```

非同期エラー処理の詳細については、256 ページの「エラーイベントへの応答とステータス」を参照してください。

- 不明な (キャッチされない) 例外 : catch ステートメントのような、応答のための対応ロジックなしで発生するエラーです。アプリケーションでエラーが発生したときに、エラー処理のための該当する catch ステートメントまたはイベントハンドラがそのレベルまたは上位レベルで見つからない場合、そのエラーは不明な例外と見なされます。

  実行時に Flash Player は、エラーによって現在の SWF ファイルが停止しない場合、不明なエラーを意図的に無視して再生を継続しようとします。ユーザー自身が必ずしもエラーを解決できるわけではないからです。不明なエラーを無視するプロセスは " 説明のない異常終了 " と呼ばれ、これはアプリケーションのデバッグを複雑にする可能性があります。不明なエラーに応答するために、デバッグ版の Flash Player では、実行中のスクリプトを終了し、その不明なエラーを trace ステートメントの出力に表示するかエラーメッセージをログファイルに書き込みます。また、例外オブジェクトが Error クラスまたはそのサブクラスのインスタンスである場合は、getStackTrace() メソッドが呼び出され、スタックトレース情報が trace ステートメントまたはログファイルに出力されます。デバッグ版の Flash Player の使用の詳細については、249 ページの「デバッガ版の Flash Player の操作」を参照してください。

ActionScript 3.0 におけるエラー処理

エラーを処理するためのロジックを構築しない実行できるアプリケーションが数多くあるため、開発者はエラー処理をアプリケーションに組み入れることを後回しにする傾向があります。ただし、エラー処理がないと、何かが予想どおりに働かない場合に、アプリケーションは簡単に機能を停止するかユーザーを欲求不満にさせる可能性がありま。ActionScript 2.0 の Error クラスでは、特定のメッセージで例外をスローするためのロジックを独自の関数に組み入れることができました。エラー処理はユーザーフレンドリーなアプリケーションを作成する上で非常に重要であるため、ActionScript 3.0 ではエラーキャッチのために拡張アーキテクチャを追加しました。

メモ

『ActionScript 3.0 コンポーネントリファレンスガイド』に、多くのメソッドによってスローされる例外の記載がありますが、各メソッドについて可能性のある例外を網羅してあるわけではありません。メソッドの記述にメソッドがスローする一部の例外が記載されている場合であっても、その記述に明示的に書かれていない構文エラーまたはその他の問題についての例外がメソッドによってスローされることがあります。

ActionScript 3.0 のエラー処理要素

ActionScript 3.0 では、次のような数多くのエラー処理用ツールを導入しています。

- Error クラス : ECMAScript (ECMA-262) Edition 4 言語仕様案に準拠して、ActionScript 3.0 では、エラーオブジェクトを生成する可能性のある状況の有効範囲を広げるために、幅広い範囲の Error クラスを採用しています。システムエラー (MemoryError 状態など)、コーディングエラー (ArgumentError 状態など)、ネットワークおよび通信のエラー (URIError 状態など)、あるいはその他の状況のいずれであっても、アプリケーションが特定のエラー状態を処理しそれに応答するのに役立つ Error クラスがそれぞれに用意されています。各クラスの詳細については、259 ページの「Error クラスの比較」を参照してください。
- 説明のない異常終了の減少 : Flash Player の以前のバージョンでは、エラーは明示的に `throw` ステートメントを使用した場合にしか生成および報告されませんでした。Flash Player 9 では、ActionScript のネイティブなメソッドやプロパティでランタイムエラーがスローされるようになり、発生するこのような例外をより効果的に処理し、個々の例外に個別に対応することが可能です。
- デバッグ中に表示されるエラーメッセージの明確化 : デバッグ版の Flash Player を使用すると、コードまたは状況に問題があるときに明確な内容を示すエラーメッセージが表示されるため、特定のコードブロックで異常が発生した原因を容易に特定できます。このため、エラーの解決がより効率的になります。詳細については、249 ページの「デバッガ版の Flash Player の操作」を参照してください。
- 実行時にユーザーに表示されるエラーメッセージの改良 : Flash Player の以前のバージョンでは、`FileReference.upload()` メソッドの呼び出しに失敗すると Boolean 型の戻り値で `false` が返されましたが、その場合に考えられるエラーの内容には 5 つの種類がありました。ActionScript 3.0 では、`upload()` メソッドの呼び出しにおいてエラーが発生した場合に 4 種類の具体的なエラーをスローでき、エンドユーザーに対して的確なエラーメッセージを表示しやすくなりました。

- エラー処理の再定義：別個のエラーが多くの一般的な状況でスローされます。たとえば、ActionScript 2.0 では、FileReference オブジェクトに値を設定する前に、`name` プロパティの値は `null` です。そのため、`name` プロパティを使用または表示する前に、値が設定済みで `null` でないことを確認する必要があります。ActionScript 3.0 では、FileReference オブジェクトの内容が設定される前に `name` プロパティにアクセスしようとすると、値が未設定であることを示す IllegalOperationError 例外がスローされます。`try..catch..finally` ブロックを使用してエラーを処理することができます。詳細については、250 ページの「try..catch..finally ステートメントの使用」を参照してください。
- 従来とほとんど変わらないパフォーマンス：`try..catch..finally` ブロックによるエラー処理に必要とされるリソース量は、ActionScript の以前のバージョンと同等またはわずかに増加する程度に抑えられています。
- ErrorEvent クラス：このクラスを使用すると、特定の非同期エラーイベントにリスナーを構築することができます。詳細については、256 ページの「エラーイベントへの応答とステータス」を参照してください。

エラー処理のアプローチ

問題のある状態が見つからない限り、エラー処理ロジックがコードに組み込まれていないアプリケーションはそのまま正常に実行される可能性があります。ただし、積極的なエラー処理がないと、アプリケーションで実際に問題が見つかって異常終了してもユーザーはその原因を知ることができません。

いろいろな方法によってアプリケーションでエラー処理にアプローチすることができます。エラー処理の主要な 3 つの方法の概略を次に示します。

- `try..catch..finally` ステートメントの使用：これは発生する同期エラーをキャッチします。各種レベルのコード実行で例外をキャッチするために、ステートメントを階層にネストすることができます。詳細については、250 ページの「try..catch..finally ステートメントの使用」を参照してください。
- 独自のカスタムエラーオブジェクトの作成：Error クラスを使用して、ビルトインのエラーの種類が扱わない特定の操作をアプリケーションで追跡するために、独自のカスタムエラーオブジェクトを作成することができます。そうすると、カスタムエラーオブジェクトで `try..catch..finally` ステートメントを使用できます。詳細については、255 ページの「独自のエラークラスの作成」を参照してください。
- エラーイベントに応答するイベントリスナーおよびハンドラの記述：この方法を採用すると、類似のエラーを処理する同じような `try..catch..finally` ブロックをいろいろな場所に記述することなく、グローバルなエラーハンドラでまとめて処理できます。また、このアプローチを使用すると非同期エラーをキャッチする可能性が高くなります。詳細については、256 ページの「エラーイベントへの応答とステータス」を参照してください。

デバッガ版の Flash Player の操作

開発者のデバッグ用に特別版の Flash Player が用意されています。このデバッグ版の Flash Player を入手するには Adobe Flash CS3 Professional または Adobe Flex Builder 2 をインストールします。

デバッグ版とリリース版の Flash Player では、エラーの表示方法に重要な差異があります。デバッグ版の場合はエラーのタイプ (一般の Error、IOError、EOFError)、エラー番号、および人間が理解できるエラーメッセージが表示されますが、リリース版の場合はエラーのタイプとエラー番号しか表示されません。たとえば、次のようなコードがあるとします。

```
try
{
  tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
  tf.text = error.toString();
}
```

`readBoolean()` メソッドがデバッグ版の Flash Player で EOFError をスローした場合、`tf` テキストフィールドに、"EOFError: ファイルの終端 (EOF) が検出されました " というメッセージが表示されます。

同じコードをリリース版の Flash Player 上で実行すると、表示されるテキストは " エラー #2030" のようになります。

使用リソース量やサイズを最小限にするため、リリース版の Flash Player にはエラーメッセージのストリングが含まれていません。マニュアル (『ActionScript 3.0 コンポーネントリファレンスガイド』の付録) を参照してエラー番号からエラーメッセージを調べるか、デバッグ版の Flash Player 上で問題を再現することで省略されていないメッセージを確認してください。

アプリケーションの同期エラーの処理

最も一般的なエラー処理は同期エラー処理のロジックで、実行時に同期エラーをキャッチするためのステートメントをコードに挿入します。この種類のエラー処理では、関数失敗時のランタイムエラーをアプリケーションで認識してそこから回復できます。同期エラーをキャッチするロジックに `try..catch..finally` ステートメントがあり、これは実際に操作を試し、Flash Player からのエラー応答をキャッチし、最後に、失敗した操作を処理するために何らかの操作を別に実行します。

try..catch..finally ステートメントの使用

同期的なエラーを扱う場合は、try..catch..finally ステートメントを使用してエラーをキャッチします。ランタイムエラーが発生すると、Flash Player は例外をスローします。これは、Flash Player が通常の実行処理を中断し、Error 型の特殊なオブジェクトを作成することを意味します。この Error オブジェクトは、該当する最初の catch ブロックに送出されます。

try ステートメントは、エラーが発生する可能性のあるステートメントを囲むために使用します。try には必ず catch ステートメントが伴います。try ステートメントのブロック内にある何らかのステートメントでエラーが検出されると、その try ステートメントに対応する catch ステートメントが実行されます。

finally ステートメントで囲んだブロックは、try ブロックでエラーが発生したかどうかに関係なく実行されます。エラーが発生しなかった場合は、try ブロック内のステートメントが終了した後で、finally ブロック内のステートメントが実行されます。エラーが発生した場合は、適切な catch ステートメントがまず実行され、その後で finally ブロック内のステートメントが実行されます。

try..catch..finally ステートメントを使用する際のシンタックスを次のコードに示します。

```
try
{
   // エラーをスローする可能性があるコード
}
catch (err:Error)
{
   // エラーを処理するコード
}
finally
{
   // エラーがスローされたかどうかに関係なく実行されるコード。このコードは
   // エラー発生後のクリーンアップや、アプリケーションを続行するために必要な処理をここで行う。
}
```

個々の catch ステートメントには、そこで処理する特定の例外の型を指定します。catch ステートメントに指定できるのは Error クラスのサブクラスのみです。各 catch ステートメントは上から順にチェックされ、スローされたエラーに該当する最初の catch ステートメントだけが実行されます。したがって、先に上位レベルの Error クラスをチェックし、その Error クラスのサブクラスを後からチェックすると、上位レベルの Error クラスに対する処理しか実行されません。この動作を説明するコードを次に示します。

```
try
{
  throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
  trace("<Error> " + error.message);
}
```

```
catch (error:ArgumentError)
{
  trace("<ArgumentError> " + error.message);
}
```

上のコードを実行すると、次のように表示されます。

```
<Error> I am an ArgumentError
```

ArgumentError を正しくキャッチするには、次のコードのように、最も具体的なエラーのチェックが最初で、下に行くほど一般性の強いエラーのチェックになるような順序にする必要があります。

```
try
{
  throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
  trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
  trace("<Error> " + error.message);
}
```

Flash Player API に含まれるいくつかのメソッドとプロパティでは、実行中にエラーが発生するとランタイムエラーをスローします。たとえば、Sound クラスの `close()` メソッドでは、オーディオストリームを閉じることができない場合に IOError をスローします。次にコード例を示します。

```
var mySound:Sound = new Sound();
try
{
  mySound.close();
}
catch (error:IOError)
{
  // Error #2029 : この URLStream オブジェクトには開いているストリームがありません。
}
```

『ActionScript 3.0 コンポーネントリファレンスガイド』に慣れるに従い、各メソッドの説明に詳細が記載されているとおり、どのメソッドで例外がスローされるかがわかってくるはずです。

throw ステートメント

アプリケーション内で実行時にエラーが発生した場合は、Flash Player が例外をスローします。また、アプリケーション内で throw ステートメントを使用すれば、開発者が例外を明示的にスローすることもできます。明示的にエラーをスローした場合は、Error クラスまたはそのサブクラスのインスタンスをスローすることをお勧めします。次のコードは、Error クラス MyErr のインスタンスをスローする throw ステートメントを示し、最後に、エラーのスロー後に応答するために関数 myFunction() を呼び出しています。

```
var MyError:Error = new Error("Encountered an error with the numUsers value", 99);
var numUsers:uint = 0;
try
{
  if (numUsers == 0)
  {
    trace("numUsers equals 0");
  }
}
catch (error:uint)
{
  throw MyError; // 符号なし整数のエラーをキャッチする
}
catch (error:int)
{
  throw MyError; // 整数のエラーをキャッチする
}
catch (error:Number)
{
  throw MyError; // 数値のエラーをキャッチする
}
catch (error:*)
{
  throw MyError; // その他すべてのエラーをキャッチする
}
finally
{
  myFunction(); // 必要に応じてクリーンアップを実行する
}
```

このように、catch ステートメントは最も限定的なデータ型から順に記述するようにしてください。Number データ型に対する catch ステートメントを最初に記述した場合は、uint データ型についても int データ型についても catch ステートメントは実行されなくなります。

メモ	Java プログラミング言語では、例外をスローする可能性があるすべての関数はその旨を宣言することが義務付けられています。宣言するには、スローする可能性がある例外クラスを列挙した throws 節を関数の宣言に含めます。ActionScript では、関数でスローする可能性がある例外を宣言する必要はありません。

単純なエラーメッセージの表示

新しい例外およびエラーイベントモデルがもたらす最大のメリットの1つに、アクションが失敗したタイミングと理由をユーザーに示すことができるという点があります。コードを記述する際、メッセージの表示と応答オプションの提示があるようにします。

次のコードは、エラーをテキストフィールドに表示する単純な try..catch ステートメントを示しています。

```
package
{
  import flash.display.Sprite;
  import flash.text.TextField;

  public class SimpleError extends Sprite
  {
    public var employee:XML =
      <EmpCode>
        <costCenter>1234</costCenter>
        <costCenter>1-234</costCenter>
      </EmpCode>;

    public function SimpleError()
    {
      try
      {
        if (employee.costCenter.length() != 1)
        {
          throw new Error("Error, employee must have exactly one cost center
  assigned.");
        }
      }
      catch (error:Error)
      {
        var errorMessage:TextField = new TextField();
        errorMessage.autoSize = TextFieldAutoSize.LEFT;
        errorMessage.textColor = 0xFF0000;
        errorMessage.text = error.message;
        addChild(errorMessage);
      }
    }
  }
}
```

広範囲のエラークラスおよびビルトインのコンパイルエラーを使用することで、ActionScript 3.0 では以前のバージョンの ActionScript に比べ、何かが失敗したときにその理由に関してより多くの情報が提供されます。このため、優れたエラー処理をする、安定性の向上したアプリケーションを構築することができます。

エラーの再スロー

アプリケーションの開発時には、エラーを適切に処理できず再スローする必要が生じる場合があります。たとえば、次のコードはネストされた try..catch ブロックを含んでおり、下位の catch ブロックでエラーを処理できない場合には ApplicationError をスローしています。

```
try
{
  try
  {
    trace("<< try >>");
    throw new ArgumentError("some error which will be rethrown");
  }
  catch (error:ApplicationError)
  {
    trace("<< catch >> " + error);
    trace("<< throw >>");
    throw error;
  }
  catch (error:Error)
  {
    trace("<< Error >> " + error);
  }
}
catch (error:ApplicationError)
{
  trace("<< catch >> " + error);
}
```

このコードを実行すると次のような内容が表示されます。

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

ネストされた try ブロックで独自の ApplicationError エラーをスローすると、それに続く catch ブロックがエラーをキャッチします。このネストされた catch ブロックではエラー処理を試みますが、処理できない場合は ApplicationError オブジェクトを上位の try..catch ブロックに対してスローします。

独自のエラークラスの作成

ActionScript では、標準 Error クラスのいずれかを拡張し、目的に特化した独自のエラークラスを作成できます。独自のエラークラスを作成する理由は、次のようにいくつかあります。

- アプリケーションに特有のエラー群を識別するため :

  たとえば、Flash Player でトラップされたエラーに加え、アプリケーションのコードでスローされた特有のエラーに対し、異なる方法で対応することが必要な場合があります。`try..catch` ブロックで新しいエラーデータ型を追跡するために、Error クラスのサブクラスを作成することができます。

- アプリケーション内で生成したエラーについて独自の方法でエラー情報を表示するため :

  たとえば、何らかの決まった方法でエラーメッセージを整形表示するために新しい `toString()` メソッドを作成できます。また、必要に応じて `lookupErrorString()` メソッドを定義することもできます。このメソッドはパラメータで指定したエラーコードに対して、ユーザーの言語環境設定に応じた適切な言語のメッセージを返します。

独自のエラークラスは、ActionScript コア Error クラスを継承している必要があります。ここに示すのは、Error クラスを継承した、特化された AppError クラスの例です。

```
public class AppError extends Error
{
  public function AppError(message:String, errorID:int)
  {
    super(message, errorID);
  }
}
```

次に示すのは、プロジェクトで AppError を使用した例です。

```
try
{
  throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
  trace(error.errorID + ": " + error.message)
}
```

メモ

サブクラスで `Error.toString()` メソッドをオーバーライドする場合、受け付けるパラメータは ... (残りパラメータ) の 1 つだけとします。これは `Error.toString()` メソッドの仕様として ECMAScript (ECMA-262) Edition 3 言語仕様に定められていることであり、後方互換性を維持するため ActionScript 3.0 もこの形式に準拠しています。したがって、`Error.toString()` メソッドをオーバーライドする際は正確にこのパラメータを踏襲する必要があります。ただし、実行時に `toString()` メソッドにパラメータを渡しても無視されるため、パラメータを指定する意味はありません。

エラーイベントへの応答とステータス

ActionScript 3.0 のエラー処理で向上した最も重要な機能の 1 つは、非同期のランタイムエラーに応答するためのエラーイベント処理をサポートすることです。非同期エラーの定義については、245 ページの「エラーの種類」を参照してください。

エラーイベントに応答するためにイベントリスナーとイベントハンドラを作成することができます。多くのクラスが、他のイベントの送出時と同じ方法でエラーイベントを送出します。たとえば、XMLSocket クラスのインスタンスは通常、3 種類のイベント (`Event.CLOSE`、`Event.CONNECT`、および `DataEvent.DATA`) を送出します。ただし、問題が発生した場合には、XMLSocket クラスは `IOErrorEvent.IOError` または `SecurityErrorEvent.SECURITY_ERROR` を送出できます。イベントリスナーとイベントハンドラの詳細については、299 ページ、第 10 章の「イベントの処理」を参照してください。

エラーイベントは、次の 2 つのカテゴリのいずれかに該当します。

- ErrorEvent クラスを拡張したエラーイベント

  flash.events.ErrorEvent クラスは、ネットワークおよび通信の操作に関係した Flash Player ランタイムエラーを管理するためのプロパティおよびメソッドを含みます。AsyncErrorEvent、IOErrorEvent、および SecurityErrorEvent クラスは ErrorEvent クラスを拡張したものです。デバッグ版の Flash Player を使用している場合は、リスナー関数が検出されなくても、実行時にダイアログボックスでエラーイベントが通知されます。

- ステータスに基づくエラーイベント

  ステータスに基づくエラーイベントは、ネットワークと通信のクラスの `netStatus` および `status` プロパティに関係します。データの読み取りまたは書き込み時に Flash Player で問題が検出されると、使用中のクラスオブジェクトに応じて、`netStatus.info.level` または `status.level` プロパティが値 "`error`" に設定されます。このエラーに応答するために、イベントハンドラ関数で `level` プロパティに値 "`error`" があるかどうかをチェックします。

エラーイベントの操作

ErrorEvent クラスとそのサブクラスは、データの読み取りまたは書き込みをしようとするときに Flash Player から送出されるエラーを処理するためにエラーの種類を格納しています。

次の例では、ローカルファイルを読み取ろうとして検出されたエラーを表示するために、`try..catch` ステートメントおよびエラーイベントハンドラの両方を使用しています。もっと高度な処理コードを追加することで、ユーザーにオプションを提供するか、そうでなければ、コメントの「ここにエラー処理コードを記述」の場所でエラーを自動的に処理できます。

```
package
{
  import flash.display.Sprite;
  import flash.errors.IOError;
```

```
  import flash.events.IOErrorEvent;
  import flash.events.TextEvent;
  import flash.media.Sound;
  import flash.media.SoundChannel;
  import flash.net.URLRequest;
  import flash.text.TextField;

  public class LinkEventExample extends Sprite
  {
    private var myMP3:Sound;
    public function LinkEventExample()
    {
      myMP3 = new Sound();
      var list:TextField = new TextField();
      list.autoSize = TextFieldAutoSize.LEFT;
      list.multiline = true;
      list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
      list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
      addEventListener(TextEvent.LINK, linkHandler);
      addChild(list);
    }

    private function playMP3(mp3:String):void
    {
      try
      {
        myMP3.load(new URLRequest(mp3));
        myMP3.play();
      }
      catch (err:Error)
      {
        trace(err.message);
        // ここにエラー処理コードを記述
      }
      myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
    }

    private function linkHandler(linkEvent:TextEvent):void
    {
      playMP3(linkEvent.text);
      // ここにエラー処理コードを記述
    }

    private function errorHandler(errorEvent:IOErrorEvent):void
    {
      trace(errorEvent.text);
      // ここにエラー処理コードを記述
    }
  }
}
```

ステータス変化イベントの操作

Flash Player では、level プロパティをサポートするクラスの netStatus.info.level または status.level プロパティの値は動的に変化します。netStatus.info.level プロパティを持つクラスは、NetConnection、NetStream、および SharedObject です。status.level プロパティを持つクラスは、HTTPStatusEvent、Camera、Microphone、および LocalConnection です。level 値の変化に応答して通信エラーを追跡するために、ハンドラ関数を記述します。

次の例では、netStatusHandler( ) 関数を使用して、level プロパティの値をテストしています。level プロパティがエラーの検出を示している場合は、"Video stream failed" というメッセージをトレースします。

```
package
{
  import flash.display.Sprite;
  import flash.events.NetStatusEvent;
  import flash.events.SecurityErrorEvent;
  import flash.media.Video;
  import flash.net.NetConnection;
  import flash.net.NetStream;

  public class VideoExample extends Sprite
  {
    private var videoUrl:String = "Video.flv";
    private var connection:NetConnection;
    private var stream:NetStream;

    public function VideoExample()
    {
      connection = new NetConnection();
      connection.addEventListener(NetStatusEvent.NET_STATUS,
  netStatusHandler);
      connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
  securityErrorHandler);
      connection.connect(null);
    }

    private function netStatusHandler(event:NetStatusEvent):void
    {
      if (event.info.level = "error")
      {
        trace("Video stream failed")
      }
      else
      {
        connectStream();
      }
    }
```

```
    private function securityErrorHandler(event:SecurityErrorEvent):void
    {
      trace("securityErrorHandler: " + event);
    }

    private function connectStream():void
    {
      var stream:NetStream = new NetStream(connection);
      var video:Video = new Video();
      video.attachNetStream(stream);
      stream.play(videoUrl);
      addChild(video);
    }
  }
}
```

Error クラスの比較

ActionScript には多数の定義済み Error クラスがあります。それらの多くは Flash Player で使用されていますが、アプリケーションのコードでも同じ Error クラスを使用して差し支えありません。ActionScript 3.0 の Error クラスは、ActionScript コア Error クラスと flash.error パッケージ Error クラスの 2 つに大別されます。コア Error クラスは、ECMAScript (ECMA-262) Edition 4 言語仕様案で規定されています。flash.error パッケージの内容は、ActionScript 3.0 のアプリケーション開発およびデバッグを支援するために導入された追加のクラスです。

ECMAScript コア Error クラス

ECMAScript コアエラークラスとしては、Error、EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URIError の各クラスがあります。いずれのクラスもトップレベルの名前空間に属します。

クラス名	説明	注意事項
Error	Error クラスは例外をスローするために使用されるクラスであり、ECMAScript に定義されている他の例外クラス (EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URIError) の基本クラスです。	Error クラスは Flash Player からスローされるすべてのランタイムエラーの基本クラスであるだけでなく、独自に作成するエラークラスの基本クラスとすることが推奨されています。
EvalError	EvalError 例外は、Function クラスのコンストラクタに何かパラメータが渡された場合または `eval()` 関数がユーザーコードで呼び出された場合にスローされます。	ActionScript 3.0 では `eval()` 関数のサポートが除外されており、この機能を使用するとエラーがスローされます。Flash Player の以前のバージョンでは、変数、プロパティ、オブジェクト、ムービークリップに名前でアクセスする際に `eval()` 関数が使用されていました。
RangeError	RangeError 例外は、数値の値が許容される範囲に収まらない場合にスローされます。	たとえば、Timer クラスに対する遅延の指定が負の値や無限である場合は RangeError がスローされます。また、表示オブジェクトを無限の深度に追加しようとした場合も RangeError がスローされます。
ReferenceError	ReferenceError 例外は、sealed 指定された (動的でない) オブジェクトに対して未定義プロパティを参照しようとした場合にスローされます。ActionScript 3.0 より前のバージョンでは、`undefined` であるプロパティにアクセスしようとしても ActionScript コンパイラのエラーは発生しませんでした。しかし、ECMAScript の新しい仕様では、この条件の場合にエラーをスローするよう定められているため、ActionScript 3.0 では ReferenceError 例外をスローするようになっています。	未定義の変数に関する例外は潜在的なバグの発見につながり、ソフトウェア品質の向上に役立ちます。しかし、変数を初期化する必要がない従来の仕様に慣れた開発者は、ActionScript の動作が変更されたことによりコーディング上の習慣を変更する必要があります。

クラス名	説明	注意事項
SyntaxError	SyntaxError 例外は、ActionScript コード内で解析エラーが発生した場合にスローされます。 詳細については、Edition 4 が利用可能になるまでは ECMAScript (ECMA-262) Edition 3 言語仕様のセクション 15.11.6.4 (www.ecma-international.org/publications/standards/Ecma-262.htm) と、ECMAScript for XML (E4X) 仕様 (ECMA-357 Edition 2) のセクション 10.3.1 (www.ecma-international.org/publications/standards/Ecma-357.htm) を参照してください。	SyntaxError は次の状況でスローされます。 • RegExp クラスで無効な正規表現を解析した場合 • XMLDocument クラスで無効な XML を解析した場合
TypeError	TypeError 例外は、オペランドに要求される型と実際の型が異なる場合にスローされます。 詳細については、ECMAScript 仕様のセクション 15.11.6.5 (www.ecma-international.org/publications/standards/Ecma-262.htm) と、E4X 仕様のセクション 10.3 (www.ecma-international.org/publications/standards/Ecma-357.htm) を参照してください。	TypeError は次の状況でスローされます。 • 関数またはメソッドの実際のパラメータを、形式上必要とされるパラメータの型に変換できない場合 • 変数に割り当てられた値を、その変数の型に変換できない場合 • `is` または `instanceof` 演算子の右辺が有効な型でない場合 • `super` キーワードの使用方法が不正な場合 • プロパティへの参照が複数のバインディングに解決され、結果があいまいである場合 • 互換性のないオブジェクトに対してメソッドが呼び出された場合 (RegExp クラスのメソッドが汎用オブジェクトに移植されて呼び出された場合など)
URIError	URIError 例外は、グローバルな URI 処理関数のいずれかが定義に合わない方法で使用された場合にスローされます。 詳細については、ECMAScript 仕様のセクション 15.11.6.6 (www.ecma-international.org/publications/standards/Ecma-262.htm) を参照してください。	URIError は次の状況でスローされます。 有効な URL を必要とする Flash Player API 関数 (`Socket.connect()` など) に対し、無効な URI を指定した場合

ActionScript コア Error クラス

ActionScript には、ECMAScript コア Error クラスの他にも、ActionScript に特有のエラー条件やエラー処理機能に関するクラスがいくつか追加されています。

これらのクラスは ECMAScript Edition 4 言語仕様案に対する ActionScript 独自の拡張ですが、有用な機能追加と判断されて仕様案に反映される可能性を考慮して、flash.error のようなパッケージ内ではなくトップレベルの名前空間に配置されています。

クラス名	説明	注意事項
ArgumentError	ArgumentError クラスは、関数呼び出しで指定されたパラメータの値がその関数の定義に適合していないために発生するエラーを表します。	たとえば、次のような状況でパラメータエラーが発生します。 • メソッドに対して指定されたパラメータの個数が多すぎる、または少なすぎる場合 • 列挙のメンバーを指定することが要求されるパラメータに、それ以外の値が指定された場合
SecurityError	SecurityError 例外は、セキュリティ違反が発生してアクセスが拒否された場合に発生します。	たとえば、次のような状況でセキュリティエラーが発生します。 • セキュリティ Sandbox の境界をまたいで、許可されていないプロパティアクセスやメソッド呼び出しを実行した場合 • セキュリティ Sandbox で許可されていない URL にアクセスしようとした場合 • ポリシーファイルが存在しない環境で、許可されていないポート番号 (たとえば 1024 未満) に対してソケット接続を確立しようとした場合 • ユーザーのカメラやマイクにアクセスしようとしたが、そのアクセス要求がユーザーによって拒否された場合
VerifyError	VerifyError 例外は、不正な形式または破損した SWF ファイルが検出された場合にスローされます。	SWF ファイルが別の SWF ファイルをロードした場合、親の SWF はロードされた SWF が生成する VerifyError をキャッチできます。

flash.error パッケージ Error クラス

flash.error パッケージの Error クラスは、Flash Player API の一部と見なされます。前述した他の Error クラスと違い、flash.error パッケージは Flash Player に特有のエラーイベントを伝達するために使用されます。

クラス名	説明	注意事項
EOFError	EOFError 例外は、取得できるデータの末尾よりも後の部分を読み取ろうとした場合にスローされます。	たとえば、IDataInput インターフェイスの読み取りメソッドのいずれかを呼び出したとき、その読み取り要求を満たす量のデータが存在しない場合は EOFError がスローされます。
IllegalOperationError	IllegalOperationError 例外は、メソッドが実装されていないか、使用方法に実装が対応していない場合にスローされます。	たとえば、次のような状況で無効な操作エラーの例外が発生します。 • 基本クラス (DisplayObjectContainer など) に、ステージでサポートされている範囲を超えた機能がある場合。たとえば、Stage のマスクを (`stage.mask` を使用して) 取得または設定しようとすると、Flash Player から "Stage クラスは、このプロパティまたはメソッドを実装しません " というメッセージと共に IllegalOperationError がスローされます。 • サブクラスで、継承する必要がなくサポートすると不都合が生じるようなメソッドを継承した場合 • アクセシビリティのサポートを含めずにコンパイルされた Flash Player の環境で、特定のアクセシビリティ関連メソッドを呼び出した場合 • ランタイム版 Flash Player でオーサリング専用の機能を呼び出した場合 • タイムライン上に配置されたオブジェクトに対して名前を設定しようとした場合
IOError	IOError 例外は、ある種の I/O 例外が発生した場合にスローされます。	たとえば、未接続または切断済みのソケットに対して読み書き操作を実行しようとすると、このエラーが発生します。

クラス名	説明	注意事項
MemoryError	MemoryError 例外は、メモリ割り当て要求が失敗した場合にスローされます。	デフォルトでは、ActionScript 仮想マシン 2 は ActionScript プログラムが割り当てを受けられるメモリの量を制限しません。デスクトップ PC 環境では、メモリ割り当て失敗は頻繁に発生するエラーではありません。操作に必要とされるメモリをシステムが割り当てられないという状況は、デスクトップ PC ではほとんど発生しないからです。32 ビット Microsoft® Windows® 版のプログラム (アドレス空間が 2 GB) で 3 GB の割り当てを要求した場合などは、要求を満たすことが不可能であるため、この例外が発生します。
ScriptTimeoutError	ScriptTimeoutError 例外は、スクリプトのタイムアウト間隔である 15 秒が経過した場合にスローされます。ScriptTimeoutError 例外をキャッチすると、スクリプトのタイムアウトをより安全な方法で処理できます。例外ハンドラを用意しない場合は、不明な例外のハンドラによってダイアログボックスにエラーメッセージが表示されます。	悪意のある開発者によってこの例外がキャッチされ、無限ループから脱出できない事態が発生するのを防ぐために、1 つのスクリプトについて実行中にスローされる ScriptTimeoutError 例外は最初の 1 回しかキャッチできません。それ以降の ScriptTimeoutError 例外はアプリケーションのコードではキャッチされず、不明な例外のハンドラが直ちに実行されます。
StackOverflowError	StackOverflowError 例外は、当該スクリプトで使用できるスタックを使い切った場合にスローされます。	StackOverflowError 例外は、無限の再帰が発生したことを示す場合があります。

例 : CustomErrors アプリケーション

ここでは CustomErrors アプリケーションを使って、アプリケーション構築時のカスタムエラーを操作する手法について示します。それらの手法は、次のとおりです。

- XML パケットの検証
- カスタムエラーの記述
- カスタムエラーのスロー
- エラースロー時のユーザーへの通知

このサンプルのアプリケーションファイルを入手するには、
www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。
CustomErrors アプリケーションのファイルは、"Samples/CustomError" フォルダにあります。
アプリケーションは、次のファイルで構成されています。

ファイル	説明
CustomErrors.mxml または CustomErrors.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/errors/ApplicationError.as	FatalError および WarningError の両方の基本エラークラスとして機能するクラス。
com/example/programmingas3/errors/FatalError.as	アプリケーションによるスローの可能性がある FatalError エラーを定義するクラス。このクラスは独自の ApplicationError クラスを拡張します。
com/example/programmingas3/errors/Validator.as	ユーザー指定の従業員 XML パケットを検証する単一のメソッドを定義するクラス。
com/example/programmingas3/errors/WarningError.as	アプリケーションによるスローの可能性がある WarningError エラーを定義するクラス。このクラスは独自の ApplicationError クラスを拡張します。

CustomErrors アプリケーションの概要

CustomErrors.mxml ファイルは、カスタムエラーアプリケーション用のユーザーインターフェイスと一部のロジックを含みます。アプリケーションの creationComplete イベントが送出されると、initApp() メソッドが呼び出されます。このメソッドは、Validator クラスによって検証されるサンプルの XML パケットを定義します。initApp() メソッドのコードを次に示します。

```
private function initApp():void
{
  employeeXML =
    <employee id="12345">
      <firstName>John</firstName>
      <lastName>Doe</lastName>
      <costCenter>12345</costCenter>
      <costCenter>67890</costCenter>
    </employee>;
}
```

XML パケットは後で、TextArea コンポーネントインスタンスの Stage に表示されます。このため、再検証を試みる前に XML パケットを変更できます。

ユーザーが Validate ボタンをクリックすると、validateData() メソッドが呼び出されます。このメソッドは、Validator クラスの validateEmployeeXML() メソッドを使用して、従業員 XML パケットを検証します。validateData() メソッドのコードを次に示します。

```
public function validateData():void
{
  try
  {
    var tempXML:XML = XML(xmlText.text);
    Validator.validateEmployeeXML(tempXML);
    status.text = "The XML was successfully validated.";
  }
  catch (error:FatalError)
  {
    showFatalError(error);
  }
  catch (error:WarningError)
  {
    showWarningError(error);
  }
  catch (error:Error)
  {
    showGenericError(error);
  }
}
```

最初に、TextArea コンポーネントインスタンス xmlText の内容を使用して、一時的な XML オブジェクトが作成されます。次に、独自の Validator クラス (com.example.programmingas3/errors/Validator.as) の validateEmployeeXML() メソッドが呼び出され、一時的な XML オブジェクトをパラメータとして渡します。XML パケットが有効な場合、Label コンポーネントインスタンス status が成功メッセージを表示して、アプリケーションが終了します。validateEmployeeXML() メソッドがカスタムエラーをスローした (つまり、FatalError、WarningError、または汎用 Error が発生した) 場合は、適切な catch ステートメントを実行して、showFatalError()、showWarningError()、または showGenericError() メソッドのいずれかを呼び出します。これらの各メソッドは、適切なメッセージを Alert コンポーネントに表示して、発生した具体的なエラーをユーザーに通知します。各メソッドは、具体的なメッセージで Label コンポーネントインスタンス status の更新も行います。

従業員 XML パケットを検証しようとしたときに致命的なエラーが発生した場合は、次のコードが示すように、そのエラーメッセージが Alert コンポーネントに表示され、TextArea コンポーネントインスタンス xmlText および Button コンポーネントインスタンス validateBtn が無効になります。

```
public function showFatalError(error:FatalError):void
{
   var message:String = error.message + "\n\n" + "Click OK to end.";
   var title:String = error.getTitle();
   Alert.show(message, title);
   status.text = "This application has ended.";
   this.xmlText.enabled = false;
   this.validateBtn.enabled = false;
}
```

致命的なエラーではなく警告エラーが発生した場合は、そのエラーメッセージが Alert コンポーネントに表示されますが、TextField および Button コンポーネントインスタンスは無効になりません。showWarningError() メソッドが、カスタムエラーメッセージを Alert コンポーネントインスタンスに表示します。このメッセージでは、XML の検証を続行するかスクリプトを中止するかの質問もユーザーに表示します。showWarningError() メソッドの抜粋コードを次に示します。

```
public function showWarningError(error:WarningError):void
{
   var message:String = error.message + "\n\n" + "Do you want to exit this
   application?";
   var title:String = error.getTitle();
   Alert.show(message, title, Alert.YES | Alert.NO, null, closeHandler);
   status.text = message;
}
```

Yes または No ボタンを使用してユーザーが Alert コンポーネントインスタンスを閉じると、closeHandler() メソッドが呼び出されます。closeHandler() メソッドの抜粋コードを次に示します。

```
private function closeHandler(event:CloseEvent):void
{
   switch (event.detail)
   {
```

```
      case Alert.YES:
        showFatalError(new FatalError(9999));
        break;
      case Alert.NO:
        break;
    }
}
```

警告エラーで Yes をクリックしてユーザーがスクリプトの中止を選択した場合は、FatalError がスローされ、その結果アプリケーションは終了します。

独自のバリデータの構築

独自の Validator クラスは、1 つのメソッド `validateEmployeeXML()` を含みます。`validateEmployeeXML()` メソッドは、1 つのパラメータ `employee` (検証対象の XML パケット) を受け取ります。`validateEmployeeXML()` メソッドを次に示します。

```
public static function validateEmployeeXML(employee:XML):void
{
  // checks for the integrity of items in the XML
  if (employee.costCenter.length() < 1)
  {
    throw new FatalError(9000);
  }
  if (employee.costCenter.length() > 1)
  {
    throw new WarningError(9001);
  }
  if (employee.ssn.length() != 1)
  {
    throw new FatalError(9002);
  }
}
```

検証のため、従業員は 1 つ (1 つのみ) の原価部門に属していなければなりません。従業員がいずれの原価部門にも属していない場合は、FatalError がスローされてメインアプリケーションファイルの `validateData()` メソッドに移ります。従業員が複数の原価部門に属している場合は、WarningError がスローされます。XML バリデータの最終チェックは、ユーザーに定義されている社会保障番号が厳密に 1 つであることの確認です (XML パケットの `ssn` ノード)。存在する `ssn` ノードが厳密に 1 つでない場合は、FatalError エラーがスローされます。

`validateEmployeeXML()` メソッドに追加のチェックを加えることができます。たとえば、`ssn` ノードに格納されている番号が有効であることを確認する場合や、従業員に少なくとも 1 つの電話番号と電子メールアドレスを割り当て、その両方の値が有効であることを確認する場合などです。各従業員に一意の ID を割り当てその上司の ID を指定するために、XML を変更することもできます。

ApplicationError クラスの定義

ApplicationError クラスは、FatalError および WarningError の両方の基本クラスとして機能します。ApplicationError クラスは Error クラスを拡張して、独自にメソッドおよびプロパティを定義します。エラー ID、重大度、独自のエラーコードおよびメッセージを格納する XML オブジェクトなどを定義します。このクラスでは、エラーの種類ごとの重大度を定義するのに使用する 2 つの静的定数も定義します。

ApplicationError クラスのコンストラクタメソッドを次に示します。

```
public function ApplicationError()
{
  messages =
    <errors>
      <error code="9000">
        <![CDATA[Employee must be assigned to a cost center.]]>
      </error>
      <error code="9001">
        <![CDATA[Employee must be assigned to only one cost center.]]>
      </error>
      <error code="9002">
        <![CDATA[Employee must have one and only one SSN.]]>
      </error>
      <error code="9999">
        <![CDATA[The application has been stopped.]]>
      </error>
    </errors>;
}
```

XML オブジェクトの各エラーコードは、一意の数値コードとエラーメッセージを格納します。次の getMessageText() メソッドが示すように、エラーメッセージは、E4X を使用してエラーコードで簡単に調べることができます。

```
public function getMessageText(id:int):String
{
  var message:XMLList = messages.error.(@code == id);
  return message[0].text();
}
```

getMessageText() メソッドは、1 つの整数パラメータ id を受け取り、ストリングを返します。id パラメータには、調べるエラーのエラーコードを指定します。たとえば、id を 9001 で渡すと、「従業員は 1 つのみの原価部門に割り当てなければならない」ことを示すエラーを受け取ります。複数のエラーでエラーコードが同じ場合、ActionScript は見つかった最初の結果のみにエラーメッセージを返します (返された XMLList オブジェクトの message[0])。

このクラスの次のメソッドである getTitle() はパラメータを 1 つも受け取らず、この特定のエラーのエラー ID を格納するストリング値を返します。この値は Alert コンポーネントのタイトルで使用すると、XML パケットの違反時に発生した正確なエラーを簡単に識別するのに役立ちます。getTitle() メソッドの抜粋コードを次に示します。

```
public function getTitle():String
{
  return "Error #" + id;
}
```

ApplicationError クラスの最後のメソッドは toString() です。このメソッドは、エラーメッセージの表現をカスタマイズするために、Error クラスに定義されている関数をオーバーライドします。発生した特定のエラー番号とメッセージを識別するストリングを返します。

```
public override function toString():String
{
  return "[APPLICATION ERROR #" + id + "] " + message;
}
```

FatalError クラスの定義

FatalError クラスは独自の ApplicationError クラスを拡張して、3 つのメソッド (FatalError コンストラクタ、getTitle()、および toString() を定義します。最初のメソッドである FatalError コンストラクタは、1 つの整数パラメータ errorID を受け取り、ApplicationError クラスに定義されている静的定数値を使用してエラーの重大度を設定し、ApplicationError クラスの getMessageText() メソッドを呼び出して特定のエラーのエラーメッセージを取得します。FatalError コンストラクタを次に示します。

```
public function FatalError(errorID:int)
{
  id = errorID;
  severity = ApplicationError.FATAL;
  message = getMessageText(errorID);
}
```

FatalError クラスの次のメソッドである getTitle() は、ApplicationError クラスで以前に定義された getTitle() メソッドをオーバーライドし、致命的なエラーが発生したことをユーザーに通知するためにタイトルにテキスト "-- FATAL" を追加します。getTitle() メソッドを次に示します。

```
public override function getTitle():String
{
  return "Error #" + id + " -- FATAL";
}
```

このクラスの最後のメソッドである toString() は、ApplicationError に定義されている toString() メソッドをオーバーライドします。toString() メソッドを次に示します。

```
public override function toString():String
{
  return "[FATAL ERROR #" + id + "] " + message;
}
```

WarningError クラスの定義

WarningError クラスは ApplicationError クラスを拡張し、FatalError クラスとほぼ同一です。ただし、次のコードが示すようなマイナー変更があり、エラー重大度は ApplicationError.FATAL ではなく ApplicationError.WARNING に設定されます。

```
public function WarningError(errorID:int)
{
  id = errorID;
  severity = ApplicationError.WARNING;
  message = super.getMessageText(errorID);
}
```

第 9 章

正規表現の使用

9

正規表現とは、ストリング内で条件に一致するテキストの検索および操作に使用するパターンの表記法です。正規表現はストリングのように見えますが、その内容には、パターンや繰り返しを表す特殊なコードが含まれています。たとえば、次の正規表現は、先頭が A であり、その後に 1 桁以上の数字が続くようなストリングに一致します。

```
/A\d+/
```

この章では、正規表現を作成するための基本的なシンタックスについて説明します。正規表現には複雑な要素や微妙に意味が異なる機能が数多く存在するため、詳細については、Web や書籍で他の参考資料を参照してください。ただし、プログラミング環境が異なると正規表現の実装も異なっている場合があるため注意が必要です。ActionScript 3.0 では、ECMAScript Edition 3 言語仕様 (ECMA-262) の定義に基づいて正規表現を実装しています。

目次

正規表現の基礎 274
正規表現のシンタックス 276
ストリングに対して正規表現を使用するメソッド 292
例 : Wiki パーサー 293

正規表現の基礎

正規表現の使用の概要

正規表現は、文字のパターンを表します。一般的に、正規表現はテキスト値が特定のパターンに準拠していることを確認する場合 (ユーザーが入力した電話番号の桁数が適切であることを確認する場合など) や、特定のパターンと一致するテキストを置き換える場合に使用されます。

正規表現には、単純なものもあります。たとえば、特定のストリングが "ABC" に一致するかどうかを確認する場合や、ストリング内に出現するすべての "ABC" を別のテキストに置き換える場合を考えてください。その場合、文字 A、B、C が順に連続して出現するパターンを定義する次の正規表現を使用できます。

```
/ABC/
```

正規表現リテラルを囲む区切り文字はスラッシュ (/) です。

正規表現では、複雑なパターンや解読困難なパターンが使用されることもあります。たとえば、次の正規表現は有効な電子メールアドレスに一致します。

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

最も一般的には、ストリング内のパターンを検索し、文字を置換するために正規表現を使用します。そうした場合は、正規表現オブジェクトを作成し、それをパラメータとしていずれかの String クラスのメソッドで使用します。正規表現のパラメータを指定できる String クラスのメソッドとしては、`match()`、`replace()`、`search()`、および `split()` があります。これらのメソッドの詳細については、200 ページの「ストリング内のパターンの検索とサブストリングの置換」を参照してください。

RegExp クラスには、`test()` メソッドおよび `exec()` メソッドが含まれます。詳細については、292 ページの「ストリングに対して正規表現を使用するメソッド」を参照してください。

一般的な正規表現のタスク

次に、この章で詳細を説明する正規表現の一般的な用途をいくつか示します。

- 正規表現パターンの作成
- パターン内での特殊文字の使用
- 複数の文字シーケンス ("2 桁の数字 "、"7 ～ 10 桁の文字 " など) の識別
- ある文字範囲または数字範囲に含まれる任意の文字 ("a ～ m" の任意の文字 " など) の識別
- 使用可能な文字セット内の文字の識別
- サブシーケンス (パターン内のセグメント) の識別
- パターンに基づくテキストの一致と置換

重要な概念と用語

次の参照リストに、この章で使用する重要な用語を示します。

- エスケープ文字：後に続く文字がリテラル文字ではなくメタ文字として処理する必要があることを示す文字。正規表現シンタックスでは、バックスラッシュ文字 (\) はエスケープ文字であるため、別の文字の後続するバックスラッシュは単なる文字自体ではなく特別なコードです。
- フラグ : 大文字と小文字を区別するかどうかなど、正規表現パターンの使用方法に関するいくつかのオプションを指定する文字。
- メタ文字：パターン内で文字を逐語的に表すのではなく、正規表現パターン内で特別な意味を持つ文字。
- 制御文字 : パターンの一部の反復回数を示す文字 (または複数の文字)。たとえば、米国の郵便番号では 5 桁または 9 桁の番号を含む必要があることを指定するために制御文字を使用します。
- 正規表現 : パターンが他のストリングに一致するかどうかを確認するため、またはストリングの一部を置換するために使用できる文字パターンを定義するプログラムステートメント。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章のコードリストは主に正規表現のパターンで構成されているので、サンプルをテストするには、次の手順が必要です。

1. 新しい Flash ドキュメントを作成します。
2. キーフレームを選択して、[アクション] パネルを開きます。
3. 次のような RegExp (正規表現) 変数を作成します。

   ```
   var pattern:RegExp = /ABC/;
   ```

4. サンプルからパターンをコピーして、RegExp 変数の値として代入します。たとえば、前のコード行では、等号の右側のコード部分 (セミコロンを含まない) がパターンです (/ABC/)。
5. 正規表現をテストするのに適したストリングを含む、String 変数を作成します。たとえば、正規表現を作成してメールアドレスが有効かどうかをテストする場合は、有効なメールアドレスと無効なメールアドレスを含んだ String 変数を作成します。

   ```
   var goodEmail:String = "bob@example.com";
   var badEmail:String = "5@$2.99";
   ```

6. String 変数をテストして正規表現パターンに一致するかどうかを判別するコード行を追加します。これらの値を画面に出力するには、trace( ) 関数を使用するか、ステージ上のテキストフィールドに書き込みます。

```
trace(goodEmail, ":", pattern.test(goodEmail));
trace(badEmail, ":", pattern.test(badEmail));
```

たとえば、pattern に有効なメールアドレスの正規表現パターンが定義されている場合、前のコード行は、このテキストを [出力] パネルに書き込みます。

```
bob@example.com is valid: true
5@$2.99 is valid: false
```

テキストフィールドインスタンスに値を書き込むことにより、または trace( ) 関数を使用して値を [出力] パネルに出力することにより値をテストする方法の詳細については、60 ページの「章内のコード例のテスト」を参照してください。

正規表現のシンタックス

このセクションでは、ActionScript の正規表現シンタックスを構成するすべての要素について説明します。以降で説明するように、正規表現は複雑であり、微妙な相違を把握することが必要です。詳細については、Web や書籍で他の参考資料を参照してください。ただし、プログラミング環境が異なると正規表現の実装も異なっている場合があるため注意が必要です。ActionScript 3.0 では、ECMAScript Edition 3 言語仕様 (ECMA-262) の定義に基づいて正規表現を実装しています。

多くの場合、正規表現は単純な文字の連続ではなく複雑なパターンを見つけるために使用します。たとえば、次の正規表現では、文字 A、B、C が順に連続して出現し、その後に任意の数字が続くパターンを定義しています。

```
/ABC\d/
```

\d というコードは、" 任意の数字 " を意味します。円記号 (\) はエスケープ文字と呼ばれ、正規表現の中では、直後に続く文字 (この場合は d) との組み合わせによって特別な意味を表現します。この章では、各種のエスケープ文字シーケンスと、正規表現シンタックスに関するその他の機能についても説明します。

次の正規表現では、ABC という一連の文字が出現し、その後に任意の個数の数字が続くパターンを定義しています (アスタリスクが付いていることに注意してください)。

```
/ABC\d*/
```

このアスタリスク文字 (*) は、" メタ文字 " の一種です。メタ文字とは、正規表現の中で特別な意味を持つ文字のことです。アスタリスクは " 繰り返し制御文字 " と呼ばれる種類のメタ文字で、文字または文字グループの繰り返しを指定する場合に使用します。詳細については、283 ページの「繰り返し制御文字」を参照してください。

正規表現には、パターンに加え、その正規表現の一致方法を指定するフラグを必要に応じて設定できます。たとえば、次の正規表現で使用している i フラグは、ストリング内の一致箇所を探す際に大文字と小文字を区別しないことを指定するものです。

```
/ABC\d*/i
```

詳細については、288 ページの「フラグとプロパティ」を参照してください。

正規表現は、String クラスの match()、replace()、および search() メソッドで使用できます。これらのメソッドの詳細については、200 ページの「ストリング内のパターンの検索とサブストリングの置換」を参照してください。

正規表現のインスタンスの作成

正規表現のインスタンスを作成するには 2 つの方法があります。1 つは正規表現を区切り文字のスラッシュ (/) で囲んで記述する方法であり、もう 1 つは new コンストラクタを使用する方法です。たとえば、次の正規表現はいずれも同等です。

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

ストリングリテラルを引用符で囲むのと同じように、正規表現リテラルを記述する場合はスラッシュで囲みます。正規表現のうち 2 つのスラッシュで囲んだ部分が " パターン " です。それに加え、後ろのスラッシュに続いて " フラグ " を指定することもできます。フラグは正規表現を構成する要素と見なされますが、パターンとは区別されています。

new コンストラクタを使用する場合は、2 つのストリングによって正規表現を定義します。次のように、第 1 のストリングでパターンを指定し、第 2 のストリングでフラグを指定します。

```
var pattern2:RegExp = new RegExp("bob", "i");
```

スラッシュで囲んだ正規表現定義の中にスラッシュを含める場合は、中のスラッシュの直前にエスケープ文字の円記号 (\) を置く必要があります。たとえば、次の正規表現は 1/2 というパターンに一致します。

```
var pattern:RegExp = /1\/2/;
```

new コンストラクタによる正規表現定義の中に引用符を含める場合は、ストリングリテラルの場合と同じように、中の引用符の直前にエスケープ文字の円記号 (\) を置く必要があります。たとえば、次の正規表現は eat at "joe's" というパターンに一致します。

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

スラッシュで囲んだ正規表現定義の中に引用符を含める場合は、エスケープ文字の円記号を付けないでください。同じように、new コンストラクタによる正規表現定義の中にスラッシュを含める場合は、エスケープ文字の円記号を付けないでください。次の正規表現は同等で、いずれも 1/2 "joe's" というパターンに一致します。

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

また、new コンストラクタによる正規表現定義の中で、円記号 (\) から始まるメタシーケンス (任意の数字を意味する \d など) を使用する場合は、次のように円記号を二重に入力する必要があります。

```
var pattern:RegExp = new RegExp("\\d+", ""); // 1 桁以上の数字に一致する
```

この場合は、RegExp( ) コンストラクタメソッドの最初のパラメータがストリングで、ストリングリテラルの中で単一の円記号を認識させるためには円記号を二重に入力する必要があるため、円記号を二重に入力します。

以降のセクションでは、正規表現パターンの定義に使用するシンタックスについて説明します。

フラグの詳細については、288 ページの「フラグとプロパティ」を参照してください。

文字、メタ文字、およびメタシーケンス

最も単純な正規表現は、次のように一連の文字に一致するものです。

```
var pattern:RegExp = /hello/;
```

ただし、次の各文字はメタ文字と呼ばれ、正規表現の中では特別な意味を持ちます。

```
^ $ \ . * + ? ( ) [ ] { } |
```

たとえば、次の正規表現は、文字 A の後に 0 個以上の B が続き (メタ文字のアスタリスクが繰り返しを表す)、さらにその後に C が続くストリングに一致します。

```
/AB*C/
```

メタ文字に該当する文字を正規表現のパターン内に含めて、特別な意味を表さないようにするには、エスケープ文字の円記号 (\) を付ける必要があります。たとえば、次の正規表現は、文字 A の後に B、アスタリスク、および C が続くストリングに一致します。

```
var pattern:RegExp = /AB\*C/;
```

" メタシーケンス " も、メタ文字と同じように正規表現の中で特別な意味を持ちます。1 つのメタシーケンスは複数の文字によって構成されます。以降のセクションでは、メタ文字とメタシーケンスの詳細な使用方法を示します。

メタ文字について

次の表は、正規表現で使用できるメタ文字の一覧です。

メタ文字	説明
^ (キャレット)	ストリングの先頭に一致します。また、m (multiline) フラグを設定した場合は、行の先頭にも一致します。289 ページの「m (multiline) フラグ」を参照してください。ただし、文字クラスの先頭にキャレットを指定した場合は、ストリングの先頭ではなく補集合を意味します。詳細については、281 ページの「文字クラス」を参照してください。
$ (ドル記号)	ストリングの末尾に一致します。また、m (multiline) フラグを設定した場合は、改行文字 (\n) の直前の位置にも一致します。詳細については、289 ページの「m (multiline) フラグ」を参照してください。
\ (円記号)	特殊文字が持つ特別な意味を無効化します。 また、/1\/2/ のように、正規表現リテラルの中でスラッシュ文字を使用する場合も円記号を使用します (文字 1、スラッシュ、文字 2 の連続に一致)。
. (ドット)	任意の 1 文字に一致します。 ただし、改行文字 (\n) には s (dotall) フラグを設定した場合のみ一致します。詳細については、290 ページの「s (dotall) フラグ」を参照してください。
* (アスタリスク)	直前のアイテムについて 0 回以上の繰り返しに一致します。 詳細については、283 ページの「繰り返し制御文字」を参照してください。
+ (プラス)	直前のアイテムについて 1 回以上の繰り返しに一致します。 詳細については、283 ページの「繰り返し制御文字」を参照してください。
? (疑問符)	直前のアイテムについて 0 回または 1 回だけ一致します。 詳細については、283 ページの「繰り返し制御文字」を参照してください。
(および)	正規表現の中でグループを定義します。グループは次の場合に使用します。 • 選択制御文字 | の適用範囲を限定する場合。例 : /(a|b|c)d/ • 繰り返し制御文字の適用範囲を指定する場合。例 : /(walla.){1,2}/ • 後方参照を使用する場合。たとえば、次の正規表現に含まれている \1 は、このパターン内にある最初のグループ化括弧に一致したストリングに一致します。 /(\w*) is repeated: \1/ 詳細については、285 ページの「グループ」を参照してください。

メタ文字	説明
[および]	文字クラス (該当位置に一致可能な文字を表す集合) を定義します。 /[aeiou]/ は、角括弧内に指定したいずれか 1 つの文字に一致します。 文字クラス内でハイフン (-) を使用すると、文字の範囲を指定できます。 /[A-Z0-9]/ は、大文字の A ～ Z または 0 ～ 9 に一致します。 文字クラス内で] および - をエスケープするには、円記号を使用します。 /[+\-]\d+/ は、+ または - とそれに続く 1 桁以上の数字に一致します。 文字クラス内では、通常メタ文字として扱われる他の文字は、メタ文字でなく一般の文字として扱われます。円記号でエスケープする必要はありません。 /[$£]/ は、$ または £ に一致します。 詳細については、281 ページの「文字クラス」を参照してください。
| (パイプ)	選択肢を示し、左側または右側のいずれかにある部分に一致します。 /abc|xyz/ は、abc または xyz に一致します。

メタシーケンスについて

メタシーケンスは、正規表現パターンの中で特別な意味を持つ一連の文字です。メタシーケンスについて次の表で説明します。

メタシーケンス	説明
{*n*} {*n*,} および {*n*,*n*}	直前のアイテムについて、繰り返し回数または回数の範囲を指定します。 /A{27}/ は、文字 A の 27 回の繰り返しに一致します。 /A{3,}/ は、文字 A の 3 回以上の繰り返しに一致します。 /A{3,5}/ は、文字 A の 3 回以上 5 回以下の繰り返しに一致します。 詳細については、283 ページの「繰り返し制御文字」を参照してください。
\b	単語構成文字とそれ以外の文字の境界となる位置に一致します。ストリングの先頭または末尾が単語構成文字である場合は、その先頭または末尾にも一致します。
\B	2 つの単語構成文字に挟まれた位置に一致します。また、単語構成文字でない 2 つの文字に挟まれた位置にも一致します。
\d	10 進数の数字 1 文字に一致します。
\D	数字以外の任意の 1 文字に一致します。
\f	改ページ文字に一致します。
\n	改行文字に一致します。
\r	復帰文字に一致します。
\s	任意の空白文字 (スペース、タブ、改行、復帰) に一致します。
\S	空白文字以外の任意の 1 文字に一致します。
\t	タブ文字に一致します。

メタシーケンス	説明
`\u`*nnnn*	16 進数 *nnnn* で指定される文字コードを持つ Unicode 文字に一致します。 たとえば、`\u263a` はスマイリー文字です。
`\v`	垂直フィード文字に一致します。
`\w`	単語構成文字 (`A` ～ `Z`、`a` ～ `z`、`0` ～ `9`、または _) に一致します。 `\w` は、é、ñ、ç など、英語以外の文字には一致しません。
`\W`	単語構成文字以外の任意の 1 文字に一致します。
`\x`*nn*	16 進数 *nn* で指定される文字コードを持つ ASCII 文字に一致します。

文字クラス

文字クラスは、正規表現の中で該当する位置に一致可能な文字を表すリストを指定するために使用します。文字クラスを定義するには、リストを角括弧 (`[` および `]`) で囲みます。たとえば、次の正規表現で定義している文字クラスは、`bag`、`beg`、`big`、`bog`、`bug` のいずれにも一致します。

```
/b[aeiou]g/
```

文字クラス内のエスケープシーケンス

正規表現において通常は特別な意味を持つメタ文字やメタシーケンスのほとんどは、文字クラス内では特別な意味を持ちません。たとえば、アスタリスクは正規表現の中で繰り返しを意味しますが、文字クラス内に出現するアスタリスクはその例外です。次の文字クラスは、リストで指定されている他の文字と同じようにアスタリスク文字そのものにも一致します。

```
/[abc*123]/
```

ただし、次の表に示す 3 つの文字は、文字クラス内で特別な意味を持つメタ文字として機能します。

メタ文字	文字クラス内での意味
`]`	文字クラスの末尾を示します。
`-`	文字の範囲を示します。282 ページの「文字クラス内の文字範囲」を参照してください。
`\`	メタシーケンスを指定します。また、メタ文字の特別な意味を無効化します。

これらの文字のいずれかをリテラル文字 (特別な意味のない一般の文字) として認識させるには、その文字の直前にエスケープ文字の円記号を付ける必要があります。たとえば、次の正規表現で定義している文字クラスは、`$`、`\`、`]`、`-` の 4 つの記号いずれにも一致します。

```
/[$\\\]\-]/
```

文字クラス内で特別な意味を持つメタ文字があるのと同じように、次のメタシーケンスは文字クラス内でもメタシーケンスとして機能します。

メタシーケンス	文字クラス内での意味
`\n`	改行文字に一致します。
`\r`	復帰文字に一致します。
`\t`	タブ文字に一致します。
`\u`*nnnn*	16 進数 *nnnn* で指定される文字コードを持つ Unicode 文字に一致します。
`\x`*nn*	16 進数 *nn* で指定される文字コードを持つ ASCII 文字に一致します。

その他の正規表現メタシーケンスおよびメタ文字は、文字クラス内では一般の文字として扱われます。

文字クラス内の文字範囲

ハイフンを使用すると、文字の範囲を指定できます (例 : `A-Z`、`a-z`、`0-9`)。指定する文字は、使用している文字セットにおいて有効な範囲を構成するものである必要があります。たとえば、次の文字クラスは、a ～ z の範囲または任意の数字のうち 1 つに一致します。

```
/[a-z0-9]/
```

また、ASCII 文字コードを示す `\x`*nn* の形式を使用し、ASCII 値の範囲を指定することもできます。たとえば、次の文字クラスは、拡張 ASCII 文字のセットに属する任意の文字に一致します (é や ê など)。

```
/[\x80-\x9A]/
```

文字クラスの補集合

文字クラスの先頭にキャレット文字 (^) を指定すると、クラスの指定が反転し、リストに含まれない任意の文字に一致するという意味になります。たとえば、次の文字クラスは、a ～ z の範囲および数字を除く任意の 1 文字に一致します。

```
/[^a-z0-9]/
```

補集合を表すキャレット文字 (^) は、文字クラスの先頭に記述する必要があります。それ以外の場所に記述した場合は、単に文字クラスにキャレット文字を含める意味になります。たとえば、次の文字クラスは、いろいろな記号 (キャレットを含む) のいずれかに一致します。

```
/[!.,#+*%$&^]/
```

繰り返し制御文字

繰り返し制御文字は、パターンの中で文字またはシーケンスの繰り返しを指定するために使用します。次に例を示します。

繰り返し制御のメタ文字	説明
* (アスタリスク)	直前のアイテムについて 0 回以上の繰り返しに一致します。
+ (プラス)	直前のアイテムについて 1 回以上の繰り返しに一致します。
? (疑問符)	直前のアイテムについて 0 回または 1 回だけ一致します。

`{n}`
`{n,}`
および
`{n,n}`
	直前のアイテムについて、繰り返し回数または回数の範囲を指定します。
	`/A{27}/` は、文字 A の 27 回の繰り返しに一致します。
	`/A{3,}/` は、文字 A の 3 回以上の繰り返しに一致します。
	`/A{3,5}/` は、文字 A の 3 回以上 5 回以下の繰り返しに一致します。

次のように、繰り返し制御文字は単一の文字に適用することも、グループに適用することもできます。

- `/a+/` は、文字 a の 1 回以上の繰り返しに一致します。
- `/\d+/` は、1 桁以上の数字に一致します。
- `/[abc]+/` は、文字 a、b、c の任意の組み合わせで構成される 1 文字以上の繰り返しに一致します。
- `/(very, )*/` は、very という単語の後にカンマと空白が続くストリングの 0 回以上の繰り返しに一致します。

繰り返し制御文字をグループ化括弧内で使用すると、グループに繰り返し制御を適用できます。たとえば、次の繰り返し制御文字は、word というストリングにも、word-word-word というストリングにも一致します。

```
/\w+(-\w+)*/
```

デフォルトでは、正規表現は " 最長一致 " の検索ロジックを使用します。すなわち、正規表現に含まれる各サブパターン (.* など) について、ストリング内で可能な限り多くの文字に一致する方法を探し、それから正規表現内の次の部分に処理を移します。たとえば、次のような正規表現とストリングがあるとします。

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

この正規表現はストリング全体に一致します。

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

しかし、場合によっては、1 組の ... グループのみに一致させることが本来の目的であることもあります。その場合は次のようにします。

```
<p>Paragraph 1</p>
```

任意の繰り返し制御文字の直後に疑問符 (?) を付けると、その繰り返し制御文字は、" 最短一致 " の検索ロジックを使用するように動作が変更されます。たとえば、次の正規表現では、*? という最短一致の繰り返し制御を使用しています。これは、 と最小限の文字の後に が続くストリングに一致します。

```
/<p>.*?<\/p>/
```

繰り返し制御文字については、次の点に注意してください。

- {0} または {0,0} と指定しても、対象のアイテムが一致から除外されるわけではありません。
- /abc+*/ のように複数の繰り返し制御文字を重複して指定することはできません。
- ドット (.) に * などの繰り返し制御を付けても、s (dotall) フラグを設定しない限り複数行にわたって一致することはありません。たとえば、次のようなコードがあるとします。

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
  // 出力 : <p>Test
  //          Multiline</p>
```

詳細については、290 ページの「s (dotall) フラグ」を参照してください。

選択制御

正規表現で | (パイプ) 文字を使用すると、複数ある選択肢のいずれかに一致することを指定できます。たとえば、次の正規表現は cat、dog、pig、rat のいずれかの単語に一致します。

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

選択制御文字 | の適用範囲を限定するには、括弧を使用してグループを定義します。次の正規表現は、cat の後に nap または nip が続くストリングに一致します。

```
var pattern:RegExp = /cat(nap|nip)/;
```

詳細については、285 ページの「グループ」を参照してください。

次の 2 つの正規表現は、一方では選択制御文字 | を、もう一方では文字クラス ([と] で指定) を使用していますが、いずれも同等です。

```
/1|3|5|7|9/
/[13579]/
```

詳細については、281 ページの「文字クラス」を参照してください。

グループ

次のように括弧を使用すると、正規表現の中でグループを定義できます。

```
/class-(\d*)/
```

グループはパターンの部分的なセクションであり、次の目的で使用できます。

- 繰り返し制御を複数の文字に適用する場合
- 選択制御 (| 文字) を適用する対象のサブパターンを区切る場合
- 一致したサブストリングをキャプチャする場合。たとえば、正規表現の中で \1 を使用すると、前出のグループが一致したストリングと同じ内容に一致します。また、String クラスの replace() メソッドでは同様の目的に $1 を使用できます。

以降のセクションでは、グループの使用方法について詳しく説明します。

繰り返し制御に対するグループの使用

グループを使用しない場合、次のように、繰り返し制御文字はその直前にある 1 つの文字または文字クラスに適用されます。

```
var pattern:RegExp = /ab*/ ;
// 文字 a と、その後に続く
// 文字 b の 0 回以上の繰り返しに一致する

pattern = /a\d+/;
// 文字 a と、その後に続く
// 1 桁以上の数字に一致する

pattern = /a[123]{1,3}/;
// 文字 a と、その後に続く
// 1、2、3 いずれかの 1 回以上 3 回以下の繰り返しに一致する
```

グループを使用すると、次のように、繰り返し制御文字を複数の文字や文字クラスに適用できます。

```
var pattern:RegExp = /(ab)*/;
// 文字 a とその後に続く
// b を組み合わせた 0 回以上の繰り返しに一致する。例 : ababab

pattern = /(a\d)+/;
// 文字 a とその後に続く
// 数字 1 桁を組み合わせた 1 回以上の繰り返しに一致する。例 : a1a5a8a3

pattern = /(spam ){1,3}/;
// spam とそれに続く空白を組み合わせた 1 ～ 3 回の繰り返しに一致する
```

繰り返し制御文字の詳細については、283 ページの「繰り返し制御文字」を参照してください。

選択制御文字 (|) に対するグループの使用

グループは、次のように選択制御文字 (|) の適用対象となる範囲を指定するために使用できます。

```
var pattern:RegExp = /cat|dog/;
// cat または dog に一致する

pattern = /ca(t|d)og/;
//catog または cadog に一致する
```

グループによる一致したサブストリングのキャプチャ

正規表現では、パターン内に通常のグループ化括弧を指定すると、後でそのグループに一致した内容を参照できます。これは " 後方参照 " と呼ばれる機能で、後方参照に使用するグループを " キャプチャグループ " といいます。たとえば、次の正規表現に含まれている \1 というシーケンスは、その前の括弧で指定したキャプチャグループに一致したサブストリングと同じ内容に一致します。

```
var pattern:RegExp = /(\d+)-by-\1/;
// 次に一致します : 48-by-48
```

後方参照は、\1、\2、...、\99 のように記述することにより、1 つの正規表現の中で最大 99 個まで使用できます。

同じように、String クラスの replace() メソッドでは $1 〜 $99 と記述することにより、キャプチャグループに一致したサブストリングを置換ストリング内に挿入できます。

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
// 出力 : Bob, hello.
```

また、RegExp クラスの exec() メソッドまたは String クラスの match() メソッドでキャプチャグループを使用すると、グループに一致したサブストリングをメソッドの戻り値として取得できます。

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@test.com,bob,example,com
```

非キャプチャグループと先読みグループの使用

非キャプチャグループとは、キャプチャおよび番号による後方参照の機能を持たない、グループ化だけを目的としたグループです。非キャプチャグループを定義するには、次のように (?: と) を使用します。

```
var pattern = /(?:com|org|net);
```

次の例は、(com|org) というパターンをキャプチャグループおよび非キャプチャグループで使用した場合の違いを示しています。全体の一致を調べた後でキャプチャグループの一致内容を列挙するために exec() メソッドを使用します。

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example,com
```

```
// 非キャプチャ :
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example
```

特殊な非キャプチャグループとして、" 先読みグループ " と呼ばれるものがあります。さらに、これは " 肯定先読みグループ " と " 否定先読みグループ " の 2 つに分類されます。

肯定先読みグループは (?= と) で定義され、該当位置に一致する必要があるサブパターンを示しますが、肯定先読みグループに一致した部分は、同じ正規表現の中にある後続のパターンについても一致の対象となります。たとえば、次のコードに含まれている (?=e) は肯定先読みグループなので、これに一致する文字 e は、正規表現の続き (この例ではキャプチャグループの \w*)) についても一致を調べる対象となります。

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

否定先読みグループは (?! と) で定義され、該当位置に一致してはならないサブパターンを示します。次に実際の使用例を示します。

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

名前付きグループの使用

名前付きグループとは、正規表現で使用するグループの一種で、名前による識別子を指定したものを指します。名前付きグループを定義するには、(?P< 名前 > と) を使用します。たとえば、次の正規表現では、digits という識別子を持つ名前付きグループを指定しています。

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

exec() メソッドを使用すると、次のように、一致する名前付きグループが result 配列のプロパティとして追加されます。

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

次の例では、name および dom という識別子を持つ 2 つの名前付きグループを使用しています。

```
var emailPattern:RegExp =
  /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // 例
```

メモ

名前付きグループは、ECMAScript の言語仕様には規定されていない、ActionScript 3.0 独自の拡張機能です。

フラグとプロパティ

次の表に、正規表現に対して設定できる 5 種類のフラグを示します。各フラグには正規表現オブジェクトのプロパティとしてアクセスできます。

フラグ	プロパティ	説明
g	global	複数箇所に一致します。
i	ignoreCase	一致の判定において大文字と小文字を区別しません。A 〜 Z および a 〜 z の文字に適用されますが、É や é などの拡張文字には適用されません。
m	multiline	このフラグを設定した場合、$ および ^ はそれぞれ行末と行頭にも一致します。
s	dotall	このフラグを設定した場合、.(ドット) は改行文字 (\n) にも一致します。
x	extended	正規表現の拡張機能を有効にします。パターンの一部と見なされない空白を正規表現の中に挿入できるようになります。これにより、正規表現コードの読みやすさを向上できます。

これらのプロパティは読み取り専用です。次のように、正規表現変数を設定する際にはフラグ (g、i、m、s、x) を設定できます。

```
var re:RegExp = /abc/gimsx;
```

しかし、名前付きプロパティの値を直接設定することはできません。たとえば、次のコードではエラーが発生します。

```
var re:RegExp = /abc/;
re.global = true; // エラーが発生する
```

デフォルトでは、正規表現の宣言内で指定しない限りフラグは設定されず、対応する各プロパティの値は false となります。

正規表現には上記の他にも、次の 2 つのプロパティがあります。

- lastIndex プロパティは、正規表現に対する次回の exec() または test() メソッド呼び出しにおいて使用する、ストリング内のインデックス位置を示します。
- source プロパティは、正規表現の中でパターンを定義する部分のストリングを示します。

g (global) フラグ

g (global) フラグを設定しない場合、正規表現が複数の場所に一致することはありません。たとえば、次の正規表現には g フラグを設定していないので、String.match( ) メソッドは一致したサブストリングを1つしか返しません。

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // 出力 : she
```

次のように g フラグを設定すると、Sting.match( ) メソッドは複数の一致箇所を返します。

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// 同じパターンだが、今度は g フラグを設定する
trace(str.match(pattern)); // 出力 : she,shells,shore
```

i (ignoreCase) フラグ

正規表現の一致を判定する際、デフォルトでは大文字と小文字が区別されますが、i (ignoreCase) フラグを設定した場合は大文字と小文字が区別されません。たとえば、次の正規表現に含まれている小文字の s は、ストリングの先頭にある大文字の S には一致しません。

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // 出力 : 13 -- Not the first character
```

この正規表現に i フラグを設定すると、大文字の S にも一致するようになります。

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // 出力 : 0
```

i フラグによって大文字と小文字が無視されるのは、A-Z および a-z 文字についてのみであり、É や é では無視されません。

m (multiline) フラグ

m (multiline) フラグを設定しない場合、^ はストリングの先頭に、$ はストリングの末尾に一致します。m フラグを設定した場合、これらの文字はそれぞれ行頭と行末にも一致します。次の例では、ストリング内に改行文字が含まれています。

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // ストリングの先頭にある単語に一致する
```

正規表現に g (global) フラグを設定していても、^ に一致する場所は1つ (ストリングの先頭) しか存在しないので、match( )- メソッドが返すサブストリングは1つだけです。したがって、結果の出力は次のようになります。

```
Test
```

次の例も同じコードですが、今度は m フラグを設定しています。

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // 行頭にある単語に一致する
```

この場合、出力にはそれぞれの行頭にある単語が含まれます。

```
Test,Multiline
```

行末を示す文字は、\n だけです。次の文字は行末と見なされません。

- 復帰文字 (\r)
- Unicode の行区切り文字 (\u2028)
- Unicode の段落区切り文字 (\u2029)

s (dotall) フラグ

s (dotall、つまり " ドットを全文字対象にする ") フラグを設定しない場合、正規表現パターン内のドット (.) は改行文字 (\n) には一致しません。したがって、次の例では一致する場所がありません。

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

ただし、s フラグを設定すると、ドットが改行文字にも一致するようになります。

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

この場合は、<p> タグに囲まれたサブストリング全体 (改行文字を含む) に一致します。

```
<p>Test
Multiline</p>
```

x (extended) フラグ

正規表現は解読が困難になる場合があります。特に、メタ文字やメタシーケンスを多用した正規表現は難解になる場合があります。たとえば、次のようになります。

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

正規表現に x (extended) フラグを設定した場合、パターン内に入力した空白文字がすべて無視されます。たとえば、次の正規表現は前の例と同等です。

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

x フラグを設定している場合に空白文字の一致を調べるには、その空白文字の直前に円記号を付けます。たとえば、次の 2 つの正規表現は同等です。

```
/foo bar/
/foo \ bar/x
```

lastIndex プロパティ

lastIndex プロパティは、ストリング内で次回の検索を開始するインデックス位置を示します。このプロパティは、exec( ) および test( ) メソッドの呼び出しに対して g フラグを true に設定した場合に有効です。たとえば、次のようなコードがあるとします。

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
  trace(pattern.lastIndex);
  result = pattern.exec(str);
}
```

lastIndex プロパティは、デフォルトでは 0 (ストリングの先頭から検索を開始) に設定されていますが、検索を実行するたびに、一致した場所に続くインデックス位置に設定されます。したがって、前の例で示したコードの出力は次のようになります。

```
0
5
11
18
25
36
44
```

global フラグを false に設定している場合、exec( ) および test( ) メソッドでは lastIndex プロパティを使用しません。

String クラスの match( )、replace( )、search( ) 各メソッドでは、検索処理を常にストリングの先頭から実行します。呼び出しに使用する正規表現の lastIndex プロパティにどのような値を設定しても影響はありません (ただし、match( ) メソッドを呼び出すと lastIndex には 0 が設定されます)。

lastIndex プロパティに値を設定すれば、ストリング内で正規表現による検索を開始する位置を変更できます。

source プロパティ

source プロパティは、正規表現の中でパターンを定義する部分のストリングを示します。次に実際の使用例を示します。

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

ストリングに対して正規表現を使用するメソッド

RegExp クラスには、exec() および test() の 2 つのメソッドが含まれます。

RegExp クラスの exec() メソッドおよび test() メソッドに加えて、String クラスにはストリング内で正規表現に一致する match()、replace()、search()、および splice() の各メソッドが含まれます。

test() メソッド

RegExp クラスの test() メソッドは、次の例で示すように、指定したストリング内に正規表現に一致する部分があるかどうかを単に判定します。

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // 出力 : true
```

exec() メソッド

RegExp クラスの exec() メソッドは、指定したストリング内に正規表現に一致する部分があるかどうかを調べ、次の内容を格納した配列を返します。

- パターンに一致したサブストリング
- 正規表現に含まれるいずれかのグループ化括弧に一致したサブストリング

また、この配列には、サブストリングの一致を開始したインデックス位置を示す index プロパティがあります。

たとえば、次のようなコードがあるとします。

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; // 米国の電話番号
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7  -  415-555-1212
```

正規表現に g (global) フラグを設定して、exec() メソッドを繰り返し呼び出すことにより、次のように複数のサブストリングについて一致を調べることができます。

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
  trace(result.index, "\t", pattern.lastIndex, "\t", result);
  result = pattern.exec(str);
}
// 出力 :
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

RegExp パラメータを使用する String のメソッド

正規表現のパラメータを指定できる String クラスのメソッドとしては、`match()`、`replace()`、`search()`、および `split()` があります。これらのメソッドの詳細については、200 ページの「ストリング内のパターンの検索とサブストリングの置換」を参照してください。

例 : Wiki パーサー

この単純な Wiki テキスト変換の例で、正規表現の使用方法をいくつか示します。

- ソースの Wiki パターンに一致するテキスト行を適切な HTML 出力ストリングに変換する。
- 正規表現を使用して、URL パターンを HTML の `<a>` ハイパーリンクタグに変換する。
- 正規表現を使用して、米ドルストリング (`"$9.95"` など) をユーロストリング (`"8.24 €"` など) に変換する。

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。WikiEditor アプリケーションのファイルは、"Samples/WikiEditor" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
WikiEditor.mxml または WikiEditor.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル
com/example/programmingas3/regExpExamples/WikiParser.as	正規表現を使用して Wiki 入力テキストパターンを同等の HTML 出力に変換するメソッドが含まれているクラス
com/example/programmingas3/regExpExamples/URLParser.as	正規表現を使用して URL ストリングを HTML の `<a>` ハイパーリンクタグに変換するメソッドが含まれているクラス
com/example/programmingas3/regExpExamples/CurrencyConverter.as	正規表現を使用して米ドルストリングをユーロストリングに変換するメソッドが含まれているクラス

WikiParser クラスの定義

WikiParser クラスには、Wiki 入力テキストを同等の HTML 出力に変換するメソッドが含まれます。この Wiki 変換アプリケーションは堅牢ではありませんが、パターンマッチングおよびストリング変換を行う正規表現の使用方法を示します。

コンストラクタ関数は `setWikiData()` メソッドと共に、Wiki 入力テキストのサンプルストリングを次のように単純に初期化します。

```
public function WikiParser()
{
  wikiData = setWikiData();
}
```

ユーザーがサンプルアプリケーションの [Test] ボタンをクリックすると、アプリケーションは WikiParser オブジェクトの `parseWikiString()` メソッドを呼び出します。このメソッドは、他の複数のメソッドを呼び出し、結果の HTML ストリングを組み立てます。

```
public function parseWikiString(wikiString:String):String
{
  var result:String = parseBold(wikiString);
  result = parseItalic(result);
  result = linesToParagraphs(result);
  result = parseBullets(result);
  return result;
}
```

呼び出されるメソッド (`parseBold()`、`parseItalic()`、`linesToParagraphs()`、および `parseBullets()`) は、入力 Wiki テキストを HTML 形式のテキストに変換するために、ストリングの `replace()` メソッドを使用して、正規表現で定義された一致パターンを置き換えます。

ボールドおよびイタリックのパターンの変換

`parseBold()` メソッドは、Wiki のボールドテキストパターン (`'''foo'''` など) を探し、次のように、同等の HTML (`<b>foo</b>` など) に変換します。

```
private function parseBold(input:String):String
{
  var pattern:RegExp = /'''(.*?)'''/g;
  return input.replace(pattern, "<b>$1</b>");
}
```

正規表現の `(.?*)` の部分は、2 つの定義パターン `'''` の間にある任意の数の文字 (*) に一致することに注意してください。? 繰り返し制御文字を使用すると、最短一致でマッチングが行われます。そのため、`'''aaa''' bbb '''ccc'''` のようなストリングの場合、最初に一致するストリングは `'''aaa'''` になります。`'''` パターンで開始し、終了するストリング全体ではありません。

正規表現内の括弧は、キャプチャグループを定義します。replace() メソッドは、置換ストリングの $1 コードを使用して、このグループを参照します。正規表現の g (global) フラグによって、replace() メソッドは (最初の 1 つだけではなく) ストリング内のすべての一致を置き換えます。

parseItalic() メソッドは parseBold() メソッドと同じように動作しますが、イタリックテキストの区切り記号として 2 つのアポストロフィ ('') を探します (3 つではありません)。

```
private function parseItalic(input:String):String
{
  var pattern:RegExp = /''(.*?)''/g;
  return input.replace(pattern, "<i>$1</i>");
}
```

箇条書きパターンの変換

次の例で示すように、parseBullet() メソッドは Wiki の箇条書き行パターン (* foo など) を探し、同等の HTML (<li>foo</li> など) に変換します。

```
private function parseBullets(input:String):String
{
  var pattern:RegExp = /^\*(.*)/gm;
  return input.replace(pattern, "<li>$1</li>");
}
```

正規表現の先頭の ^ 記号は、行の先頭に一致します。正規表現で m (multiline) フラグを使用した場合、^ 記号は、単純にストリングの先頭ではなく、行頭とマッチングされます。

* パターンは、アスタリスク文字に一致します (* 繰り返し制御文字ではなく、リテラルのアスタリスクであることを示すために、円記号を使用します)。

正規表現内の括弧は、キャプチャグループを定義します。replace() メソッドは、置換ストリングの $1 コードを使用して、このグループを参照します。正規表現の g (global) フラグによって、replace() メソッドは (最初の 1 つだけではなく) ストリング内のすべての一致を置き換えます。

Wiki 段落パターンの変換

linesToParagraphs() メソッドは、入力 Wiki ストリングの各行を HTML の <p> 段落タグに変換します。メソッド内の次の行によって、入力 Wiki ストリングから空の行を削除します。

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

正規表現の ^ 記号は行の先頭に一致し、$ 記号は行の末尾に一致します。正規表現で m (multiline) フラグを使用した場合、^ 記号は単純にストリングの先頭ではなく、行頭とマッチングされます。

replace() メソッドは、すべての一致するサブストリング (空の行) を空のストリング ("") で置き換えます。正規表現の g (global) フラグによって、replace() メソッドは (最初の 1 つだけではなく) ストリング内のすべての一致を置き換えます。

URL から HTML の <a> タグへの変換

ユーザーがサンプルアプリケーションの [Test] ボタンをクリックしたとき、[urlToATag] チェックボックスがオンであると、アプリケーションは URLParser.urlToATag() 静的メソッドを呼び出して、入力 Wiki ストリングの URL ストリングを HTML の <a> タグに変換します。

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart,
                       "ig");
var result:String = input.replace(urlPattern,
                 "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

RegExp() コンストラクタ関数を使用して、複数の要素部分から得られた正規表現を組み立てます (urlPattern)。これらの要素部分は、正規表現パターンの一部を定義するストリングです。

正規表現パターンの最初の部分は、protocol ストリングで定義され、URL プロトコル (http://または ftp://) を定義します。括弧は ? 記号で示され、非キャプチャグループを定義します。これは、括弧が | 選択制御パターン用のグループの定義にのみ使用されるという意味です。このグループは replace() メソッドの置換ストリングの後方参照コード ($1、$2、$3) に一致しません。

正規表現のその他の要素部分は、それぞれキャプチャグループを使用し (パターン内の括弧で示されます)、replace() メソッドの置換ストリングの後方参照コード ($1、$2、$3) で使用されます。

urlPart ストリングで定義されたパターン部分は、文字 a-z、0-9、_、または - の 1 つ以上に一致します。+ 繰り返し制御文字は、少なくとも 1 つの文字に一致することを示します。\. は、必須のドット (.) 文字を示します。残りは、文字 a 〜 z、0 〜 9、_、または - の 1 つ以上で構成される別のストリングに一致します。

optionalUrlPart ストリングで定義されたパターン部分は、0 個以上の任意の数の英数字文字 (_ と - を含む) が続くドット (.) 文字に一致します。* 繰り返し制御文字は、0 個以上の文字が一致することを示します。

replace() メソッドの呼び出しでは、正規表現を使用し、後方参照を使用して置換 HTML ストリングが組み立てられます。

urlToATag() メソッドは、次に emailToATag() メソッドを呼び出します。このメソッドは、似た手法を使用して、電子メールパターンを HTML の <a> ハイパーリンクストリングに置き換えます。このサンプルファイルで HTTP、FTP、および電子メールの URL とのマッチングに使用する正規表現は、例を示すことが目的なので単純化されています。より正確にこれらの URL とマッチングするには、より複雑な正規表現を使用します。

米ドルストリングからユーロストリングへの変換

ユーザーがサンプルアプリケーションの [Test] ボタンをクリックしたとき、[dollarToEuro] チェックボックスがオンであると、アプリケーションは CurrencyConverter.usdToEuro() 静的メソッドを呼び出して、次のように米ドルストリング ("$9.95" など) をユーロストリング ("8.24 €" など) に変換します。

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

最初の行で、米ドルストリングに一致する単純なパターンを定義します。$ 文字の前にバックスラッシュ (\) エスケープ文字が付いていることに注意してください。

replace() メソッドは、正規表現を使用してパターンマッチングを行い、usdStrToEuroStr() 関数を呼び出して置換ストリング (ユーロでの値) を決定します。

関数名を replace() メソッドの第 2 パラメータとして使用すると、次の要素がパラメータとして呼び出し元関数に渡されます。

- ストリング内の一致する部分。
- グループ化括弧によってキャプチャされた一致。この方法で渡される引数の数は、グループ化括弧によってキャプチャされた一致数によって異なります。グループ化括弧によってキャプチャされた一致数は、関数コード内の arguments.length - 3 を確認して特定することができます。
- ストリング内で一致部分が始まる場所のインデックス位置。
- ストリング全体。

usdStrToEuroStr() メソッドは、次のように米ドルストリングパターンをユーロストリングに変換します。

```
private function usdToEuro(...args):String
{
   var usd:String = args[1];
   usd = usd.replace(",", "");
   var exchangeRate:Number = 0.828017;
   var euro:Number = Number(usd) * exchangeRate;
   trace(usd, Number(usd), euro);
   const euroSymbol:String = String.fromCharCode(8364); // €
   return euro.toFixed(2) + " " + euroSymbol;
}
```

args[1] は、usdPrice 正規表現に一致した、キャプチャされた括弧指定グループを表します。これは、米ドルストリングの数字部分です。すなわち、$ 記号が付いていない金額です。メソッドは、為替レート変換を適用し、結果のストリングを返します (前に $ 記号を付ける代わりに、後ろに € 記号を付けます)。

第 10 章

イベントの処理

10

イベント処理システムは、ユーザー入力やシステムイベントにプログラムが応答するための便利な仕組みです。ActionScript 3.0 のイベントモデルは単に便利であるだけでなく、標準規格の仕様に準拠しており、しかも、Adobe Flash Player 9 の表示リストの機能とよく統合されています。業界標準のイベント処理アーキテクチャであるドキュメントオブジェクトモデル (DOM) Level 3 Events 仕様に基づき、新しいイベントモデルは ActionScript プログラマ用の強力かつ直感性のあるイベント処理ツールを提供します。

この章は 5 つのセクションで構成されます。そのうち最初の 2 つのセクションでは、ActionScript のイベント処理に関する基本事項について説明します。残り 3 つのセクションでは、イベントモデルのベースとなっている主要な概念である、イベントフロー、イベントオブジェクトおよびイベントリスナーについて説明します。ActionScript 3.0 のイベント処理システムは表示リストと緊密に協調して機能するため、この章の内容は、表示リストについて基本事項を理解している読者が対象となっています。詳細については、353 ページの「表示のプログラミング」を参照してください。

目次

イベント処理の基本 300
ActionScript 3.0 イベント処理の以前のバージョンとの違い 303
イベントフロー 306
イベントオブジェクト 308
イベントリスナー 313
例 : Alarm Clock. 320

イベント処理の基本

イベント処理の概要

イベントとは、SWF ファイル内でのプログラマとして関心のあるさまざまな事象の発生と考えることができます。たとえば、大部分の SWF ファイルは、マウスクリックに応答する単純なものから、フォームに入力されたデータの受け入れまたは処理のような複雑なものまで、何らかのユーザーの操作をサポートします。ユーザーと SWF ファイルとの間で行われるそのような対話操作は、イベントの一種と考えられます。また、ユーザーによる直接的な操作が行われていないときも、サーバーからのデータのロードが終了した場合や、接続されたカメラがアクティブになった場合などにイベントが発生することがあります。

ActionScript 3.0 では、1 回のイベントを 1 個のイベントオブジェクト (Event クラスまたはそのサブクラスのインスタンス) として表現します。イベントオブジェクトには、具体的なイベントに関する情報が格納されているのに加え、イベントオブジェクトを操作する際に役立つメソッドが備わっています。たとえば Flash Player では、マウスクリックを検出すると、その特定のマウスクリックイベントを表すイベントオブジェクト (MouseEvent クラスのインスタンス) を作成します。

次に、Flash Player は作成したイベントオブジェクトを " 送出 " します。これは、イベントのターゲットとなるオブジェクトにイベントオブジェクトを引き渡すという意味です。イベントオブジェクトの送出先となるオブジェクトを、" イベントターゲット " と呼びます。たとえば、コンピュータに取り付けられているカメラがアクティブになった場合、Flash Player はイベントターゲット (この場合はカメラを表すオブジェクト) に対して直接にイベントオブジェクトを送出します。ただし、イベントターゲットが表示リスト内にある場合、イベントオブジェクトは表示リストの階層内を下ってイベントターゲットまで伝達されていきます。場合によっては、この表示リスト階層内の経路をイベントオブジェクトが逆に " 浮上 " (バブリング) していくこともあります。この表示リスト階層の移動を " イベントフロー " と呼びます。

アプリケーションのコードでイベントを受け取る (監視する) には、イベントリスナーを使用します。" イベントリスナー " とは、特定のイベントに応答するために記述する関数やメソッドです。プログラムがイベントに常に応答するようにするには、イベントターゲットまたはイベントオブジェクトのイベントフローの一部である表示リストオブジェクトのいずれかにイベントリスナーを追加する必要があります。

イベントリスナーコードを作成する場合は、常に次の基本構造に従います (太字の要素は、特定の状況で入力するプレースホルダです)。

```
function eventResponse(eventObject:EventType):void
{
   // イベントに応じて実行されるアクションがここで実行されます。
}

eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse);
```

このコードは 2 つの操作を実行します。まず、関数を定義します。これはイベントに対応して実行するアクションを指定する方法です。次に、ソースオブジェクトの addEventListener( ) メソッドを呼び出します。実際には、イベントの発生時に関数のアクションが実行されるように、指定されたイベントに関数を " サブスクライブ " します。実際にイベントが発生すると、イベントリスナーとして登録されたすべての関数とメソッドのリストが、イベントターゲットによりチェックされます。その後、関数とメソッドが順に呼び出され、パラメータとしてイベントオブジェクトが渡されます。

独自のイベントリスナーを作成するには、このコード内の 4 つの項目を変更する必要があります。最初に、関数の名前を使用したい名前に変更します (eventResponse と記述された 2 か所を変更します)。2 番目に、監視したいイベントにより送出されるイベントオブジェクトに適切なクラス名を指定します (コード内の EventType)。また、特定のイベントに対して適切な定数を指定します (リスト内の EVENT_NAME)。3 番目に、イベントを送出するオブジェクトの addEventListener( ) メソッドを呼び出します (コード内の eventTarget)。オプションで、関数のパラメータとして使用する変数の名前も変更できます (コード内の eventObject)。

一般的なイベント処理のタスク

この章では、次のような一般的なイベント処理タスクについて説明します。

- イベントに応答するコードの記述
- コードのイベント応答停止
- イベントオブジェクトの操作
- イベントフローの操作
  - イベントフロー情報の特定
  - イベントフローの停止
  - デフォルト動作の回避
- クラスからのイベントの送出
- カスタムイベントタイプの作成

重要な概念と用語

次の参照リストに、この章で使われる重要な用語を示します。

- デフォルト動作 : 一部のイベントでは、イベントにともなって動作が実行されます。これをデフォルト動作と呼びます。たとえば、ユーザーがテキストフィールドにテキストを入力すると、テキスト入力イベントが生成されます。このイベントに対するデフォルト動作は、テキストフィールドに入力された文字を実際に表示することです。ただし、入力した文字を何らかの理由で表示したくない場合などは、このデフォルト動作をオーバーライドすることができます。
- 送出 : イベントの発生をイベントリスナーに通知する処理です。

- イベント：あるオブジェクトに発生した事象で、そのオブジェクトから他のオブジェクトに通知できます。
- イベントフロー：表示リスト上のオブジェクト (画面に表示されたオブジェクト) に対してイベントが発生すると、そのオブジェクトを含むすべてのオブジェクトにイベントが通知され、次にそのイベントリスナーに通知されます。このプロセスはステージで開始し、表示リストからイベントが発生した実際のオブジェクトへと移動して、再びステージに戻ります。このプロセスをイベントフローと呼びます。
- イベントオブジェクト：特定のイベントの発生に関する情報を含むオブジェクトで、イベントが送出されるとすべてのリスナーに送信されます。
- イベントターゲット：実際にイベントを送出するオブジェクトです。たとえば、ステージ内の Sprite に含まれるボタンをユーザーがクリックした場合、そのステージ内のすべてのオブジェクトがイベントを送出しますが、イベントターゲットとは実際にイベントが発生したオブジェクトであり、この場合はクリックされたボタンのことです。
- リスナー：オブジェクトに登録したオブジェクトまたは関数で、特定のイベントの発生時に通知を受けることを示します。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。基本的に、この章のすべてのコードリストには、コードの結果をテストするための `trace()` 関数呼び出しが含まれています。この章のコードリストをテストするには：

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   コードリストの `trace()` 関数の結果が [出力] パネルに表示されます。

コードリストの一部はより複雑で、クラスとして作成されています。これらの例をテストするには：

1. 空の Flash ドキュメントを作成して、コンピュータに保存します。
2. 新しい ActionScript ファイルを作成して、Flash ドキュメントと同じディレクトリに保存します。ファイル名は、コードリスト内のクラスの名前と同じ名前にしてください。たとえば、コードリストで EventTest という名前のクラスが定義されている場合は、EventTest.as という名前で ActionScript ファイルを保存します。
3. コードリストを ActionScript ファイルにコピーして、ファイルを保存します。

4. Flash ドキュメントで、ステージの何もない部分または作業スペースをクリックして、ドキュメントのプロパティインスペクタを起動します。
5. プロパティインスペクタの [ドキュメントクラス] フィールドに、テキストからコピーした ActionScript クラスの名前を入力します。
6. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   例の結果が [出力] パネルに表示されます。

サンプルコードリストをテストする手法の詳細については、60 ページの「章内のコード例のテスト」を参照してください。

ActionScript 3.0 イベント処理の以前のバージョンとの違い

ActionScript 3.0 のイベント処理と、ActionScript の以前のバージョンにおけるイベント処理との最も目立つ違いは、ActionScript 3.0 にはイベント処理に関するシステムが 1 つしかないのに対し、以前のバージョンには複数の異なるイベント処理システムが存在していたことです。このセクションでは、まず、以前のバージョンにおけるイベント処理の仕組みを概観し、続いて、それが ActionScript 3.0 でどのように変更されたかを説明します。

ActionScript の以前のバージョンにおけるイベント処理

ActionScript 3.0 より前のバージョンでは、イベント処理に関して次のように複数の方法が存在していました。

- Button インスタンスや MovieClip インスタンスに直接配置できる `on()` イベントハンドラ
- MovieClip インスタンスに直接配置できる `onClipEvent()` イベントハンドラ
- `XML.onload` や `Camera.onActivity` などのコールバック関数プロパティ
- `addListener()` メソッドを使用して登録するイベントリスナー
- DOM イベントモデルの一部を実装した UIEventDispatcher クラス

いずれのメカニズムにも、それぞれに特有のメリットとデメリットがあります。`on()` ハンドラと `onClipEvent()` ハンドラの場合は、気軽に使用できる反面、ボタンやムービークリップにコードを直接配置すると見通しが悪いため以後のプロジェクト保守作業が困難になります。コールバック関数の場合も、実装作業は簡単ですが、1 種類のイベントについてコールバック関数を 1 つしか作成できないという制約があります。イベントリスナーの実装が困難になり、リスナーオブジェクトや関数の作成だけではなく、イベントを生成するオブジェクトを備えたリスナーの登録が必要です。ただし、このオーバーヘッドの増加により、複数のリスナーオブジェクトを作成し、それらのすべてを同じイベントに登録することが可能になります。

ActionScript 2.0 用コンポーネントが開発される際にもう 1 つのイベントモデルが導入され、この新しいモデルは、UIEventDispatcher クラスで実装され、DOM Events 仕様のサブセットに基づいています。コンポーネントのイベント処理に詳しい開発者にとって、新しい ActionScript 3.0 イベントモデルへの移行は比較的容易です。

従来の各種イベントモデルの間には、使用するシンタックスがさまざまな形で重複していたり、また別の面では違っていたりするという問題がありました。たとえば、ActionScript 2.0 における一部のプロパティ (`TextField.onChanged` など) は、コールバック関数として使用することも、イベントリスナーとして使用することもできました。ところが、リスナーをサポートする 6 つのクラスと UIEventDispatcher クラスとではリスナーオブジェクトを登録する際のシンタックスが違っていました。つまり、Key、Mouse、MovieClipLoader、Selection、Stage、TextField の各クラスでは `addListener()` メソッドを使用する一方、コンポーネントのイベント処理については `addEventListener()` メソッドを呼び出す必要がありました。

他にも、イベント処理モデルが複数あることに起因して、イベントハンドラ関数のスコープが使用メカニズムによって異なるという問題がありました。このため、`this` キーワードの指す対象が一貫せず、使用するイベント処理システムによって異なっていました。

ActionScript 3.0 におけるイベント処理

ActionScript 3.0 ではイベント処理モデルが一本化され、以前のバージョンに存在していた複数のイベント処理メカニズムはそれによって置き換えられました。新しいイベントモデルは、ドキュメントオブジェクトモデル (DOM) Level 3 Events 仕様に基づいています。SWF ファイルの形式は DOM の標準仕様に準拠していませんが、表示リストと DOM の構造には、DOM イベントモデルを実装するために十分な程度の共通点があります。表示リスト上のオブジェクトは DOM 階層構造のノードに類似しており、" 表示リストオブジェクト " と " ノード " という用語は、ここでの説明を通じて同じ意味で使用されます。

Flash Player における DOM イベントモデルの実装には、" デフォルト動作 " という概念が導入されています。" デフォルト動作 " とは、特定のイベントの通常の結果として Flash Player が実行するアクションです。

デフォルト動作

イベントに応答するコードは、開発者が記述しなくてはならないのが普通です。しかし、イベントに対して何らかの決まった動作が非常によく行われる場合については、一般的な処理が Flash Player によって自動的に実行されるようになっています (開発者が特にこの動作をキャンセルするためのコードを記述した場合を除く)。Flash Player が動作を自動的に表示するので、このような動作をデフォルト動作と呼びます。

たとえば、ユーザーによって TextField オブジェクトにテキストが入力されたとき、たいていの場合は、入力されたテキストをその TextField オブジェクトに表示する必要があります。これは非常に一般的な動作なので、Flash Player に組み込まれています。この動作が不要な場合は、新しいイベント処理システムを使用してデフォルト動作をキャンセルできます。ユーザーによって TextField オブジェクトにテキストが入力されると、その入力を表す TextEvent クラスのインスタンスが Flash Player によって作成されます。入力テキストが Flash Player によって TextField オブジェクト内に表示されるのを防ぐには、その特定の TextEvent インスタンスにアクセスして `preventDefault()` メソッドを呼び出す必要があります。

デフォルト動作の中には、キャンセルできないものもあります。たとえば、ユーザーによって TextField オブジェクト内の単語がダブルクリックされると、Flash Player から MouseEvent オブジェクトが生成されます。回避できないデフォルト動作は、カーソル位置にある単語がハイライト表示される動作です。

多くの種類のイベントオブジェクトには、デフォルト動作が関連付けられていません。たとえば、ネットワーク接続が確立すると Flash Player から connect イベントが送出されますが、このイベントに関連付けられたデフォルト動作はありません。Event クラスとそのサブクラスの API ドキュメントは、各タイプのイベントを示し、関連するデフォルト動作とその動作を回避できるかどうかについて説明します。

デフォルト動作は Flash Player から送出されるイベントオブジェクトにしか関連付けられないため、ActionScript のプログラムで送出したイベントオブジェクトに対しては実行されません。この点を理解しておくことは重要です。たとえば、EventDispatcher クラスのメソッドを使用して `textInput` タイプのイベントオブジェクトを送出することはできますが、その場合はイベントオブジェクトにデフォルト動作が関連付けられません。したがって、プログラムから `textInput` イベントを送出しても、その結果 Flash Player によって TextField オブジェクトに文字が表示されることはありません。

ActionScript 3.0 のイベントリスナーに関する新機能

ActionScript 2.0 の `addListener()` メソッドを使用した経験があれば、ActionScript 2.0 のイベントリスナーモデルと ActionScript 3.0 のイベントモデルとの違いがよくわかります。次のリストでは、2 つのイベントモデル間のいくつかの主な違いについて説明します。

- ActionScript 2.0 でイベントリスナーを追加するには、ある場合には `addListener()` を使用し、他の場合には `addEventListener()` を使用します。一方、ActionScript 3.0 では、すべての場合に `addEventListener()` を使用します。
- ActionScript 2.0 にはイベントフローの仕組みがないため、`addListener()` メソッドを呼び出せるのは当該イベントをブロードキャストするオブジェクトに対してのみでした。ActionScript 3.0 では、イベントフローの一部となっているすべてのオブジェクトに対して `addEventListener()` メソッドを呼び出せます。
- ActionScript 2.0 では、関数、メソッド、オブジェクトのいずれをイベントリスナーとして使用することもできました。ActionScript 3.0 では、関数またはメソッドのみイベントリスナーとして使用できます。

イベントフロー

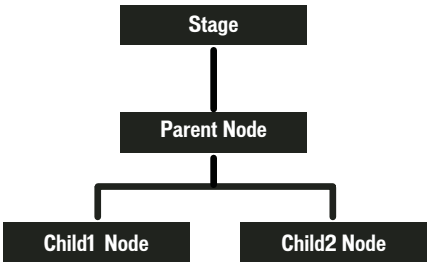
Flash Player では、イベントが発生するたびにイベントオブジェクトが送出されます。イベントターゲットが表示リスト内にない場合、Flash Player からイベントターゲットに対して直接にイベントオブジェクトが送出されます。たとえば、progress イベントオブジェクトは URLStream オブジェクトに対して直接に送出されます。イベントターゲットが表示リスト内にある場合、イベントオブジェクトは表示リストに対して送出され、表示リストの階層内を経由してイベントターゲットに到達します。

イベントフローは、表示リスト内をイベントオブジェクトが伝わる経路です。表示リストの構造はツリー状の階層になっており、そのトップの階層はステージです。ステージは特殊な表示オブジェクトコンテナであり、表示リストのルートとして機能します。ステージは flash.display.Stage クラスによって表され、表示オブジェクトを介してのみアクセスできます。すべての表示オブジェクトには、そのアプリケーションのステージを参照する `stage` という名前のプロパティが設定されています。

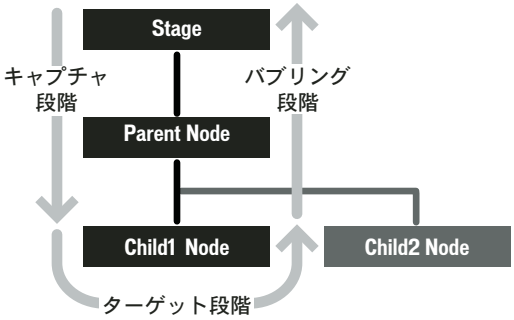
Flash Player によってイベントオブジェクトが送出されると、そのイベントオブジェクトは、ステージから " ターゲットノード " までの経路を往復して移動します。DOM Events 仕様では、ターゲットノードとはイベントターゲットを表すノードのことであると定義されています。これを言い換えれば、ターゲットノードとは当該イベントが発生した表示リストオブジェクトのことです。たとえば、ユーザーが `child1` という名前の表示リストオブジェクトをクリックすると、Flash Player は `child1` をターゲットノードとして使用するイベントオブジェクトを送出します。

概念上、イベントフローは 3 つの部分に分けられます。第 1 の部分はキャプチャ段階と呼ばれ、ステージからターゲットノードの親までのノードすべてがこれに含まれます。第 2 の部分はターゲット段階と呼ばれ、ターゲットノードだけがこれに含まれます。第 3 の部分はバブリング段階と呼ばれ、バブリング段階は、ターゲットノードの親からステージに戻る帰り道で検出されるノードで構成されます。

次の図のように、ステージを最上位とする縦の階層として表示リストの構造を考えると、各段階に付けられた名前の意味がよりよくわかります。



ユーザーによって Child1 Node がクリックされると、Flash Player により、イベントオブジェクトがイベントフローに対して送出されます。次の図で示すように、オブジェクトは Stage から始まる伝達経路を下っていき、Parent Node を経て Child1 Node に到達してから、もう一度 " バブリング " によって Parent Node から Stage へと逆方向の経路を浮上していきます。



この例で、イベントオブジェクトが下に向かって伝達されていくキャプチャ段階には、Stage および Parent Node が含まれています。ターゲット段階には Child1 Node が含まれています。バブリング段階は、ルートノードに戻る上方向への移動中に検出される Parent Node と Stage で構成されます。

このイベントフローは、従来よりいっそう強力な ActionScript 用イベント処理システムの実現に寄与しています。ActionScript の以前のバージョンでは、イベントフローの仕組みが存在しなかったため、イベントの生成元となるオブジェクトにイベントリスナーを登録することしかできませんでした。ActionScript 3.0 では、ターゲットノード自体の他、イベントフローに含まれる任意のノードにイベントリスナーを登録できます。

イベントリスナーをイベントフローに追加する機能は、ユーザーインターフェイスコンポーネントが複数のオブジェクトで構成されている場合に役立ちます。たとえばボタンオブジェクトは、ボタンのラベルとして機能するテキストオブジェクトを含むことがよくあります。リスナーをイベントフローに追加する機能がないと、ボタンの任意の場所で発生したクリックイベントに関する通知を受信できるように、ボタンオブジェクトとテキストオブジェクトの両方にリスナーを追加する必要が生じます。しかし、イベントフローがあることで、単一のイベントリスナーをボタンオブジェクトに配置し、テキストオブジェクトと、テキストオブジェクトで隠されていないボタンオブジェクトの領域の両方で発生したクリックイベントを処理できます。

ただし、イベントオブジェクトのタイプによっては、イベントフローの 3 段階のうち一部しか処理されないことがあります。たとえば、enterFrame や init などのタイプはターゲットノードに対して直接に送出され、キャプチャ段階とバブリング段階を経ることがありません。また、その他のイベントでは表示リストにないオブジェクトがターゲットになることがあります。Socket クラスのインスタンスに対して送出されるイベントなどがこれに該当します。これらのイベントオブジェクトは、キャプチャ段階またはバブリング段階を経ずに、ターゲットオブジェクトにも直接フローします。

具体的な各種イベントタイプの動作については、API ドキュメントを参照するか、実際のイベントオブジェクトのプロパティを確認してください。イベントオブジェクトのプロパティについては、次のセクションで説明します。

イベントオブジェクト

イベントオブジェクトは、新しいイベント処理システムで 2 つの主な目的に使用されます。1 つは、具体的な個々のイベントを表現し、当該イベントに関する情報を各種プロパティに格納することです。2 つ目は、イベントオブジェクトを操作し、イベント処理システムに影響を及ぼすことができるいくつかのメソッドをイベントオブジェクトに含めることです。

それらのプロパティとメソッドにアクセスしやすいよう、Flash Player API には、イベントオブジェクトすべての基本クラスとなる Event クラスが定義されています。Event クラスは、すべてのイベントオブジェクトに共通の基本的なプロパティとメソッドのセットを定義します。

このセクションでは、まず Event クラスのプロパティと、次に Event クラスのメソッドについて順に説明し、最後に、Event クラスにサブクラスが存在する理由について説明します。

Event クラスのプロパティについて

Event クラスには、イベントオブジェクトに関する重要な情報を提供する多数の読み取り専用プロパティと定数があります。次のプロパティは特に重要です。

- Event オブジェクトのタイプは定数によって表され、`Event.type` プロパティに格納されます。
- イベントのデフォルト動作を回避できるかどうかはブール値で表され、`Event.cancelable` プロパティに格納されます。
- イベントフローに関する情報はその他のプロパティに格納されます。

イベントオブジェクトのタイプ

すべてのイベントオブジェクトには、それぞれイベントタイプが設定されます。イベントタイプはストリング値として `Event.type` プロパティに格納されます。コードでイベントオブジェクトのタイプを知ることができると、タイプに応じてオブジェクトの処理方法を区別できて便利です。たとえば、次のコードでは、`myDisplayObject` に渡されるマウスクリックのイベントオブジェクトすべてに対して `clickHandler()` リスナー関数が応答するように指定しています。

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

20 数種類のイベントが Event クラスそのものに関連付けられ、Event クラス定数によって表されます。その内のいくつかを Event クラス定義から抜粋して次に示します。

```
package flash.events
{
  public class Event
  {
    // クラス定数
    public static const ACTIVATE:String = "activate";
    public static const ADDED:String    = "added";
    // 残余の定数は簡略化のために省略
  }
}
```

これらの定数を使用すると、特定のイベントタイプを簡単に参照することができます。各定数が表すストリングを直接に使用することは避け、定数名で参照するようにしてください。コードに入力した定数名にスペルミスがあった場合はコンパイラによって検出できます。しかし、ストリングの値にスペルミスがあった場合はコンパイル時のエラーとならず実行時に予期しない動作が生じ、困難なデバッグ作業が必要となる可能性があります。たとえば、イベントリスナーを登録する際は次のようなコードを使用してください。

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

次のコードは使用しません。

```
myDisplayObject.addEventListener("click", clickHandler);
```

デフォルト動作に関する情報

コードで cancelable プロパティを調べると、特定のイベントオブジェクトに対するデフォルト動作をキャンセルできるかどうか知ることができます。cancelable プロパティの値は Boolean 型で、デフォルト動作に対するキャンセルの可否を示します。デフォルト動作のキャンセルが可能なイベントは少数ですが、該当する場合は、関連付けられた動作を preventDefault() メソッドで無効化できます。詳細については、312 ページの「デフォルトイベント動作のキャンセル」を参照してください。

イベントフロー情報

Event クラスの他のプロパティには、次に示すとおり、イベントオブジェクト自体やイベントフローとの関係に関する重要な情報が格納されます。

- bubbles プロパティには、そのイベントオブジェクトの伝達経路となるイベントフローの構成要素に関する情報を含まれます。
- eventPhase プロパティは、イベントフローが現在どの段階にあるかを示します。
- target プロパティには、イベントターゲットへの参照が格納されます。
- currentTarget プロパティには、当該イベントオブジェクトを現在処理している表示リストオブジェクトへの参照が格納されます。

bubbles プロパティ

イベントオブジェクトがバブリング段階にあるとき、そのオブジェクトによって表されるイベントは " 浮上 " 中、つまり、ターゲットノードからステージへと階層の上位に向かって戻る経路の途中にあります。Event.bubbles プロパティには、そのイベントオブジェクトがバブリング段階を経由するかどうかを示すブール値が格納されます。バブリングの対象となるイベントは、必ずキャプチャ段階とターゲット段階の対象にもなるので、イベントフローの 3 段階すべてを経ることになります。このプロパティの値が true の場合、当該イベントオブジェクトは 3 つの段階をすべて経由します。値が false の場合、イベントオブジェクトはバブリング段階の対象になりません。

eventPhase プロパティ

eventPhase プロパティを調べると、そのイベントオブジェクトがイベントのどの段階にあるかを知ることができます。eventPhase プロパティには、イベントフローの 3 つの段階いずれかを示す符号なし整数値が格納されます。Flash Player API は、次のコードの抜粋に示すように、3 つの符号なし整数値に対応する 3 つの定数を含む別個の EventPhase クラスを定義します。

```
package flash.events
{
  public final class EventPhase
  {
    public static const CAPTURING_PHASE:uint = 1;
    public static const AT_TARGET:uint       = 2;
    public static const BUBBLING_PHASE:uint  = 3;
  }
}
```

これらの定数は、eventPhase プロパティの 3 つの有効な値に対応します。定数を使用すると、コードの読みやすさを向上できます。たとえば、イベントオブジェクトがターゲット段階にある場合のみ myFunc() という関数を呼び出すようにするには、次のようなコードで条件を判断します。

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
  myFunc();
}
```

target プロパティ

target プロパティには、当該イベントのターゲットであるオブジェクトへの参照が格納されます。ターゲットの扱いは単純な場合とそうでない場合があります。たとえば、マイクがアクティブになったことを示すイベントでは、イベントオブジェクトのターゲットは Microphone オブジェクトであることが明確です。しかし、ターゲットが表示リスト内にある場合は、表示リストの階層構造を考慮する必要があります。たとえば、複数の表示リストオブジェクトの重なり合う地点がユーザーによってマウスでクリックされた場合、Flash Player では、ステージから最も遠い階層にあるオブジェクトがイベントターゲットとして選ばれます。

複雑な SWF ファイルでは、target プロパティがうまく機能しないことがあります。各ボタンに対して子オブジェクトを使用した定型的な修飾処理が施されているような場合は、target プロパティの参照先がボタンではなく修飾用の子オブジェクトになってしまう状況が発生しやすいためです。このような状況では、ボタンにイベントリスナーを登録し、currentTarget プロパティを使用するのが一般的です。そうすれば、target プロパティが子オブジェクトを参照してしまう場合にも currentTarget プロパティでボタンを参照できます。

currentTarget プロパティ

currentTarget プロパティには、当該イベントオブジェクトを現在処理している表示リストオブジェクトへの参照が格納されます。調べる対象のイベントオブジェクトが現在どのノードで処理されているかが事前にわかっていないとは奇妙な状況のようにも聞こえますが、実際は特別なことではありません。なぜなら、当該イベントオブジェクトのイベントフローに含まれる任意の表示オブジェクトにリスナー関数が登録されている可能性があり、また、リスナー関数自体が任意の場所に配置されている可能性があるからです。さらに、1つのリスナー関数が複数の表示オブジェクトに登録されている場合もあります。プロジェクトのサイズが拡大し、複雑さが増すと、currentTarget プロパティがますます有用になります。

Event クラスのメソッドについて

Event クラスのメソッドは、次の 3 種類に大別されます。

- ユーティリティメソッド：イベントオブジェクトのコピーを作成する、またはストリングに変換する
- イベントフローメソッド : イベントオブジェクトをイベントフローから取り除く
- デフォルト動作メソッド：デフォルト動作をキャンセルする、またはキャンセル済みかどうかを確認する

Event クラスのユーティリティメソッド

Event クラスには 2 つのユーティリティメソッドがあります。clone() メソッドは、イベントオブジェクトのコピーを作成します。toString() メソッドは、イベントオブジェクトの各種プロパティとそれらの値を表すストリング表現を生成します。これらのメソッドの両方がイベントモデルシステムで内部的に使用されますが、開発者の一般的な使用のために公開されています。

上級開発者が Event クラスのサブクラスを作成する場合は、イベントサブクラスを正常に機能させるために、両方のユーティリティメソッドをオーバーライドして独自のバージョンを実装する必要があります。

イベントフローの停止

イベントオブジェクトがイベントフローで処理されるのを停止するには、Event.stopPropogation() メソッドまたは Event.stopImmediatePropogation() メソッドを呼び出します。これら 2 つのメソッドはほとんど同じですが、現在のノードの他のイベントリスナーを実行できるかどうかという点のみが異なります。

- Event.stopPropogation() メソッドを呼び出すと、当該イベントオブジェクトがイベントフローで次のノードに伝達されるのを防ぐことができます。ただし、現在のノードに登録されている他のイベントリスナーはすべて実行されます。

- `Event.stopImmediatePropogation()` メソッドを呼び出すと、当該イベントオブジェクトがイベントフローで次のノードに伝達されるのを防ぎ、また現在のノードに登録されている他のイベントリスナーの実行も防ぐことができます。

いずれのメソッドを呼び出した場合も、イベントに関連付けられているデフォルト動作が実行されるかどうかには影響しません。デフォルト動作をキャンセルするには、Event クラスのデフォルト動作メソッドを使用する必要があります。

デフォルトイベント動作のキャンセル

デフォルト動作のキャンセルに関しては、`preventDefault()` メソッドと `isDefaultPrevented()` メソッドの 2 つがあります。`preventDefault()` メソッドを使用すると、イベントに関連付けられたデフォルト動作をキャンセルできます。`preventDefault()` が既にイベントオブジェクトで呼び出されているかどうかをチェックするには、`isDefaultPrevented()` メソッドを呼び出します。メソッドが既に呼び出されている場合は値 `true` が返され、そうでない場合は `false` が返されます。

`preventDefault()` メソッドは、デフォルト動作のキャンセルが可能であるイベントに対してのみ使用できます。キャンセルの可否については、API ドキュメントで該当するイベントタイプの項目を参照するか、ActionScript で当該イベントオブジェクトの `cancelable` プロパティを確認してください。

デフォルト動作をキャンセルしても、イベントフローによるイベントオブジェクト処理の進行には影響しません。イベントオブジェクトをイベントフローから取り除くには、イベントフローメソッドを使用する必要があります。

Event クラスのサブクラス

多くのイベントは、Event クラスに定義されている共通のプロパティセットを使用すれば十分に表現できます。しかし、Event クラスのプロパティでは表現できない独特の性質を持ったイベントもあります。そのようなイベントのために、Flash Player API には Event クラスのサブクラスがいくつか定義されています。

各サブクラスには、そのイベントのカテゴリに特有のプロパティおよびイベントタイプが追加されています。たとえば、マウスの入力に関するイベントには、Event クラスに定義されているプロパティでは表現できない独特の性質があります。MouseEvent クラスは、Event クラスを拡張し、マウスイベントの発生位置や、発生時に特定のキーが押されていたかどうかなどの情報を格納するプロパティを追加したものです。

また、Event のサブクラスには、当該サブクラスに関連付けられたイベントタイプを表す定数が格納されます。たとえば MouseEvent クラスの場合、マウス関連のイベントタイプである `click`、`doubleClick`、`mouseDown`、`mouseUp` を表す定数がそれぞれ定義されています。

311 ページの「Event クラスのユーティリティメソッド」で説明したように、Event サブクラスを作成する場合は、`clone()` と `toString()` メソッドをオーバーライドして、サブクラスに固有の機能を提供する必要があります。

イベントリスナー

イベントハンドラとも呼ばれるイベントリスナーは、Flash Player が特定のイベントに応答して実行する関数です。イベントリスナーを登録するには 2 つの手順を実行します。まず、目的のイベントに応答して Flash Player に実行させるための関数またはクラスメソッドを作成します。これは、リスナー関数またはイベントハンドラ関数と呼ばれることもあります。次に addEventListener() メソッドを使用して、そのリスナー関数を、イベントのターゲットまたは該当するイベントフローに含まれる任意の表示リストオブジェクトに登録します。

リスナー関数の作成

ActionScript 3.0 のイベントモデルのうち、リスナー関数の作成に関する部分は DOM イベントモデルに準拠していません。DOM イベントモデルではイベントリスナーとリスナー関数が明確に区別されています。イベントリスナーは EventListener インターフェイスを実装するクラスのインスタンスで、リスナー関数はその handleEvent() というクラスのメソッドです。DOM イベントモデルにおいては、リスナー関数自体を登録するのではなく、リスナー関数を含んだクラスインスタンスを登録する形式をとります。

ActionScript 3.0 のイベントモデルには、イベントリスナーとリスナー関数との区別がありません。ActionScript 3.0 には EventListener インターフェイスがなく、リスナー関数はクラス内、クラス外のいずれに定義することもできます。さらに、リスナー関数は handleEvent() で指定する必要がありません。任意の有効な識別子を使用して指定することができます。ActionScript 3.0 においては、リスナー関数自体の名前を登録する形式をとります。

クラス外に定義したリスナー関数

次のコードは、赤い正方形のシェイプを表示する単純な SWF ファイルです。クラスの一部ではない clickHandler() という名前のリスナー関数は、赤い正方形上のマウスクリックイベントを監視します。

```
package
{
   import flash.display.Sprite;

   public class ClickExample extends Sprite
   {
      public function ClickExample()
      {
         var child:ChildSprite = new ChildSprite();
         addChild(child);
      }
   }
}

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

```
class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, clickHandler);
  }
}

function clickHandler(event:MouseEvent):void
{
  trace("clickHandler detected an event of type: " + event.type);
  trace("the this keyword refers to: " + this);
}
```

ユーザーが、正方形をクリックして生成される SWF ファイルを操作すると、Flash Player は以下のトレース出力を生成します。

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

イベントオブジェクトは、引数として clickHandler() に渡されるようになっています。これにより、リスナー関数の中でイベントオブジェクトを調べることができます。この例では、イベントオブジェクトの type プロパティを使用し、イベントがクリックイベントであることを確認しています。

また、この例では this キーワードの値も確認しています。this はグローバルオブジェクトを参照しています。なぜなら、リスナー関数をカスタムクラスやオブジェクトに属さない外部で定義しているからです。

クラスメソッドとして定義されたリスナー関数

次の例では、前出の例と同じく ClickExample クラスを定義していますが、clickHandler() 関数を ChildSprite クラスのメソッドとして定義している点が異なります。

```
package
{
  import flash.display.Sprite;

  public class ClickExample extends Sprite
  {
    public function ClickExample()
    {
      var child:ChildSprite = new ChildSprite();
      addChild(child);
    }
  }
}

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

```
class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, clickHandler);
  }
  private function clickHandler(event:MouseEvent):void
  {
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
  }
}
```

ユーザーが、赤い正方形をクリックして生成される SWF ファイルを操作すると、Flash Player は以下のトレース出力を生成します。

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

this キーワードは child という名前の ChildSprite インスタンスを参照するようになっています。この点に関しては動作が変更されており、ActionScript 2.0 とは異なります。ActionScript 2.0 のコンポーネントでは、UIEventDispatcher.addEventListener() に対してクラスメソッドを指定した場合、リスナーメソッドのスコープが、メソッドが定義されているクラスではなくイベントをブロードキャストしたコンポーネントに基づいて決まるようになっていました。つまり、この例と同じテクニックを ActionScript 2.0 で使用したとすると、this キーワードは、ChildSprite インスタンスではなくイベントをブロードキャストしたコンポーネントを参照することになります。

この従来の仕様では、リスナーメソッドからそのメソッドがあるクラス内の他のメソッドやプロパティにアクセスできないため、場合によっては非常に大きな問題が生じていました。これを回避するために、ActionScript 2.0 には、mx.util.Delegate クラスを使用してリスナーメソッドのスコープを変更するという方法が用意されていました。ActionScript 3.0 では addEventListener() の呼び出し時にバインドメソッドが作成されるため、この回避策は不要です。これにより、this キーワードで child という名前の ChildSprite インスタンスを参照し、プログラムから ChildSprite クラスの他のメソッドとプロパティにアクセスできるようになります。

望ましくないイベントリスナー登録の方法

第 3 のテクニックとして、汎用オブジェクトを作成し、そのオブジェクトのプロパティによって動的にリスナー関数を割り当てる方法がありますが、これを使用することはお勧めできません。この方法を説明するのは、ActionScript 2.0 で一般的に使用されていたという理由からです。ActionScript 3.0 では使用しないでください。このテクニックでは this キーワードの参照先がリスナーオブジェクトではなくグローバルオブジェクトになるため、望ましくありません。

次の例では、前出の例と同じく ClickExample クラスを定義していますが、リスナー関数を myListenerObj という汎用オブジェクトの一部として定義している点が異なります。

```
package
{
  import flash.display.Sprite;

  public class ClickExample extends Sprite
  {
    public function ClickExample()
    {
      var child:ChildSprite = new ChildSprite();
      addChild(child);
    }
  }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
  public function ChildSprite()
  {
    graphics.beginFill(0xFF0000);
    graphics.drawRect(0,0,100,100);
    graphics.endFill();
    addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
  }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

トレースの結果は次のようになります。

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

this が myListenerObj を参照し、その結果トレース出力が [object Object] となると考えられる場合でも、this キーワードはグローバルオブジェクトを参照します。動的なプロパティ名を addEventListener() に引数として渡す場合、Flash Player はバインドメソッドを作成できません。なぜなら、その場合に listener パラメータとして渡されるのはリスナー関数の単なるメモリアドレスであり、Flash Player がそのメモリアドレスと myListenerObj インスタンスを結び付ける手段がないからです。

イベントリスナーの管理

リスナー関数を管理するには、IEventDispatcher インターフェイスのメソッドを使用します。IEventDispatcher インターフェイスは、DOM イベントモデルにおける EventTarget インターフェイスの ActionScript 3.0 バージョンです。IEventDispatcher という名前からは、Event オブジェクトを送信 (送出) することが主な目的であるように思われますが、実際には、イベントリスナーを登録、確認、および削除する目的でメソッドを使用する機会が最も多いといえます。

IEventDispatcher インターフェイスは、次のコードに示すように、5 つのメソッドを定義します。

```
package flash.events
{
  public interface IEventDispatcher
  {
    function addEventListener(eventName:String,
              listener:Object,
              useCapture:Boolean=false,
              priority:Integer=0,
              useWeakReference:Boolean=false):Boolean;

    function removeEventListener(eventName:String,
           listener:Object,
           useCapture:Boolean=false):Boolean;

    function dispatchEvent(eventObject:Event):Boolean;

    function hasEventListener(eventName:String):Boolean;
    function willTrigger(eventName:String):Boolean;
  }
}
```

Flash Player API により IEventDispatcher インターフェイスが EventDispatcher クラスに実装され、イベントターゲットまたはイベントフローの一部になることができるすべてのクラスの基本クラスとして機能します。たとえば、DisplayObject クラスは EventDispatcher クラスを継承しています。つまり、表示リストの任意のオブジェクトが IEventDispatcher インターフェイスのメソッドにアクセスできることを意味します。

イベントリスナーの追加

`addEventListener()` は、IEventDispatcher インターフェイスで最もよく使用されるメソッドです。リスナー関数を登録する際にはこのメソッドを使用します。`type` および `listener` の 2 つのパラメータは必須です。`type` パラメータでは、イベントのタイプを指定します。`listener` パラメータでは、目的のイベントが発生したときに実行するリスナー関数を指定します。`listener` パラメータには、関数への参照またはクラスメソッドへの参照のいずれかを指定できます。

メモ

`listener` パラメータを指定するときは括弧を使用しないでください。たとえば、次の `addEventListener()` メソッド呼び出しで、`clickHandler()` 関数は括弧なしで指定されます。

```
addEventListener(MouseEvent.CLICK, clickHandler)
```

addEventListener( ) メソッドの useCapture パラメータを使用すると、イベントフローのどの段階においてリスナーをアクティブにするかを制御できます。useCapture に true を指定すると、そのリスナーはイベントフローのキャプチャ段階でアクティブになります。useCapture に false を指定すると、そのリスナーはイベントフローのターゲット段階とバブリング段階でアクティブになります。イベントフローのすべての段階でイベントを受け取るには、addEventListener( ) を 2 回、つまり useCapture を true に設定して 1 回、false に設定して 1 回呼び出す必要があります。

addEventListener( ) メソッドの priority パラメータは、DOM Level 3 イベントモデルの正式なパラメータではありません。これは、イベントリスナーをより柔軟に編成できるようにするため、ActionScript 3.0 で独自に提供されているパラメータです。addEventListener( ) を呼び出す際に priority パラメータに整数値を指定することで、イベントリスナーの優先度を設定できます。デフォルト値は 0 ですが、負の整数値または正の整数値を指定することができます。指定した数値が高いほど、イベントリスナーが早い順序で実行されます。同じ優先度で登録された複数のイベントリスナーがある場合、それらは登録順に実行されます。つまり、先に登録したイベントリスナーほど早く実行されます。

useWeakReference パラメータを使用すると、そのリスナー関数に対する参照を通常の参照にするか、弱参照にするかを指定できます。このパラメータに true を指定すると、リスナー関数が使用されなくなった後もメモリ上に残り続けるのを防ぐことができます。Flash Player では、" ガベージコレクション " というテクニックによって、使用されなくなったオブジェクトをメモリから削除します。あるオブジェクトに対して参照が 1 つも存在しなくなったとき、そのオブジェクトはもう使用されないと見なされます。弱参照はガベージコレクションのシステムで無視されるため、弱参照による参照しか受けていないリスナー関数はガベージコレクションの対象となります。

このパラメータの重要な結果の 1 つに表示オブジェクトのイベントの操作があります。通常、表示オブジェクトを表示リストから削除すると、表示オブジェクトはメモリから削除されるものと考えられます。ただし、useWeakReference パラメータを false (デフォルト) に設定して、他のオブジェクトをその表示オブジェクトのリスナーとしてサブスクライブしている場合、表示オブジェクトは、画面には表示されなくても、引き続き Flash Player のメモリに存在します。この問題を回避するには、useWeakReference パラメータを true に設定して、すべてのリスナーを表示オブジェクトにサブスクライブするか、removeEventListener( ) メソッドを使用して、すべてのイベントリスナーを表示オブジェクトから削除します。

イベントリスナーの削除

不要になったイベントリスナーを削除するには、removeEventListener( ) メソッドを使用します。不要になったリスナーは削除することをお勧めします。必須のパラメータは eventName および listener の 2 つで、addEventListener( ) メソッドの必須パラメータと同じです。前述したとおり、イベントフローのすべての段階においてイベントを受け取る場合は、useCapture に true を指定した場合と false を指定した場合の両方について、合計 2 回 addEventListener( ) を呼び出します。両方のイベントリスナーを削除するには、removeEventListener( ) を 2 回、つまり useCapture を true に設定して 1 回、false に設定して 1 回呼び出す必要があります。

イベントの送出

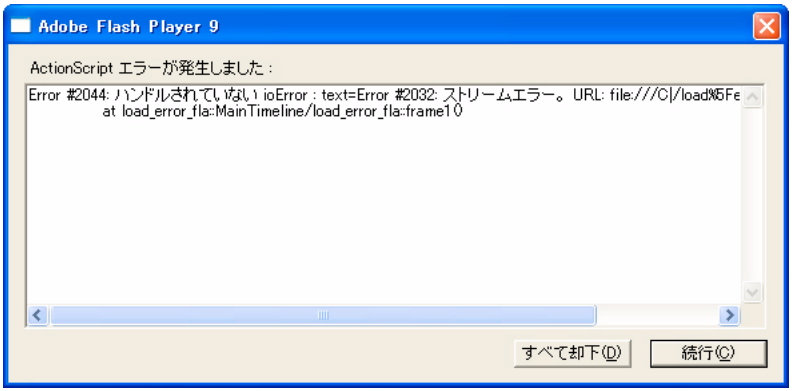
dispatchEvent() メソッドは、独自のイベントオブジェクトをイベントフローに送出する場合に使用する上級開発者向けのメソッドです。このメソッドに指定できるパラメータは 1 つだけで、Event クラスまたはそのサブクラスのインスタンスを指定します。イベントを送出すると、イベントオブジェクトの target プロパティが dispatchEvent() を呼び出したオブジェクトに設定されます。

既存のイベントリスナーの確認

IEventDispatcher インターフェイスが備える残り 2 つのメソッドは、イベントリスナーの存在に関する有用な情報を取得するために使用します。指定した表示リストオブジェクトに、指定したイベントタイプのイベントリスナーが登録されている場合、hasEventListener() メソッドは true を返します。willTrigger() メソッドも、指定した表示リストオブジェクトについてリスナーが登録されている場合に true を返します。ただし、willTrigger() では指定した表示オブジェクト自体を確認するだけでなく、イベントフローのすべての段階におけるその表示リストオブジェクトの祖先もすべて確認します。

リスナーを使用しない場合のエラーイベント

エラー処理に関して、ActionScript 3.0 における最も重要なメカニズムはイベントではなく例外ですが、非同期の操作 (ファイルのロードなど) には例外処理を使用できません。非同期の操作を実行中にエラーが発生すると、Flash Player によりエラーイベントオブジェクトが送出されます。エラーイベントに対するリスナーを作成していない場合、デバッグ版の Flash Player では、発生したエラーに関する情報がダイアログボックスに表示されます。たとえば、ファイルのロード時に無効な URL を使用すると、Flash Player のデバッガバージョンで以下のダイアログボックスが表示されます。



エラーイベントのほとんどは ErrorEvent クラスに基づいているため、Flash Player で表示するためのエラーメッセージを格納する `text` というプロパティがあります。ただし、StatusEvent クラスと NetStatusEvent クラスの 2 つはこれに該当しません。これらのクラスには `level` プロパティ (`StatusEvent.level` および `NetStatusEvent.info.level`) があります。`level` プロパティの値が "`error`" の場合、これらのイベントタイプはエラーイベントと見なされます。

エラーイベントが発生しても、それによって SWF ファイルの実行が停止することはありません。エラーイベントは、ブラウザプラグインやスタンドアロンプレーヤーのデバッガバージョンではダイアログボックスとして、オーサリングプレーヤーの出力パネルにはメッセージとして、Adobe Flex Builder 2 ではログファイルのエントリとして表示されるだけです。Flash Player のリリース版では一切表示されません。

例 : Alarm Clock

Alarm Clock の例は、アラームが鳴り出す時刻をユーザーが指定できる時計と、その時刻に表示されるメッセージで構成されます。Alarm Clock の例は、第 5 章の「日付と時刻の操作」の SimpleClock アプリケーションを基に作成されています。Alarm Clock は、次のような ActionScript 3.0 のイベントを操作するいくつかの機能を示しています。

- イベントの監視と応答
- リスナーへのイベントの通知
- カスタムイベントタイプの作成

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。Alarm Clock アプリケーションのファイルは、"Samples/AlarmClock" フォルダにあります。アプリケーションには次のファイルが含まれます。

ファイル	説明
AlarmClockApp.mxml または AlarmClockApp.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/clock/AlarmClock.as	SimpleClock クラスを拡張し、目覚まし時計の機能を追加するクラス
com/example/programmingas3/clock/AlarmEvent.as	AlarmClock クラスの `alarm` イベントのイベントオブジェクトとして機能するカスタムイベントクラス (flash.events.Event のサブクラス)
com/example/programmingas3/clock/AnalogClockFace.as	時計の丸い文字盤と時間に合わせた時針、分針、秒針を描画するコード (SimpleClock の例で説明)
com/example/programmingas3/clock/SimpleClock.as	単純な時刻合わせ機能を持つ時計インターフェイスコンポーネント (SimpleClock の例で説明)

Alarm Clock の概要

この例に示す時計の主な機能は、時刻を追跡して時計のディスプレイに表示すると共に、186 ページの「例 : 単純なアナログ時計」で説明しているとおり、SimpleClock アプリケーションコードを再利用することです。AlarmClock クラスは、アラーム時刻の設定やアラームが " 鳴り出す " ときの通知の指定など、目覚まし時計に必要な機能を追加することで、前の例の SimpleClock クラスを拡張します。

イベントが作成された目的は、何かが発生したときの通知を行うためです。AlarmClock クラスが Alarm イベントを公開し、他のオブジェクトは Alarm イベントを監視して、必要なアクションを実行できます。また、AlarmClock クラスは Timer クラスのインスタンスを使用して、アラームをトリガする時刻を判断します。AlarmClock クラスと同様に、Timer クラスは指定された時間が経過すると、他のオブジェクト (この例では AlarmClock インスタンス) に通知するイベントを提供します。ほとんどの ActionScript アプリケーションと同様に、イベントは Alarm Clock サンプルアプリケーションの機能の重要な部分を形成します。

アラームのトリガ

既に説明したように、AlarmClock クラスが実際に提供する機能は、アラームの設定とトリガに関する機能だけです。組み込みの Timer クラス (flash.utils.Timer) を使用して、指定された時間が経過した後で実行するようにコードを定義できます。AlarmClock クラスは Timer インスタンスを使用して、アラームを鳴らす時刻を判断します。

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * アラームに使用する Timer
 */
public var alarmTimer:Timer;
...
/**
 * 指定されたサイズの新しい AlarmClock のインスタンス化
 */
public override function initClock(faceSize:Number = 200):void
{
  super.initClock(faceSize);
  alarmTimer = new Timer(0, 1);
  alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

AlarmClock クラスで定義された Timer インスタンスは alarmTimer と呼ばれます。AlarmClock インスタンスに必要なセットアップ操作を実行する initClock() メソッドは、alarmTimer 変数を使用して 2 つの処理を行います。まず、Timer インスタンスが 0 ミリ秒待機し、タイマーイベントを 1 回だけトリガするように指示するパラメータを使用して、変数をインスタンス化します。alarmTimer をインスタンス化した後、コードは変数の addEventListener() メソッドを呼び出し、変数の timer イベントを監視しようとしていることを示します。指定された時間が経過した後、timer イベントを送出することで、Timer インスタンスが動作します。AlarmClock クラスは、独自のアラームを鳴らすために、いつ timer イベントが送出されたかを知る必要があります。addEventListener() を呼び出すことで、AlarmClock コードは自分自身をリスナーとして alarmTimer に登録します。2 つのパラメータは、AlarmClock クラスが timer イベントを監視しようとしていることを示し (定数 TimerEvent.TIMER で指定)、イベントが発生したときに、イベントに対する応答として AlarmClock クラスの onAlarm() メソッドを呼び出す必要があることを示しています。

実際にアラームを設定するために、AlarmClock クラスの setAlarm() メソッドが次のように呼び出されます。

```
/**
 * アラームを鳴らす時刻を設定する
 * パラメータ hour は、アラーム時刻の時間部分
 * パラメータ minutes は、アラーム時刻の分部分
 * パラメータ message は、アラームが鳴り出したときに表示するメッセージ
 * 戻り値は、アラームが鳴り出す時刻
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String =
"Alarm!"):Date
{
  this.alarmMessage = message;
  var now:Date = new Date();
  // この時刻を今日の日付で作成する
  alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

  // 指定した時刻が今日既に経過しているかどうかを判定する
  if (alarmTime <= now)
  {
    alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
  }

  // 現在設定されているアラームタイマーを停止する
  alarmTimer.reset();
  // アラームが鳴るまでの時間 ( アラーム時刻と現在の時刻の差 )
  // をミリ秒単位で計算し、
  // アラームタイマーの遅延値を設定する
  alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
  alarmTimer.start();

  return alarmTime;
}
```

このメソッドは、アラームメッセージの格納、Date オブジェクトの作成 (alarmTime)、アラームが鳴る実際の時刻の表示など、複数の機能を実行します。この章と最も関係がある部分として、メソッドの最後の数行で、alarmTimer 変数のタイマーが設定され、アクティブ化されます。まず、reset() メソッドが呼び出され、タイマーが停止され、既に実行されていた場合はリセットされます。次に、現在時刻 (now 変数で表現) が alarmTime 変数の値から減算され、アラームを鳴らすまでに経過する時間がミリ秒単位で決定されます。Timer クラスは timer イベントを絶対時間ではトリガしないため、この相対的な時間差が alarmTimer の delay プロパティに割り当てられます。最後に start() メソッドが呼び出され、タイマーが実際に開始されます。

指定された時間が経過すると、alarmTimer は timer イベントを送出します。AlarmClock クラスは、このイベントのリスナーとして onAlarm() メソッドを登録しているため、timer イベントが発生すると、onAlarm() が呼び出されます。

```
/**
 * timer イベントが送出されたときに呼び出される
 */
public function onAlarm(event:TimerEvent):void
{
  trace("Alarm!");
  var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
  this.dispatchEvent(alarm);
}
```

イベントリスナーとして登録されたメソッドは、適切なシグネチャ (つまり、メソッドのパラメータと戻り値の型のセット) を使用して定義する必要があります。メソッドを Timer クラスの timer イベントのリスナーとして使用するには、データ型が Event クラスのサブクラス TimerEvent (flash.events.TimerEvent) であるパラメータを 1 つ定義する必要があります。Timer インスタンスはイベントリスナーを呼び出すときに、TimerEvent インスタンスをイベントオブジェクトとして渡します。

アラームのその他の事項の通知

Timer クラスと同様に、AlarmClock クラスはイベントを提供し、これによってアラームが鳴り出すときに他のコードが通知を受信できます。ActionScript に組み込まれているイベント処理フレームワークをクラスで使用するには、そのクラスで flash.events.IEventDispatcher インターフェイスを実装する必要があります。最も一般的な方法として、IEventDispatcher の標準実装を提供する flash.events.EventDispatcher クラスを拡張する方法があります (または、EventDispatcher のサブクラスのいずれかを拡張します)。既に説明したように、AlarmClock クラスは SimpleClock クラスを拡張し、今度は Sprite クラスを拡張し、さらに (継承のチェーンを通じて) EventDispatcher クラスを拡張します。すなわち、AlarmClock クラスには、既に独自のイベントを提供する組み込み機能があります。

他のコードは、AlarmClock が EventDispatcher から継承した addEventListener() メソッドを呼び出すことで、AlarmClock クラスの alarm イベントの通知を受けるように登録できます。alarm イベントが生成されたことを AlarmClock インスタンスが他のコードに通知する準備ができている場合は、同様に EventDispatcher から継承された dispatchEvent() メソッドを呼び出して通知できます。

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

これらのコード行は、AlarmClock クラスの onAlarm() メソッドから取得されます (全体は前に示したとおりです)。AlarmClock インスタンスの dispatchEvent() メソッドが呼び出され、次に、AlarmClock インスタンスの alarm イベントがトリガされたことが登録済みのすべてのリスナーに通知されます。dispatchEvent() に渡されるパラメータは、リスナーメソッドに渡されるイベントオブジェクトです。この例では、特に作成された Event サブクラスである AlarmEvent クラスのインスタンスです。

カスタムアラームイベントの提供

すべてのイベントリスナーが、トリガされる特定のイベントに関する情報が設定されているイベントオブジェクトパラメータを受け取ります。多くの場合、イベントオブジェクトは Event クラスのインスタンスです。ただし、場合によっては、追加の情報をイベントリスナーに提供すると便利なことがあります。この章で既に説明したように、一般的にはこれを実現するために、新しいクラス (Event クラスのサブクラス) を定義し、このクラスのインスタンスをイベントオブジェクトとして使用します。この例では、AlarmClock クラスの alarm イベントが送出されたときに、AlarmEvent インスタンスがイベントオブジェクトとして使用されています。ここで示す AlarmEvent クラスは、alarm イベントに関する追加の情報 (具体的にはアラームメッセージ) を提供します。

```
import flash.events.Event;

/**
 * このカスタム Event クラスによって、基本の Event に message プロパティを追加
 */
public class AlarmEvent extends Event
{
  /**
   * 新しい AlarmEvent タイプの名前
   */
  public static const ALARM:String = "alarm";

  /**
   * このイベントオブジェクトと共にイベントハンドラに渡すことができる
   * テキストメッセージ
   */
  public var message:String;

  /**
   * コンストラクタ
```

```
     * パラメータ message は、アラームが鳴り出したときに表示するテキスト
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
      super(ALARM);
      this.message = message;
    }
    ...
  }
```

カスタムイベントオブジェクトクラスを作成する最善の方法は、上述の例で示したように、Event クラスを拡張するクラスを定義することです。継承された機能を補うために、AlarmEvent クラスでは、イベントに関連付けられたアラームメッセージのテキストを含むプロパティ message を定義しています。message 値はパラメータとして AlarmEvent コンストラクタに渡されます。AlarmEvent クラスは、定数 ALARM も定義します。これは、AlarmClock クラスの addEventListener() メソッドを呼び出すときに、特定のイベント (alarm) を参照するために使用できます。

カスタム機能を追加するほか、Event サブクラスでは、ActionScript イベント処理フレームワークの一部として継承された clone() メソッドをオーバーライドする必要があります。Event サブクラスでは、オプションで、toString() メソッドが呼び出されたときに返される値にカスタムイベントのプロパティを含めるように、継承された toString() メソッドをオーバーライドすることもできます。

```
  /**
   * 現在のインスタンスのコピーを作成して返す
   * 戻り値は、現在のインスタンスのコピー
   */
  public override function clone():Event
  {
    return new AlarmEvent(message);
  }

  /**
   * 現在のインスタンスのすべてのプロパティを含む String を
   * 返す
   * 戻り値は、現在のインスタンスのストリング表現
   */
  public override function toString():String
  {
    return formatToString("AlarmEvent", "type", "bubbles", "cancelable",
  "eventPhase", "message");
  }
```

オーバーライドされた clone() メソッドは、現在のインスタンスに一致するすべてのカスタムプロパティセットを持つカスタム Event サブクラスの新しいインスタンスを返す必要があります。オーバーライドされた toString() メソッドは、ユーティリティメソッド formatToString() (Event から継承) を使用して、カスタムタイプの名前およびカスタムタイプのすべてのプロパティの名前と値が設定されたストリングを提供します。

第 11 章

XML の操作

11

ActionScript 3.0 には、ECMAScript for XML (E4X) 仕様 (ECMA-357 Edition 2) に基づいたクラス群が追加されています。これらのクラスは、XML データの処理に関する強力で扱いやすい機能を備えています。E4X を使用すれば、XML データを処理するコードを、従来のプログラミングテクニックと比べて短期間で開発できます。しかも、コードの読みやすさも従来より向上します。

この章では、E4X による XML データの処理方法について説明します。

目次

XML の基礎 328
E4X による XML 処理. 332
XML オブジェクト 334
XMLList オブジェクト 336
XML 変数の初期化. 338
XML オブジェクトの構築と変形 339
XML 階層構造内の移動 340
XML 名前空間の使用 346
XML の型変換. 347
外部 XML ドキュメントの読み込み 349
インターネットから RSS データをロードする 349

XML の基礎

XML の操作の概要

XML は、コンピュータを用意に操作でき、ユーザーが無理なく簡単に記述および理解できるように、構造化情報を表現する標準手段です。XML は、eXtensible Markup Language の略語です。XML 標準は、http://www.w3.org/XML/ で参照できます。

XML は、データを分類し、その読み取り、アクセス、および操作を容易にする標準の便利な方法を提供するものです。XML は、HTML に似たツリー構造とタグ構造を使用します。次に、XML データの簡単な例を示します。

```
<song>
  <title>What you know?</title>
  <artist>Steve and the flubberblubs</artist>
  <year>1989</year>
  <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

XML データはまた、他のタグや、属性、その他の構造コンポーネント内にネストされるタグにより、複雑になる場合があります。次に、XML データのより複雑な例を示します。

```
<album>
  <title>Questions, unanswered</title>
  <artist>Steve and the flubberblubs</artist>
  <year>1989</year>
  <tracks>
    <song tracknumber="1" length="4:05">
      <title>What do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:31</lastplayed>
    </song>
    <song tracknumber="2" length="3:45">
      <title>Who do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:35</lastplayed>
    </song>
    <song tracknumber="3" length="5:14">
      <title>When do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:39</lastplayed>
    </song>
    <song tracknumber="4" length="4:19">
      <title>Do you know?</title>
      <artist>Steve and the flubberblubs</artist>
      <lastplayed>2006-10-17-08:44</lastplayed>
    </song>
  </tracks>
</album>
```

この XML ドキュメント内には、他の完全な XML 構造が含まれることに注意してください (子タグを持つ song タグなど)。この例ではまた、属性 (song タグ内の tracknumber および length) など他の XML 構造や、データではなく他のタグを含むタグ (tracks タグなど) を示しています。

XML 入門

ここでは、XML についてあまり知識をお持ちでない方のために、XML データの最も一般的な側面について簡単に説明します。XML データは、情報を構造化された形式で編成するための特別なシンタックスによりプレーンテキスト形式で記述されます。一般に、XML データセット一式を "XML ドキュメント " と言います。XML 形式ではデータは、階層構造により " エレメント " (単一のデータアイテムまたは他のエレメントのコンテナ) 内に編成されます。各 XML ドキュメントは、最上位アイテム (メインアイテム) として単一のエレメントを持ちます。このルートエレメント内部には、単一の情報が含まれることもありますが、多くの場合他のエレメントが含まれ、さらにそのエレメントに他のエレメントが含まれます (以降のエレメントについても同様です)。たとえば、この XML ドキュメントには音楽アルバムに関する情報が含まれます。

```
<song tracknumber="1" length="4:05">
  <title>What do you know?</title>
  <artist>Steve and the flubberblubs</artist>
  <mood>Happy</mood>
  <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

各エレメントは、一連の " タグ " により区別されます。エレメントの名前は、山括弧 (より小さい記号とより大きい記号) で囲まれます。エレメントの先頭を示す開始タグには、エレメント名が含まれます。

```
<title>
```

エレメントの末尾を示す終了タグでは、エレメント名の前にスラッシュが付きます。

```
</title>
```

エレメントにコンテンツが含まれない場合は、空のエレメント (自己終了エレメントとも呼ばれます) を記述できます。XML では、このエレメントは、

```
<lastplayed/>
```

次のエレメントと同じです。

```
<lastplayed></lastplayed>
```

開始タグと終了タグの間に含まれるエレメントのコンテンツに加えて、エレメントにはエレメントの開始タグで定義される " 属性 " と呼ばれる他の値を含めることもできます。たとえば、次の XML エレメントでは値 “4:19” を持つ length という名前の単一の属性を定義しています。

```
<song length="4:19"></song>
```

各 XML エレメントには、単一の値、1 つ以上の XML エレメント、または空 (空のエレメント) のいずれかであるコンテンツが含まれます、

XML の詳細について

次の Web サイトを含めて、XML の詳細について学習する場合に役立つ数多くの書籍および資料が存在します。

- W3Schools XML チュートリアル : http://w3schools.com/xml/
- XML.com: http://www.xml.com/
- XMLpitstop チュートリアル、ディスカッションリスト、その他 : http://xmlpitstop.com/

XML を操作するための ActionScript クラス

ActionScript 3.0 には、XML 構造化情報を操作するためのクラスがいくつか含まれています。次に、2 つのメインクラスを示します。

- XML : 単一の XML エレメント。複数の子エレメントまたはドキュメント内に単一値のエレメントを持つ XML ドキュメント。
- XMLList : XML エレメントセット。兄弟エレメント (XML ドキュメント階層で同じレベルにあり、同じ親に含まれているエレメント) である複数の XML エレメントが存在する場合、XMLList オブジェクトが使用されます。たとえば XMLList インスタンスは、こうした XML エレメントセット (おそらく XML ドキュメントに含まれます) を操作するための最も容易な手段として使用できます。

```
<artist type="composer">Fred Wilson</artist>
<artist type="conductor">James Schmidt</artist>
<artist type="soloist">Susan Harriet Thurndon</artist>
```

XML 名前空間にかかわるより高度な使用を可能にするために、ActionScript には Namespace クラスおよび QName クラスも含まれています。詳細については、346 ページの「XML 名前空間の使用」を参照してください。

XML を操作するためのビルトインクラスに加えて、ActionScript 3.0 には XML データにアクセスして操作するための特別な機能を提供するいくつかの演算子も含まれています。こうしたクラスおよび演算子を使用して XML を操作するこのアプローチを ECMAScript for XML (E4X) と言います。これは、ECMA-357 Edition 2 仕様により定義されています。

一般的な XML タスク

ActionScript で XML を操作する場合、次のタスクを実行することが一般的です。

- XML ドキュメントの構築 (エレメントおよび値の追加)
- XML のエレメント、値、および属性へのアクセス
- XML エレメントのフィルタ (検索)
- XML エレメントセットのループ処理
- XML クラスと String クラス間のデータの変換
- XML 名前空間の操作
- 外部 XML ファイルのロード

重要な概念と用語

次の参照リストに、この章で使用する重要な用語を示します。

- エレメント : XML ドキュメント内の単一アイテム。開始タグと終了タグ (タグも含みます) の間にあるコンテンツにより識別されます。XML エレメントには、テキストデータ、その他のエレメントを含めることができます。また、空であってもかまいません。
- 空のエレメント : 子エレメントを含まない XML エレメント。空のエレメントは、自己終了タグ (`<element/>` など) として記述されることがよくあります。
- ドキュメント : 単一の XML 構造。XML ドキュメントには、任意の数のエレメントを含めることができます (また、単一の空のエレメントのみで構成することもできます)。ただし、XML ドキュメントには、ドキュメント内の他のすべてのエレメントを含む単一の最上位エレメントが存在する必要があります。
- ノード : XML エレメントの別名。
- 属性 : エレメント内部にネストする個別の子エレメントとしてではなく、`attributename="value"` の形式でエレメントの開始タグに記述されるエレメントに関連する名前付き値。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。基本的に、この章のすべてのコードリストには、適切な `trace()` 関数呼び出しが含まれています。この章のコードリストをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   `trace()` 関数の結果が [出力] パネルに表示されます。

サンプルコードリストをテストする方法の詳細については、60 ページの「章内のコード例のテスト」を参照してください。

E4X による XML 処理

ECMAScript for XML 仕様には、XML データを処理するためのクラス群や機能が定義されています。これらのクラスは総称して E4X と呼ばれています。ActionScript 3.0 は、E4X 準拠のクラスとして、XML、XMLList、QName、および Namespace を備えています。

E4X クラスのメソッド、プロパティ、演算子についての策定作業では、次の特徴を実現することが目標とされました。

- 簡易であること : E4X では、XML データを処理するコードをできるだけ簡単に作成および解読できるようになっています。
- 一貫性があること : E4X で採用されているメソッドや考え方は、E4X それ自体の中で首尾一貫しており、また、ActionScript の他の部分とも一貫性があります。
- なじみやすいこと : XML データの操作には、よく知られているドット (.) などの演算子を使用します。

メモ

ActionScript 2.0 に存在していた従来の XML クラスは、E4X 仕様の一部である ActionScript 3.0 XML クラスと競合しないように、ActionScript 3.0 では XMLDocument というクラス名に変更されました。XMLDocument、XMLNode、XMLParser、XMLTag の各クラスは、ActionScript 3.0 では、主にレガシー機能のサポートを目的として flash.xml パッケージに含まれています。E4X の新しいクラスはコアクラスなので、使用の際にパッケージを読み込む必要はありません。この章では、ActionScript 2.0 のレガシー XML クラスについては詳しく説明しません。これらの詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の flash.xml パッケージを参照してください。

E4X を使用したデータ処理の例を次に示します。

```
var myXML:XML =
  <order>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

XML データは、実際のアプリケーションでは Web サービスや RSS フィードなどの外部ソースからロードすることが多いと考えられますが、ここでは、例を単純にするために XML データをリテラルで割り当てます。

次のコードに示すとおり、E4X には、XML のプロパティや属性へのアクセスに使用するドット (.) や属性識別用の接頭辞 (@) などのわかりやすい演算子が用意されています。

```
trace(myXML.item[0].menuName); // 出力 : burger
trace(myXML.item.(@id==2).menuName); // 出力 : fries
trace(myXML.item.(menuName=="burger").price); // 出力 : 3.95
```

XML のノードに新しい子ノードを割り当てるには、次のように appendChild() メソッドを使用します。

```
var newItem:XML =
  <item id="3">
    <menuName>medium cola</menuName>
    <price>1.25</price>
  </item>

myXML.appendChild(newItem);
```

@ および . 演算子は、読み取りだけでなく、次のようにデータの割り当てにも使用できます。

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

次のように for ループを使用すると、XML の一連のノードに対する繰り返し処理ができます。

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
  var q:int = Number(property.@quantity);
  var p:Number = Number(property.price);
  var itemTotal:Number = q * p;
  total += itemTotal;
  trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

XML オブジェクト

1 つの XML オブジェクトは、1 つの XML エレメント、属性、コメント、処理命令、またはテキストエレメントを表します。

XML オブジェクトは、" 単純内容 " を持つものと " 複合内容 " を持つものに分類されます。複合内容の XML オブジェクトとは、子ノードを持つ XML オブジェクトです。単純内容の XML オブジェクトとは、属性、コメント、処理命令、テキストノードのいずれか 1 つを含んだ XML オブジェクトです。

たとえば、次の XML オブジェクトは、コメントと処理命令が含まれた複合内容の XML オブジェクトです。

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
  <order>
    <!-- これはコメントです。-->
    <?PROC_INSTR sample ?>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

次の例に示すとおり、`comments()` および `processingInstructions()` メソッドを使用して、コメントおよび処理命令の新しい XML オブジェクトを作成できるようになりました。

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

XML プロパティ

XML クラスには次の 5 つの静的プロパティがあります。

- `ignoreComments` および `ignoreProcessingInstructions` プロパティでは、XML オブジェクトの解析時にコメントまたは処理命令を無視するかどうかを指定します。
- `ignoreWhitespace` プロパティでは、空白文字だけで区切られているエレメントタグや埋め込み式について、空白文字を無視するかどうかを指定します。
- `prettyIndent` および `prettyPrinting` プロパティは、XML クラスの `toString()` および `toXMLString()` メソッドが返すテキストを整形するために使用します。

これらのプロパティの詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

XML メソッド

次の各メソッドは、XML オブジェクトの階層構造に関する操作に使用します。

- appendChild()
- child()
- childIndex()
- children()
- descendants()
- elements()
- insertChildAfter()
- insertChildBefore()
- parent()
- prependChild()

次の各メソッドは、XML オブジェクトの属性に関する操作に使用します。

- attribute()
- attributes()

次の各メソッドは、XML オブジェクトのプロパティに関する操作に使用します。

- hasOwnProperty()
- propertyIsEnumerable()
- replace()
- setChildren()

次の各メソッドは、修飾名および名前空間に関する操作に使用します。

- addNamespace()
- inScopeNamespaces()
- localName()
- name()
- namespace()
- namespaceDeclarations()
- removeNamespace()
- setLocalName()
- setName()
- setNamespace()

次の各メソッドは、XML の内容のタイプに関する操作および判定に使用します。

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

次の各メソッドは、XML オブジェクトからストリングへの変換および整形に使用します。

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

次の各メソッドはその他の用途に使用します。

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

これらのメソッドの詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

XMLList オブジェクト

XMLList インスタンスは、XML オブジェクトの任意のコレクションを表し、完全な XML ドキュメント、XML フラグメント、または XML クエリの結果を含むことができます。

次の各メソッドは、XMLList オブジェクトの階層構造に関する操作に使用します。

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

次の各メソッドは、XMLList オブジェクトの属性に関する操作に使用します。

- attribute()
- attributes()

次の各メソッドは、XMLList オブジェクトのプロパティに関する操作に使用します。

- hasOwnProperty()
- propertyIsEnumerable()

次の各メソッドは、XML の内容のタイプに関する操作および判定に使用します。

- comments()
- hasComplexContent()
- hasSimpleContent()
- processingInstructions()
- text()

次の各メソッドは、XMLList オブジェクトからストリングへの変換および整形に使用します。

- normalize()
- toString()
- toXMLString()

次の各メソッドはその他の用途に使用します。

- contains()
- copy()
- length()
- valueOf()

これらのメソッドの詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

XML エレメントを 1 つだけ含んでいる XMLList オブジェクトに対しては、XML クラスのプロパティとメソッドをすべて使用できます。これは、XML エレメントを 1 つだけ含んでいる XMLList オブジェクトは XML オブジェクト 1 つと同等に扱われるためです。たとえば、次のコードにある doc.div は、エレメントを 1 つだけ含んでいる XMLList オブジェクトなので、XML クラスの appendChild() メソッドを使用できます。

```
var doc:XML =
     <body>
       <div>
         <p>Hello</p>
       </div>
     </body>;
doc.div.appendChild(<p>World</p>);
```

XML プロパティおよびメソッドの一覧については、334 ページの「XML オブジェクト」を参照してください。

XML 変数の初期化

XML オブジェクトに対しては、次のようにして XML リテラルを割り当てることができます。

```
var myXML:XML =
  <order>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

また、次のコードに示すとおり、XML データを含んだストリングから new コンストラクタを使用して XML オブジェクトのインスタンスを作成することもできます。

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                 + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

ストリング内の XML データが整形式でない場合 (終了タグが欠落している場合など) は、ランタイムエラーが発生します。

また、次のように XML オブジェクトに対してデータを参照で (他の変数から) 渡すこともできます。

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = “5”;
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
  // 出力 : <item id="5">Chicken</item>
```

指定した URL から XML データをロードするには、次のように URLLoader クラスを使用します。

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
  var loader:URLLoader = event.target as URLLoader;
  if (loader != null)
  {
```

```
      externalXML = new XML(loader.data);
      trace(externalXML.toXMLString());
   }
   else
   {
      trace("loader is not a URLLoader!");
   }
}
```

ソケット接続から XML データを読み取るには、XMLSocket クラスを使用します。詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の XMLSocket を参照してください。

XML オブジェクトの構築と変形

XML オブジェクトが持つプロパティリストの先頭または末尾にプロパティを追加するには、次のように prependChild() メソッドまたは appendChild() メソッドを使用します。

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
   // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

特定のプロパティの直前または直後にプロパティを追加するには、次のように insertChildBefore() メソッドまたは insertChildAfter() メソッドを使用します。

```
var x:XML =
   <body>
      <p>Paragraph 1</p>
      <p>Paragraph 2</p>
   </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

また、XML オブジェクトの作成時に次のようにして中括弧演算子 ({ および }) を使用し、参照によってデータを他の変数から渡すこともできます。

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
   var newnode:XML = new XML();
   newnode =
      <employee id={ids[i]}>
         <last>{names[i][0]}</last>
```

```
        <first>{names[i][1]}</first>
      </employee>;

    x = x.appendChild(newnode)
}
```

XML オブジェクトにプロパティまたは属性を割り当てるには、次のように = 演算子を使用します。

```
var x:XML =
  <employee>
    <lastname>Smith</lastname>
  </employee>
x.firstname = "Jean";
x.@id = "239";
```

これを実行すると、x という XML オブジェクトは次のように設定されます。

```
<employee id="239">
  <lastname>Smith</lastname>
  <firstname>Jean</firstname>
</employee>
```

+ 演算子および += 演算子を使用すると、次のように複数の XMLList オブジェクトを連結できます。

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

これを実行すると、xList という XMLList オブジェクトは次のように設定されます。

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

XML 階層構造内の移動

XML が備えている強力な機能の 1 つは、複雑にネストされたデータを、テキストの文字による連続的なストリングによって表現できることです。データを XML オブジェクトにロードすると ActionScript によって解析され、階層構造のデータとしてメモリにロードされます (XML データが整形式でない場合はランタイムエラーが発生します)。

XML オブジェクトと XMLList オブジェクトの演算子やメソッドを使用すれば、この XML データの構造内を簡単に移動できます。

XML オブジェクトの子プロパティにアクセスするには、ドット (.) 演算子と子孫アクセサ (..) 演算子を使用します。たとえば、次のような XML オブジェクトがあるとします。

```
var myXML:XML =
  <order>
    <book ISBN="0942407296">
      <title>Baking Extravagant Pastries with Kumquats</title>
```

```
      <author>
        <lastName>Contino</lastName>
        <firstName>Chuck</firstName>
      </author>
      <pageCount>238</pageCount>
    </book>
    <book ISBN="0865436401">
      <title>Emu Care and Breeding</title>
      <editor>
        <lastName>Case</lastName>
        <firstName>Justin</firstName>
      </editor>
      <pageCount>115</pageCount>
    </book>
  </order>
```

オブジェクト myXML.book は XMLList オブジェクトであり、myXML オブジェクトの book という名前を持つ子プロパティを含んでいます。それらは、myXML オブジェクトが持つ 2 つの book プロパティに対応する 2 つの XML オブジェクトです。

オブジェクト myXML..lastName は XMLList オブジェクトであり、lastName という名前を持つすべての子孫プロパティを含んでいます。それらは、myXML オブジェクトが持つ 2 つの lastName プロパティに対応する 2 つの XML オブジェクトです。

オブジェクト myXML.book.editor.lastName は XMLList オブジェクトであり、myXML オブジェクトの book という子の editor という子の、lastName という名前を持つすべての子を含んでいます。この場合、これは XML オブジェクトを 1 つだけ含んだ XMLList オブジェクトです (値が "Case" である lastName プロパティ)。

親ノードおよび子ノードへのアクセス

parent() メソッドは、XML オブジェクトの親を返します。

子リストでは、序数のインデックス値を指定することにより特定の子オブジェクトにアクセスできます。たとえば、XML オブジェクト myXML に book という名前のプロパティが 2 つある場合、book という名前の各子プロパティには、次のようにインデックス番号が対応しています。

```
myXML.book[0]
myXML.book[1]
```

特定の孫にアクセスするには、子の名前と孫の名前の両方に対してインデックス番号を指定します。

```
myXML.book[0].title[0]
```

ただし、x.book[0] に title という子が 1 つしかない場合は、次のようにインデックス参照を省略できます。

```
myXML.book[0].title
```

同じように、x に book という子が 1 つしかなく、それに title オブジェクトが 1 つしか含まれない場合は、次のように両方のインデックス参照を省略できます。

```
myXML.book.title
```

変数や式で子の名前を指定してアクセスするには、次のように child() メソッドを使用します。

```
var myXML:XML =
     <order>
       <book>
         <title>Dictionary</title>
       </book>
     </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // 出力 : Dictionary
```

属性へのアクセス

XML オブジェクトまたは XMLList オブジェクトの属性にアクセスするには、次のコードのように @ シンボル (属性識別用の演算子) を使用します。

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.@id); // 6401
```

XML オブジェクトまたは XMLList オブジェクトが持つすべての属性にアクセスするには、次のコードのように、@ シンボルにワイルドカードの * シンボルを付けます。

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

また、次のコードのように attribute() または attributes() メソッドを使用して、XML オブジェクトまたは XMLList オブジェクトが持つ特定の属性またはすべての属性にアクセスすることもできます。

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
```

```
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

メモ：属性へのアクセスには次のシンタックスを使用することもできます。その場合は下の例のようになります。

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

これらはいずれも employee.@id と同等ですが、employee.@id のシンタックスを使用することをお勧めします。

属性またはエレメントの値によるフィルタ処理

括弧演算子の (および) を使用すると、エレメント名または属性値を指定してエレメントをフィルタ処理できます。たとえば、次のような XML オブジェクトがあるとします。

```
var x:XML =
  <employeeList>
    <employee id="347">
      <lastName>Zmed</lastName>
      <firstName>Sue</firstName>
      <position>Data analyst</position>
    </employee>
    <employee id="348">
      <lastName>McGee</lastName>
      <firstName>Chuck</firstName>
      <position>Jr. data analyst</position>
    </employee>
  </employeeList>
```

この場合、次の式はいずれも有効なコードです。

- x.employee.(lastName == "McGee")－：2 番目の employee ノード
- x.employee.(lastName == "McGee").firstName－：2 番目の employee ノードが持つ firstName プロパティ
- x.employee.(lastName == "McGee").@id：2 番目の employee ノードが持つ id 属性の値
- x.employee.(@id == 347)：最初の employee ノード
- x.employee.(@id == 347).lastName：最初の employee ノードが持つ lastName プロパティ
- x.employee.(@id > 300)：両方の employee プロパティを含んだ XMLList
- x.employee.(position.toString().search("analyst") > -1)：両方の position プロパティを含んだ XMLList

存在しない属性またはエレメントを指定してフィルタ処理を実行しようとすると、Adobe Flash Player により例外がスローされます。たとえば、次のコード例では最後の行を実行しようとした時点でエラーが発生します。これは、2 番目の p エレメントに id 属性がないためです。

```
var doc:XML =
        <body>
          <p id='123'>Hello, <b>Bob</b>.</p>
          <p>Hello.</p>
        </body>;
trace(doc.p.(@id == '123'));
```

同じように、次のコード例では最後の行を実行しようとした時点でエラーが発生します。これは、2 番目の p エレメントに b プロパティがないためです。

```
var doc:XML =
        <body>
          <p id='123'>Hello, <b>Bob</b>.</p>
          <p>Hello.</p>
        </body>;
trace(doc.p.(b == 'Bob'));
```

これらのエラーを避けるには、次のコードのように、attribute() メソッドおよび elements() メソッドを使用して、一致する属性またはエレメントを持つプロパティを識別します。

```
var doc:XML =
        <body>
          <p id='123'>Hello, <b>Bob</b>.</p>
          <p>Hello.</p>
        </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

また、次のコードのように hasOwnProperty() メソッドを使用するともできます。

```
var doc:XML =
        <body>
          <p id='123'>Hello, <b>Bob</b>.</p>
          <p>Hello.</p>
        </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

for..in および for each..in ステートメントの使用

XMLList オブジェクトに対して繰り返し処理を実行する場合、ActionScript 3.0 では `for..in` ステートメントと `for each..in` ステートメントを使用できます。たとえば、次のような XML オブジェクト `myXML` と XMLList オブジェクト `myXML.item` があるとします。XMLList オブジェクトの `myXML.item` は、XML オブジェクトの `item` という 2 つのノードで構成されています。

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2' quantity='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>;
```

XMLList が持つプロパティ名のセットに対し、`for..in` ステートメントを使用して繰り返し処理を実行するには、次のようにします。

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
  total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

XMLList が持つ各種のプロパティに対し、`for each..in` ステートメントを使用して繰り返し処理を実行するには、次のようにします。

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
  total2 += prop.@quantity * prop.price;
}
```

XML 名前空間の使用

XML オブジェクト (またはドキュメント) に含まれるデータの種類は、そのオブジェクトの名前空間によって特定されます。たとえば、メッセージ通信プロトコルとして SOAP を使用する Web サービスに対して XML データを送信または提供する場合は、その XML の開始タグにおいて次のように名前空間を宣言します。

```
var message:XML =
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <soap:Body xmlns:w="http://www.test.com/weather/">
      <w:getWeatherResponse>
        <w:tempurature >78</w:tempurature>
      </w:getWeatherResponse>
    </soap:Body>
  </soap:Envelope>;
```

この名前空間は soap という接頭辞を持ち、http://schemas.xmlsoap.org/soap/envelope/ という URI において定義されています。

ActionScript 3.0 には、XML の名前空間を扱うための Namespace クラスが用意されています。前の例に示した XML オブジェクトに対する Namespace クラスの使用例を次に示します。

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // 出力 : http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

XML クラスには、名前空間の扱いに関するメソッドとして、addNamespace()、inScopeNamespaces()、localName()、name()、namespace()、namespaceDeclarations()、removeNamespace()、setLocalName()、setName()、および setNamespace() があります。

default xml namespace ディレクティブは、XML オブジェクトのデフォルトの名前空間を割り当てるために使用します。たとえば、次の例では、x1 と x2 がいずれも同じデフォルトの名前空間を持ちます。

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

XML の型変換

XML オブジェクトと XMLList オブジェクトは、String 値に変換できます。同様に、ストリングを XML オブジェクトまたは XMLList オブジェクトに変換することもできます。また、XML の属性値、名前、テキスト値はすべてストリングです。以降のセクションでは、これらすべてに関する XML 型変換について説明します。

XML および XMLList オブジェクトからストリングへの変換

XML クラスと XMLList クラスには、toString() メソッドと toXMLString() メソッドがあります。toXMLString() メソッドは、XML オブジェクトのすべてのタグ、属性、名前空間宣言、および内容を含んだストリングを返します。複合内容 (子エレメント) を持つ XML オブジェクトの場合、toString() メソッドは toXMLString() メソッドとまったく同じように機能します。テキストエレメントだけを含む単純内容の XML オブジェクトの場合、toString() メソッドは、次の例のように当該エレメントのテキストの内容だけを返します。

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
  <order>;

trace(myXML.item[0].menuName.toXMLString());
  // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
  // burger
```

toString() または toXMLString() を指定せずに trace() メソッドを使用すると、次のコードのように、データはデフォルトで toString() メソッドを使用して変換されます。

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
  <order>;

trace(myXML.item[0].menuName);
  // burger
```

コードのデバッグ用に trace() メソッドを使用する際は、toXMLString() メソッドを使用し、より完全な形でデータを出力することがしばしば役立ちます。

ストリングから XML オブジェクトへの変換

次のように new XML() コンストラクタを使用すると、ストリングから XML オブジェクトを作成できます。

```
var x:XML = new XML("<a>test</a>");
```

XML として有効でないストリングや、整形式の XML になっていないストリングを XML に変換しようとすると、次のようにランタイムエラーが発生します。

```
var x:XML = new XML("<a>test"); // エラーがスローされる
```

ストリングから属性値、名前、テキスト値への変換

XML の属性値、名前、テキスト値はすべて String データ型で表現されており、場合によっては他のデータ型に変換する必要があります。たとえば、次のコードでは Number() 関数を使用してテキスト値を数値に変換します。

```
var myXML:XML =
               <order>
                 <item>
                   <price>3.95</price>
                 </item>
                 <item>
                   <price>1.00</price>
                 </item>
               </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
  myXML.total.children()[0] = Number(myXML.total.children()[0])
                            + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.35;
```

このコードで Number() 関数を使用しないとすると、+ 演算子がストリング連結の演算子と解釈され、その結果、最終行にある trace() メソッドの出力は次のようになります。

```
01.003.95
```

外部 XML ドキュメントの読み込み

URLLoader クラスを使用すると、指定した URL から XML データをロードできます。次のコードをアプリケーションで使用する際には、XML_URL の値を実際の有効な URL に置き換えてください。

```
var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener("complete", xmlLoaded);

function xmlLoaded(event:Event):void
{
  myXML = XML(myLoader.data);
  trace("Data loaded.");
}
```

また、XMLSocket クラスを使用すると、サーバーとの間に非同期の XML ソケット接続を確立できます。詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

インターネットから RSS データをロードする

RSSViewer サンプルアプリケーションを使用して、次のような、ActionScript で XML を操作する多くの機能を示します。

- XML メソッドを使用して、RSS フィード形式の XML データにアクセスする。
- XML メソッドを使用して、テキストフィールドで使用する HTML の形式に XML データを構成する。

RSS 形式は、XML でニュースを受け取るために広く使用されています。単純な RSS データファイルの例を示します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
  <title>Alaska - Weather</title>
  <link>http://www.nws.noaa.gov/alerts/ak.html</link>
  <description>Alaska - Watches, Warnings and Advisories</description>

  <item>
    <title>
      Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
    </title>
    <link>
      http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
    </link>
    <description>
      Short Term Forecast Issued At: 2005-04-11T19:00:00
```

```
        Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
        Homepage: http://pajk.arh.noaa.gov
      </description>
    </item>
    <item>
      <title>
        Short Term Forecast - Haines Borough (Alaska)
      </title>
      <link>
        http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
      </link>
      <description>
        Short Term Forecast Issued At: 2005-04-11T19:00:00
        Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
        Homepage: http://pajk.arh.noaa.gov
      </description>
    </item>
</channel>
</rss>
```

SimpleRSS アプリケーションは、インターネットから RSS データを読み取り、ヘッドライン (タイトル)、リンク、および説明のデータを解析し、そのデータを返します。SimpleRSSUI クラスは UI を提供し、SimpleRSS クラスを呼び出します。SimpleRSS クラスはすべての XML 処理を行います。

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。RSSViewer アプリケーションのファイルは、"Samples/RSSViewer" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
RSSViewer.mxml または RSSViewer.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/rssViewer/RSSParser.as	E4X を使用して RSS (XML) データにアクセスし、対応する HTML 表現を生成するメソッドが含まれているクラス。
RSSData/ak.rss	サンプル RSS ファイル。アプリケーションは、アドビ システムズ社がホストする Flex RSS フィードで、Web から RSS データを読み取るように設定されています。ただし、使用するスキーマが Flex RSS フィードのスキーマとやや異なるこのドキュメントの RSS データを読み取るように、簡単にアプリケーションを変更できます。

XML データの読み取りと解析

RSSParser クラスには、rssXML 変数に保存されている入力 RSS データを HTML 形式の出力を含むストリングである rssOutput に変換する xmlLoaded() メソッドがあります。

ソース RSS データにデフォルトの名前空間が含まれている場合にデフォルトの XML 名前空間を設定するコードが、メソッドの先頭近くにあります。

```
if (rssXML.namespace("") != undefined)
{
  default xml namespace = rssXML.namespace("");
}
```

次の行は、ソース XML データの内容でループし、item という子孫のプロパティを確認します。

```
for each (var item:XML in rssXML..item)
{
  var itemTitle:String = item.title.toString();
  var itemDescription:String = item.description.toString();
  var itemLink:String = item.link.toString();
  outXML += buildItemHTML(itemTitle,
                itemDescription,
                itemLink);
}
```

最初の 3 行は、XML データの item プロパティの title、description、および link プロパティを表すストリング変数を単純に設定しています。次の行は、3 つの新しいストリング変数をパラメータとして使用して buildItemHTML() メソッドを呼び出し、XMLList オブジェクトの形式で HTML データを取得しています。

XMLList データの構成

HTML データ (XMLList オブジェクト) の形式は、次のとおりです。

```
<b>itemTitle</b>
<p>
  itemDescription
  <br />
  <a href="link">
    <font color="#008000">More...</font>
  </a>
</p>
```

メソッドの最初の行で、デフォルトの xml 名前空間を消去します。

```
default xml namespace = new Namespace();
```

default xml namespace ディレクティブのスコープは、関数ブロックレベルです。すなわち、この宣言のスコープは buildItemHTML() メソッドです。

次の行で、関数に渡されたストリング引数に基づいて、XMLList を構成します。

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
     // <font color="#008000"></font></a>
     // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

この XMLList オブジェクトは、ActionScript HTML テキストフィールドに適したストリングデータを表します。

`xmlLoaded()` メソッドは、`buildItemHTML()` メソッドの戻り値を使用して、ストリングに変換します。

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

RSS フィードのタイトル抽出とカスタムイベントの送信

`xmlLoaded()` メソッドは、ソース RSS XML データの情報に基づいて、`rssTitle` ストリング変数を設定します。

```
rssTitle = rssXML.channel.title.toString();
```

最後に、`xmlLoaded()` メソッドが、データが解析され使用可能になったことをアプリケーションに通知するイベントを生成します。

```
       dataWritten = new Event("dataWritten", true);
```

第 12 章

表示のプログラミング

12

ActionScript 3.0 での表示のプログラミングを使用して、Adobe Flash Player 9 のステージで表示されるエレメントを操作できます。この章では、画面上のエレメントを操作するための基本概念について説明します。ビジュアルエレメントのプログラムによる構成の詳細および表示オブジェクトのカスタムクラスの作成について学習します。

目次

表示のプログラミングの基本 354
コア表示クラス 358
表示リストによるアプローチの利点 360
表示オブジェクトの操作 363
表示オブジェクトの操作 376
表示オブジェクトのマスク 395
オブジェクトのアニメーション化 398
例 : SpriteArranger 404

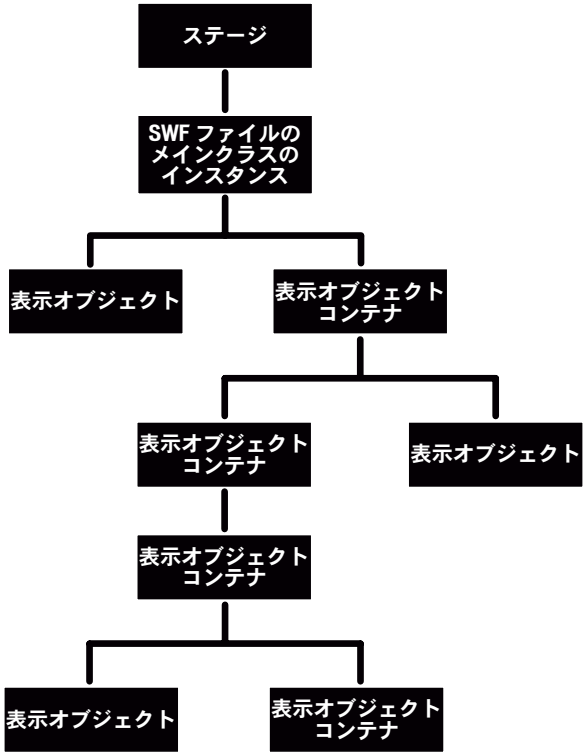
表示のプログラミングの基本

表示のプログラミングの概要

ActionScript 3.0 で構築された各アプリケーションには、" 表示リスト " と呼ばれる、表示されるオブジェクトの階層があります。表示リストには、アプリケーション内で表示されるエレメントがすべて含まれます。表示エレメントは、次の 1 つまたは複数のグループに分類されます。

- Stage

  ステージは、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には 1 つの Stage オブジェクトがあり、この中に画面の表示オブジェクトがすべて含まれます。ステージは、最上位コンテナで、表示リスト階層の最上位に位置します。



それぞれの SWF ファイルには、関連する ActionScript クラスがあります。これは、SWF ファイルのメインクラスと呼ばれます。Flash Player が HTML ページに埋め込まれた SWF ファイルを開くときにそのクラスのコンストラクタ関数が呼び出され、作成されるインスタンス (常に表示オブジェクトの型になります) 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 (詳細については、360 ページの「表示リストによるアプローチの利点」を参照してください)。

DisplayObject インスタンスの `stage` プロパティからステージにアクセスできます。詳細については、370 ページの「Stage プロパティの設定」を参照してください。

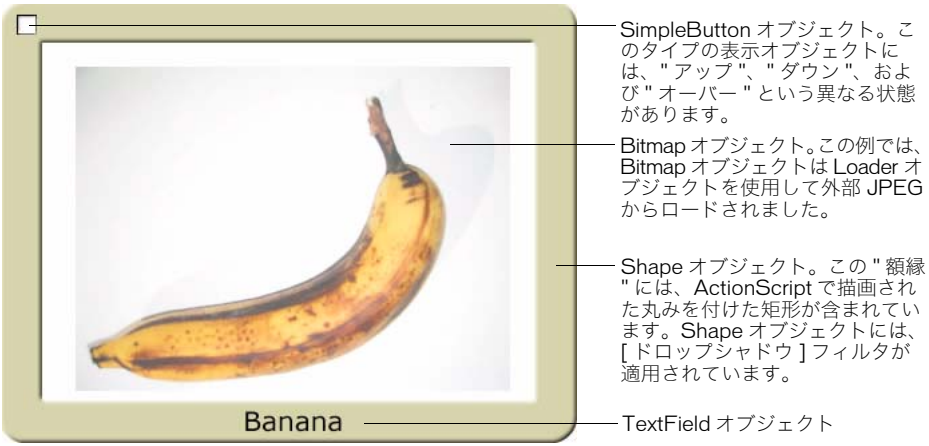
- 表示オブジェクト

ActionScript 3.0 では、アプリケーション内で画面上に表示されるすべてのエレメントのタイプは、" 表示オブジェクト " です。flash.display パッケージには、他の多くのクラスにより拡張される基本クラスである DisplayObject クラスが含まれます。これらのさまざまなクラスは、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などの各種の表示オブジェクトを表します。これらのクラスの概要については、360 ページの「表示リストによるアプローチの利点」を参照してください。

- 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特殊な型の表示オブジェクトであり、独自の視覚的表現を持つだけでなく、それ自体が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ます。

DisplayObjectContainer クラスは、DisplayObject クラスのサブクラスです。DisplayObjectContainer オブジェクトには、子リスト内の複数の表示リストを含めることができます。たとえば次の図は、さまざまな表示オブジェクトを含む Sprite という DisplayObjectContainer オブジェクトの型を示しています。



表示オブジェクトの説明では、DisplayObjectContainer オブジェクトは " 表示オブジェクトコンテナ "、または単に " コンテナ " と呼ばれます。

表示可能な表示オブジェクトはすべて DisplayObject クラスを継承しますが、そのタイプは DisplayObject クラスの特定のサブクラスです。たとえば、Shape クラスまたは Video クラスにはコンストラクタ関数がありますが、DisplayObject クラスにはコンストラクタ関数はありません。

前に説明したように、ステージは表示オブジェクトコンテナです。

一般的な表示のプログラミングのタスク

ビジュアルエレメントの作成と操作には多くの ActionScript プログラミングが関係するため、表示のプログラミングに関するタスクは数多く存在します。この章では、すべての表示オブジェクトに適用される一般的なタスクについて説明します。次のようなタスクがあります。

- 表示リストおよび表示オブジェクトコンテナの操作
  - 表示リストへの表示オブジェクトの追加
  - 表示リストからのオブジェクトの削除
  - 表示コンテナ間でのオブジェクトの移動
  - 他のオブジェクトの手前または背後でのオブジェクトの移動
- ステージの操作
  - フレームレートの設定
  - ステージの拡大 / 縮小の制御
  - フルスクリーンモードの操作
- 表示オブジェクトイベントの処理
- 表示オブジェクトの配置 (ドラッグ & ドロップ操作の作成を含む)
- サイズ変更、拡大 / 縮小、表示オブジェクトの回転
- 表示オブジェクトへのブレンドモード、カラー変換、透明度の適用
- 表示オブジェクトのマスク
- 表示オブジェクトのアニメーション化
- 外部表示コンテンツのロード (SWF ファイルやイメージなど)

本マニュアルの後半の章では、表示オブジェクトを操作するための追加タスクについて説明します。この追加タスクには、任意の表示オブジェクトに適用するタスク、および特定の型の表示オブジェクトに関連付けられたタスクの両方が含まれます。

- 表示オブジェクト上での ActionScript によるベクターグラフィックの描画 (427 ページ、第 14 章の「描画 API の使用」を参照)
- 表示オブジェクトへの図形変換の適用 (411 ページ、第 13 章の「ジオメトリの操作」を参照)
- 表示オブジェクトへのグラフィックフィルタ効果 (ぼかし、グロー、ドロップシャドウなど) の適用 (443 ページ、第 15 章の「表示オブジェクトのフィルタ処理」を参照)
- MovieClip 固有の特性の操作 (469 ページ、第 16 章の「ムービークリップの操作」を参照)

- TextField オブジェクトの操作 (485 ページ、第 17 章の「テキストの操作」を参照)
- ビットマップグラフィックの操作 (515 ページ、第 18 章の「ビットマップの操作」を参照)
- ビデオエレメントの操作 (529 ページ、第 19 章の「ビデオの操作」を参照)

重要な概念と用語

次の参照リストには、この章で使用されている重要な用語が含まれます。

- アルファ : カラーの透明の度合い (より正確には不透明の度合い) を表すカラー値。たとえば、アルファチャンネル値が 60% のカラーは、全強度の 60% だけを表示し、透明度 40% になります。
- ビットマップグラフィック : コンピュータで、カラーピクセルのグリッド (行および列) として定義されているグラフィック。一般に、ビットマップグラフィックにはデジタル写真や類似したイメージが含まれます。
- ブレンドモード : 2 つの重なり合うイメージの内容の相互関係を指定する仕様。一般に、別のイメージの上にある不透明なイメージは、その下にあるイメージを単に遮り、まったく見えなくしますが、それ以外のブレンドモードでは、そのイメージのカラーがさまざまなに混ざり合い、その内容は 2 つのイメージの組み合わせになります。
- 表示リスト : Flash Player によって目に見える画面の内容としてレンダリングされる表示オブジェクトの階層。ステージは表示リストのルートであり、ステージまたはステージのいずれかの子に関連付けられたすべての表示オブジェクトが表示リストを形成します (オブジェクトが、ステージの境界の外部にあるなど、実際にはレンダリングされていない場合でも)。
- 表示オブジェクト : Flash Player のある種の視覚的なコンテンツを表すオブジェクト。表示オブジェクトだけを表示リストに入れることができ、すべての表示オブジェクトクラスは、DisplayObject クラスのサブクラスです。
- 表示オブジェクトコンテナ : 特殊な型の表示オブジェクトであり、(通常は) 独自の視覚的表現を持つだけでなく、子オブジェクトを含むこともできます。
- SWF ファイルのメインクラス : SWF ファイルの最も外側の表示オブジェクトのビヘイビアを定義するクラスであり、概念上は、その SWF ファイル自体のクラスです。たとえば、Flash オーサリングで作成された SWF には、その他のすべてのタイムラインが含まれる " メインタイムライン " があり、SWF ファイルのメインクラスは、そのメインタイムラインがインスタンスになるクラスです。
- マスク : イメージの特定の部分を非表示にする (または、逆にイメージの特定の部分だけを表示する) 技術。イメージの非表示の部分が透明になるので、その下の内容が透けて見えます。この用語は、ペンキが特定の場所に付かないように使用するマスキングテープと似ています。
- ステージ : SWF の目に見えるすべての内容の基礎または背景である視覚的なコンテナ。
- 変換 : 図形の視覚的特性を調整すること。オブジェクトの回転、尺度の変更、シェイプの傾斜や歪み、色の変更など。
- ベクターグラフィック : 特定の特性 (太さ、長さ、サイズ、角度、位置) を使用して描かれる線またはシェイプとしてコンピュータで定義されるグラフィック。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章はビジュアルコンテンツの作成と操作に関するものであり、基本的にこの章のすべてのコードリストでビジュアルオブジェクトを作成し、それを画面に表示します。サンプルのテストには、以前の章のように変数の値を確認するのではなく、結果を Flash Player で確認する必要があります。この章のコードリストをテストするには :

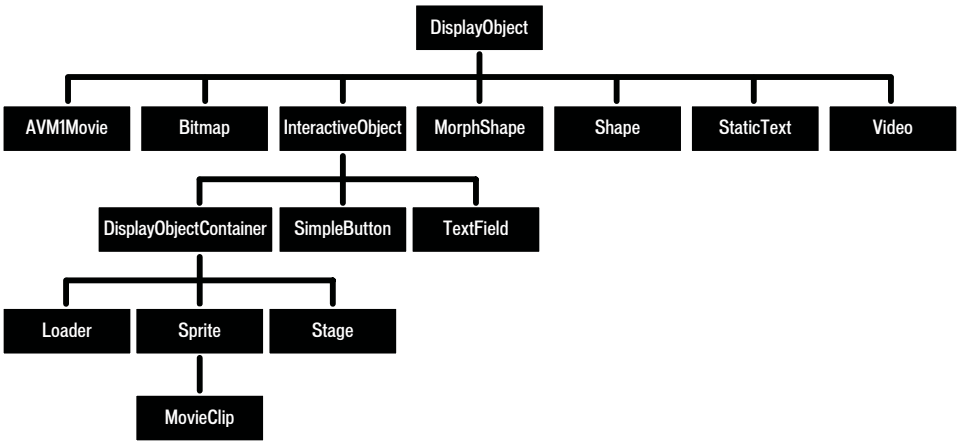
1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   コードの結果が画面に表示され、`trace()` 関数呼び出しが [出力] パネルに表示されます。

サンプルコードリストのテストの詳細については、60 ページの「章内のコード例のテスト」を参照してください。

コア表示クラス

ActionScript 3.0 flash.display パッケージには、Flash Player で表示できるビジュアルオブジェクトのクラスが含まれています。次の図は、このコア表示オブジェクトクラスのサブクラスの関係を示します。



この図は、表示オブジェクトクラスのクラス継承を示します。StaticText、TextField、および Video などの一部のクラスは、flash.display パッケージに含まれていませんが、DisplayObject クラスを継承します。

DisplayObject クラスを拡張するすべてのクラスはそのメソッドとプロパティを継承します。詳細については、363 ページの「DisplayObject クラスのプロパティとメソッド」を参照してください。

flash.display パッケージに含まれる次のクラスのオブジェクトをインスタンス化することができます。

- Bitmap - Bitmap クラスを使用して、外部ファイルからロードされたビットマップオブジェクト、または ActionScript でレンダリングされたビットマップオブジェクトを定義します。Loader クラスを使用して外部ファイルからビットマップをロードできます。ロードできるのは、GIF、JPG、PNG ファイルです。カスタムデータを使用して BitmapData オブジェクトを作成した後、そのデータを使用する Bitmap オブジェクトを作成することもできます。ビットマップがロードされたか ActionScript で作成されたかに関係なく、BitmapData クラスのメソッドを使用してビットマップを変更できます。詳細については、400 ページの「表示オブジェクトのロード」および 515 ページ、第 18 章の「ビットマップの操作」を参照してください。
- Loader - Loader クラスを使用して、外部アセット (SWF ファイルまたはグラフィック) をロードします。詳細については、400 ページの「表示コンテンツの動的ロード」を参照してください。
- Shape - Shape クラスを使用して、ベクターグラフィック (矩形、線、円など) を作成します。詳細については、427 ページ、第 14 章の「描画 API の使用」を参照してください。
- SimpleButton - SimpleButton オブジェクトは、Flash ボタンシンボルの ActionScript 表現です。SimpleButton インスタンスには、アップ、ダウン、オーバーの 3 つのボタン状態があります。
- Sprite - Sprite オブジェクトには独自のグラフィックを格納でき、子表示オブジェクトを格納することもできます。Sprite クラスは DisplayObjectContainer クラスを拡張します。詳細については、364 ページの「表示オブジェクトコンテナの操作」および 427 ページ、第 14 章の「描画 API の使用」を参照してください。
- MovieClip - MovieClip オブジェクトは、Flash オーサリングで作成されたムービークリップシンボルの ActionScript 形式です。実際に、MovieClip は、タイムラインがあること以外、Sprite オブジェクトと似ています。詳細については、469 ページ、第 16 章の「ムービークリップの操作」を参照してください。

flash.display パッケージ内にない次のクラスは、DisplayObject クラスのサブクラスです。

- flash.text パッケージに含まれる TextField クラスは、テキストの表示と入力用の表示オブジェクトです。詳細については、485 ページ、第 17 章の「テキストの操作」を参照してください。
- flash.media パッケージに含まれる Video クラスは、ビデオファイルの表示に使用される表示オブジェクトです。詳細については、529 ページ、第 19 章の「ビデオの操作」を参照してください。

flash.display パッケージに含まれる次のクラスは DisplayObject クラスを拡張しますが、これらのクラスのインスタンスを作成することはできません。その代わり、これらのクラスは他の表示オブジェクトの親クラスとなり、一般的な機能を単一のクラスに結合します。

- AVM1Movie - AVM1Movie クラスを使用して、ActionScript 1.0 または 2.0 でオーサリングされたロード済み SWF ファイルを表現します。
- DisplayObjectContainer - Loader、Stage、Sprite、MovieClip の各クラスは、DisplayObjectContainer クラスを拡張します。詳細については、364 ページの「表示オブジェクトコンテナの操作」を参照してください。
- InteractiveObject - InteractiveObject は、マウスとキーボードの操作に使用されるすべてのオブジェクトの基本クラスです。SimpleButton、TextField、Video、Loader、Sprite、Stage、および MovieClip オブジェクトはすべて InteractiveObject クラスのサブクラスです。マウスおよびキーボード操作の作成の詳細については、599 ページ、第 21 章の「ユーザー入力のキャプチャ」を参照してください。
- MorphShape - このオブジェクトは、Flash オーサリングツールでシェイプトゥイーンを作成したときに作成されます。MorphShape オブジェクトは、ActionScript を使用してインスタンス化することはできませんが、表示リストからはアクセスできます。
- Stage - Stage クラスは、DisplayObjectContainer クラスを拡張します。アプリケーションには Stage インスタンスが 1 つあります。Stage インスタンスは表示リスト階層の最上位にあります。Stage にアクセスするには、任意の DisplayObject インスタンスの `stage` プロパティを使用します。詳細については、370 ページの「Stage プロパティの設定」を参照してください。

また、flash.text パッケージ内の StaticText クラスは DisplayObject クラスを拡張しますが、インスタンスをコードで作成することはできません。静止テキストフィールドは、Adobe Flash CS3 Professional だけで作成できます。

表示リストによるアプローチの利点

ActionScript 3.0 では、各種の表示オブジェクトに対して別個のクラスがあります。ActionScript 1.0 および 2.0 では、同じタイプの多くのオブジェクトが、すべて 1 つのクラス、MovieClip クラスに含まれます。

このようなクラスの個別化および表示リストの階層構造には、次のような利点があります。

- レンダリングの効率化とメモリ使用量の削減
- 深度管理の向上
- 表示リスト内の自由な移動
- リスト外表示オブジェクト
- 表示オブジェクトのより容易なサブクラス化

上記の利点については、次のセクションで説明します。

レンダリングの効率化とファイルサイズの縮小

ActionScript 1.0 および 2.0 では、MovieClip オブジェクトでのみシェイプを描画できました。ActionScript 3.0 には、シェイプを描画できる簡単な表示オブジェクトクラスがあります。ActionScript 3.0 表示オブジェクトクラスには、MovieClip オブジェクトに含まれるメソッドとプロパティがすべて含まれるわけではないため、メモリおよびプロセッサリソースへの負担が軽減されます。

たとえば、各 MovieClip オブジェクトにはムービークリップのタイムラインのプロパティが含まれますが、Shape オブジェクトには含まれません。タイムラインを制御するためのプロパティは、大量のメモリおよびプロセッサリソースを使用する場合があります。ActionScript 3.0 では、Shape オブジェクトを使用することでパフォーマンスが向上します。Shape オブジェクトは、複雑な MovieClip オブジェクトよりオーバーヘッドが少なくなります。Flash Player では、使用されていない MovieClip プロパティを管理する必要はないため、スピードが向上し、オブジェクトが使用するメモリフットプリントが削減されます。

深度管理の向上

ActionScript 1.0 および 2.0 では、`getNextHighestDepth()` のような線状の深度管理スキームとメソッドを使用して深度を管理します。

ActionScript 3.0 には、表示オブジェクトの深度を管理するために便利なメソッドおよびプロパティが含まれる DisplayObjectContainer クラスがあります。

ActionScript 3.0 では、表示オブジェクトを DisplayObjectContainer インスタンスの子リスト内の新しい位置に移動すると、表示オブジェクトコンテナ内の他の子は、自動的に位置が変更され、表示オブジェクトコンテナ内の適切な子インデックス位置が割り当てられます。

ActionScript 3.0 では、常に表示オブジェクトコンテナの子オブジェクトをすべて見つけることができます。どの DisplayObjectContainer インスタンスにも、表示オブジェクトコンテナの子の数を一覧表示する `numChildren` プロパティがあります。また、表示オブジェクトコンテナの子リストは常にインデックスリストであるため、リスト内のインデックス位置 0 から最後のインデックス位置 (`numChildren - 1`) までのすべてのオブジェクトを調べることができます。これは、ActionScript 1.0 および 2.0 の MovieClip オブジェクトのメソッドとプロパティでは不可能でした。

ActionScript 3.0 では、表示リスト内を順に移動するのは簡単です。表示オブジェクトコンテナの子リストのインデックス番号に欠落番号がないためです。表示リスト内の移動とオブジェクトの深度管理は、ActionScript 1.0 および 2.0 より簡単になりました。ActionScript 1.0 および 2.0 では、ムービークリップに、深度順に不連続な欠落があるオブジェクトが含まれていたため、オブジェクトのリスト内を移動するのが困難でした。ActionScript 3.0 では、表示オブジェクトコンテナのそれぞれの子リストが 1 つの配列として内部にキャッシュされるので、インデックスによるルックアップが非常に高速に実行されます。また、表示オブジェクトコンテナのすべての子を非常に高速にループすることもできます。

ActionScript 3.0 では、DisplayObjectContainer クラスの `getChildByName()` メソッドを使用して表示オブジェクトコンテナの子にアクセスすることもできます。

表示リスト内の自由な移動

ActionScript 1.0 および 2.0 では、ベクターシェイプなど、Flash オーサリングツールで描画されたオブジェクトにはアクセスできませんでした。ActionScript 3.0 では、表示リスト内のすべてのオブジェクトにアクセスできます。これには、ActionScript を使用して作成されたオブジェクト、および Flash オーサリングツールで作成されたすべての表示オブジェクトの両方が含まれます。詳細については、368 ページの「表示リスト内の移動」を参照してください。

リスト外表示オブジェクト

ActionScript 3.0 では、表示リストに表示されない表示オブジェクトを作成できます。これらのオブジェクトを " リスト外 " 表示オブジェクトと呼びます。表示オブジェクトは、既に表示リストに追加されている DisplayObjectContainer インスタンスの `addChild()` または `addChildAt()` メソッドを呼び出したときにのみ、表示可能な表示リストに追加されます。

リスト外表示オブジェクトを使用すると、複数の表示オブジェクトが含まれる複数の表示オブジェクトコンテナを持つ表示オブジェクトなど、複雑な表示オブジェクトを構成できます。表示オブジェクトをリスト外にしておくことで、これらの表示オブジェクトをレンダリングする処理時間をかけずに、複雑なオブジェクトを構成できます。これにより、必要に応じてリスト外表示オブジェクトを表示リストに追加できます。また、表示オブジェクトコンテナの子を表示リスト内および外に、または表示リスト内の必要な場所に任意に移動できます。

表示オブジェクトのより容易なサブクラス化

ActionScript 1.0 および 2.0 では、通常は SWF ファイルに新しい MovieClip オブジェクトを追加して、基本シェイプを作成するか、またはビットマップを表示します。ActionScript 3.0 では、DisplayObject クラスに Shape、Bitmap などの多くのビルトインサブクラスが含まれています。ActionScript 3.0 のクラスは特定のタイプのオブジェクト用に特化されているので、ビルトインクラスの基本サブクラスを簡単に作成できます。

たとえば、ActionScript 2.0 で円を描画するには、カスタムクラスのオブジェクトをインスタンス化したときに MovieClip クラスを拡張する CustomCircle クラスを作成できました。しかし、このクラスには、クラスに適用されない MovieClip クラスの多数のプロパティやメソッド (`totalFrames` など) も含まれてしまいます。ActionScript 3.0 では、Shape オブジェクトを拡張する CustomCircle クラスを作成できるため、MovieClip クラスに含まれている関係のないプロパティやメソッドが含まれることはありません。次のコードは、CustomCircle クラスの例です。

```
import flash.display.*;

private class CustomCircle extends Shape
{
  var xPos:Number;
  var yPos:Number;
```

```
  var radius:Number;
  var color:uint;
  public function CustomCircle(xInput:Number,
                               yInput:Number,
                               rInput:Number,
                               colorInput:uint)
  {
    xPos = xInput;
    yPos = yInput;
    radius = rInput;
    color = colorInput;
    this.graphics.beginFill(color);
    this.graphics.drawCircle(xPos, yPos, radius);
  }
}
```

表示オブジェクトの操作

ステージ、表示オブジェクト、表示オブジェクトコンテナ、および表示リストの基本概念を理解できたので、このセクションでは、ActionScript 3.0 での表示オブジェクトの操作について詳しく説明します。

DisplayObject クラスのプロパティとメソッド

すべての表示オブジェクトは、DisplayObject クラスのサブクラスです。従って、表示オブジェクトは DisplayObject クラスのプロパティとメソッドを継承します。継承したプロパティは、すべての表示オブジェクトに適用される基本プロパティです。たとえば、表示オブジェクトには、表示オブジェクトコンテナ内のオブジェクトの位置を指定する x プロパティと y プロパティがあります。

DisplayObject クラスコンストラクタを使用して DisplayObject インスタンスを作成することはできません。Sprite など別の型のオブジェクト (DisplayObject クラスのサブクラスであるオブジェクト) を作成することにより、new 演算子でオブジェクトをインスタンス化する必要があります。また、カスタム表示オブジェクトクラスを作成する場合は、たとえば、Shape クラスや Sprite クラスなど、使用可能なコンストラクタ関数を持ついずれかの表示オブジェクトサブクラスのサブクラスを作成する必要があります。詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の DisplayObject クラスの説明を参照してください。

表示リストへの表示オブジェクトの追加

表示オブジェクトをインスタンス化する場合、表示リストにある表示オブジェクトコンテナに表示オブジェクトインスタンスを追加するまで、表示オブジェクトは画面 (ステージ) に表示されません。たとえば、次のコードでは、コードの最後の行を省略すると myText TextField オブジェクトは表示されません。コードの最後の行で、this キーワードは、既に表示リストに追加されている表示オブジェクトコンテナを参照する必要があります。

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

ステージにビジュアルエレメントを追加すると、そのエレメントは、Stage オブジェクトの " 子 " になります。アプリケーション内にロードされた最初の SWF ファイル (HTML ページに埋め込む SWF ファイルなど) は、自動的に Stage オブジェクトの子として追加されます。これは、Sprite クラスを拡張する型のオブジェクトです。

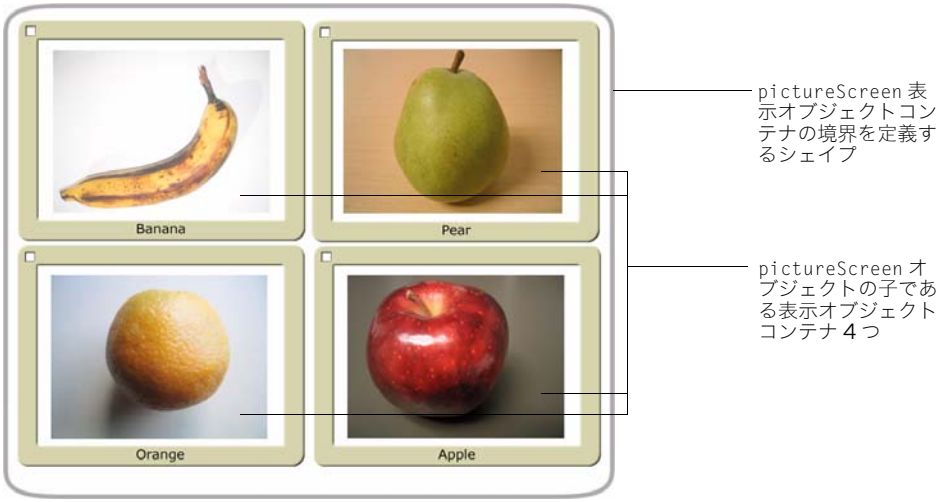
ActionScript を使用せずに作成した表示オブジェクトは、表示リストに追加されます。たとえば、Adobe Flex Builder 2 で MXML タグを追加したり、Flash のステージにアイテムを配置することで作成した表示オブジェクトです。これらの表示オブジェクトは ActionScript を使用して追加されていませんが、ActionScript からアクセスできます。たとえば、次のコードは、ActionScript ではなくオーサリングツールで追加された button1 というオブジェクトの幅を調整します。

```
button1.width = 200;
```

表示オブジェクトコンテナの操作

DisplayObjectContainer オブジェクトが表示リストから削除されたか、何らかの方法で移動または変換された場合、DisplayObjectContainer 内の各表示オブジェクトも、削除、移動、変換されます。

表示オブジェクトコンテナはそれ自体が表示オブジェクトの 1 つの型であり、別の表示オブジェクトコンテナに追加できます。たとえば、次の図は、アウトラインシェイプ 1 つと (PictureFrame 型の) 表示オブジェクトコンテナ 4 つを含む表示オブジェクトコンテナ pictureScreen を示します。



表示リストに表示オブジェクトを表示するには、表示リスト上にある表示オブジェクトコンテナに表示オブジェクトを追加する必要があります。これには、コンテナオブジェクトの addChild() メソッドまたは addChildAt() メソッドを使用します。たとえば、次のコードの最後の行がなければ、myTextField オブジェクトは表示されません。

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

このサンプルコードでは、コードを格納する MovieClip 表示オブジェクトコンテナを this.root によって指定します。実際のコードでは、別のコンテナを指定できます。

表示オブジェクトコンテナの子リスト内の特定の位置に子を追加するには、addChildAt() メソッドを使用します。子リスト内の 0 から始まるインデックス位置は、表示オブジェクトのレイヤー (前から後ろへの順) に対応しています。たとえば、次のような 3 つの表示オブジェクトがあるとします。各オブジェクトは、Ball と呼ばれるカスタムクラスから作成されました。

コンテナ内のこれらの表示オブジェクトのレイヤーは、addChildAt() メソッドを使用して調整できます。たとえば、次のようなコードがあるとします。

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

このコードを実行後、表示オブジェクトは、container DisplayObjectContainer オブジェクト内で次のように配置されます。オブジェクトのレイヤーに注意してください。

表示リストの一番上にオブジェクトを再配置するには、リストにオブジェクトを再度追加するだけです。たとえば、前のコードの実行後、ball_A をスタックの最上部に移動するには、次のコード行を使用します。

```
container.addChild(ball_A);
```

このコードは、ball_A を container の表示リスト内から削除してリストの最上部に再度追加します。その結果、スタックの最上部に移動されます。

表示オブジェクトのレイヤーの順序を確認するには、getChildAt() メソッドを使用します。getChildAt() メソッドは、メソッドに渡すインデックス番号に基づいてコンテナの子オブジェクトを返します。たとえば、次のコードは、container DisplayObjectContainer オブジェクトの子リスト内の異なる位置にある表示オブジェクトの名前を表示します。

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

親コンテナの子リストから表示オブジェクトを削除すると、リスト上でそれより上位にあるエレメントの子インデックス内の位置がそれぞれ下がります。たとえば、引き続き前のコードを使用すると、次のコードは、子リスト内の下位にある表示オブジェクトが削除された場合、container DisplayObjectContainer 内の位置 2 にあった表示オブジェクトが位置 1 に移動することを示します。

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

removeChild() および removeChildAt() メソッドは、表示オブジェクトインスタンスを完全に削除しません。コンテナの子リストから削除するだけです。インスタンスは、別の変数で参照できますオブジェクトを完全に削除するには、delete 演算子を使用します。

表示オブジェクトには親コンテナが 1 つだけあるので、その 1 つだけの表示オブジェクトコンテナに表示オブジェクトのインスタンスを追加できます。たとえば、次のコードは、表示オブジェクト tf1 が 1 つだけのコンテナ (この例では、DisplayObjectContainer クラスを拡張する Sprite) 内にあることを示します。

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // テキスト 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // テキスト 1
```

表示オブジェクトコンテナ内に含まれる表示オブジェクトを別の表示オブジェクトコンテナに追加すると、表示オブジェクトは最初の表示オブジェクトコンテナの子リストから削除されます。

上記で説明したメソッド以外にも、DisplayObjectContainer クラスでは、子表示オブジェクトを操作するための次のようなメソッドが定義されています。

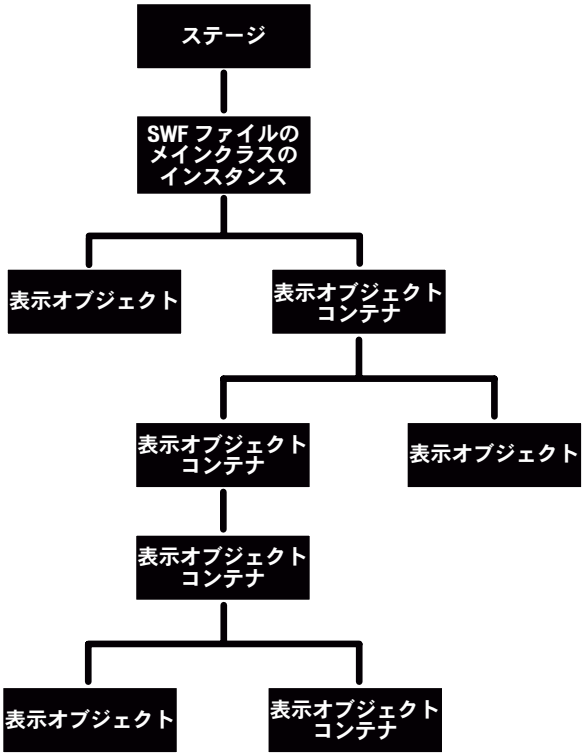
- contains(): 表示オブジェクトが DisplayObjectContainer の子であるかどうかを指定します。
- getChildByName(): 表示オブジェクトを名前によって取得します。
- getChildIndex(): 表示オブジェクトのインデックス位置を返します。
- setChildIndex(): 子表示オブジェクトの位置を変更します。
- swapChildren(): 2 つの表示オブジェクトの重ね順を入れ替えます。
- swapChildrenAt(): 指定されたインデックス位置にある 2 つの表示オブジェクトの重ね順を入れ替えます。

詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

表示リスト外の表示オブジェクト、すなわち Stage の子である表示オブジェクトコンテナに含まれない表示オブジェクトを " リスト外 " 表示オブジェクトと呼びます。

表示リスト内の移動

既に見たように、表示リストはツリー構造です。ツリーの一番上には、複数の表示オブジェクトを含めることができるステージがあります。これらの表示オブジェクトは、それ自体が表示オブジェクトコンテナであり、別の表示オブジェクト、つまり表示オブジェクトコンテナを含めることができます。



DisplayObjectContainer クラスには、表示オブジェクトコンテナの子リストを使用して表示リスト内を移動するためのプロパティとメソッドが含まれています。たとえば、次のようなコードがあるとします。このコードは、2 つの表示オブジェクト `title` と `pict` を `container` オブジェクト (Sprite、Sprite クラスは DisplayObjectContainer クラスを拡張します) に追加します。

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
```

```
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

getChildAt() メソッドは、特定のインデックス位置にある表示リストの子を返します。

```
trace(container.getChildAt(0) is TextField); // true
```

子オブジェクトに名前でアクセスすることもできます。表示オブジェクトには name プロパティがあります。name プロパティを割り当てなかった場合は、Flash Player では "instance1" などのデフォルト値が割り当てられます。たとえば、次のコードは、getChildByName() メソッドを使用して "banana loader" という名前で子表示オブジェクトにアクセスする方法を示します。

```
trace(container.getChildByName("banana loader") is Loader); // true
```

getChildByName() メソッドを使用すると、getChildAt() メソッドを使用するよりパフォーマンスが低下します。

表示オブジェクトコンテナには表示リストの子オブジェクトとして別の表示オブジェクトコンテナを含めることができるため、ツリーのようにアプリケーションの表示リスト内を移動できます。たとえば、前述のコードの抜粋では、pict Loader オブジェクトのロード操作が完了すると、pict オブジェクトにビットマップである子表示オブジェクトがロードされます。このビットマップ表示オブジェクトにアクセスするために、pict.getChildAt(0) を記述することができます。また、container.getChildAt(0) == pict であるため、container.getChildAt(0).getChildAt(0) を記述することもできます。

次の関数は、表示オブジェクトコンテナから表示リストのインデントされた trace() を出力します。

```
function traceDisplayList(container:DisplayObjectContainer,
                          indentString:String = ""):void
{
  var child:DisplayObject;
  for (var i:uint=0; i < container.numChildren; i++)
  {
    child = container.getChildAt(i);
    trace(indentString, child, child.name);
    if (container.getChildAt(i) is DisplayObjectContainer)
    {
      traceDisplayList(DisplayObjectContainer(child), indentString + "")
    }
  }
}
```

Stage プロパティの設定

Stage クラスは、DisplayObject クラスのほとんどのプロパティとメソッドをオーバーライドします。オーバーライドされたプロパティまたはメソッドのいずれかを呼び出した場合、Flash Player は例外をスローします。たとえば、Stage オブジェクトは、アプリケーションのメインコンテナとして位置が固定されているため、x プロパティや y プロパティを持ちません。x プロパティと y プロパティは、コンテナを基準として表示オブジェクトの位置を参照します。Stage オブジェクトは別の表示オジェクトコンテナに含まれていないので、この 2 つのプロパティは適用されません。

メモ

Stage クラスの一部のプロパティおよびメソッドは、最初にロードされた SWF ファイルと同じセキュリティサンドボックス内にないオブジェクトを表示するために使用することはできません。詳細については、744 ページの「ステージのセキュリティ」を参照してください。

再生フレームレートの制御

Stage クラスの `framerate` プロパティは、アプリケーションにロードされたすべての SWF ファイルのフレームレートを設定するために使用します。詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

ステージの拡大 / 縮小の制御

Flash Player 画面のサイズが変更されると、Flash Player はそれに合わせてステージの内容を自動的に調節します。ステージの内容を調節する方法は、Stage class の `scaleMode` プロパティで指定します。このプロパティは、flash.display.StageScaleMode クラスの定数として定義された 4 つの異なる値に設定できます。

`scaleMode` の 3 つの値 (`StageScaleMode.EXACT_FIT`、`StageScaleMode.SHOW_ALL`、および `StageScaleMode.NO_BORDER`) については、Flash Player はステージの内容を境界線に合わせて拡大 / 縮小します。3 つのオプションごとに、拡大 / 縮小の方法が異なります。

- `StageScaleMode.EXACT_FIT` は、縦横比を変えずに SWF を拡大 / 縮小します。
- `StageScaleMode.SHOW_ALL` は、テレビでワイドスクリーンサイズの映画を見るときのように黒い帯状のボーダーを表示するかどうかを指定します。
- `StageScaleMode.NO_BORDER` は、内容を部分的に切り詰めるかどうかを指定します。

別の方法として、`scaleMode` を `StageScaleMode.NO_SCALE` に設定すると、ユーザーが Flash Player ウィンドウのサイズを変更してもステージの内容は定義済みのサイズのまま変わりません。この拡大 / 縮小モードでのみ、Stage クラスの `width` および `height` プロパティを使用して、サイズ変更後の Flash Player ウィンドウの実際のピクセル寸法を指定できます (他の拡大 / 縮小モードでは、`stageWidth` および `stageHeight` プロパティは常に SWF の元の幅と高さを反映します)。さらに、`scaleMode` を `StageScaleMode.NO_SCALE` に設定した場合、SWF ファイルがサイズ変更されると Stage クラスの `resize` イベントが送出されるので、それに合わせて調整することができます。

結果として、scaleMode を StageScaleMode.NO_SCALE に設定することにより、必要な場合は、ウィンドウのサイズ変更に応じて画面の内容をより詳細に制御できます。たとえば、ビデオとコントロールバーが含まれる SWF の場合、ステージがサイズ変更されたときに、コントロールバーのサイズを変えずに、ビデオウィンドウのサイズだけをステージサイズの変更に合わせて変更できます。例を次のコードに示します。

```
// videoScreen はビデオを含む表示オブジェクト（Video インスタンスなど）
// であり、ステージの左上に表示される。
// SWF がサイズ変更されると、サイズ変更される。

// controlBar はいくつかのボタンを含む表示オブジェクト（Sprite など）
// であり、常にステージの左下角（videoScreen の下）に表示される。
// SWF がサイズ変更されても、
// サイズ変更されない。

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = videoScreen.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;

function resizeDisplay(event:Event):void
{
  var swfWidth:int = swfStage.stageWidth;
  var swfHeight:int = swfStage.stageHeight;

  // Resize the video window.
  var newVideoHeight:Number = swfHeight - controlBar.height;
  videoScreen.height = newVideoHeight;
  videoScreen.scaleX = videoScreen.scaleY;

  // Reposition the control bar.
  controlBar.y = newVideoHeight;
}

swfStage.addEventListener(Event.RESIZE, resizeDisplay);
```

フルスクリーンモードの操作

フルスクリーンモードを使用すると、ビューアのモニタ全体に SWF を表示できるようになります。境界線やメニューバーなどは表示されません。Stage クラスの displayState プロパティは、SWF のフルスクリーンモードのオンとオフを切り替えるために使用されます。displayState プロパティは、flash.display.StageDisplayState クラスの定数で定義されるいずれかの値に設定できます。フルスクリーンモードをオンにするには、displayState を StageDisplayState.FULL_SCREEN に設定します。

```
// mySprite は Sprite インスタンスで、表示リストに追加済み
mySprite.stage.displayState = StageDisplayState.FULL_SCREEN;
```

フルスクリーンモードを終了するには、displayState プロパティを StageDisplayState.NORMAL に設定します。

```
mySprite.stage.displayState = StageDisplayState.NORMAL;
```

また、別のウィンドウにフォーカスを切り替えるか、ESC キー (すべてのプラットフォーム共通)、Ctrl キーと W キー (Windows の場合)、Command キーと W キー (Mac の場合)、Alt キーと F4 キー (Windows の場合) のいずれかの組み合わせを使用することで、フルスクリーンモードを終了することができます。

フルスクリーンモードでのステージの拡大 / 縮小操作は、通常モードの場合と同様です。拡大 / 縮小は、Stage クラスの scaleMode プロパティで制御します。従来どおり、scaleMode プロパティを StageScaleMode.NO_SCALE に設定した場合、Stage の stageWidth プロパティと stageHeight プロパティは、SWF が占める画面領域のサイズ (この場合、画面全体) に従って変化します。

Stage クラスの fullScreen イベントを使用すると、フルスクリーンモードのオンとオフを検出して応答することができます。たとえばこの例のように、フルスクリーンモードに移行するとき、またはフルスクリーンモードを終了するときに、画面アイテムの再配置、追加、削除を行うことができます。

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
   if (event.fullScreen)
   {
      // 入力テキストフィールドを削除する。
      // フルスクリーンモードを終了するボタンを追加する。
   }
   else
   {
      // 入力テキストフィールドを再度追加する。
      // フルスクリーンモードを終了するボタンを削除する。
   }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

このコードが示すように、fullScreen イベントのイベントオブジェクトは、flash.events.FullScreenEvent クラスのインスタンスです。このクラスには、フルスクリーンモードが有効 (true) であるか無効 (false) であるかを示す fullScreen プロパティがあります。

ActionScript でフルスクリーンモードを操作するときは、次の点に注意してください。

- フルスクリーンモードは、マウスがクリックされた場合 (右クリックを含む) またはキーが押された場合の応答として、ActionScript によってのみ初期化することができます。
- マルチモニタの場合、SWF コンテンツが画面一杯に拡大されるのは、1 つのモニタだけです。Flash Player はメトリックを使用して SWF コンテンツの割合が一番大きいモニタを判別し、このモニタをフルスクリーンモード用に使用します。
- HTML ページに埋め込まれた SWF ファイルの場合、Flash Player を埋め込む HTML コードには、名前と値をそれぞれ allowFullScreen と true に設定した param タグと embed 属性を次のように指定する必要があります。

  ```
  <object>
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowfullscreen="true" />
  </object>
  ```

  Web ページ内の JavaScript を使用して SWF 埋め込みタグを生成する場合、JavaScript を変更して、allowFullScreen パラメータのタグや属性を追加する必要があります。たとえば、HTML ページで AC_FL_RunContent( ) 関数 (Flex Builder および Flash の両方で生成された HTML ページで使用される) を使用する場合、allowFullScreen パラメータを該当の関数呼び出しに対して次のように追加する必要があります。

  ```
  AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
  ```

  これは、スタンドアローンの Flash Player で実行される SWF ファイルには適用されません。
- キーボードイベントや TextFields インスタンスへのテキスト入力など、キーボードに関連する ActionScript はすべて、フルスクリーンモードでは無効になります。ただし、フルスクリーンモードを終了するためのキーボードショートカットは除きます。

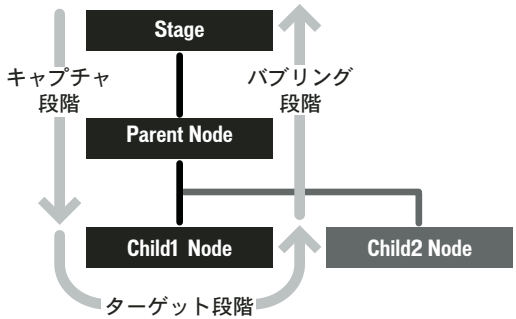
セキュリティに関する重要な制約は、この他にもいくつかあります。これらについては、732 ページの「セキュリティサンドボックス」で説明します。

表示オブジェクトのイベントの処理

DisplayObject クラスは EventDispatcher クラスを継承します。これは、どの表示オブジェクトでもイベントモデルに完全に参加できることを意味します (299 ページ、第 10 章の「イベントの処理」を参照してください)。各表示オブジェクトは、EventDispatcher クラスから継承した独自の `addEventListener()` メソッドを使用して、特定のイベントを監視することができます。ただし、これは、監視しているオブジェクトがそのイベントのイベントフローの一部である場合に限ります。

Flash Player によってイベントオブジェクトが送出されると、そのイベントオブジェクトはステージからイベントが発生した表示オブジェクトへと往復します。たとえば、ユーザーが `child1` という名前の表示オブジェクトをクリックすると、Flash Player により、ステージから表示リスト階層を下がって `child1` 表示オブジェクトまでイベントオブジェクトが送出されます。

次の図に示されているように、概念上イベントフローは 3 段階に分けられます。



詳細については、299 ページ、第 10 章の「イベントの処理」を参照してください。

表示オブジェクトのイベント操作で注意すべき点は、表示オブジェクトを表示リストから削除するとき、(ガベージコレクションにより) 表示オブジェクトが自動的にメモリから削除されたかどうかによってイベントリスナーの効果が異なることです。イベントのリスナーとしてサブスクライブされたオブジェクトを含む表示オブジェクトの場合、この表示オブジェクトには該当のリスナーオブジェクトへの参照が設定されているため、表示リスからの削除時にも削除されません。詳細については、317 ページの「イベントリスナーの管理」を参照してください。

DisplayObject サブクラスの選択

表示オブジェクトを操作する場合に重要なのは、複数のオプションを選択する場合に、どの表示オブジェクトをどの目的で使用するかを決定することです。これを決定する場合のガイドラインを次に示します。これらのガイドラインは、クラスのインスタンスが必要か、あるいは作成するクラスの基本クラスを選択するかを判断する場合にも適用できます。

- 他の表示オブジェクトのコンテナとなるオブジェクトを必要としない場合 (すなわち、スタンドアローンの画面エレメントとなるオブジェクトのみを必要とする場合)、次の DisplayObject サブクラスまたは InteractiveObject サブクラスのいずれかを使用目的に合わせて選択します。
  - ビットマップイメージを表示するための Bitmap。
  - テキストを追加するための TextField。
  - ビデオを表示するための Video。
  - 画面上のコンテンツを描画するための " キャンバス " 用の Shape。具体的には、画面上でシェイプを描画するためのインスタンスを作成する必要があり、このオブジェクトが他の表示オブジェクト用のコンテナとはならない場合、Sprite や MovieClip ではなく Shape を使用することで、パフォーマンスが大幅に改善します。
  - Flash オーサリング固有のアイテム用の MorphShape、StaticText、SimpleButton (これらのクラスのインスタンスをプログラムによって作成することはできません。ただし、これらのデータ型を持つ変数を作成し、Flash オーサリングツールで作成されたアイテムを参照することはできます)。
- メインの Stage を参照する変数が必要な場合、そのデータ型として Stage クラスを使用します。
- 外部 SWF ファイルやイメージファイルをロードするためのコンテナが必要な場合、Loader インスタンスを使用します。ロードされたコンテンツは、Loader インスタンスの子として表示リストに追加されます。そのデータ型は、ロードされたコンテンツの特性によって次のように異なります。
  - ロードされたイメージは Bitmap インスタンスになります。
  - ActionScript 3.0 で記述された SWF ファイルをロードした場合、Sprite または MovieClip インスタンス (または、コンテンツ作成者の指定により、これらのクラスのサブクラスのインスタンス) になります。
  - ActionScript 1.0 または ActionScript 2.0 で記述された SWF ファイルをロードした場合、AVM1Movie インスタンスになります。
- 他の表示オブジェクトのコンテナとなるオブジェクトが必要な場合 (ActionScript を使用して表示オブジェクトに描画するかどうかに関わらず)、次のいずれかの DisplayObjectContainer サブクラスを選択します。
  - ActionScript のみを使用してオブジェクトを作成する場合、または ActionScript のみによって作成および操作されるカスタム表示オブジェクトの基本クラスとしては、Sprite を選択します。
  - Flash オーサリングツールで作成されたムービークリップシンボルを参照するための変数を作成する場合、MovieClip を選択します。

- Flash ライブラリ内のムービークリップシンボルと関連付けられるクラスを作成する場合は、クラスの基本クラスとして、次のいずれかの DisplayObjectContainer サブクラスを選択します。
  - 関連付けられたムービークリップシンボルのコンテンツが複数のフレーム上に存在する場合、MovieClip を選択します。
  - 関連付けられたムービークリップシンボルのコンテンツが最初のフレームのみに存在する場合、Sprite を選択します。

表示オブジェクトの操作

どの表示オブジェクトを使用するかに関わらず、画面上に表示されるエレメントとして、すべての表示オブジェクトに対して共通に適用される操作が多数あります。たとえば、これらの操作はすべて画面上に配置でき、表示オブジェクトの積み重ね順序を前後に移動できます。また、拡大 / 縮小、回転などの操作も可能です。すべての表示オブジェクトは、共通の基本クラス (DisplayObject) からこの関数を継承しているため、TextField インスタンス、Video インスタンス、Shape インスタンス、その他のどの表示オブジェクトを操作するかに関わらず、この関数は同様に機能します。次のセクションでは、こうした共通の表示オブジェクト操作について詳しく説明します。

位置の変更

あらゆる表示オブジェクトに対する最も基本的な操作は、画面上への配置です。表示オブジェクトの位置を設定するには、オブジェクトの x プロパティと y プロパティを変更します。

```
myShape.x = 17;
myShape.y = 212;
```

表示オブジェクトの配置システムでは、Stage を直交座標系 (水平方向の x 軸と垂直方向の y 軸を使用する一般的なグリッド方式) として処理します。座標系の原点 (x 軸と y 軸が交わる 0,0 座標) は、ステージの左上隅になります。この位置から右方向が x 値の正方向、左方向が負方向になります。一方、(一般的なグラフとは異なり) 下方向が y 値の正方向、上方向が負方向になります。たとえば上記のコード行の場合、オブジェクト `myShape` は、x 座標 17 (原点から右方向へ 17 ピクセル)、y 座標 212 (原点から下方向へ 212 ピクセル) に移動します。

デフォルトでは、ActionScript を使用して表示オブジェクトを作成する場合、x プロパティと y プロパティはいずれも 0 に設定され、表示オブジェクトは親コンテンツの左上隅に配置されます。

ステージを基準とした位置の変更

x プロパティと y プロパティは、親表示オブジェクト軸の 0,0 座標を基準とした表示オブジェクトの位置を常に参照することに注意してください。このため、Sprite インスタンス内の Shape インスタンス (円など) の場合、Shape オブジェクトの x プロパティと y プロパティを 0 に設定すると、円は Sprite の左上隅に配置されます。これは、必ずしもステージの左上隅とは限りません。グローバルステージ座標を基準としてオブジェクトを配置するには、任意の表示オブジェクトの `globalToLocal()` メソッドを使用して、次のようにグローバル (ステージ) 座標からローカル (表示オブジェクトコンテナ) 座標に変換します。

```
// ステージの左上隅にシェイプを配置
// 親の位置には関係なく配置する。

// Sprite を作成し、x:200、y:200 に配置する。
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// 参照用に、Sprite の 0,0 座標の位置にドットを描画する。
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();

// 円の Shape インスタンスを作成する。
var circle:Shape = new Shape();
mySprite.addChild(circle);

// 半径 50 で、Shape の x:50、y:50 に中心点がある円を描画する。
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// ステージの 0, 0 座標に左上隅が来るようにシェイプを移動する。
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

同様に、DisplayObject クラスの `localToGlobal()` メソッドを使用して、ローカル座標をステージ座標に変換できます。

ドラッグ & ドロップ操作の作成

表示オブジェクトを移動する一般的な方法の 1 つは、ドラッグ & ドロップ操作を作成し、ユーザーがオブジェクトをクリックしてからマウスボタンを離すまで、オブジェクトがマウスの動きに合わせて移動するようにする方法です。ActionScript でドラッグ & ドロップ操作を作成する方法は 2 つあります。いずれの方法でも、2 つのマウスイベントを使用します。つまり、マウスボタンが押された場合はマウスカーソルに従って移動するようにオブジェクトに指示を出し、マウスボタンが離された場合はマウスカーソルに従って停止するようにオブジェクトに指示を出します。

最初の方法は startDrag() メソッドを使用した方法です。これは簡単な方法ですが、制約が多くなります。マウスボタンを押すと、ドラッグされる表示オブジェクトの startDrag() メソッドが呼び出されます。マウスボタンを離すと、stopDrag() メソッドが呼び出されます。

```
// このコードは、startDrag() 方式を使用してドラッグ & ドロップ操作を
// 作成する。
// 四角形は DisplayObject (MovieClip または Sprite インスタンスなど )。

import flash.events.MouseEvent;

// この関数は、マウスボタンが押されたときに呼び出される。
function startDragging(event:MouseEvent):void
{
  square.startDrag();
}

// この関数は、マウスボタンが離されたときに呼び出される。
function stopDragging(event:MouseEvent):void
{
  square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

この方法の場合、startDrag() を使用してドラッグできるのは一度に 1 つのアイテムだけという重大な制約があります。ある表示オブジェクトがドラッグされているときに startDrag() メソッドが別の表示オブジェクトに対して呼び出された場合、最初の表示オブジェクトはマウスに従って直ちに停止します。たとえば、startDragging() 関数を次のように変更した場合、square.startDrag() メソッド呼び出しにも関わらず、circle オブジェクトだけがドラッグされます。

```
function startDragging(event:MouseEvent):void
{
  square.startDrag();
  circle.startDrag();
}
```

startDrag() を使用してドラッグできるのは、一度に 1 つのオブジェクトのみです。そのため、表示オブジェクトに対して stopDrag() メソッドが呼びだされると、現在ドラッグ中のオブジェクトがどんなオブジュクトであってもそのオブジェクトは停止します。

複数の表示オブジェクトをドラッグする必要がある場合、または複数のオブジェクトが startDrag() を使用することで生じる可能性のある競合を回避する必要がある場合は、マウス追従方式を使用してドラッグ効果を作成するのが最適です。この方式を使用すると、マウスボタンが押されたときに、ステージの mouseMove イベントに対するリスナーとして関数がサブスクライブされます。マウスを移動するたびに呼び出されるこの関数により、ドラッグされるオブジェクトは、マウスの x, y 座標にジャンプします。マウスボタンを離すと、関数はリスナーとしてのサブスクライブを解除されます。この場合、マウスを移動しても関数が呼び出されなくなるため、オブジェクトはマウスカーソルに従って停止します。この方式を説明するコードを次に示します。

```
// このコードは、マウス追従方式を使用してドラッグ & ドロップ操作を
// 作成する。
// 円は DisplayObject (MovieClip または Sprite インスタンスなど )。

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// この関数は、マウスボタンが押されたときに呼び出される。
function startDragging(event:MouseEvent):void
{
  // マウスボタンが押されたときのカーソルの位置と
  // マウスボタンが押されたときの円の x, y 座標との
  // 差を記録する。
  offsetX = event.stageX - circle.x;
  offsetY = event.stageY - circle.y;

  // mouseMove イベントの監視を開始するよう Flash Player に指示する。
  stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// この関数は、マウスボタンが離されたときに呼び出される。
function stopDragging(event:MouseEvent):void
{
  // mouseMove イベントの監視を停止するよう Flash Player に指示する。
  stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// この関数は、マウスボタンが押されている間、
// マウスを移動するたびに呼び出される。
function dragCircle(event:MouseEvent):void
{
  // カーソルの位置に円を移動し、
  // カーソルの位置とドラッグされるオブジェクトの
  // 位置とのオフセットを保持する。
  circle.x = event.stageX - offsetX;
  circle.y = event.stageY - offsetY;
```

```
    // このイベントの後、画面を更新するよう Flash Player に指示する。
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

一般的なドラッグ & ドロップ操作には、表示オブジェクトをマウスカーソルに追従させるだけではなく、ドラッグするオブジェクトを画面の手前に移動することにより、他のすべてのオブジェクトの上に浮かせて表示させる操作もあります。たとえば、円と四角形の 2 つのオブジェクトがあり、その両方でドラッグ & ドロップ操作を使用するとします。表示リスト上で円が四角形の下にある場合、カーソルが四角形の上に来るように円をクリック & ドラッグすると、円は四角形の後ろにスライドして表示され、ドラッグ & ドロップ効果がなくなります。これとは別に、円をクリックすると、円が表示リストの一番上に移動し、常に他のすべてのコンテンツの一番上に表示される方法があります。

次のコード (前の例を応用) では、円と四角形の 2 つの表示オブジェクトに対してドラッグ & ドロップ操作を作成します。いずれかのアイテム上でマウスボタンを押すたびに、そのアイテムはステージの表示リストの最上部に移動します。その結果、ドラッグされるアイテムは常に一番上に表示されます。新しいコードと前の例から変更されたコードは、ボールド体で記述されています。

```
// このコードは、マウス追従方式を使用してドラッグ & ドロップ操作を
// 作成する。
// 円と四角形は DisplayObject (MovieClip または Sprite
// インスタンスなど )。

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

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// この関数は、マウスボタンが押されたときに呼び出される。
function startDragging(event:MouseEvent):void
{
    // ドラッグ中のオブジェクトを記憶
    draggedObject = DisplayObject(event.target);

    // マウスボタンが押されたときのカーソルの位置と
    // マウスボタンが押されたとき、ドラッグされているオブジェクトの x, y 座標との
    // 差 ( オフセット ) を記録する。
    offsetX = event.stageX - draggedObject.x;
    offsetY = event.stageY - draggedObject.y;

    // 選択したオブジェクトを表示リストの一番上に移動
    stage.addChild(draggedObject);

    // mouseMove イベントの監視を開始するよう Flash Player に指示する。
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
```

```
}

// この関数は、マウスボタンが離されたときに呼び出される。
function stopDragging(event:MouseEvent):void
{
  // mouseMove イベントの監視を停止するよう Flash Player に指示する。
  stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// この関数は、マウスボタンが押されている間、
// マウスを移動するたびに呼び出される。
function dragObject(event:MouseEvent):void
{
  // ドラッグされるオブジェクトをカーソルの位置に移動し、
  // カーソルの位置とドラッグされるオブジェクトの位置との
  // オフセットを保持する。
  draggedObject.x = event.stageX - offsetX;
  draggedObject.y = event.stageY - offsetY;

  // このイベントの後、画面を更新するよう Flash Player に指示する。
  event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

トークンやカードをパイル間で移動するゲームなどでこの効果をさらに拡張するには、ドラッグされるオブジェクトを " 拾う " ときにステージの表示リストに追加し、次にマウスボタンを離すときに別の表示リスト (オブジェクトがドロップされる " パイル " など) に追加します。

最後に、効果を強調するために、表示オブジェクトをクリックしたとき (ドラッグを開始したとき) にドロップシャドウフィルタを適用し、オブジェクトを解放したときにドロップシャドウを削除します。ActionScript でのドロップシャドウフィルタや、その他の表示オブジェクトフィルタを使用する場合の詳細については、443 ページ、第 15 章の「表示オブジェクトのフィルタ処理」を参照してください。

表示オブジェクトのパンとスクロール

表示させたい領域に対して大きすぎる表示オブジェクトがある場合、scrollRect プロパティを使用して、表示オブジェクトの可視領域を定義できます。また、ユーザー入力に応じて scrollRect プロパティを変更することで、コンテンツの左右のパンや上下のスクロールが可能になります。

scrollRect プロパティは、Rectangle クラスのインスタンスです。Rectangle クラスは、矩形領域を単一のオブジェクトとして定義する場合に必要な値を結合するクラスです。最初に表示オブジェクトの可視領域を定義するには、新しい Rectangle インスタンスを作成し、これを表示オブジェクトの scrollRect プロパティに割り当てます。その後でスクロールまたはパンするには、scrollRect プロパティを個別の Rectangle 変数に読み込み、目的のプロパティを変更します (たとえば、パンするには Rectangle インスタンスの x プロパティを、スクロールするには y プロパティを変更します)。次に、この Rectangle インスタンスを scrollRect プロパティに再度割り当て、変更された値を表示オブジェクトに通知します。

たとえば、次に示すコードでは、SWF ファイルの境界内には収まらない高さが設定された bigText という名前の TextField オブジェクトに対する可視領域を定義します。up と down という 2 つのボタンがクリックされると、TextField オブジェクトのコンテンツを上下にスクロールさせる関数が呼び出されます。この関数は、scrollRect Rectangle インスタンスの y プロパティを変更することによってスクロールを行います。

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// TextField インスタンスの初期可視領域を次のように定義する。
// 左 : 0、最上部 : 0、幅 : TextField の幅、高さ : 350 ピクセル。
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// TextField をビットマップとしてキャッシュすることにより、パフォーマンスを改善する。
bigText.cacheAsBitmap = true;

// [up] ボタンがクリックされると呼び出される
function scrollUp(event:MouseEvent):void
{
   // 現在のスクロール矩形にアクセスする。
   var rect:Rectangle = bigText.scrollRect;
   // 矩形の y 値を 20 だけ減らすと、結果として
   // 矩形は 20 ピクセルだけ下方に移動する。
   rect.y -= 20;
   // 矩形を TextField に再割り当てし、変更を " 適用 " する。
   bigText.scrollRect = rect;
}

// [down] ボタンがクリックされると呼び出される
function scrollDown(event:MouseEvent):void
{
   // 現在のスクロール矩形にアクセスする。
```

```
    var rect:Rectangle = bigText.scrollRect;
    // 矩形の y 値を 20 だけ増やすと、結果として
    // 矩形は 20 ピクセルだけ上方に移動する。
    rect.y += 20;
    // 矩形を TextField に再割り当てし、変更を " 適用 " する。
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

上記の例に示すように、表示オブジェクトの scrollRect プロパティを操作する場合、cacheAsBitmap プロパティを使用して、Flash Player が表示オブジェクトのコンテンツをビットマップとしてキャッシュするように指定するのが最適な方法です。こうすると、Flash Player は、表示オブジェクトがスクロールされるたびに、そのコンテンツ全体を再描画する必要がなくなります。代わりに、キャッシュされたビットマップを使用して、必要な部分だけを画面に直接レンダリングできます。詳細については、387 ページの「表示オブジェクトのキャッシュ」を参照してください。

サイズの変更とオブジェクトの拡大 / 縮小

表示オブジェクトのサイズを測定および操作する方法は 2 つあります。サイズのプロパティ (width および height) または拡大 / 縮小のプロパティ (scaleX および scaleY) のいずれかを使用します。

各表示オブジェクトには、width プロパティと height プロパティがあり、これらの値はオブジェクトのサイズに初期設定されています (ピクセル単位)。これらのプロパティの値を読み取って、表示オブジェクトのサイズを測定できます。また、次のように新しい値を指定して、オブジェクトのサイズを変更することもできます。

```
// 表示オブジェクトのサイズを変更する。
square.width = 420;
square.height = 420;

// 円表示オブジェクトの半径を決定する。
var radius:Number = circle.width / 2;
```

表示オブジェクトの height または width を変更すると、オブジェクトが拡大または縮小されます。つまり、オブジェクトのコンテンツが新しい領域に合わせて拡張または縮小することになります。ベクターシェイプのみを含む表示オブジェクトの場合、これらのシェイプは新しいスケールで再描画されます。その際、画質が低下することはありません。表示オブジェクト内のすべてのビットマップグラフィックエレメントは、再描画されるのではなく、拡大 / 縮小されます。このため、イメージの実際のピクセル情報よりも大きな幅と高さが設定されたデジタル写真などの場合、ピクセル化されて粗く表示されます。

表示オブジェクトの width または height プロパティを変更すると、オブジェクトの scaleX プロパティと scaleY プロパティも Flash Player によって更新されます。これらのプロパティは、表示オブジェクトのサイズを、その元のサイズと比較して相対的に表します。scaleX プロパティと scaleY プロパティは、小数 (十進分数) 値を使用してパーセンテージを表します。たとえば、表示オブジェクトの width を変更し、その幅を元のサイズの半分にした場合、このオブジェクトの scaleX プロパティの値は、50% を表す .5 となります。このオブジェクトの高さを 2 倍にした場合、その scaleY プロパティの値は、200% を表す 2 となります。

```
// 円は、幅と高さが 150 ピクセルの表示オブジェクト。
// 元のサイズの scaleX と scaleY はそれぞれ 1 (100%)。
trace(circle.scaleX);// 出力 : 1
trace(circle.scaleY);// 出力 : 1

// width プロパティと height プロパティが変更されると、
// Flash Player はこれに従って scaleX および scaleY プロパティを変更する。
circle.width = 100;
circle.height = 75;
trace(circle.scaleX);// 出力 : 0.6622516556291391
trace(circle.scaleY);// 出力 : 0.4966887417218543
```

サイズの変更は相対的ではありません。つまり、四角形の height は変更するが width は変更しない場合、その比率は同じではなくなるため、オブジェクトは四角形ではなく矩形になります。表示オブジェクトのサイズを相対的に変更したい場合、width プロパティまたは height プロパティを設定する代わりに、scaleX プロパティと scaleY プロパティの値を設定することによって、表示オブジェクトのサイズを変更できます。たとえば次のコードは、square という名前の表示オブジェクトの width を変更し、次に垂直スケール (scaleY) を変更して水平スケールに一致させることで、このオブジェクトの相対的なサイズを保ちます。

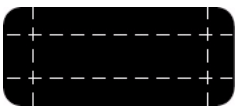
```
// 幅を直接変更する。
square.width = 150;

// 相対的なサイズ変更を維持するために
// 垂直スケールを変更して水平スケールに一致させる。
square.scaleY = square.scaleX;
```

拡大 / 縮小時の歪みの制御

表示オブジェクトを拡大 / 縮小する場合 (水平方向に伸縮する場合など)、結果として生じる歪みはオブジェクト全体で均等に拡散されるため、すべての部分が同じだけ伸縮するのが通常です。グラフィックエレメントやデザインエレメントの場合、これが問題になることはあまりありません。しかし場合によっては、表示オブジェクトの一部を伸縮し、一部を変更しないように制御したいことがあります。角の丸い矩形のボタンが、この場合の典型的な例です。通常の拡大 / 縮小操作の場合、ボタンのサイズを変更すると、ボタンの隅が伸縮し、角丸の半径が変化します。

しかしこの場合は、拡大 / 縮小を制御できれば (つまり、拡大 / 縮小する領域 (直線の辺と中央部分) としない領域 (角) を指定できれば)、表示したときの歪みが生じないため、その方が望ましいことになります。

9 スライスの拡大 / 縮小 (スケール 9) を使用すると、表示オブジェクトを作成してそのオブジェクトの拡大 / 縮小方法を制御できます。9 スライスの拡大 / 縮小を使用すると、表示オブジェクトが 9 つの個別の矩形 (三目並べの升目のような 3 x 3 のグリッド) に分割されます。矩形は必ずしも同じサイズであるとは限りません。グリッド線をどこに配置するかは、ユーザーが指定します。表示オブジェクトを拡大 / 縮小しても、4 つの角にあたる矩形 (ボタンの丸い角の部分など) 内のコンテンツのサイズは変更されません。上端中央部と下端中央部の矩形は水平方向に拡大 / 縮小しますが、垂直方向には拡大 / 縮小しません。一方、左端中央部と右端中央部の矩形は垂直方向に拡大 / 縮小しますが、水平方向には拡大 / 縮小しません。中央部の矩形は、垂直方向と水平方向の両方に拡大 / 縮小します。



表示オブジェクトを作成し、そのオブジェクトの特定のコンテンツを拡大 / 縮小したくない場合は、そのコンテンツが矩形の 4 つの角のいずれかに配置されるように、9 スライスの拡大 / 縮小のグリッド境界線を設定する必要があります。

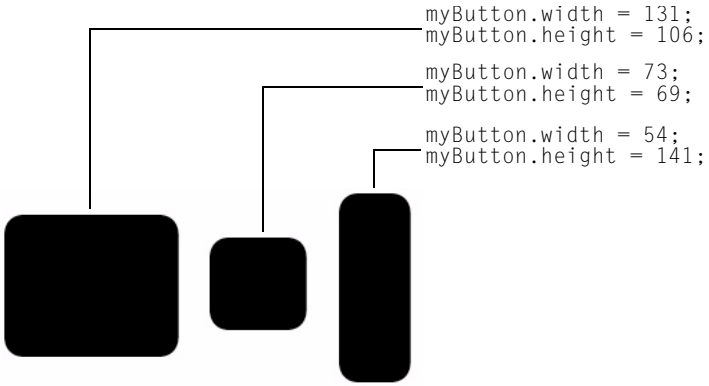
ActionScript の場合、表示オブジェクトの `scale9Grid` プロパティの値を設定すると、そのオブジェクトに対する 9 スライスの拡大 / 縮小がオンになり、オブジェクトのスケール 9 グリッドにおける矩形のサイズが定義されます。`scale9Grid` プロパティの値として、Rectangle クラスのインスタンスを次のように使用します。

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

Rectangle コンストラクタの 4 つのパラメータは、x 座標、y 座標、幅、高さです。この例の矩形の左上隅は、`myButton` という表示オブジェクト上の x: 32、y: 27 に配置されます。この矩形の幅は 71 ピクセル、高さは 65 ピクセルです (従って、その右端は表示オブジェクト上の x 座標 103、下端は表示オブジェクト上の y 座標 92 になります)。

Rectangle インスタンスによって定義される領域に含まれる実際の領域は、スケール 9 グリッドの中央部の矩形を表します。次に示すように、他の部分の矩形は Rectangle インスタンスの各辺を拡張することにより、Flash Player で算出されます。

この例では、ボタンを拡大 / 縮小しても角丸の部分は変化しませんが、他の領域は拡大 / 縮小に合わせて調整されます。



表示オブジェクトのキャッシュ

Flash 内のデザインのサイズが大きくなると、アプリケーションの作成や、複雑なスクリプトで記述されたアニメーションの作成を行っている場合、パフォーマンスと最適化について考慮する必要があります。静的な状態のままのコンテンツ (矩形の Shape インスタンスなど) がある場合、Flash によるコンテンツの最適化は行われません。したがって、矩形の位置を変更すると、Flash によって Shape インスタンス全体が再度描画されます。

指定した表示オブジェクトをキャッシュすることにより、SWF ファイルのパフォーマンスを向上させることができます。表示オブジェクトは " サーフェス " であり、基本的にはインスタンスのベクターデータのビットマップバージョンです。このデータは、SWF ファイルの処理で大きく変更されることはありません。したがって、キャッシュがオンになっているインスタンスでは、SWF ファイルの再生中に連続的に再描画されることがないので、SWF ファイルのレンダリングが高速になります。

メモ

サーフェスが再作成されるときに、ベクターデータを更新できます。このため、サーフェス内にキャッシュされたベクターデータは、SWF ファイル全体で同じである必要はありません。

表示オブジェクトの `cacheAsBitmap` プロパティを `true` に設定すると、表示オブジェクト自体のビットマップ表現がキャッシュされます。Flash は、インスタンスのサーフェスオブジェクトを作成します。これは、ベクターデータではなくキャッシュされたビットマップです。表示オブジェクトの境界を変更すると、サーフェスはサイズ変更されるのではなく、再作成されます。サーフェスは、他のサーフェス内にネストできます。子サーフェスは、親サーフェス上にビットマップをコピーします。詳細については、389 ページの「ビットマップキャッシュの有効化」を参照してください。

DisplayObject クラスの `opaqueBackground` プロパティと `scrollRect` プロパティは、`cacheAsBitmap` プロパティを使用してビットマップキャッシュに関連付けられています。この 3 つのプロパティは互いに独立していますが、`opaqueBackground` プロパティと `scrollRect` プロパティは、オブジェクトがビットマップとしてキャッシュされるときに最適に動作します。また、`opaqueBackground` プロパティと `scrollRect` プロパティによってパフォーマンスが向上するのは、`cacheAsBitmap` が `true` に設定されている場合のみです。表示オブジェクトのコンテンツのスクロールの詳細については、382 ページの「表示オブジェクトのパンとスクロール」を参照してください。不透明な背景の設定の詳細については、390 ページの「不透明な背景色の設定」を参照してください。

`cacheAsBitmap` プロパティを `true` に設定する必要があるアルファチャンネルマスクの詳細については、396 ページの「アルファチャンネルマスク」を参照してください。

キャッシュを有効にするシナリオ

表示オブジェクトのキャッシュを有効にすると、サーフェスが作成されます。これには、複雑なベクターアニメーションのレンダリングが速くなるなどいくつかの利点があります。キャッシュを有効にしたいシナリオとして、いくつかの場合が考えられます。常にキャッシュを有効にして SWF ファイルのパフォーマンスを向上させたい場合が考えられます。しかし、キャッシュを有効にしてもパフォーマンスが向上しない場合や、パフォーマンスが低下する場合もあります。このセクションでは、キャッシュを使用する必要があるシナリオと、通常の表示オブジェクトを使用する必要があるシナリオについて説明します。

キャッシュされたデータの全体的なパフォーマンスは、インスタンスのベクターデータの複雑さ、変更するデータ量、および `opaqueBackground` プロパティを設定するかどうかによって異なります。小さな領域を変更する場合、サーフェスを使用したときとベクターデータを使用したときの違いは、ごくわずかです。アプリケーションをデプロイする前に、両方のシナリオをテストする必要があります。

ビットマップキャッシュを使用するシナリオ

ビットマップキャッシュを有効にすることで大きな利点が得られる代表的なシナリオを次に示します。

- 複雑な背景イメージ : ベクターデータの詳細で複雑な背景イメージ (ビットマップのトレースコマンドを適用したイメージや Adobe Illustrator で作成したアートワークなど) を含むアプリケーション。背景で文字をアニメーション化することがあります。この場合、背景で連続的にベクターデータを再生成する必要があるため、アニメーションが遅くなります。パフォーマンスを改善するには、背景の表示オブジェクトの `opaqueBackground` プロパティを `true` に設定します。これにより、背景はビットマップとしてレンダリングされるため素早く再描画でき、アニメーションの再生がより高速になります。
- テキストフィールドのスクロール : スクロールテキストフィールドに大量のテキストを表示するアプリケーション。スクロール境界 (`scrollRect` プロパティ) によってスクロール可能に設定された表示オブジェクト内に、テキストフィールドを配置できます。これにより、指定したインスタンスの高速なピクセルスクロールが可能になります。ユーザーが表示オブジェクトインスタンスをスクロールすると、Flash はスクロールされたピクセルを上方向に移動し、テキストフィールド全体を再生成するのではなく、新しく表示された領域だけを生成します。
- ウィンドウシステム : 重なり合うウィンドウを持つ複雑なアプリケーション。各ウィンドウは、開いて閉じることができます (Web ブラウザウィンドウなど)。各ウィンドウをサーフェスとしてマークした場合 (`cacheAsBitmap` プロパティを `true` に設定することによりマーク)、各ウィンドウは分離されてキャッシュされます。ユーザーは複数のウィンドウをドラッグして重ね合わせることができるため、各ウィンドウでベクターコンテンツを再生成する必要はありません。
- アルファチャンネルマスク : アルファチャンネルマスクを使用する場合は、`chacheAsBitmap` プロパティを `true` に設定する必要があります。詳細については、396 ページの「アルファチャンネルマスク」を参照してください。

これらのシナリオでビットマップキャッシュを有効にすると、ベクターグラフィックの最適化により、アプリケーションの応答性とインタラクティブ性が向上します。

また、表示オブジェクトに対してフィルタを適用するたびに、cacheAsBitmap が (明示的に false 設定されている場合でも) Flash Player によって自動的に true に設定されます。表示オブジェクトのすべてのフィルタをクリアすると、cacheAsBitmap は最後に設定した値に戻ります。

ビットマップキャッシュの使用を避けるべきシナリオ

ビットマップキャッシュ機能を誤って使用すると、SWF ファイルに悪影響を及ぼすことがあります。ビットマップキャッシュを使用する場合は、次のガイドラインを覚えておく必要があります。

- サーフェス (キャッシュを有効にした表示オブジェクト) を多用しないでください。各サーフェスは、通常の表示オブジェクトより多くのメモリを使用します。サーフェスを有効にするのは、レンダリングのパフォーマンスを改善する必要がある場合のみにしてください。

  キャッシュされたビットマップは、通常の表示オブジェクトに比べてはるかに多くのメモリを使用します。たとえば、ステージ上の Sprite インスタンスのサイズが 250 ピクセル x 250 ピクセルの場合、通常の (キャッシュされていない) Sprite インスタンスでは 1 KB のメモリ使用量が、キャッシュされると 250 KB になることがあります。
- キャッシュされたサーフェスのズームインは避けてください。ビットマップキャッシュを多用しすぎると、大量のメモリが消費されますが (前の項目を参照)、特にコンテンツをズームインした場合にメモリ使用量が大きくなります。
- 大部分が静的な (アニメーション化されていない) 表示オブジェクトインスタンスに対しては、サーフェスを使用してください。インスタンスのドラッグや移動はできますが、インスタンスのコンテンツのアニメーション化や大幅な変更は避ける必要があります (アニメーションや Video インスタンスを含む MovieClip インスタンスの場合、アニメーション化やコンテンツの変更が行われる可能性が高くなります)。たとえば、インスタンスの回転や変換を行うと、インスタンスはサーフェスからベクターデータ (またはその逆) へ変更されます。この処理は複雑なため、SWF ファイルに悪影響を与えます。
- サーフェスとベクターデータを混在させた場合、Flash Player (場合によってはコンピュータ) で実行する処理量が増加します。たとえば、ウィンドウアプリケーションを作成するときなどは、可能な限りサーフェスをグループ化してください。

ビットマップキャッシュの有効化

表示オブジェクトのビットマップキャッシュを有効にするには、cacheAsBitmap プロパティを true に設定します。

```
mySprite.cacheAsBitmap = true;
```

cacheAsBitmap プロパティを true に設定すると、表示オブジェクトのピクセルが全体の座標に自動的に吸着します。SWF ファイルをテストすると、複雑なベクターイメージで実行されるアニメーションのレンダリングが大幅に速くなっていることがわかります。

cacheAsBitmap を true に設定しても、次のうち 1 つまたは複数の条件に当てはまる場合、サーフェス (キャッシュされたビットマップ) は作成されません。

- ビットマップの高さと幅のいずれかが 2880 ピクセルを超えている場合。
- ビットマップが割り当てに失敗した場合 (メモリ不足エラー)。

不透明な背景色の設定

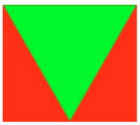
表示オブジェクトに不透明な背景を設定できます。たとえば、背景に複雑なベクターアートが使用されている場合、opaqueBackground プロパティを指定の色 (通常はステージと同じ色) に設定できます。色は数値で指定します (通常は 16 進数のカラー値)。これにより、背景はビットマップとして処理されるため、パフォーマンスの最適化に役立ちます。

cacheAsBitmap を true に設定して opaqueBackground プロパティを指定の色に設定すると、opaqueBackground プロパティによって内部ビットマップが不透明になり、レンダリングが高速になります。cacheAsBitmap を true に設定しなかった場合、opaqueBackground プロパティによって表示オブジェクトの背景に不透明なベクター四角形シェイプが追加されます。ビットマップは自動的には作成されません。

次の例は、表示オブジェクトの背景を設定してパフォーマンスを最適化する方法を示しています。

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

この例では、myShape という名前の Shape インスタンスの背景色を赤 (0xFF0000) に設定します。白い背景のステージ上に描画された緑の三角形が Shape インスタンスに含まれている場合、緑の三角形の背景部分として、Shape インスタンスの境界ボックス (Shape の周りを囲む矩形) 内の空き領域が赤で表示されます。



もちろんこのコードは、単色の赤の背景が設定されたステージで使用するのが実際的です。別の色の背景上では、その色が代わりに指定される可能性が高くなります。たとえば、白い背景を持つ SWF では、opaqueBackground プロパティは 0xFFFFFF (白) に設定される可能性が高くなります。

ブレンドモードの適用

ブレンドモードとは、あるイメージ (ベースイメージ) の色と別のイメージ (ブレンドイメージ) の色を組み合わせて第 3 のイメージを作成することです。結果として作成されたイメージが、画面上で実際に表示されるイメージになります。あるイメージ内の各ピクセル値を他のイメージの対応するピクセル値で処理することにより、結果のイメージの同じ位置のピクセル値が生成されます。

各表示オブジェクトには blendMode プロパティがあり、次のいずれかのブレンドモードに設定できます。これらのモードは、BlendMode クラスで定義されている定数です。これらの代わりに、定数の実際の値である String 値 (括弧内の値) を使用することもできます。

- BlendMode.ADD ("add"): 2 つのイメージ間で色を明るくする Dissolve エフェクトのアニメーションを作成する場合によく使用されます。
- BlendMode.ALPHA ("alpha"): 前面の透明度を背景に適用する場合によく使用されます。
- BlendMode.DARKEN ("darken"): 重ね合わせタイプによく使用されます。
- BlendMode.DIFFERENCE ("difference"): 鮮やかな色を作成する場合によく使用されます。
- BlendMode.ERASE ("erase"): 前面のアルファを使用して背景の一部を切り取る (消去する) 場合によく使用されます。
- BlendMode.HARDLIGHT ("hardlight"): シャドウ効果を作成する場合によく使用されます。
- BlendMode.INVERT ("invert"): 背景を反転する場合に使用されます。
- BlendMode.LAYER ("layer"): 特定の表示オブジェクトを事前構成するための一時バッファーを強制的に作成する場合に使用されます。
- BlendMode.LIGHTEN ("lighten"): 重ね合わせタイプによく使用されます。
- BlendMode.MULTIPLY ("multiply"): シャドウ効果や深度効果を作成する場合によく使用されます。
- BlendMode.NORMAL ("normal"): ベースイメージのピクセル値をブレンドイメージのピクセル値で上書きする場合に使用されます。
- BlendMode.OVERLAY ("overlay"): シャドウ効果を作成する場合によく使用されます。
- BlendMode.SCREEN ("screen"): ハイライトとレンズのフレアを作成する場合によく使用されます。
- BlendMode.SUBTRACT ("subtract"): 2 つのイメージ間で色を暗くする Dissolve エフェクトのアニメーションを作成する場合によく使用されます。

DisplayObject の色の調整

ColorTransform クラス (flash.geom.ColorTransform) のメソッドを使用して、表示オブジェクトの色を調整できます。各表示オブジェクトには、transform プロパティがあります。このプロパティは Transform クラスのインスタンスであり、表示オブジェクトに適用される各種の変換 (回転、スケールの変更、位置の変更など) に関する情報を格納します。また、Transform クラスは、図形変換に関する情報を格納するだけではなく、colorTransform プロパティも備えています。このプロパティは ColorTransform クラスのインスタンスであり、このプロパティによって表示オブジェクトのカラーを調整できます。表示オブジェクトのカラー変換情報にアクセスするには、次のようなコードを使用します。

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

ColorTransform インスタンスを作成したらそのプロパティ値を読み取り、適用済みのカラー変換を確認できます。また、これらの値を設定して、表示オブジェクトに対してカラー変更を加えることもできます。何らかの変更後に表示オブジェクトを更新するには、ColorTransform インスタンスを transform.colorTransform プロパティに再割り当てする必要があります。

```
var colorInfo:ColorTransform = my DisplayObject.transform.colorTransform;

// ここでカラー変更を行う。

// 変更を確定する。
myDisplayObject.transform.colorTransform = colorInfo;
```

コードを使用したカラー値の設定

ColorTransform クラスの color プロパティを使用すると、特定の赤、緑、青 (RGB) のカラー値を表示オブジェクトに割り当てることができます。次の例では、color プロパティを使用し、blueBtn という名前のボタンがクリックされたときに、square という名前の表示オブジェクトの色が青に変わるようにします。

```
// square は、ステージ上の表示オブジェクト。
// blueBtn、redBtn、greenBtn、および blackBtn は、ステージ上のボタン。

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// square と関連付けられた ColorTransform インスタンスにアクセスする。
var colorInfo:ColorTransform = square.transform.colorTransform;

// この関数は、blueBtn がクリックされると呼び出される。
function makeBlue(event:MouseEvent):void
{
  // ColorTransform オブジェクトの色を設定する。
  colorInfo.color = 0x003399;
  // 表示オブジェクトに変更を適用
  square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

color プロパティを使用して表示オブジェクトの色を変更すると、変更前のオブジェクトに複数の色があった場合でも、オブジェクト全体の色が完全に変更されます。上部に黒いテキストのある緑の円を含む表示オブジェクトがある場合、そのオブジェクトに関連付けられた ColorTransform インスタンスの color プロパティを赤の陰影に設定すると、オブジェクト、円、テキストがすべて赤になります (従って、テキストを他のオブジェクト部分から判別することはできなくなります)。

コードを使用したカラーおよび明度効果の変更

複数の色を持つ表示オブジェクト (デジタル写真など) があり、オブジェクトの色を全面的に変更したくない (つまり、既存の色に基づいて表示オブジェクトの色を調整したい) という場合を想定します。このシナリオでは、こうした調整を作成するための乗数プロパティとオフセットプロパティが ColorTransform クラスに設定されています。redMultiplier、greenMultiplier、blueMultiplier、alphaMultiplier という名前の乗数プロパティは、色付きの写真フィルタ (または色付きサングラス) と同様に動作し、表示オブジェクト内の特定の色を強くしたり弱くしたりします。オフセットプロパティ (redOffset、greenOffset、blueOffset、alphaOffset) は、オブジェクトに特定の色を余分に追加したり、特定の色に対して設定できる最小値を指定する場合に使用されます。

これらの乗数プロパティとオフセットプロパティの共通点は、プロパティインスペクタで [カラー] ポップアップメニューから [詳細] を選択すると、Flash オーサリングツールのムービークリップシンボルに対して詳細なカラー設定を利用できるようになることです。

次のコードでは、JPEG イメージをロードしてカラー変換を適用します。これにより、x 軸と y 軸に沿ってマウスポインタを移動すると、赤のチャンネルと緑のチャンネルが調整されます。この例ではオフセット値が指定されていないため、画面上に表示される各カラーチャンネルのカラー値は、イメージ内の元のカラー値のパーセンテージになります。つまり、特定のピクセルで表示されるほとんどの赤や緑は、そのピクセル内の元の赤や緑の量と同じになります。

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// ステージ上にイメージをロードします。
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/
  image1.jpg");
loader.load(url);
this.addChild(loader);

// この関数は、ロードされたイメージ上にマウスを移動すると呼び出される。
function adjustColor(event:MouseEvent):void
{
  // Loader ( イメージの格納 ) 用 ColorTransform オブジェクトにアクセス
  var colorTransformer:ColorTransform = loader.transform.colorTransform;

  // マウスの位置に従って、赤と緑の乗数を設定する。
  // 赤の値は、カーソルが左にあるときの 0% ( 赤はなし ) から
  // カーソルが右にあるときの 100% 赤 ( 通常のイメージ外観 ) までの範囲。
  // 緑のチャンネルにも同様に適用するが、
  // y 軸でのマウスの位置によって制御される場合は除く。
  colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
  colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;
```

```
  // 表示オブジェクトに変更を適用する。
  loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

オブジェクトの回転

rotation プロパティを使用して、表示オブジェクトを回転することができます。この値を読み取って、オブジェクトが回転されたかどうかを確認したり、オブジェクトに適用される回転量を表す数値 (度数) をこのプロパティで設定して、オブジェクトを回転したりすることができます。たとえば、このコード行では、square という名前のオブジェクトを 45 度 (完全な 1 回転の 8 分の 1) 回転します。

```
square.rotation = 45;
```

別の方法として、411 ページ、第 13 章の「ジオメトリの操作」で説明されている変換行列を使用して表示オブジェクトを回転することもできます。

オブジェクトのフェード

表示オブジェクトの透明度を制御して、表示オブジェクトを部分的に (または完全に) 透明にしたり、または透明度を変更して、オブジェクトをフェードインまたはフェードアウトさせたりすることができます。DisplayObject クラスの alpha プロパティでは、表示オブジェクトの透明度 (より正確には不透明度) を定義します。alpha プロパティは、0 と 1 の間の任意の値に設定できます。ここで、0 は完全に透明、1 は完全に不透明であることを意味します。たとえば、このコード行では、マウスがクリックされたときに、myBall という名前のオブジェクトを部分的に (50%) 透明にします。

```
function fadeBall(event:MouseEvent):void
{
  myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

ColorTransform クラスを通じて利用できるカラー調整を使用して、表示オブジェクトの透明度を変更することもできます。詳細については、391 ページの「DisplayObject の色の調整」を参照してください。

表示オブジェクトのマスク

表示オブジェクトをマスクとして使用することにより、空洞を作成し、この空洞から別の表示オブジェクトのコンテンツを表示することができます。

マスクの定義

別の表示オブジェクトのマスクとなる表示オブジェクトを指定するには、マスクされる表示オブジェクトの mask プロパティとしてマスクオブジェクトを設定します。

```
// オブジェクト maskSprite をオブジェクト mySprite のマスクにする。
mySprite.mask = maskSprite;
```

マスクされる表示オブジェクトは、マスクとして機能する表示オブジェクトのすべての不透明な (透明でない) 領域で表示されます。たとえば、次のコードでは、赤色で 100 x 100 ピクセルの四角形を含む Shape インスタンスと、青色で半径 25 ピクセルの円を含む Sprite インスタンスを作成します。円をクリックすると、円は四角形のマスクとして設定されます。結果的に、四角形の表示される部分は、円の単体部分によって覆い隠された部分のみとなります。すなわち、赤い円しか表示されません。

```
// このコードは、MovieClip または Sprite インスタンスなどの
// 表示オブジェクト内で実行されることを前提としている。

import flash.display.Shape;

// 四角形を描画し、表示リストに追加する。
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// 円を描画し、表示リストに追加する。
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);

function maskSquare(event:MouseEvent):void
{
  square.mask = circle;
  circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

マスクとして機能する表示オブジェクトは、ドラッグ、アニメーション化、動的なサイズ変更が可能なほか、単一のマスク内で別個のシェイプを使用できます。マスク表示オブジェクトは、必ずしも表示リストに追加する必要があるわけではありません。ただし、ステージが拡大 / 縮小されるときにマスクオブジェクトを拡大 / 縮小したい場合、または、マスクを使ったユーザー操作 (ユーザー制御のドラッグやサイズ変更など) を可能にしたい場合は、マスクオブジェクトを表示リストに追加する必要があります。マスクオブジェクトを表示リストに追加する間、表示オブジェクトの実際の z 順序 (重ね順) は関係ありません (マスクオブジェクトは、マスクとして以外に画面上に表示されることはありません)。マスクオブジェクトが、複数のフレームのある MovieClip インスタンスであった場合、このオブジェクトがマスクとして機能しなかった場合と同様に、すべてのフレームがそのタイムライン内で再生されます。mask プロパティを次のように null に設定すると、マスクを削除できます。

```
// mySprite のマスクを削除
mySprite.mask = null;
```

別のマスクをマスクするためにマスクを使用することはできません。マスク表示オブジェクトの alpha プロパティを設定することはできません。マスクとして使用される表示オブジェクトでは、塗りのみを使用できます (線は無視されます)。

デバイスフォントのマスクについて

表示オブジェクトを使用して、デバイスフォントで設定されたテキストをマスクすることができます。表示オブジェクトを使用して、デバイスフォントで設定されたテキストをマスクする場合、マスクの矩形境界ボックスがマスクシェイプとして使用されます。すなわち、デバイスフォントテキストに対する非矩形の表示オブジェクトマスクを作成する場合、SWF ファイルに表示されるマスクは、マスク自体のシェイプではなく、マスクの矩形境界ボックスのシェイプになります。

アルファチャンネルマスク

アルファチャンネルマスクがサポートされるのは、マスクおよびマスクされる側の両方の表示オブジェクトがビットマップキャッシュを使用している場合です。

```
// maskShape は、グラデーションの塗りを含む Shape インスタンス。
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

たとえば、アルファチャンネルマスクの適用例の 1 つは、マスクされる表示オブジェクトに適用されるフィルタとは別にマスクオブジェクト上でフィルタを使用することです。

次に示す例では、外部のイメージファイルをステージ上にロードします。このイメージ (より正確には、イメージがその中にロードされる Loader インスタンス) は、マスクされる表示オブジェクトになります。グラデーションのある楕円形 (単色の黒の中心から端に向かって透明にフェード) がイメージに描画されます。これが、アルファマスクです。両方の表示オブジェクトで、ビットマップキャッシュがオンになっています。楕円形はイメージのマスクとして設定され、次にドラッグ可能に設定されます。

```
// このコードは、MovieClip または Sprite インスタンスなどの
// 表示オブジェクト内で実行されることを前提としている。

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// イメージをロードし、表示リストに追加する。
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/
  image1.jpg");
loader.load(url);
this.addChild(loader);

// Sprite を作成する。
var oval:Sprite = new Sprite();
// グラデーションのある楕円形を描画する。
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
var matrix:Matrix = new Matrix();
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                    colors,
                    alphas,
                    ratios,
                    matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// Sprite を表示リストに追加
this.addChild(oval);

// cacheAsBitmap を設定 = 両方の表示オブジェクトに対して true。
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// 楕円形をローダー ( およびその子、ロードされたイメージ ) のマスクとして設定
loader.mask = oval;

// 楕円形をドラッグ可能にする。
oval.startDrag(true);
```

オブジェクトのアニメーション化

アニメーションは、時間の経過と共に何かを動かす、または、何かを変化させる処理です。スクリプトで記述されたアニメーションは、ビデオゲームの基本的部分であり、他のアプリケーションを洗練したり、便利な操作上のヒントを提供したりするために頻繁に使用されます。

スクリプトで記述されたアニメーションの基本的概念は、変化が生じる必要があり、その変化が時間の経過と共にインクリメントに分割される必要があるということです。ActionScript では、一般的なループステートメントを使用して、何かを繰り返す処理が簡単にできます。ただし、ループは、表示を更新するまで、すべての繰り返しを通じて実行されます。スクリプトで記述されたアニメーションを作成するには、時間の経過と共に繰り返しアクションを実行する ActionScript を記述し、実行するたびに画面を更新する必要があります。

たとえば、画面でボールが移動するような、単純なアニメーションを作成するとします。ActionScript には、時間の推移を追跡し、これに従って画面を更新できるシンプルなメカニズムが備わっています。これはすなわち、ボールが目的地に到達するまで、時間と共に少しずつ移動するようなコードを記述できることを意味します。各動きごとに画面を更新すると、ステージをまたがるモーションがビューアに表示されます。

実用的な観点からは、スクリプトで記述されたアニメーションを SWF ファイルのフレームレートと同期させる (すなわち、新しいフレームが表示されるか、または表示されようとするたびにアニメーションを変化させる) ことに意味があります。Flash Player はこの方法により、画面を速く更新できるためです。各表示オブジェクトには、SWF ファイルのフレームレートに従って送出される `enterFrame` イベントがあります (各フレームに 1 イベント)。スクリプトで記述されたアニメーションを作成する開発者の多くは、`enterFrame` イベントを、時間と共に繰り返すアクションを作成する方法として使用しています。`enterFrame` イベントをリッスンするコードを記述して、アニメーション化されたボールを各フレームごとに一定量動かすことができます。画面の更新 (各フレーム) 後、ボールは新しい位置で再描画され、モーションが生成されます。

メモ

時間の経過と共にアクションを繰り返し実行するためのもう 1 つの方法は、Timer クラスを使用することです。Timer インスタンスは、指定された時間が経過するごとに、イベント通知をトリガします。Timer クラスの timer イベントを処理して、時間間隔を短く (1 秒の数分の 1) に設定することにより、アニメーションを実行するコードを記述できます。Timer クラスの使用の詳細については、184 ページの「タイマー間隔の制御」を参照してください。

次の例では、`circle` という名前の円 Sprite インスタンスを Stage 上に作成します。円がクリックされると、スクリプトで記述されたアニメーションのシーケンスが開始し、円が完全に透明になるまで `circle` がフェードします (その `alpha` プロパティが減少します)。

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// 円を描画し、表示リストに追加する。
```

```
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// アニメーションの開始時、この関数は各フレームごとに呼び出される。
// この関数によって行われる変更（各フレームごとに
// 画面に更新される）は次のとおり。
function fadeCircle(event:Event):void
{
  circle.alpha -= .05;

  if (circle.alpha <= 0)
  {
    circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
  }
}

function startAnimation(event:MouseEvent):void
{
  circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

円がクリックされると、fadeCircle() が enterFrame イベントのリスナーとしてサブスクライブされます（フレームごとに1回呼び出され始めます）。この関数では、alpha プロパティを変更することにより、circle をフェードさせます。したがって、フレームごとに1回、円の alpha が 0.05 (5%) ずつ減少し、画面が更新されます。最終的には、alpha 値が 0 (circle が完全に透明) になると、fadeCircle() 関数はリスナーとして削除され、アニメーションは終わります。

同じコードを使用して、たとえば、フェードではなく、モーションのアニメーションを作成できます。enterFrame イベントリスナーである関数内で、alpha に対するプロパティを置き換えることにより、このプロパティが代わりにアニメーション化されます。たとえば、次の行を

```
  circle.alpha -= .05;
```

次の行に

```
  circle.x += 5;
```

置き換えると、x プロパティがアニメーション化され、結果的に円がステージの右へ移動します。目的の x 座標に達したら、アニメーションを終了する条件を変更して、アニメーションを終了できます (enterFrame リスナーがサブスクライブ解除されます)。

表示コンテンツの動的ロード

次の外部表示アセットを ActionScript 3.0 アプリケーションにロードできます。

- ActionScript 3.0 でオーサリングされた SWF ファイル – Sprite、MovieClip、または Sprite を拡張する任意のクラスです。
- イメージファイル – JPG、PNG、および GIF ファイルが含まれます。
- AVM1 SWF ファイル – ActionScript 1.0 または 2.0 で記述された SWF ファイルです。

これらのアセットをロードするには、Loader クラスを使用します。

表示オブジェクトのロード

Loader オブジェクトは、SWF ファイルおよびグラフィックファイルをアプリケーションにロードするために使用されます。Loader クラスは、DisplayObjectContainer クラスのサブクラスです。Loader オブジェクトは、その表示リストに子表示オブジェクト (ロードされる SWF またはグラフィックファイルを表す表示オブジェクト) を 1 つしか含むことができません。表示リストに Loader オブジェクトを追加するときは、次のコードのように、子表示オブジェクトのロードが完了したら、ロードされた子表示オブジェクトも表示リストに追加します。

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

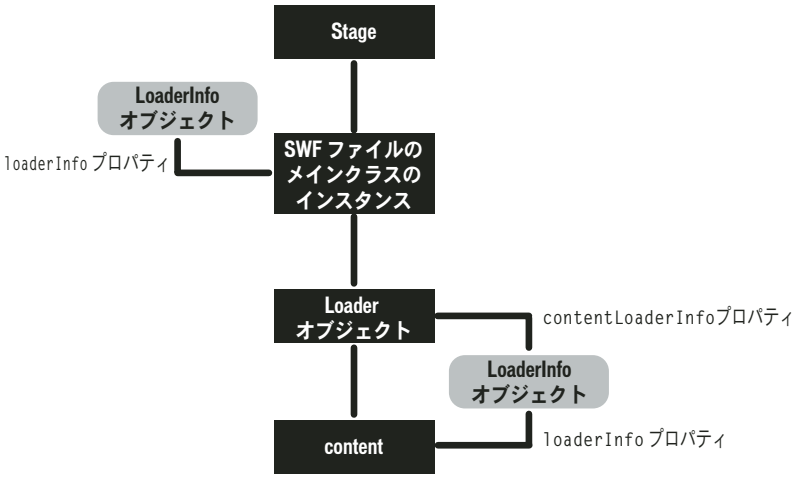
SWF ファイルまたはイメージがロードされたら、ロードされた表示オブジェクトを、次の例に示す `container` DisplayObjectContainer オブジェクトのような別の表示オブジェクトコンテナに移動できます。

```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
  container.addChild(pictLdr.content);
}
```

ロードの進行状況の監視

ファイルがロードを開始すると、LoaderInfo オブジェクトが作成されます。LoaderInfo オブジェクトは、ロードの進行状況、ローダする側およびロードされる側の URL、メディアの総バイト数、メディアの規格高さと幅などの情報を提供します。LoaderInfo オブジェクトは、ロードの進行状況を監視するためのイベントも送出します。

次の図は、LoaderInfo オブジェクトの各種使用方法を示します。LoaderInfo オブジェクトは、SWF ファイルのメインクラスのインスタンス、Loader オブジェクト、および Loader オブジェクトによってロードされたオブジェクトの場合でそれぞれ使用方法が異なります。



LoaderInfo オブジェクトは、Loader オブジェクトと、ロードされた表示オブジェクトの両方のプロパティとしてアクセスできます。LoaderInfo オブジェクトは、ロードが開始されると同時に、Loader オブジェクトの `contentLoaderInfo` プロパティを介してアクセスできます。また、表示オブジェクトのロード完了後は、表示オブジェクトの `loaderInfo` プロパティを介してロードされた表示オブジェクトのプロパティとして LoaderInfo オブジェクトにアクセスすることができます。ロードされた表示オブジェクトの `loaderInfo` プロパティは、Loader オブジェクトの `contentLoaderInfo` プロパティと同じ LoaderInfo オブジェクトを参照します。つまり、LoaderInfo オブジェクトは、ロードされたオブジェクトファイルとロードした Loader オブジェクト間 (ロードする側およびロードされる側の間) で共有されます。

ロードされたコンテンツのプロパティにアクセスするには、LoaderInfo オブジェクトにイベントリスナーを次のコードのように追加する必要があります。

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
  var content:Sprite = event.target.content;
  content.scaleX = 2;
}
```

詳細については、299 ページ、第 10 章の「イベントの処理」を参照してください。

ロードするコンテンツの指定

Loader クラスの `load()` または `loadBytes()` メソッドを介して Flash Player に外部ファイルをロードする場合、オプションで `context` パラメータを指定できます。このパラメータは LoaderContext オブジェクトです。

LoaderContext クラスには、ロードされたコンテンツの使用方法のコンテキストを定義できる次の 3 つのプロパティが含まれています。

- `checkPolicyFile`: このプロパティは、SWF ファイルではなくイメージファイルをロードする場合にのみ使用します。このプロパティを `true` に設定した場合、Loader は元のサーバーでクロスドメインポリシーファイルをチェックします (726 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」を参照)。これは、Loader オブジェクトを含む SWF ファイルのドメイン以外のドメインにあるコンテンツにのみ必要です。サーバーから Loader ドメインにアクセス許可が与えられている場合、Loader ドメイン内の SWF ファイルの ActionScript はロードされたイメージ内のデータにアクセスできます。つまり、`BitmapData.draw()` コマンドを使用して、ロードされたイメージ内のデータにアクセスできます。

  Loader オブジェクトのドメイン以外の他のドメインにある SWF ファイルは、`Security.allowDomain()` を呼び出して特定のドメインを許可できます。

- `securityDomain`: このプロパティは、イメージではなく SWF ファイルをロードする場合にのみ使用します。Loader オブジェクトを含むファイルのドメイン以外のドメインの SWF ファイルに対してこのプロパティを指定します。このオプションを指定すると、Flash Player はクロスドメインポリシーファイルが存在するかどうかをチェックします。ファイルが存在する場合は、クロスドメインポリシーファイルで許可されているドメインの SWF ファイルはロードされた SWF コンテンツをクロススクリプトできます。このパラメータとして `flash.system.SecurityDomain.currentDomain` を指定できます。
- `applicationDomain`: ActionScript 1.0 や 2.0 で記述されたイメージまたは SWF ファイルではなく、ActionScript 3.0 で記述された SWF ファイルをロードする場合にのみ、このプロパティを使用します。ファイルをロードするとき、`applicationDomain` パラメータを `flash.system.ApplicationDomain.currentDomain` に設定することで、ファイルが Loader オブジェクトと同じアプリケーションドメインに含まれるように指定できます。ロードされた SWF ファイルを同じアプリケーションドメインに置くと、そのクラスに直接アクセスできます。これは、関連付けられたクラス名を使用してアクセスできる埋め込みメディアを含む SWF ファイルをロードする場合に便利です。詳細については、666 ページの「ApplicationDomain クラスの使用」を参照してください。

次に示す例では、別のドメインからビットマップをロードするとき、クロスドメインポリシーファイルをチェックします。

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/
  photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

次に示す例では、別のドメインから SWF をロードするときに、クロスドメインポリシーファイルをチェックし、このファイルを Loader オブジェクトと同じセキュリティサンドボックス内に配置します。また、このコードは、ロードされた SWF ファイルのクラスを Loader オブジェクトと同じアプリケーションドメインに追加します。

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/
  library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の LoaderContext クラスを参照してください。

例 : SpriteArranger

SpriteArranger サンプルアプリケーションは、個別に説明した Geometric Shapes サンプルアプリケーション上に構築されます (170 ページの「例 : GeometricShapes」を参照)。

SpriteArranger サンプルアプリケーションは、表示オブジェクトの操作に関する次のような概念を示しています。

- 表示オブジェクトクラスの拡張
- 表示リストへのオブジェクトの追加
- 表示オブジェクトのレイヤーと表示オブジェクトコンテナの操作
- 表示オブジェクトイベントへの応答
- 表示オブジェクトのプロパティとメソッドの使用

このサンプルのアプリケーションファイルを入手するには、
www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。
SpriteArranger アプリケーションのファイルは、"Examples/SpriteArranger" フォルダに入っています。アプリケーションは、次のファイルで構成されています。

ファイル	説明
SpriteArranger.mxml または SpriteArranger.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/SpriteArranger/CircleSprite.as	画面上で円をレンダリングする Sprite オブジェクトの型を定義するクラス。
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	キャンバスを定義するクラス。キャンバスは、GeometricSprite オブジェクトを格納する表示オブジェクトコンテナです。
com/example/programmingas3/SpriteArranger/SquareSprite.as	画面上で四角形をレンダリングする Sprite オブジェクトの型を定義するクラス。
com/example/programmingas3/SpriteArranger/TriangleSprite.as	画面上で三角形をレンダリングする Sprite オブジェクトの型を定義するクラス。
com/example/programmingas3/SpriteArranger/GeometricSprite.as	Sprite オブジェクトを拡張するクラス。画面上のシェイプを定義するのに使用されます。CircleSprite、SquareSprite、および TriangleSprite は、それぞれこのクラスを拡張します。
com/example/programmingas3/geometricshapes/IGeometricShape.as	すべての図形クラスによって実装されるメソッドを定義する基本インターフェイス。
com/example/programmingas3/geometricshapes/IPolygon.as	複数の辺を持つ図形クラスによって実装されるメソッドを定義するインターフェイス。

ファイル	説明
com/example/programmingas3/geometricshapes/RegularPolygon.as	シェイプの中心から対称の位置に等しい長さの辺を有する一種の幾何学図形。
com/example/programmingas3/geometricshapes/Circle.as	円を定義する一種の幾何学図形。
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	正三角形を定義する RegularPolygon のサブクラス。
com/example/programmingas3/geometricshapes/Square.as	正四角形を定義する RegularPolygon のサブクラス。
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	指定された種類およびサイズでシェイプを作成するための " ファクトリメソッド " を含むクラス。

SpriteArranger クラスの定義

SpriteArranger アプリケーションを使用して、画面上の " キャンバス " に、各種表示オブジェクトを追加することができます。

DrawingCanvas クラスは、ユーザーが画面上のシェイプを追加できる描画領域、すなわち表示オブジェクトコンテナのタイプを定義します。これらの画面上のシェイプは、GeometricSprite クラスのいずれかのサブクラスのインスタンスです。

DrawingCanvas クラス

DrawingCanvas クラスは Sprite クラスを拡張するもので、この継承は次のように DrawingCanvas クラス宣言で定義されます。

```
public class DrawingCanvas extends Sprite
```

Sprite クラスは DisplayObjectContainer クラスと DisplayObject クラスのサブクラスであり、DrawingCanvas クラスは、これらのクラスのメソッドとプロパティを使用します。

`DrawingCanvas()` コンストラクタメソッドは、Rectangle オブジェクト、`bounds` を設定します。bounds は、後でキャンバスのアウトラインを描画するために使用されるプロパティです。その後、コンストラクタメソッドは次のようにして `initCanvas()` メソッドを呼び出します。

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

次の例に示すように、initCanvas() メソッドは DrawingCanvas オブジェクトの各種プロパティを定義します。これらのプロパティは、コンストラクタ関数へ引数として渡されたものです。

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

その後、initCanvas() メソッドは drawBounds() メソッドを呼び出し、そのメソッドにより、DrawingCanvas クラスの graphics プロパティを使用してキャンバスが描画されます。graphics プロパティは、Shape クラスから継承されます。

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
               bounds.top - 1,
               bounds.width + 2,
               bounds.height + 2);
this.graphics.endFill();
```

次に示す DrawingCanvas クラスの追加メソッドは、アプリケーションに対するユーザーの操作に基づいて呼び出されます。

- addShape() および describeChildren() メソッド。これらについては、407 ページの「キャンバスへの表示オブジェクトの追加」で説明します。
- moveToBack()、moveDown()、moveToFront()、および moveUp() メソッド。これらについては、410 ページの「表示オブジェクトのレイヤーの再配置」で説明します。
- onMouseUp() メソッド。これについては、409 ページの「表示オブジェクトのクリックとドラッグ」で説明します。

GeometricSprite クラスとそのサブクラス

ユーザーがキャンバスに追加できる表示オブジェクトは、GeometricSprite クラスの次のいずれかのサブクラスのインスタンスです。

- CircleSprite
- SquareSprite
- TriangleSprite

GeometricSprite クラスは、flash.display.Sprite クラスを拡張します。

```
public class GeometricSprite extends Sprite
```

GeometricSprite クラスは、すべての GeometricSprite オブジェクトに共通するいくつかのプロパティを含んでいます。それらは、関数に渡されたパラメータに基づいてコンストラクタ関数の中で設定されます。次に実際の使用例を示します。

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

GeometricSprite クラスの geometricShape プロパティは、IGeometricShape インターフェイスを定義し、このインターフェイスはシェイプの数学プロパティを定義しますが、ビジュアルプロパティは定義しません。IGeometricShape インターフェイスを実装するクラスは、GeometricShapes サンプルアプリケーションで定義されています (170 ページの「例 : GeometricShapes」を参照)。

GeometricSprite クラスは drawShape() メソッドを定義し、このメソッドは GeometricSprite の各サブクラス内のオーバーライド定義の中でさらに改良されます。詳細については、この後の「キャンバスへの表示オブジェクトの追加」の節を参照してください。

GeometricSprite クラスは、次のメソッドも備えています。

- onMouseDown() および onMouseUp() メソッド。これらについては、409 ページの「表示オブジェクトのクリックとドラッグ」で説明します。
- showSelected() および hideSelected() メソッド。これらについては、409 ページの「表示オブジェクトのクリックとドラッグ」で説明します。

キャンバスへの表示オブジェクトの追加

ユーザーが Add Shape ボタンをクリックすると、アプリケーションは DrawingCanvas クラスの addShape() メソッドを呼び出します。アプリケーションは、次の例に示すように、新しい GeometricSprite のインスタンス化するために、いずれかの GeometricSprite サブクラスの適切なコンストラクタ関数を呼び出します。

```
public function addShape(shapeName:String, len:Number):void
{
  var newShape:GeometricSprite;
  switch (shapeName)
  {
    case "Triangle":
      newShape = new TriangleSprite(len);
      break;

    case "Square":
      newShape = new SquareSprite(len);
      break;

    case "Circle":
      newShape = new CircleSprite(len);
      break;
```

```
    }
    newShape.alpha = 0.8;
    this.addChild(newShape);
}
```

各コンストラクタメソッドは drawShape() メソッドを呼び出し、このメソッドは、Sprite クラスから継承したクラスの graphics プロパティを使用して適切なベクターグラフィックを描画します。たとえば、CircleSprite クラスの drawShape() メソッドには、次のコードが含まれています。

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

addShape() 関数の最後から 2 番目の行は、DisplayObject クラスから継承した表示オブジェクトの alpha プロパティを設定します。これにより、キャンバスに追加される表示オブジェクトはわずかに透明になり、ユーザーはその背後にあるオブジェクトを見ることができます。

addChild() メソッドの最後の行は、既に表示リストに載っている新しい表示オブジェクトを DrawingCanvas クラスのインスタンスの子リストに追加します。これにより、新しい表示オブジェクトがステージに表示されます。

アプリケーションのインターフェイスには、selectedSpriteTxt と outputTxt の 2 つのテキストフィールドが含まれています。これらのテキストフィールドのテキストプロパティは、キャンバスに追加されるかユーザーによって選択された GeometricSprite オブジェクトに関する情報で更新されます。GeometricSprite クラスは、次のように toString() メソッドをオーバーライドすることにより、この情報報告タスクを処理します。

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " +
    this.y;
}
```

shapeType プロパティは、各 GeometricSprite サブクラスのコンストラクタメソッドの中で、適切な値に設定されます。たとえば、toString() メソッドは、DrawingCanvas インスタンスに新たに追加された CircleSprite インスタンスの次の値を返す場合があります。

```
Circle of size 50 at 0, 0
```

DrawingCanvas クラスの describeChildren() メソッドは、DisplayObjectContainer クラスから継承した numChildren プロパティを使用して for ループの限度を設定し、キャンバスの子リストをループします。このメソッドは、次のように、それぞれの子をリストしたストリングを生成します。

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
    child = this.getChildAt(i);
    desc += i + ": " + child + '\n';
}
```

結果のストリングは、outputTxt テキストフィールドの text プロパティを設定するために使用されます。

表示オブジェクトのクリックとドラッグ

ユーザーが GeometricSprite インスタンスをクリックすると、アプリケーションは onMouseDown() イベントハンドラを呼び出します。次に示すように、このイベントハンドラは GeometricSprite クラスのマウスダウンイベントを受け取るように設定されています。

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

次に、onMouseDown() メソッドは GeometricSprite オブジェクトの showSelected() メソッドを呼び出します。このオブジェクトに対してこのメソッドが最初に呼び出された場合は、selectionIndicator という名前の新しい Shape オブジェクトが作成され、次のように、その Shape オブジェクトの graphics プロパティを使用して赤いハイライトの矩形が描画されます。

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1,
  this.size + 1);
this.addChild(this.selectionIndicator);
```

onMouseDown() メソッドの呼び出しが最初でない場合、このメソッドは、単に DisplayObject クラスから継承した selectionIndicator シェイプの visible プロパティを次のように設定します。

```
this.selectionIndicator.visible = true;
```

hideSelected() メソッドは、前に選択されたオブジェクトの selectionIndicator シェイプを、その visible プロパティを false に設定することによって非表示にします。

また、onMouseDown() イベントハンドラは、次のコードを含んでいる startDrag() メソッドも呼び出します。このメソッドは、Sprite クラスから継承されたものです。

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

これにより、ユーザーは選択したオブジェクトを、キャンバスの boundsRect 矩形によって設定された境界内でドラッグできます。

ユーザーがマウスボタンを離すと、mouseUp イベントが送出されます。DrawingCanvas のコンストラクタメソッドは、次のイベントリスナーを設定します。

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

このイベントリスナーは、個々の GeometricSprite オブジェクト用でなく、DrawingCanvas オブジェクト用に設定されます。なぜなら、GeometricSprite オブジェクトは、ドラッグされた場合、マウスボタンが離されたときに、別の表示オブジェクト (別の GeometricSprite オブジェクト) の背後で終了する可能性があるからです。前景の表示オブジェクトは、マウスアップイベントを受け取りますが、ユーザーがドラッグしている表示オブジェクトは、それを受け取りません。DrawingCanvas オブジェクトにリスナーを追加すると、そのイベントが常に確実に処理されます。

onMouseUp() メソッドは GeometricSprite オブジェクトの onMouseUp() メソッドを呼び出し、後者のメソッドは GeometricSprite オブジェクトの stopDrag() メソッドを呼び出します。

表示オブジェクトのレイヤーの再配置

アプリケーションのユーザーインターフェイスには、Move Back、Move Down、Move Up、および Move to Front というラベルの付いたボタンが含まれます。ユーザーがこれらのボタンをクリックすると、アプリケーションは DrawingCanvas クラスの対応するメソッド (moveToBack()、moveDown()、moveUp()、または moveToFront()) を呼び出します。たとえば、moveToBack() メソッドには次のコードが含まれています。

```
public function moveToBack(shape:GeometricSprite):void
{
  var index:int = this.getChildIndex(shape);
  if (index > 0)
  {
    this.setChildIndex(shape, 0);
  }
}
```

このメソッドは、DisplayObjectContainer クラスから継承された setChildIndex() メソッドを使用して、表示オブジェクトを DrawingCanvas インスタンス (this) の子リスト内のインデックス位置 0 に位置付けます。

moveDown() メソッドもほとんど同じ働きをしますが、次のように、DrawingCanvas インスタンスの子リスト内で表示オブジェクトのインデックス位置を 1 ずつデクリメントする点が異なります。

```
public function moveDown(shape:GeometricSprite):void
{
  var index:int = this.getChildIndex(shape);
  if (index > 0)
  {
    this.setChildIndex(shape, index - 1);
  }
}
```

moveUp() および moveToFront() メソッドは、moveToBack() および moveDown() メソッドと同様の働きをします。

第 13 章

ジオメトリの操作

13

flash.geom パッケージには、ポイント、矩形、変換マトリックスなどのジオメトリオブジェクトを定義するクラスがあります。これらのクラスは、他のクラスで使用されるオブジェクトのプロパティを定義する際に使用します。

目次

ジオメトリの基本 411
Point オブジェクトの使用 414
Rectangle オブジェクトの使用 416
Matrix オブジェクトの使用 420
例 : マトリックス変換の表示オブジェクトへの適用 421

ジオメトリの基本

ジオメトリの操作の概要

学校でジオメトリ (幾何学) を学んだときには最小限の知識で切り抜けたという人が多いかもしれませんが、少しでもジオメトリの知識があると、ActionScript で非常に役立ちます。

flash.geom パッケージには、ポイント、矩形、変換マトリックスなどのジオメトリオブジェクトを定義するクラスがあります。これらのクラス自体は、必ずしも機能を提供するものではなく、他のクラスで使用されるオブジェクトのプロパティを定義する際に使用します。

すべてのジオメトリクラスは、画面上の位置は 2 次元の面で表されるという概念に基づいています。画面は、水平 (x) 軸と垂直 (y) 軸を持つ平面のグラフのように扱われます。画面上の任意の位置 (ポイント) は、x 値と y 値の組み合わせ、つまり座標で表すことができます。

ステージなどのすべての表示オブジェクトは、座標空間を持ちます。この座標空間が、子表示オブジェクトや描画などを配置するための、その表示オブジェクトにおけるグラフになります。通常は、原点 (x 軸と y 軸が交わる座標 0, 0 の位置) は表示オブジェクトの左上隅になります。これはステージには常に当てはまりますが、他の表示オブジェクトには当てはまらない場合もあります。標準の 2 次元座標系と同様に、x 軸の値が大きくなるほど右へ、小さくなるほど左へ移動します。原点の左側では x 座標は負の値になります。ただし、通常の座標系とは異なり、ActionScript では y 軸上の値が大きくなるほど画面の下へ、小さくなるほど上へ移動します。原点の上側では、y 座標は負の値になります。ステージの左上隅が座標空間の原点であるため、ステージ上のオブジェクトはいずれも x 座標が 0 より大きくなり、幅はステージよりも小さくなります。また、y 座標は 0 よりも大きくなり、高さはステージよりも小さくなります。

座標空間の個々のポイントを表すには、Point クラスのインスタンスを使用します。座標空間で矩形の領域を表すには、Rectangle インスタンスを作成します。上級ユーザーの場合は、Matrix インスタンスを使用して、表示オブジェクトに複数変換または複合変換を適用できます。回転、位置の変更、拡大 / 縮小倍率の変更などの多くの単純な変換は、オブジェクトのプロパティを使用して表示オブジェクトに直接適用できます。表示オブジェクトのプロパティを使用した変換の詳細については、376 ページの「表示オブジェクトの操作」を参照してください。

一般的なジオメトリのタスク

ActionScript のジオメトリクラスを使用して、次のような作業を実行できます。

- 2 つのポイント間の距離の計算
- 異なる座標空間のポイントの座標の決定
- 角度と距離を使用した表示オブジェクトの移動
- Rectangle インスタンスの操作
  - Rectangle インスタンスの再配置
  - Rectangle インスタンスのサイズ変更
  - 複数の Rectangle インスタンスの合計サイズや重なり領域の決定
- Matrix オブジェクトの作成
- 表示オブジェクトを変換するための Matrix オブジェクトの使用

重要な概念と用語

次の参照用の一覧に、この章で使用される重要な用語を示します。

- 直交座標 : 座標は一般に 2 つの数値で表されます (5, 12 や 17, -23 など)。この 2 つの数値はそれぞれ、x 座標と y 座標です。
- 座標空間 : 表示オブジェクトに含まれる座標グラフで、子エレメントがここに配置されます。
- 原点 : 座標空間で x 軸と y 軸が交わるポイント。このポイントの座標は 0, 0 です。
- ポイント : 座標空間内の 1 つの場所。ActionScript で使用する 2 次元座標系では、ポイントは x 軸と y 軸 における場所 (ポイントの座標) によって定義されます。
- 基準点 : 表示オブジェクトの座標空間の原点 (0, 0 座標)。
- 拡大 / 縮小 : オブジェクトの元のサイズを基準にしたサイズです。拡大 / 縮小は、オブジェクトを伸縮させてサイズを変更することを意味します。
- 平行移動 : ポイント座標を 1 つの座標空間から他の座標空間に変更すること。
- 変換 : 図形の視覚的特性を調整すること。オブジェクトの回転、尺度の変更、シェイプの傾斜や歪み、色の変更など。
- X 軸 : ActionScript で使用する 2 次元座標系の水平軸。
- Y 軸 : ActionScript で使用する 2 次元座標系の垂直軸。

章内の例の使用

この章の例の多くは、計算または値の変更を示す例です。これらの例のほとんどに、コードの結果を示すための適切な `trace()` 関数呼び出しが含まれています。これらの例をテストするには、次の操作を行います。

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   コードリストの `trace()` 関数の結果が [出力] パネルに表示されます。

この章の一部の例は、表示オブジェクトへの変形の適用を示しています。このような例では、例の結果は、テキスト出力ではなく視覚的に確認できます。変形の例をテストするには、次の操作を行います。

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. ステージ上にムービークリップシンボルインスタンスを作成します。たとえば、図形を描画し、それを選択してから [変更]-[シンボルに変換] を選択し、シンボルに名前を付けます。

5. ステージのムービークリップを選択した状態で、プロパティインスペクタでインスタンスにインスタンス名を付けます。名前は、サンプルコードリスト内の表示オブジェクトの名前と同じにします。たとえば、コードリストで myDisplayObject という名前のオブジェクトに変形を適用する場合は、ムービークリップインスタンスにも myDisplayObject という名前を付けます。
6. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   画面上で、コードリストでの指定に基づいてオブジェクトに変形が適用された結果を確認できます。

サンプルコードリストのテストの詳細については、60 ページの「章内のコード例のテスト」を参照してください。

Point オブジェクトの使用

Point オブジェクトは、座標の直交ペアを定義します。2 次元の座標系において、水平軸を表す x と垂直軸を表す y により 1 つのポイントを表現します。

Point オブジェクトを定義する際は、次のように x プロパティと y プロパティを設定します。

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

2 つのポイント間の距離の検索

Point クラスの distance() メソッドを使用すると、同じ座標空間内にある 2 つのポイント間について距離を調べることができます。たとえば、次のコードでは、同じ表示オブジェクトコンテナ内にある 2 つの表示オブジェクト circle1 および circle2 が持つ基準点間の距離を取得します。

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

座標空間の変換

2 つの表示オブジェクトがそれぞれ異なる表示オブジェクトコンテナ内にある場合は、それぞれの属する座標区間が異なっている可能性があります。DisplayObject クラスの localToGlobal() メソッドを使用すると、それらの座標を共通のグローバルな座標空間 (ステージ座標) に変換できます。たとえば、次のコードでは、別々の表示オブジェクトコンテナ内にある 2 つの表示オブジェクト circle1 および circle2 が持つ基準点間の距離を測定します。

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle1.x, circle1.y);
```

```
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

同様に、ステージ上の特定の点から表示オブジェクト target の基準点までの距離を検索するには、DisplayObject クラスの localToGlobal() メソッドを使用できます。

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

表示オブジェクトの指定された角度と距離の移動

移動距離と角度を指定してオブジェクトを移動するには、Point クラスの polar() メソッドを使用します。たとえば、次のコードでは、myDisplayObject オブジェクトを 60 度の方向に 100 ピクセル分だけ移動します。

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Point クラスのその他の使用

Point オブジェクトは次のメソッドおよびプロパティで使用できます。

クラス	メソッドまたはプロパティ	説明
DisplayObjectContainer	areInaccessibleObjectsUnderPoint() getObjectsUnderPoint()	表示オブジェクトコンテナ内のポイントに存在するオブジェクトのリストを返すために使用します。
BitmapData	hitTest()	BitmapData オブジェクト内のピクセルと、衝突のチェックに使用するポイントを定義するために使用します。

クラス	メソッドまたはプロパティ	説明
BitmapData	applyFilter() copyChannel() merge() paletteMap() pixelDissolve() threshold()	操作の対象とする長方形の位置を定義するために使用します。
Matrix	deltaTransformPoint() transformPoint()	変換を適用するポイントを定義するために使用します。
Rectangle	bottomRight size topLeft	これらのプロパティを定義するために使用します。

Rectangle オブジェクトの使用

Rectangle オブジェクトは、矩形領域を定義します。Rectangle オブジェクトには、位置 (左上隅の x 座標と y 座標で定義される) と、width プロパティ、height プロパティがあります。新しい Rectangle オブジェクトに対してこれらのプロパティを指定するには、次のように Rectangle() コンストラクタを呼び出します。

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

Rectangle オブジェクトのサイズ変更と再配置

Rectangle オブジェクトをサイズ変更および移動するには多数の方法があります。

x および y プロパティを変更すると、直接に Rectangle オブジェクトの位置を移動できます。ただし、Rectangle オブジェクトの幅と高さをこの方法で変更しても効果はありません。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

次のコードに示すように、Rectangle オブジェクトの left または top プロパティを変更した場合、オブジェクトはその x および y プロパティをそれぞれ left と top プロパティに一致させて再配置されます。ただし、Rectangle オブジェクトの左下隅の位置は変化せず、その代わりに長方形のサイズが変化します。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=30, y=20, w=70, h=30)
```

同様に、次の例で示すように、Rectangle オブジェクトの bottom または right プロパティを変更すると、左上隅の位置は変化せずに長方形のサイズが変化します。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

次のように、Rectangle オブジェクトの offset() メソッドでも位置を移動できます。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

offsetPt() メソッドも同様に機能しますが、パラメータとして x および y のオフセット値ではなく Point オブジェクトを指定します。

Rectangle オブジェクトの inflate() メソッドに dx および dy の 2 つのパラメータを指定してサイズを変更することもできます。dx パラメータは、長方形の中心に対して右辺と左辺を移動する量 (ピクセル単位) を表します。dy パラメータは、長方形の中心に対して上辺と下辺を移動する量 (ピクセル単位) を表します。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

inflatePt() メソッドも同様に機能しますが、パラメータとして dx および dy の値ではなく Point オブジェクトを指定します。

Rectangle オブジェクトの結合と交差の検索

union() メソッドは、2 つの矩形の境界で形成された矩形領域の検索に使用します。

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

intersection() メソッドは、2 つの矩形の重なり合う領域で形成される矩形領域の検索に使用します。

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

2 つの長方形に重なり合う部分があるかどうかを調べるには、intersects() メソッドを使用します。また、表示オブジェクトがステージ上の特定の領域内に配置されているかどうかを調べる場合にも intersects() メソッドを使用できます。次のコード例では、circle オブジェクトを含んだ表示オブジェクトコンテナの座標空間がステージの座標空間と同じであることが前提となっています。この例は、intersects() メソッドを使用して、表示オブジェクト circle が target1 および target2 Rectangle オブジェクトで定義されたステージの指定された領域と交差しているかどうかを判断する方法を示しています。

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

同様に、2 つの表示オブジェクトの境界を示す矩形に重なり合う部分があるかどうかを調べる場合にも intersects() メソッドを使用できます。DisplayObject クラスの getRect() メソッドを使用すると、表示オブジェクトの線により境界範囲に挿入される空白を考慮することができます。

Rectangle オブジェクトのその他の使用

Rectangle オブジェクトは次のメソッドおよびプロパティで使用されます。

クラス	メソッドまたはプロパティ	説明
BitmapData	applyFilter()、colorTransform()、copyChannel()、copyPixels()、draw()、fillRect()、generateFilterRect()、getColorBoundsRect()、getPixels()、merge()、paletteMap()、pixelDissolve()、setPixels()、および threshold()	BitmapData オブジェクトの領域を定義するパラメータの型として使用します。
DisplayObject	getBounds()、getRect()、scrollRect、scale9Grid	プロパティのデータ型または返されるデータ型として使用します。
PrintJob	addPage()	printArea パラメータを定義するために使用します。
Sprite	startDrag()	bounds パラメータを定義するために使用します。

クラス	メソッドまたはプロパティ	説明
TextField	getCharBoundaries()	戻り値型として使用します。
Transform	pixelBounds	データ型として使用します。

Matrix オブジェクトの使用

Matrix クラスは、2 つの座標空間の間におけるポイントのマッピング方法を決定する変換マトリックス (変換行列) を表します。Matrix オブジェクトのプロパティを設定し、その Matrix オブジェクトを Transform オブジェクトの matrix プロパティに適用し、さらに、その Transform オブジェクトを表示オブジェクトの transform プロパティに適用することで、表示オブジェクトに対してさまざまなグラフィック変換を実行できます。実現できる変換機能としては、平行移動 (x 位置および y 位置の移動)、回転、拡大 / 縮小、傾斜などがあります。

Matrix オブジェクトの定義

マトリックスは、プロパティ (a、b、c、d、tx、ty) を調整して直接に定義することもできますが、createBox() メソッドを使用して定義するほうが簡単です。このメソッドのパラメータでは、作成するマトリックスに定義する拡大 / 縮小、回転、平行移動の効果を直接に定義できます。たとえば、次のコードで作成している Matrix オブジェクトには、水平方向 2.0 倍の拡大、垂直方向 3.0 倍の拡大、45 度の回転、右に 10 ピクセルの移動、および下に 20 ピクセルの移動の効果があります。

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

Matrix オブジェクトの拡大 / 縮小、回転、平行移動のエフェクトは、scale()、rotate()、および translate() メソッドを使用して調整することもできます。これらのメソッドで指定した値は、既存の Matrix オブジェクトに設定されている値と組み合わされます。たとえば、次のコードでは scale() と rotate() を 2 回呼び出しているため、Matrix オブジェクトの効果は 4 倍の拡大と 60 度の回転になります。

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

Matrix オブジェクトに傾斜変換を適用するには、オブジェクトの b または c プロパティをそれぞれ調整します。b プロパティを調整するとマトリックスが垂直方向に傾斜し、c プロパティを調整するとマトリックスが水平方向に傾斜します。次のコードでは、myMatrix Matrix オブジェクトが垂直に 2 倍傾斜しています。

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

Matrix による変換は、表示オブジェクトの transform プロパティに適用できます。たとえば、次のコードでは myDisplayObject という表示オブジェクトにマトリックス変換を適用しています。

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

1 行目では、myDisplayObject 表示オブジェクトで使用される既存の変換マトリックスに Matrix オブジェクトが設定されます (myDisplayObject 表示オブジェクトの transformation プロパティの matrix プロパティ)。こうすることで、表示オブジェクトの現状の位置、拡大率、回転角をふまえた上に、Matrix クラスメソッドの実行による効果を累積的に適用できます。

メモ

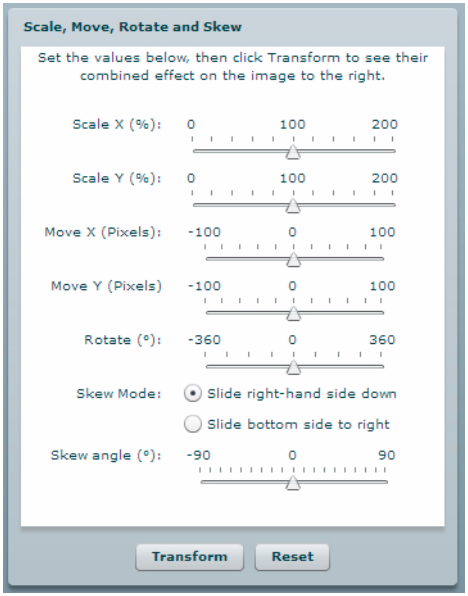
flash.geometry パッケージには ColorTransform クラスも含まれていますが、このクラスは Transform オブジェクトの colorTransform プロパティを設定する際に使用するものです。幾何学変換には使用しないため、この章では説明しません。詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の ColorTransform クラスを参照してください。

例: マトリックス変換の表示オブジェクトへの適用

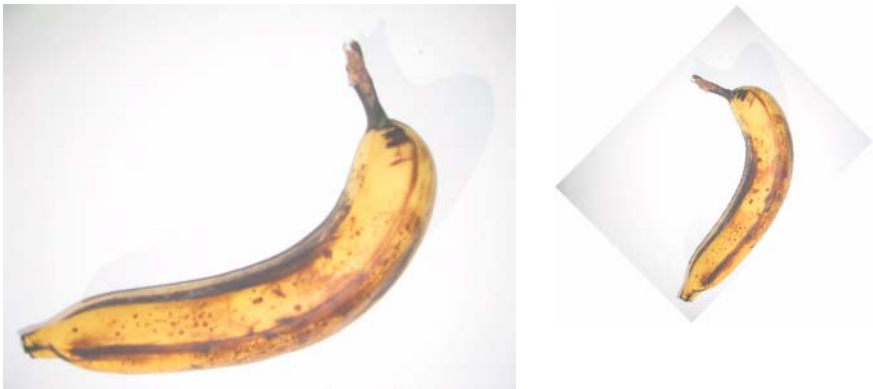
DisplayObjectTransformer サンプルアプリケーションには、Matrix クラスを使用して表示オブジェクトを変換する、次のような機能がいくつか示されます。

- 表示オブジェクトの回転
- 表示オブジェクトの拡大 / 縮小
- 表示オブジェクトの平行移動 (再配置)
- 表示オブジェクトの傾斜

このアプリケーションでは、マトリックス変換のパラメータを次のように調整するためのインターフェイスが使用できます。



ユーザーが [変換] ボタンをクリックすると、アプリケーションにより適切な変換が適用されます。



元の表示オブジェクトと、-45° 回転し、50% 拡大 / 縮小した表示オブジェクト

このサンプルのアプリケーションファイルを入手するには、
www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。
DisplayObjectTransformer アプリケーションのファイルは、"Samples/DisplayObjectTransformer" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
DisplayObjectTransformer.mxml または DisplayObjectTransformer.fla	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル
com/example/programmingas3/geometry/MatrixTransformer.as	マトリックス変換を適用するためのメソッドが含まれているクラス
img/	アプリケーションが使用するサンプルイメージファイルが保存されているディレクトリ

MatrixTransformer クラスの定義

MatrixTransformer クラスには、Matrix オブジェクトの図形変換を適用する静的メソッドが含まれます。

transform() メソッド

`transform()` メソッドには、次の各パラメータが含まれます。

- `sourceMatrix` – メソッドで変換される入力マトリックス
- `xScale` と `yScale` – x と y の縮尺率
- `dx` と `dy` – x と y の変換ピクセル数
- `rotation` – 回転度数
- `skew` – 傾斜率 (%)
- `skewType` – 傾斜の方向、" 右 " または " 左 "

戻り値は、結果のマトリックスです。

`transform()` メソッドは、クラスの次の静的メソッドを呼び出します。

- `skew()`
- `scale()`
- `translate()`
- `rotate()`

それぞれ、ソースマトリックスに変換を適用して返します。

skew() メソッド

skew( ) メソッドは、マトリックスの b および c プロパティを調整して、マトリックスを傾斜します。オプションのパラメータ unit によって、傾斜角度の定義に使用する単位が決まります。必要に応じて、メソッドは angle 値をラジアンに変換します。

```
if (unit == "degrees")
{
  angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
  angle = Math.PI * 2 * angle / 100;
}
```

傾斜変換を適用するために、skewMatrix Matrix オブジェクトが作成され、調整されます。最初は、次のように、同一のマトリックスです。

```
var skewMatrix:Matrix = new Matrix();
```

skewSide パラメータによって、傾斜を適用する側が決まります。"right" が設定されている場合、次のコードでマトリックスの b プロパティが設定されます。

```
skewMatrix.b = Math.tan(angle);
```

それ以外の場合、次のように Matrix の c プロパティが調整され、下側が傾斜されます。

```
skewMatrix.c = Math.tan(angle);
```

次の例で示すように、2 つのマトリックスを連結することによって、結果の傾斜が既存のマトリックスに適用されます。

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

scale () メソッド

次の例で示すように、scale( ) メソッドはまず、縮尺率がパーセントで指定されている場合、調整します。次に、マトリックスオブジェクトの scale( ) メソッドを使用します。

```
if (percent)
{
  xScale = xScale / 100;
  yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

translate () メソッド

translate( ) メソッドは、次のようにマトリックスオブジェクトの translate( ) メソッドを呼び出し、単純に、dx および dy 平行移動係数を適用します。

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

rotate () メソッド

rotate( ) メソッドは、入力回転角度をラジアンに変換し (度またはグラデーションで指定されている場合)、マトリックスオブジェクトの rotate( ) メソッドを呼び出します。

```
if (unit == "degrees")
{
  angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
  angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

アプリケーションからの MatrixTransformer.transform() メソッドの呼び出し

アプリケーションには、ユーザーが変換パラメータを指定するためのユーザーインターフェイスがあります。パラメータが指定されると、次のように、パラメータと表示オブジェクトの transform プロパティの matrix プロパティを Matrix.transform( ) メソッドに渡します。

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
                                         xScaleSlider.value,
                                         yScaleSlider.value,
                                         dxSlider.value,
                                         dySlider.value,
                                         rotationSlider.value,
                                         skewSlider.value,
                                         skewSide );
```

次に、アプリケーションは戻り値を表示オブジェクトの transform プロパティの matrix プロパティに適用します。これによって、変換がトリガされます。

```
img.content.transform.matrix = tempMatrix;
```

第 14 章

描画 API の使用

14

イメージやアートワークの読み込みは重要ですが、描画 API という機能を使用すると、ActionScript で線やシェイプを描画できます。これにより、コンピュータ上で白いキャンバスに任意のイメージを作成するアプリケーションを起動することもできるようになります。独自のグラフィックを作成する機能により、アプリケーションの可能性が広がります。この章で説明する方法を使用すると、描画プログラムの作成、アニメーション化されたインタラクティブアートの作成、プログラムを使用した独自のユーザーインターフェイスエレメントの作成をはじめ、さまざまな処理を行うことができます。

目次

描画 API の使用の基礎 428
Graphics クラスについて 429
線と曲線の描画 430
ビルトインメソッドを使用したシェイプの描画 433
グラデーションの線と塗りの作成 434
描画メソッドでの Math クラスの使用 438
描画 API でのアニメーション化 439
例 : Algorithmic Visual Generator 440

描画 API の使用の基礎

描画 API の使用の概要

描画 API は、ActionScript に組み込まれた機能の名前です。この機能を使用すると、線、曲線、シェイプ、塗り、グラデーションなどのベクターグラフィックを作成し、ActionScript でそれらを画面上に表示できます。flash.display.Graphics クラスがこの機能を提供します。Shape、Sprite、または MovieClip の各クラスに定義された `graphics` プロパティで、任意の Shape、Sprite、または MovieClip インスタンスに対して ActionScript を使用して描画できます。各クラスの `graphics` プロパティは、実際は Graphics クラスのインスタンスです。

コードを使用した描画に不慣れな場合、Graphics クラスには、円、楕円、矩形、および角の丸い矩形などの一般的なシェイプを簡単に描画するためのいくつかのメソッドがあります。それらは表示されない線としても、塗りつぶされたシェイプとしても描画できます。より高度な機能が必要な場合は、Graphics クラスには線および二次ベジェ曲線を描画するためのメソッドもあり、Math クラスの三角関数と組み合わせて、任意のシェイプを作成できます。

描画 API の一般的なタスク

この章では、ActionScript の描画 API を使用して実行できる次のタスクについて説明しています。

- シェイプを描画するための線のスタイルと塗りのスタイルの定義
- 直線と曲線の描画
- 円、楕円、および矩形などのシェイプを描画するメソッドの使用
- グラデーションの線と塗りを使用した描画
- グラデーション作成用のマトリックスの定義
- 描画 API での三角関数の使用
- 描画 API のアニメーションへの組み込み

重要な概念と用語

次の参照リストに、この章で使用されている重要な用語を示します。

- アンカーポイント : 二次ベジェ曲線の 2 つの端点の 1 つ。
- コントロールポイント ： 二次ベジェ曲線の方向と量を定義するポイント。曲線はコントロールポイントを通りません。コントロールポイントに引き付けられているような曲線が描画されます。
- 座標空間 : 表示オブジェクトに含まれる座標グラフで、子エレメントがここに配置されます。
- 塗り : 色で塗りつぶされた輪郭のあるシェイプの内側の単色の部分、または輪郭のないシェイプ全体。
- グラデーション : ある色から別の 1 つ以上の色への段階的な変化を含む色 (単色とは対照的な概念)。

- ポイント : 座標空間内の 1 つの場所。ActionScript で使用する 2 次元座標系では、ポイントは x 軸と y 軸における場所 (ポイントの座標) によって定義されます。
- 二次ベジェ曲線 : 特定の数式で定義される一種の曲線。この種の曲線のシェイプは、アンカーポイント (曲線の端点) の位置と、曲線の量と方向を定義するコントロールポイントの位置を基づいて計算されます。
- 拡大 / 縮小 : オブジェクトの元のサイズを基準とするサイズです。拡大 / 縮小とは、オブジェクトを伸縮させてサイズを変更することを意味します。
- 境界線 : シェイプの輪郭の、色で塗りつぶされた線の部分。または塗りのないシェイプの輪郭。
- 平行移動 : ポイント座標を 1 つの座標空間から他の座標空間に変更すること。
- X 軸 : ActionScript で使用する 2 次元座標系の水平軸。
- Y 軸 : ActionScript で使用する 2 次元座標系の垂直軸。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章はビジュアルコンテンツに関するものなので、コードリストをテストするには、コードを実行し、作成された SWF で結果を確認する必要があります。コードリストをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   作成された SWF ファイルで、コードリストの結果を確認できます。

Graphics クラスについて

Shape、Sprite、および MovieClip の各オブジェクトには、Graphics クラスのインスタンスである `graphics` プロパティがあります。Graphics クラスには、線、塗り、およびシェイプを描画するためのプロパティおよびメソッドがあります。表示オブジェクトを、コンテンツを描画するキャンバスとしてのみ使用する場合は、Shape インスタンスを使用します。Shape インスタンスは、他の表示オブジェクトよりも描画の性能が優れています。Sprite クラスや MovieClip クラスにある追加機能によるオーバーヘッドがないためです。表示オブジェクト上にグラフィックコンテンツを描画して、他の表示オブジェクトも含める場合は、Sprite インスタンスを使用します。さまざまなタスクにどの表示オブジェクトを使用するかを判断する方法の詳細については、375 ページの「DisplayObject サブクラスの選択」を参照してください。

線と曲線の描画

Graphics インスタンスを使用して行うすべての描画は、線と曲線の基本的な描画に基づいています。そのため、すべての ActionScript 描画は同じ一連の手順で実行される必要があります。

- 線と塗りのスタイルの定義
- 最初の描画位置の設定
- 線、曲線、およびシェイプの描画 (オプションで、描画ポイントの移動)
- 塗りの作成の完了 (必要な場合)

線と塗りのスタイルの定義

Shape、Sprite、または MovieClip インスタンスの `graphics` プロパティを使用して描画するには、まず、描画に使用するスタイル (線のサイズと色、塗りの色) を定義する必要があります。Adobe Flash CS3 Professional などの描画アプリケーションで描画ツールを使用するときと同様に、ActionScript を使用して描画するときも、境界線や塗りの色の有無を指定して描画できます。境界線の外観は `lineStyle()` メソッドまたは `lineGradientStyle()` メソッドを使用して指定します。単色の線を作成するには `lineStyle()` メソッドを使用します。このメソッドを呼び出す場合に指定する最も一般的な値は、最初の 3 つのパラメータである、線の太さ、色、およびアルファです。たとえば、次のコード行は、`myShape` という名前の Shape に、線の太さが 2 ピクセルで、色が赤 (0x990000)、不透明度が 75 % の線を描画するよう指定しています。

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

アルファパラメータのデフォルト値は 1.0 (100%) であるため、完全に不透明な線を描画する場合はこのパラメータを指定しないこともできます。`lineStyle()` メソッドは、ピクセルのヒンティングと拡大 / 縮小モードの 2 つの追加パラメータも受け取ります。これらのパラメータの使用方法の詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の `Graphics.lineStyle()` メソッドの説明を参照してください。

グラデーションの線を作成するには、`lineGradientStyle()` メソッドを使用します。このメソッドについては、434 ページの「グラデーションの線と塗りの作成」を参照してください。

塗りのシェイプを作成する場合、描画を開始する前に、`beginFill()` メソッド、`beginGradientFill()` メソッド、または `beginBitmapFill()` メソッドを呼び出します。これらのメソッドの中で最も基本的な `beginFill()` メソッドは、塗りの色と塗りの色のアルファ値 (オプション) の 2 つのパラメータを受け取ります。たとえば、単色の緑の塗りでシェイプを描画するには、次のコードを使用します (`myShape` という名前のオブジェクトに描画すると仮定)。

```
myShape.graphics.beginFill(0x00FF00);
```

塗りメソッドを呼び出すと、新しい塗りを開始する前に、暗黙的に前の塗りを終了します。境界線のスタイルを指定したメソッドを呼び出すと、前の境界線が置き換えられますが、前に指定した塗りは変更されません。逆の場合も同様です。

線のスタイルと塗りのプロパティを指定したら、次に描画の開始点を指定します。Graphics インスタンスには、用紙上のペン先のような描画ポイントがあります。描画ポイントが位置するところから、次の描画アクションが開始されます。初期状態では、Graphics オブジェクトは、描画するオブジェクトの座標空間のポイント 0, 0 という描画ポイントから描画を開始します。別のポイントから描画を開始するには、まず `moveTo()` メソッドを呼び出してから、いずれかの描画メソッドを呼び出します。これは、ペン先を用紙から上げて新しい場所に移動するのと似ています。

描画ポイントを適切な場所に配置してから、`lineTo()` メソッド (直線の描画) や `curveTo()` メソッド (曲線の描画) に対する一連の呼び出しを使用して描画します。

ヒント	描画中はいつでも `moveTo()` メソッドを呼び出して、描画せずに描画ポイントを新しい位置に移動することができます。

塗りの色を指定している場合、描画中に `endFill()` メソッドを呼び出して塗りを終了するように指定できます。閉じたシェイプを描画していない場合 (つまり、`endFill()` を呼び出したときに描画ポイントがそのシェイプの開始点にない場合)、`endFill()` メソッドを呼び出した時点で、現在の描画ポイントから最後に `moveTo()` 呼び出しで指定した位置まで自動的に直線が描画され、閉じたシェイプが描画されます。塗りを開始して `endFill()` を呼び出していない場合、`beginFill()` (または他のいずれかの塗りメソッド) を呼び出すと、現在の塗りが終了し、新しい塗りが開始されます。

直線の描画

`lineTo()` メソッドを呼び出すと、Graphics オブジェクトは現在の描画ポイントからこのメソッド呼び出しの 2 つのパラメータで指定した座標まで、指定した線のスタイルで直線を描画します。たとえば、次のコード行では、描画ポイントをポイント 100, 100 に配置し、ポイント 200, 200 まで線を描画します。

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

次の例では、高さ 100 ピクセルの赤と緑の三角形を描画します。

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight/2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight/2, 0);

// green triangle, starting at point 200, 0
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight/2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
```

```
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight/2, 0);

this.addChild(triangle);
```

曲線の描画

curveTo( ) メソッドは、二次ベジェ曲線を描画します。このメソッドでは、2 つのポイント (アンカーポイント) をつなぐ弧を、3 番目のポイント (コントロールポイント) に向けて曲げながら描画します。Graphics オブジェクトは現在の描画位置を最初のアンカーポイントとして使用します。curveTo( ) メソッドを呼び出す場合、コントロールポイントの x 座標と y 座標、2 番目のアンカーポイントの x 座標と y 座標の順に、4 つのパラメータを渡します。たとえば、次のコードでは、ポイント 100, 100 から始まりポイント 200, 200 で終わる曲線を描画します。コントロールポイントがポイント 175, 125 にあるため、まず右に、その後下に移動する曲線を作成します。

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

次の例では、幅と高さが 100 ピクセルの赤と緑の丸いオブジェクトを作成します。二次ベジェ曲線の特徴として、これらのオブジェクトは完全な円ではありません。

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);

// green circular shape
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

ビルトインメソッドを使用したシェイプの描画

ActionScript 3.0 には、円、楕円、矩形、および角の丸い矩形などの一般的なシェイプを描画できる便利なメソッドがあります。それは、Graphics クラスの drawCircle()、drawEllipse()、drawRect()、drawRoundRect()、および drawRoundRectComplex() メソッドです。これらのメソッドは lineTo() メソッドおよび curveTo() メソッドの代わりに使用できます。ただし、これらのメソッドを呼び出す前にも、線と塗りのスタイルを指定しておく必要があります。

次の例は、前に示した例と同様に、幅と高さが 100 ピクセルの赤、緑、および青の四角形を描画します。このコードでは、drawRect() メソッドを使用し、さらに塗りの色に 50 % (0.5) のアルファを指定しています。

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

Sprite オブジェクトまたは MovieClip オブジェクトでは、graphics プロパティを使用して作成された描画コンテンツは常に、オブジェクトに含まれているすべての子表示オブジェクトの背面に表示されます。また、graphics プロパティのコンテンツは個別の表示オブジェクトではないため、Sprite オブジェクトまたは MovieClip オブジェクトの子のリストには表示されません。たとえば、次の Sprite オブジェクトには graphics プロパティを使用して描画した円があり、その子表示オブジェクトのリストには TextField オブジェクトがあります。

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

TextField は、グラフィックオブジェクトで描画された円の最上部に表示されます。

グラデーションの線と塗りの作成

グラフィックオブジェクトでは、線と塗りを単色ではなく、グラデーションで描画することもできます。グラデーションの境界線は lineGradientStyle() メソッドで作成し、グラデーションの塗りは beginGradientFill() メソッドで作成します。

どちらのメソッドも同じパラメータを受け取ります。最初の 4 つのパラメータ、タイプ、色、アルファ、および比率は必須です。残りの 4 つのパラメータはオプションですが、高度なカスタマイズに役立ちます。

- 最初のパラメータには、作成するグラデーションのタイプを指定します。指定できる値は、GradientFill.LINEAR または GradientFill.RADIAL です。
- 2 番目のパラメータには、使用するカラー値の配列を指定します。線状グラデーションの場合、色は左から右に配置されます。放射状グラデーションの場合、色は内側から外側に配置されます。配列での色の順序は、グラデーションで色が描画される順序を表します。
- 3 番目のパラメータには、前のパラメータの色に対応するアルファ透明度値を指定します。
- 4 番目のパラメータには、各色のグラデーションの中での強調度を示すための比率を指定します。指定できる値の範囲は 0 ～ 255 です。これらの値は幅や高さではなく、グラデーション内での位置を表します。0 がグラデーションの始まりを表し、255 がグラデーションの終わりを表します。比率の配列は、順に増やしていく必要があり、2 番目および 3 番目のパラメータで指定した色およびアルファの配列の両方とエントリの数が同じである必要があります。

5 番目のパラメータである変換マトリックスはオプションですが、グラデーションの外観を制御する簡単で強力な方法であるため、よく使用されます。このパラメータは Matrix インスタンスを受け取ります。グラデーション用の Matrix オブジェクトを最も簡単に作成するには、Matrix クラスの createGradientBox() メソッドを使用します。

グラデーションで使用する Matrix オブジェクトの定義

シェイプで使用するグラデーションを定義するには、flash.display.Graphics クラスの beginGradientFill() および lineGradientStyle() メソッドを使用します。グラデーションを定義する際、これらのメソッドのパラメータでマトリックスを指定します。

マトリックスを最も簡単に定義するには、Matrix クラスの createGradientBox() メソッドを使用してグラデーション定義用のマトリックスを作成します。createGradientBox() メソッドに指定するパラメータでは、グラデーションの拡大 / 縮小、回転、および位置を定義します。

createGradientBox() メソッドは次のパラメータを受け取ります。

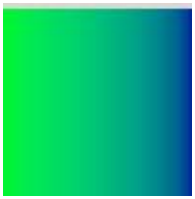
- グラデーションボックスの幅 : グラデーションが広がる幅 (ピクセル単位)
- グラデーションボックスの高さ : グラデーションが広がる高さ (ピクセル単位)
- グラデーションボックスの回転 : グラデーションに適用される回転角度 (ラジアン単位)
- 水平移動 : グラデーションが水平方向に移動する距離 (ピクセル単位)
- 垂直移動 : グラデーションが垂直方向に移動する距離 (ピクセル単位)

たとえば、次の性質を持つグラデーションを作成するとします。

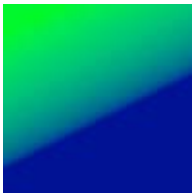
- GradientType.LINEAR
- 緑と青の２色、ratios 配列を [0, 255] に設定
- SpreadMethod.PAD
- InterpolationMethod.LINEAR_RGB

次の例で示すグラデーションは、createGradientBox() メソッドの rotation パラメータが異なりますが、その他の設定はすべて同じです。

```
width = 100;
height = 100;
rotation = 0;
tx = 0;
ty = 0;
```



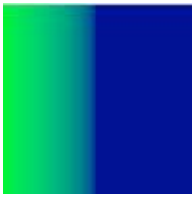
```
width = 100;
height = 100;
rotation = Math.PI/4; // 45°
tx = 0;
ty = 0;
```



```
width = 100;
height = 100;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;
```

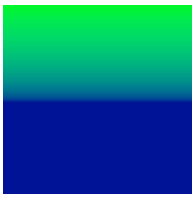
次の例で示す緑〜青の線状グラデーション効果は、createGradientBox() メソッドの rotation、tx、ty パラメータが異なりますが、その他の設定はすべて同じです。

```
width = 50;
height = 100;
rotation = 0;
tx = 0;
ty = 0;
```

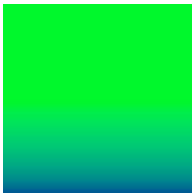


```
width = 50;
height = 100;
rotation = 0
tx = 50;
ty = 0;
```

```
width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;
```

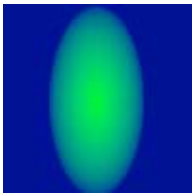


```
width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 50;
```



createGradientBox() メソッドの width、height、tx、ty パラメータは、次の例で示すように、放射状グラデーションの塗りに関するサイズと位置にも影響します。

```
width = 50;
height = 100;
rotation = 0;
tx = 25;
ty = 0;
```



次のコードは、前の図の最後に示した放射状グラデーションを生成します。

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                 colors,
                 alphas,
                 ratios,
                 matrix,
                 spreadMethod,
                 interp,
                 focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

グラデーションの塗りの幅と高さは、Graphics オブジェクトを使用して描画される幅と高さではなく、グラデーションマトリックスの幅と高さによって決定されます。Graphics オブジェクトを使用して描画する場合、グラデーションマトリックスのその座標に存在するものが描画されます。drawRect( ) などの Graphics オブジェクトのシェイプメソッドのいずれかを使用した場合でも、グラデーションは描画されるシェイプのサイズには伸縮されません。グラデーションのサイズはグラデーションマトリックス自体で指定する必要があります。

次のコードを使用して、グラデーションマトリックスのサイズと描画自体のサイズの視覚的な違いを説明します。

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000, 0x00FF00,
   0x0000FF], [1, 1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
```

```
myShape.graphics.drawRect(0, 50, 100, 40);
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

このコードは、赤、緑、青の色配分を同一に指定した塗りのスタイルを持つ 3 つのグラデーションを描画します。グラデーションは、drawRect() メソッドを使用して、それぞれピクセル幅 50、100、150 で描画されます。beginGradientFill() メソッドで指定するグラデーションマトリックスは、100 ピクセルの幅で作成されます。つまり、最初のグラデーションはグラデーション領域の半分のみを占め、2 番目は領域全体を占め、3 番目は領域全体を占めて、さらに青を 50 ピクセル右に拡張します。

lineGradientStyle() メソッドは beginGradientFill() と同様に動作しますが、グラデーションを定義するだけでなく、描画の前に、lineStyle() メソッドを使用して境界線の幅を指定する必要があります。次のコードは、赤、緑、および青のグラデーションの境界線でボックスを描画します。

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000, 0x00FF00,
   0x0000FF], [1, 1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

Matrix クラスの詳細については、420 ページの「Matrix オブジェクトの使用」を参照してください。

描画メソッドでの Math クラスの使用

Graphics オブジェクトを使用して円や四角形を描画できますが、特に、その描画メソッドを Math クラスのプロパティやメソッドと組み合わせて使用すると、より複雑なフォームを描画することもできます。Math クラスには、円周と円の直径の比を表す定数 Math.PI (約 3.14159265...) など、一般的な数学定数があります。また、Math.sin()、Math.cos()、および Math.tan() などの三角関数のメソッドも含まれます。これらのメソッドおよび定数を使用してシェイプを描画すると、特に繰り返しや再帰を使用した場合、より動的な視覚効果を作成できます。

Math クラスの多くのメソッドでは、角度の単位に度ではなくラジアンを使用します。Math クラスは、これら 2 種類の単位間の変換によく使用されます。

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

次の例では、正弦波と余弦波を作成して、特定の値での Math.sin() メソッドと Math.cos() メソッドの違いを示します。

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
  var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
  var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

  graphics.beginFill(sinWaveColor);
  graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
  graphics.beginFill(cosWaveColor);
  graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

描画 API でのアニメーション化

描画 API を使用してコンテンツを作成する利点の 1 つは、コンテンツの配置が 1 度だけに制限されないことです。描画に使用した変数を保持または変更して、描画したコンテンツを変更できます。一定期間のフレーム上でまたはタイマーを使用して、変数を変更し再描画すると、アニメーションを表示できます。

たとえば、次のコードは、Event.ENTER_FRAME イベントを監視して、通過フレームごとに表示を変更していきます。現在の度数を増やしていき、グラフィックオブジェクトをクリアして更新した位置に再描画します。

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);

addEventListener(Event.ENTER_FRAME, doEveryFrame);
```

```
function doEveryFrame(event:Event):void
{
  currentDegrees += 4;
  var radians:Number = getRadians(currentDegrees);
  var posX:Number = Math.sin(radians) * radius;
  var posY:Number = Math.cos(radians) * radius;
  satellite.graphics.clear();
  satellite.graphics.beginFill(0);
  satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
    return degrees * Math.PI / 180;
}
```

大きく異なる結果を生成するには、コードの先頭の初期シード変数 currentDegrees、radius、および satelliteRadius を変更して試してみます。たとえば、radius 変数を減らしたり、totalSatellites 変数を増やします。これは、描画 API が複雑なビジュアル表示をどれだけ簡単に作成できるかを示す一例にすぎません。

例 : Algorithmic Visual Generator

この例に示す Algorithmic Visual Generator は、ステージに複数の " サテライト " (円軌道を移動する円) を動的に描画します。ここでは、次の機能について説明します。

- 描画 API を使用した、動的な外観を持つ基本的なシェイプの描画
- 描画に使用するプロパティとユーザー操作との連動
- 各フレーム上のステージをクリアして再描画することによる、アニメーションの表示

前のサブセクションの例では、Event.ENTER_FRAME イベントを使用して、単独の " サテライト " をアニメーション化しました。この例では、前の例を拡張して、複数のサテライトのビジュアル表示を即時に更新する一連のスライダがあるコントロールパネルを作成します。この例では、コードを外部クラスに形式化し、サテライト作成コードを、各サテライトへの参照を satellites 配列に格納するループにラップします。

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。このアプリケーションのファイルは、"Samples/AlgorithmicVisualGenerator" フォルダにあります。このフォルダには次のファイルが含まれます。

ファイル	説明
AlgorithmicVisualGenerator.fla	Flash (FLA) のメインアプリケーションファイル。
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	ステージにサテライトを描画する機能や、コントロールパネルからのイベントに応答して、サテライトの描画に影響する変数を更新する機能など、アプリケーションの主要な機能を提供するクラス。
com/example/programmingas3/algorithmic/ControlPanel.as	複数のスライダを使用したユーザー操作を管理し、それに応答してイベントを送出するクラス。
com/example/programmingas3/algorithmic/Satellite.as	中心点の周囲の軌道内を回転する表示オブジェクトを表すクラス。現在の描画状態に関連するプロパティが含まれます。

リスナーの設定

このアプリケーションは、まず 3 つのリスナーを作成します。最初のリスナーは、サテライトを再構築する必要がある、コントロールパネルから送出されるイベントを監視します。2 番目のリスナーは SWF ファイルのステージのサイズ変更を監視します。3 番目のリスナーは SWF ファイルの各通過フレームを監視し、`doEveryFrame()` 関数を使用して再描画します。

サテライトの作成

これらのリスナーを設定したら、`build()` 関数を呼び出します。この関数は、最初に `clear()` 関数を呼び出します。この関数は、`satellites` 配列を空にしてステージ上の前の描画をすべてクリアします。この処理が必要なのは、カラー設定が変更された場合など、コントロールパネルがイベントを送出して `build()` 関数を再び呼び出す可能性があるためです。そのような場合は、サテライトを削除して、再作成する必要があります。

次に、この関数はサテライトを作成します。そのために、軌道のランダムな位置で始まる `position` 変数や、この例ではサテライトの作成後は変更しない `color` 変数など、作成に必要な初期プロパティを設定します。

サテライトが作成されるたびに、そのサテライトへの参照が `satellites` 配列に追加されます。`doEveryFrame()` 関数を呼び出すと、この配列のすべてのサテライトが更新されます。

サテライト位置の更新

doEveryFrame( ) 関数は、このアプリケーションのアニメーション処理の中心です。SWF ファイルのフレームレートと同じレートで、すべてのフレームに対して呼び出されます。描画の変数が少しずつ変化するため、アニメーションの外観が伝達されます。

この関数は、最初に前の描画をすべてクリアし、背景を再描画します。その後、各サテライトコンテナをループ処理して各サテライトの position プロパティを増やし、ユーザーのコントロールパネル操作によって変更された可能性のある radius プロパティと orbitRadius プロパティを更新します。最後に、Satellite クラスの draw( ) メソッドを呼び出して、サテライトを新しい位置に更新します。

カウンタ i は、visibleSatellites 変数までしか増やすことができません。これは、ユーザーがコントロールパネルで表示するサテライトの数を制限している場合、ループ内の残りのサテライトは再描画せずに非表示にする必要があるためです。これは、描画を行うループの直後のループで発生します。

doEveryFrame() 関数が完了すると、visibleSatellites の数のサテライトが画面上で新しい位置に更新されます。

ユーザー操作への応答

ユーザー操作は、ControlPanel クラスで管理されるコントロールパネルから行われます。このクラスは、各スライダの個々の最小値、最大値、およびデフォルト値と併せてリスナーを設定します。ユーザーがこれらのスライダを動かすと、changeSetting( ) 関数が呼び出されます。この関数は、コントロールパネルのプロパティを更新します。変更により表示を再構築する必要がある場合、イベントが送出され、メインアプリケーションファイルで処理されます。コントロールパネルの設定が変更されると、doEveryFrame( ) 関数は更新された変数を使用して各サテライトを描画します。

追加カスタマイズ

この例は、描画 API を使用したビジュアルの生成方法の基本的な概略を示しています。比較的少ない行数のコードを使用して、とても複雑に見えるインタラクティブ機能を作成しています。それでも、この例を少し変更して拡張できます。いくつかのアイデアを示します。

- doEveryFrame( ) 関数で、サテライトのカラー値を増やします。
- doEveryFrame( ) 関数で、サテライトの半径を時間の経過と共に縮小または拡大します。
- サテライトの軌道は円である必要はありません。たとえば、Math クラスを使用して正弦波に沿ってサテライトを移動します。
- 他のサテライトとのヒット検出を使用します。

描画 API は、実行時に基本的なシェイプを描画する、Flash オーサリング環境で視覚効果を作成する代替機能として使用できます。また、手動で作成するのが不可能な変化やスコープの視覚効果を作成するのにも使用できます。描画 API と若干の数式を使用することで、ActionScript 作成者は斬新な作品を実現できます。

第 15 章

表示オブジェクトのフィルタ処理

15

従来、ビットマップへのフィルタ効果の適用は、Adobe Photoshop® や Adobe Fireworks® などのイメージ編集に特化したソフトウェアによって処理される領域でした。ActionScript 3.0 には flash.filters パッケージがあり、このパッケージには一連のビットマップ効果フィルタクラスが含まれています。開発者は、これらのクラスを利用してプログラムでビットマップや表示オブジェクトにフィルタを適用することにより、グラフィック操作アプリケーションで適用可能な効果と同じ効果の多くを実現することができます。

目次

表示オブジェクトのフィルタ処理の基礎 443
フィルタの作成と適用 445
使用可能な表示フィルタ 450
例 : フィルタワークベンチ 468

表示オブジェクトのフィルタ処理の基礎

表示オブジェクトのフィルタ処理の概要

アプリケーションを洗練する方法の 1 つは、簡単なグラフィック効果を追加することです。たとえば、写真の背後にドロップシャドウを追加すると 3D 効果を生成できます。また、グローをボタンの周囲に追加するとボタンがアクティブであることを示すことができます。ActionScript 3.0 には 9 種類のフィルタがあり、表示オブジェクトまたは BitmapData インスタンスに適用できます。ドロップシャドウフィルタやグローフィルタなどの基本的なフィルタから、置き換えマップフィルタや畳み込みフィルタなどの多様な効果を生成する複雑なフィルタまでが用意されています。

フィルタ処理に関する一般的なタスク

次のタスクは、ActionScript でフィルタを使用するときに行うことが多いタスクです。

- フィルタの作成
- 表示オブジェクトへのフィルタの適用
- BitmapData インスタンス内のイメージデータへのフィルタの適用
- オブジェクトからのフィルタの削除
- 次のような各種フィルタ効果の作成
  - グロー
  - ぼかし
  - ドロップシャドウ
  - シャープ
  - 置き換え
  - エッジ検出
  - エンボス
  - その他の効果

重要な概念と用語

次の参照用の一覧に、この章で使用される重要な用語を示します。

- ベベル : 2 方向でピクセルを明るくし、その反対側の 2 方向でピクセルを暗くして作成されるエッジです。浮き出たボタンや凹んだボタンなどのグラフィックでよく使用される 3 次元境界効果を生成します。
- 畳み込み : 個々のピクセルの値と、隣接する一部またはすべてのピクセルの値とを、さまざまな割合で合計することによって、イメージ内のピクセルに歪みを適用することです。
- 置き換え : イメージ内のピクセルを新しい位置にシフトまたは移動することです。
- マトリックス : ある特定の数学的計算を実行するために使用される数値から成るグリッドです。この計算では、グリッドの数値がさまざまな値に適用され、その結果が合計されます。

章内の例の使用

この章の進行に合わせて、記載されているサンプルコードリストを実際にテストすることができます。この章はビジュアルコンテンツの作成と操作に関するものなので、コードリストをテストするには、コードを実行し、作成された SWF で結果を確認する必要があります。ほとんどすべての例で、描画 API を使用してコンテンツを作成するか、フィルタの適用先のイメージをロードします。

この章のコードをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   作成された SWF ファイルで、コードの結果を確認できます。

ほとんどすべてのサンプルコードにビットマップイメージを作成するコードが含まれているので、ビットマップコンテンツを用意しなくても、コードを直接テストできます。または、独自のイメージをロードするようにコードリストを変更し、サンプル中のイメージの代わりに使用することもできます。

フィルタの作成と適用

フィルタを使用すると、ドロップシャドウからベベルやぼかしまでの一連の効果を、ビットマップオブジェクトや表示オブジェクトに適用することができます。各フィルタはクラスとして定義されています。したがって、フィルタを適用するにはフィルタオブジェクトのインスタンスを作成する必要がありますが、他のオブジェクトを作成する場合との違いはありません。フィルタオブジェクトのインスタンスを作成した後は、表示オブジェクトの `filters` プロパティを使用して、そのインスタンスを表示オブジェクトに簡単に適用できます。BitmapData オブジェクトの場合は、`applyFilter()` メソッドを使用して簡単に適用することができます。

新しいフィルタの作成

新しいフィルタオブジェクトは、適切なフィルタクラスのコンストラクタメソッドを呼び出すだけで作成できます。たとえば、新しい DropShadowFilter オブジェクトを作成するには次のコードを使用します。

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

ここでは示していませんが、どのフィルタクラスのコンストラクタもそうであるように、`DropShadowFilter()` コンストラクタも、フィルタ効果の外観のカスタマイズに使用可能ないくつかの任意指定のパラメータを受け入れます。

フィルタの適用

フィルタオブジェクトを作成した後には、そのオブジェクトを表示オブジェクトまたは BitmapData オブジェクトに適用することができます。フィルタの適用方法は、フィルタを適用するオブジェクトによって異なります。

表示オブジェクトへのフィルタの適用

フィルタ効果を表示オブジェクトに適用する場合、filters プロパティを使用して適用します。表示オブジェクトの filters プロパティは Array インスタンスであり、このインスタンスのエレメントは、その表示オブジェクトに適用されるフィルタオブジェクトです。表示オブジェクトにフィルタを 1 つだけ適用する場合は、次のように、フィルタインスタンスを作成して Array インスタンスに追加した後、その Array オブジェクトを表示オブジェクトの filters プロパティに割り当てます。

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// BitmapData オブジェクトを作成して画面にレンダリングします。
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// DropShadowFilter インスタンスを作成します。
var dropShadow:DropShadowFilter = new DropShadowFilter();

// フィルタ配列を作成します。このとき、フィルタをパラメータとして
// Array() コンストラクタに渡して、配列にフィルタを追加します。
var filtersArray:Array = new Array(dropShadow);

// 表示オブジェクトにフィルタ配列を割り当てて、フィルタを適用します。
myDisplayObject.filters = filtersArray;
```

オブジェクトに複数のフィルタを割り当てる場合は、単純にすべてのフィルタを Array インスタンスに追加します。その後で、filters プロパティにそのインスタンスを割り当てます。複数のオブジェクトを配列に追加するには、配列のコンストラクタにそれらをパラメータとして渡します。たとえば、次のコードでは、既に作成されている表示オブジェクトにベベルフィルタとグローフィルタを適用します。

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// フィルタを作成して配列に追加します。
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);
```

```
// 表示オブジェクトにフィルタ配列を割り当てて、フィルタを適用します。
myDisplayObject.filters = filtersArray;
```

メモ

フィルタを含む配列を作成するときには、前の例に示したように、new Array() コンストラクタを使用して作成することができます。また、フィルタを角括弧 ([]) で囲んで、配列リテラルシンタックスを使用することもできます。たとえば、次のようなコード行があるとします。

```
var filters:Array = new Array(dropShadow, blur);
```

この行の実行結果は次のコード行と同じになります。

```
var filters:Array = [dropShadow, blur];
```

複数のフィルタを表示オブジェクトに適用すると、それらのフィルタは順番に従って累積的に適用されます。たとえば、あるフィルタ配列に 2 つのエレメントがあるとします。エレメントとしては、ベベルフィルタが最初に追加され、ドロップシャドウフィルタが 2 番目に追加されています。この場合、ドロップシャドウフィルタはベベルフィルタと表示オブジェクトの両方に適用されます。これは、フィルタ配列内でドロップシャドウフィルタの位置が 2 番目であるためです。フィルタを非累積的に適用する場合は、各フィルタを別個に表示オブジェクトの新しいコピーに適用する必要があります。

単一またはごく少数のフィルタを表示オブジェクトに適用する場合には、フィルタインスタンスの作成とオブジェクトへの適用を 1 つのステートメントで行うことができます。たとえば、次のコード行は、myDisplayObject という名前の表示オブジェクトにぼかしフィルタを適用するものです。

```
myDisplayObject.filters = [new BlurFilter()];
```

上記のコードでは、配列リテラルシンタックス (角括弧) を使用して Array インスタンスを作成し、その Array のエレメントとして新しい BlurFilter インスタンスを作成しています。さらに、myDisplayObject という名前の表示オブジェクトの filters プロパティに、その Array を割り当てています。

表示オブジェクトからのフィルタの削除

表示オブジェクトからすべてのフィルタを削除する操作は、次のように、filters プロパティに null 値を割り当てるだけの簡単な操作です。

```
myDisplayObject.filters = null;
```

オブジェクトに複数のフィルタを適用しており、それらのフィルタのうち 1 つだけを削除する必要がある場合には、複数の手順を実行して filters プロパティ内の配列を変更する必要があります。詳細については、449 ページの「実行時のフィルタの変更」を参照してください。

BitmapData オブジェクトへのフィルタの適用

BitmapData オブジェクトにフィルタを適用するには、次のように BitmapData オブジェクトの applyFilter() メソッドを使用する必要があります。

```
myBitmapData.applyFilter(sourceBitmapData);
```

applyFilter() メソッドは、ソース BitmapData オブジェクトにフィルタを適用して、フィルタ適用済みのイメージを新しく作成します。このメソッドによって元のソースイメージが変更されることはありません。その代わりに、ソースイメージにフィルタを適用した結果が、この applyFilter() メソッドが呼び出された BitmapData インスタンスに格納されます。

フィルタのしくみ

表示オブジェクトのフィルタ処理は、元のオブジェクトのコピーを透明なビットマップとしてキャッシュするしくみになっています。

フィルタが表示オブジェクトに適用されると、Adobe Flash Player は、そのオブジェクトに有効なフィルタリストがある限り、そのオブジェクトをビットマップとしてキャッシュします。このソースビットマップは、それ以降に適用されるすべてのフィルタ効果の元イメージとして使用されます。

通常、表示オブジェクトにはそれぞれ 2 つのビットマップがあります。1 つはフィルタ適用前の元の表示オブジェクトのビットマップで、もう 1 つはフィルタ適用後の最終イメージのビットマップです。最終イメージはレンダリング時に使用されます。表示オブジェクトが変更されない限り、最終イメージを更新する必要はありません。

フィルタ操作の潜在的な問題

フィルタを操作する際には、混乱やトラブルの原因になる可能性があるため注意を必要とする事項がいくつかあります。これらについて、次のセクションで説明します。

フィルタとビットマップキャッシュ

表示オブジェクトにフィルタを適用するには、そのオブジェクトのビットマップキャッシュを有効にする必要があります。cacheAsBitmap プロパティが false に設定されている表示オブジェクトにフィルタを適用すると、そのオブジェクトの cacheAsBitmap プロパティの値は、Flash Player によって自動的に true に設定されます。後でその表示オブジェクトからすべてのフィルタを削除すると、cacheAsBitmap プロパティは、現在の設定値の直前の設定値にリセットされます。

実行時のフィルタの変更

表示オブジェクトにフィルタが既に適用されている場合、その他のフィルタを filters プロパティ内の配列に追加することはできません。適用されるフィルタのセットを追加または変更するには、このような方法を使用するのではなく、filters の配列全体のコピーを作成して、この (一時的な) 配列を変更します。その後、配列の再割り当てによってこの配列を表示オブジェクトの filters プロパティに戻して、フィルタをオブジェクトに適用します。次のコードは、ここで説明した処理の例です。まず、グローフィルタが myDisplayObject という名前の表示オブジェクトに適用されます。その後、この表示オブジェクトがクリックされたときに、関数 addFilters( ) が呼び出されます。この関数では、2 つの追加フィルタが myDisplayObject に適用されます。

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
   // フィルタ配列のコピーを作成します。
   var filtersCopy:Array = myDisplayObject.filters;

   // 必要な変更をフィルタに加えます。この例ではフィルタを追加します。
   filtersCopy.push(new BlurFilter());
   filtersCopy.push(new DropShadowFilter());

   // filters プロパティに配列を再度割り当てて、変更を適用します。
   myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

フィルタとオブジェクト変形

ドロップシャドウなどのフィルタ処理された領域のうち、表示オブジェクトの境界ボックスの矩形よりも外側にある部分は、ヒット検出の対象サーフェスの一部と見なされません。ヒット検出は、インスタンス同士の重なりや交差があるかどうかを判定する処理です。DisplayObject クラスのヒット検出メソッドはベクターベースであるため、ビットマップとして生成された結果に対してヒット検出を実行することはできません。たとえば、ベベルフィルタをボタンインスタンスに適用した場合、そのインスタンスのベベル部分ではヒット検出を使用できません。

フィルタでは拡大 / 縮小、回転、および傾斜がサポートされません。フィルタ処理した表示オブジェクト自体を拡大 / 縮小しても、すなわち scaleX と scaleY を 100% 以外にしても、フィルタ効果はインスタンスと共に拡大 / 縮小されません。つまり、インスタンスの元のシェイプを回転、拡大 / 縮小、または傾斜しても、そのインスタンスと共にフィルタも回転、拡大 / 縮小、または傾斜するわけではありません。

フィルタを適用してインスタンスをアニメーション化すると、リアルな効果を生成できます。インスタンスをネスト化し、BitmapData クラスを使用してフィルタをアニメーション化しても、同じ効果を実現できます。

フィルタとビットマップオブジェクト

BitmapData オブジェクトにフィルタを適用すると、`cacheAsBitmap` プロパティが自動的に `true` に設定されます。このようにして、実際には元のオブジェクトではなくオブジェクトのコピーにフィルタが適用されます。

このコピーは、その後、メイン表示で元のオブジェクトに重ねて配置され、最も近いピクセルに可能な限り近付けられます。元のビットマップの境界が変化すると、フィルタが適用されたコピーのビットマップは、伸縮または歪曲されるのではなく、新しく再作成されます。

ビットマップオブジェクトのフィルタをすべてクリアすると、`cacheAsBitmap` プロパティは、フィルタ適用前の値にリセットされます。

使用可能な表示フィルタ

ActionScript 3.0 には次に示す 9 個のフィルタクラスがあり、表示オブジェクトと BitmapData オブジェクトに適用できます。

- ベベルフィルタ (BevelFilter クラス)
- ぼかしフィルタ (BlurFilter クラス)
- ドロップシャドウフィルタ (DropShadowFilter クラス)
- グローフィルタ (GlowFilter クラス)
- グラデーションベベルフィルタ (GradientBevelFilter クラス)
- グラデーショングローフィルタ (GradientGlowFilter クラス)
- カラーマトリックスフィルタ (ColorMatrixFilter クラス)
- 畳み込みフィルタ (ConvolutionFilter クラス)
- 置き換えマップフィルタ (DisplacementMapFilter クラス)

先頭から 6 番目までのフィルタは単純なフィルタです。それぞれ、1 つの特定の効果の生成に使用することができ、効果を若干カスタマイズすることができます。これらの 6 個のフィルタは、ActionScript を使用して適用できます。また、Adobe Flash CS3 Professional では、[フィルタ] パネルを使用してオブジェクトに適用できます。したがって、フィルタを ActionScript で適用しようとしている場合でも、この Flash オーサリングツールがあれば、ビジュアルなインターフェイスを使用して手軽にさまざまなフィルタや設定を試し、必要な効果の生成方法を把握することができます。

最後の 3 個のフィルタは、ActionScript でのみ使用できます。これらのフィルタ (カラーマトリックスフィルタ、畳み込みフィルタ、および置き換えマップフィルタ) は単一の効果を目的として最適化されたものではなく、これらのフィルタを使用して生成できる効果の種類は多岐にわたります。そのため、強力で柔軟性に優れています。たとえば、畳み込みフィルタの場合、このフィルタのマトリックスに指定する値を変更することによって、ぼかし、エンボス、シャープ、カラーエッジ検出、変換などの各種効果の生成に使用することができます。

各フィルタは、単純なフィルタであるか複雑なフィルタであるかに関係なく、プロパティを使用してカスタマイズすることができます。通常、フィルタのプロパティの設定方法は 2 つあります。どのフィルタでも、フィルタオブジェクトのコンストラクタにパラメータ値を渡すことによって、プロパティを設定できます。また、パラメータを渡す方法でフィルタのプロパティを設定したかどうかに関係なく、フィルタオブジェクトのプロパティの値を設定することによって、後からフィルタを調整できます。例として示すコードリストの大半は、プロパティを直接設定しています。これは、例をわかりやすくするためです。ただし、通常は、フィルタオブジェクトのコンストラクタに値をパラメータとして渡す方法を使用して、行数がより少ないコードで同じ結果を得ることができます。各フィルタの細部の特徴、プロパティ、およびコンストラクタパラメータの詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の flash.filters パッケージ関連の記述を参照してください。

ベベルフィルタ

BevelFilter クラスを使用すると、フィルタ処理されたオブジェクトに 3D のベベルエッジを追加することができます。このフィルタを適用すると、オブジェクトの角部分やエッジが、浮き彫りのように、または斜角面があるように見えます。

BevelFilter クラスのプロパティを使用して、ベベルの外観をカスタマイズすることができます。ハイライトカラーとシャドウカラー、ベベルエッジのぼかし、ベベルの角度、およびベベルエッジの配置を設定できます。さらに、ノックアウト効果を付加することもできます。

次の例では、外部イメージをロードし、そのイメージにベベルフィルタを適用します。

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// ステージ上にイメージをロードします。
var imageLoader:Loader = new Loader();
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);

// ベベルフィルタを作成し、フィルタのプロパティを設定します。
var bevel:BevelFilter = new BevelFilter();
```

```
bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// イメージにフィルタを適用します。
imageLoader.filters = [bevel];
```

ぼかしフィルタ

BlurFilter クラスは、表示オブジェクトとそのコンテンツに、にじみやぼかしを適用します。ぼかし効果は、オブジェクトにフォーカスがないことを印象付ける場合や、高速移動が行われているように見せる場合に便利です。ぼかしフィルタの quality プロパティを低レベルに設定すると、ソフトフォーカスレンズの効果をシミュレートすることができます。quality プロパティを高レベルに設定すると、ガウスぼかしに似たスムーズなぼかしが生成されます。

次の例では、Graphics クラスの drawCircle() メソッドを使用して円オブジェクトを作成し、そのオブジェクトにぼかしを適用します。

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// 円を描画します。
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// 円を表示リストに追加します。
addChild(redDotCutout);

// ぼかしフィルタを円に適用します。
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

ドロップシャドウフィルタ

ドロップシャドウを使用すると、ターゲットオブジェクトの上部に別個の光源があるような印象を与えることができます。この光源の位置や強さを変更して、さまざまなドロップシャドウ効果を生成することが可能です。

ドロップシャドウフィルタには、ぼかしフィルタのアルゴリズムと同様のアルゴリズムが使用されています。主な違いは、ドロップシャドウフィルタには若干多くのプロパティが用意されており、それらを変更すると光源の各種属性 (アルファ、カラー、オフセット、明度など) をシミュレートできることです。

また、ドロップシャドウフィルタでは、内側シャドウ、外側シャドウ、ノックアウト (カットアウト) モードなどのドロップシャドウのスタイルに関するカスタム変換オプションも適用することができます。

次のコードでは、四角形のボックスのスプライトを作成し、そのボックスにドロップシャドウフィルタを適用します。

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// ボックスを描画します。
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// ドロップシャドウフィルタをボックスに適用します。
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// その他のプロパティも設定できます。シャドウカラー、アルファ、
// ぼかし量、強さ、品質のほか、内側シャドウやノックアウト効果を
// 適用するオプションなどを設定できます。

boxShadow.filters = [shadow];
```

グローフィルタ

GlowFilter クラスは、表示オブジェクトに照明効果を適用します。その結果、表示オブジェクトは、背後から光で照らされて穏やかに輝いているように見えるようになります。

ドロップシャドウフィルタと同様に、グローフィルタも光源の距離、角度、およびカラーを変更できるプロパティを備えており、さまざまな効果を生成できます。また、グローフィルタには、内側グロー、外側グロー、ノックアウトモードなどのグローのスタイルを変更するためのオプションもいくつか用意されています。

次のコードでは、Sprite クラスを使用して十字形を作成し、その十字形にグローフィルタを適用します。

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// 十字形のグラフィックを作成します。
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// グローフィルタを十字形に適用します。
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

グラデーションベベルフィルタ

GradientBevelFilter クラスを使用すると、表示オブジェクトや BitmapData オブジェクトに高度なベベル効果を適用することができます。グラデーションカラーがベベルに使用されるので、ベベルの空間的な深みが大幅に向上し、エッジがいっそうリアルな 3D に見えるようになります。

次のコードでは、Shape クラスの drawRect() メソッドを使用して矩形オブジェクトを作成し、そのオブジェクトにグラデーションベベルフィルタを適用します。

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// 矩形を描画します。
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// 矩形にグラデーションベベルを適用します。
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // 45 度の 180 度反対側
```

```
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// その他のプロパティでは、フィルタの強さを設定できるほか、
// 内側ベベルやノックアウト効果を適用するオプションを設定できます。

box.filters = [gradientBevel];

// グラフィックを表示リストに追加します。
addChild(box);
```

グラデーショングローフィルタ

GradientGlowFilter クラスを使用すると、表示オブジェクトや BitmapData オブジェクトに高度なグロー効果を適用することができます。この効果を使用することによって、グローのカラー制御が充実し、グロー効果がいっそうリアルになります。また、グラデーショングローフィルタでは、オブジェクトの内側、外側、または上側のエッジにグラデーショングローを適用することができます。

次の例では、ステージ上に円を描画し、その円にグラデーショングローフィルタを適用します。マウスを右方向と下方向に移動すると、それぞれ水平方向と垂直方向で、ぼかしの量が増加します。また、ステージをクリックすると、ぼかしの強さが増加します。

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// 新しい Shape インスタンスを作成します。
var shape:Shape = new Shape();

// シェイプを描画します。
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// ステージ上にシェイプを配置します。
addChild(shape);
shape.x = 100;
shape.y = 100;
```

```
// グラデーショングローを定義します。
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;

// 2 種類のイベントを監視するための関数を定義します。
function onClick(event:MouseEvent):void
{
  gradientGlow.strength++;
  shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
  gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
  gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
  shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

例 : 基本的なフィルタの組み合わせ

次のコード例では、いくつかの基本的なフィルタをタイマーと組み合わせて使用して、繰り返されるアクションを作成し、アニメーションによる信号機のシミュレーションを作成します。

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // 45 度の 180 度反対側
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
```

```
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// 信号機の背景の矩形を描画します。
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// 3 つの灯火を表す 3 つの円を描画します。
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// グラフィックを表示リストに追加します。
addChild(box);
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// 信号機の矩形にグラデーションベベルを適用します。
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance,
  angleInDegrees, colors, alphas, ratios, blurX, blurY, strength, quality, type,
  knockout);
box.filters = [gradientBevel];

// 内側シャドウ（消灯中の灯火用）とグロー（点灯中の灯火用）
// を作成します。
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3, 3, 1,
  1, true, false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false, false);
```

```
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1, false,
  false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false,
  false);

// 開始時の灯火の状態を設定します（青は点灯、赤と黄は消灯）。
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// カウント値に基づいてフィルタを入れ替えます。
function trafficControl(event:TimerEvent):void
{
  if (count == 4)
  {
    count = 1;
  }

  switch (count)
  {
    case 1:
      stopLight.filters = [innerShadow];
      cautionLight.filters = [yellowGlow];
      goLight.filters = [innerShadow];
      break;
    case 2:
      stopLight.filters = [redGlow];
      cautionLight.filters = [innerShadow];
      goLight.filters = [innerShadow];
      break;
    case 3:
      stopLight.filters = [innerShadow];
      cautionLight.filters = [innerShadow];
      goLight.filters = [greenGlow];
      break;
  }

  count++;
}

// 3 秒間隔でフィルタを入れ替えるためのタイマーを作成します。
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

カラーマトリックスフィルタ

ColorMatrixFilter クラスは、フィルタ処理するオブジェクトのカラー値とアルファ値の操作に使用します。その結果、1 つのカラーチャンネルの値とその値を他のチャンネルに潜在的に適用する値を使用して、彩度変更、色相回転 (ある範囲のカラーから別の範囲のカラーへのパレットのシフト)、輝度アルファ変換などのカラー操作効果を生成することができます。

概念的には、フィルタでソースイメージのピクセルが 1 つずつ処理され、各ピクセルが赤、緑、青、アルファの各成分に分解されます。次に、これらのそれぞれの値とカラーマトリックス内の指定値が乗算され、その結果が合計されて、画面上に表示される各ピクセルの最終的なカラーの値が決定されます。フィルタの `matrix` プロパティは、最終的なカラーの計算に使用される 20 個の配列になっています。カラー値の計算に使用される具体的なアルゴリズムの詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の、ColorMatrixFilter クラスの `matrix` プロパティについて説明している項を参照してください。

カラーマトリックスフィルタの詳細情報と例については、Adobe デベロッパーセンター Web サイトの記事「Flash での変形、カラー調整、畳み込み効果のマトリックスの使用について」を参照してください。

畳み込みフィルタ

ConvolutionFilter クラスを使用すると、BitmapData オブジェクトまたは表示オブジェクトに、ぼかし、エッジ検出、シャープ、エンボス、ベベルなどのさまざまなイメージ変換を適用することができます。

概念的には、畳み込みフィルタでソースイメージのピクセルが 1 つずつ処理され、処理対象のピクセルとその周囲のピクセルの値を使用して、その対象ピクセルの最終的なカラーが決定されます。マトリックスは、数値の配列として指定されており、個々の隣接ピクセルの値が最終結果の値にどの程度影響するかを示します。

ここでは、最もよく使用される種類のマトリックスである 3 x 3 のマトリックスについて考えてみます。このマトリックスには、9 個の値が含まれています。

```
N N N
N P N
N N N
```

Flash Player は、畳み込みフィルタを特定のピクセルに適用する場合、そのピクセル自体 (この例では "P") のカラー値を参照すると共に、周囲のピクセル (この例では "N") の値も参照します。ただし、マトリックス内に値を設定することによって、結果として得られるイメージに対する影響の優先度をピクセルごとに指定することができます。

たとえば、次のマトリックスが畳み込みフィルタによって適用されると、イメージは適用前とまったく同じ状態のままになります。

```
0  0  0
0  1  0
0  0  0
```

イメージが変更されない理由は、最終的なピクセルカラーが決定されるときに、元のピクセルの値の相対的な強さが 1 になるのに対し、周囲のピクセルの値の相対的な強さが 0 になるためです。つまり、周囲のピクセルの色は最終イメージに影響しません。

これと同様の理由で、次のマトリックスの場合にはイメージが左方向に 1 ピクセル分シフトすることになります。

```
0  0  0
0  0  1
0  0  0
```

この場合、処理対象のピクセル自体は、最終イメージでこのピクセルの位置に表示されるピクセルの最終値に影響しません。右隣にあったピクセルの値だけが、その表示されるピクセルの最終値の決定に使用されます。

ActionScript でマトリックスを作成するときには、値を含む Array インスタンスと、マトリックスの行数および列数を指定する 2 つのプロパティとを組み合わせたものとして作成します。次の例では、イメージをロードし、イメージのロードが完了したら前述のマトリックスを使用して畳み込みフィルタをそのイメージに適用します。

```
// ステージ上にイメージをロードします。
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/
  image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
  // 畳み込みのマトリックスを作成します。
  var matrix:Array = [ 0, 0, 0,
                       0, 0, 1,
                       0, 0, 0 ];

  var convolution:ConvolutionFilter = new ConvolutionFilter();
  convolution.matrixX = 3;
  convolution.matrixY = 3;
  convolution.matrix = matrix;
  convolution.divisor = 1;

  loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

マトリックスで 1 または 0 以外の値を使用するとどのような結果になるかはこのコードからはわかりません。たとえば、上と同じマトリックスで右側の位置の 1 の代わりに数値 8 を使用した場合には、同じアクションが実行されます。つまり、左方向にピクセルがシフトします。また、イメージのカラーが影響を受け、イメージが 8 倍明るくなります。これは、最終的なピクセルカラーの値は、マトリックスの各値と元の各ピクセルのカラーを乗算し、値を合計した後、フィルタの divisor プロパティに指定されている値で除算することによって算出されるためです。例に示したコードでは、divisor プロパティは 1 に設定されています。基本的に、カラーの明度を元のイメージとほぼ同じ明度のままにする場合は、除数をマトリックスの各値の合計と等しくする必要があります。このように、値の合計が 8 になるマトリックスを使用し、除数を 1 にした場合には、結果として得られるイメージは元のイメージよりも約 8 倍明るくなります。

ここで紹介したマトリックスの効果は目立つものではありませんが、別のマトリックス値を使用すればさまざまな効果を生成することができます。次に、3 x 3 マトリックスを使用して各種効果を生成する場合の標準的なマトリックス値の組み合わせをいくつか示します。

- 基本的なぼかし (除数は 5) :

```
0 1 0
1 1 1
0 1 0
```

- シャープ (除数は 1) :

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- エッジ検出 (除数は 1) :

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- エンボス効果 (除数は 1) :

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

これらの効果のほぼすべてで除数が 1 になっています。これは、負のマトリックス値と正のマトリックス値を合計すると結果が 1 になるためです。エッジ検出では 0 になりますが、divisor プロパティを 0 にすることはできません。

置き換えマップフィルタ

DisplacementMapFilter クラスは、置き換えマップイメージとも呼ばれる BitmapData オブジェクトのピクセル値を使用して、新しいオブジェクトで置き換え効果を実行します。通常、置き換えマップイメージは、フィルタの適用先となる実際の表示オブジェクトや BitmapData インスタンスとは異なります。置き換え効果では、フィルタを適用したイメージのピクセルが置き換えられます。つまり、ピクセルが元の位置から若干シフトします。このフィルタを使用して、シフト効果、ワープ効果、または斑点効果を生成することができます。

個々のピクセルに適用される置き換えの位置と量は、置き換えマップイメージのカラー値に基づいて決定されます。このフィルタを操作するときには、マップイメージの指定に加えて、マップイメージに基づいて置き換えをどのように計算するかを制御する次の値を指定する必要があります。

- マップポイント : フィルタの適用対象となるイメージ上の位置であり、この位置に置き換えフィルタの左上隅が適用されます。イメージの一部にのみフィルタを適用する場合に使用できます。
- X コンポーネント : マップイメージのカラーチャンネルのうち、ピクセルの x 方向位置に影響するものを示します。
- Y コンポーネント : マップイメージのカラーチャンネルのうち、ピクセルの y 方向位置に影響するものを示します。
- X スケール : x 軸方向の置き換えの拡大の程度を指定する乗数値です。
- X スケール : y 軸方向の置き換えの拡大の程度を指定する乗数値です。
- フィルタモード : ピクセルのシフトによって生じた空白の領域に対する Flash Player の処理を決定します。オプションは DisplacementMapFilterMode クラスの定数として定義されており、元のピクセルを表示する (フィルタモード `IGNORE`) か、イメージの反対側でピクセルを折り返す (フィルタモード `WRAP`、デフォルト) か、最も近くにあるシフトしたピクセルを使用する (フィルタモード `CLAMP`) か、またはその領域をカラーで塗りつぶす (フィルタモード `COLOR`) ことができます。

置き換えマップフィルタのしくみの基礎を理解できるように、ここで基本的な例を示します。次のコードでは、イメージをロードし、ロードが完了したらイメージをステージの中央に移動して、置き換えマップフィルタをそのイメージに適用します。その結果、イメージ全体のピクセルが左方向に水平にシフトします。

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// ステージ上にイメージをロードします。
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/
  image3.jpg");
```

```
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
   // ロードしたイメージをステージ中央に移動します。
   loader.x = (stage.stageWidth - loader.width) / 2;
   loader.y = (stage.stageHeight - loader.height) / 2;

   // 置き換えマップイメージを作成します。
   mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

   // 置き換えフィルタを作成します。
   displacementMap = new DisplacementMapFilter();
   displacementMap.mapBitmap = mapImage;
   displacementMap.mapPoint = new Point(0, 0);
   displacementMap.componentX = BitmapDataChannel.RED;
   displacementMap.scaleX = 250;
   loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

置き換えの定義に使用されているプロパティは次のとおりです。

- マップビットマップ : 置き換えビットマップは、このコードで作成する新しい BitmapData インスタンスです。サイズは、ロードされるイメージのサイズと一致しています。したがって、置き換えはイメージ全体に適用されます。赤一色のピクセルで塗りつぶされています。
- マップポイント : この値はポイント (0, 0) に設定されています。これも、置き換えがイメージ全体に適用されるようにするためです。
- X コンポーネント : この値は定数 BitmapDataChannel.RED に設定されています。つまり、ピクセルがどの程度 x 軸方向に置き換えられる (移動する) かについては、マップビットマップの赤の値に基づいて決定されます。
- X スケール : この値は 250 に設定されています。マップイメージの色が完全な赤で、置き換え量が最大 (フル) になる場合でも、それに対応する実際のイメージの置き換えはごくわずかな量 (1/2 ピクセル程度) です。そのため、この値が 1 に設定されていると、イメージは 0.5 ピクセルしか水平方向にシフトしません。250 に設定すると、イメージは約 125 ピクセル分シフトします。

これらの設定の結果、フィルタが適用されたイメージは 125 ピクセル左方向にシフトするようになります。シフトの方向 (左または右) と量は、マップイメージ内のピクセルのカラー値に基づいて決まります。概念的には、フィルタが適用されたイメージのピクセルが Flash Player によって 1 つずつ処理されます。少なくとも、フィルタが適用された領域のピクセル (ここで示した例ではすべてのピクセル) が処理されます。ピクセルごとに次の処理が行われます。

1. マップイメージ内の対応するピクセルが特定されます。たとえば、フィルタが適用されたイメージの左上隅のピクセルの置き換え量が計算されるときには、マップイメージの左上隅のピクセルが参照されます。
2. マップ上のピクセルで、指定されたカラーチャンネルの値が判定されます。ここで示した例では、x コンポーネントに指定されたカラーチャンネルは赤チャンネルです。そのため、特定されたピクセルの位置でマップイメージの赤チャンネルの値が調べられます。マップイメージは赤一色なので、このピクセルの赤チャンネルの値は 0xFF、すなわち 255 です。これは置き換え値として使用されます。
3. 置き換え値が " 中間 " 値と比較されます。中間値は、0 ～ 255 の中間にあたる 127 です。置き換え値が中間値よりも小さい場合、正の方向 (x 置き換えでは右方向、y 置き換えでは下方向) にピクセルがシフトします。一方、ここで示した例のように置き換え値が中間値よりも大きい場合には、負の方向 (x 置き換えでは左方向、y 置き換えでは上方向) にピクセルがシフトします。さらに正確に説明すると、127 から置き換え値が減算され、その結果 (正または負) が、適用される相対的な置き換え量になります。
4. 最後に、実際の置き換え量が決定されます。そのために、相対的な置き換え量の値が最大置き換えの何パーセントに相当するかが判定されます。ここで示した例では、完全な赤の場合に 100% の置き換えになります。続いて、そのパーセント値と x スケール値または y スケール値が乗算されて、適用される置き換えピクセル数が決定されます。ここで示した例では、100% に乗数 250 を掛けて、置き換え量が決定されます。これは左方向に約 125 ピクセルになります。

y コンポーネントと y スケールについては、値が指定されていないため、デフォルト設定が使用されます。デフォルト設定では置き換えが発生しません。イメージが垂直方向にシフトしないのはこのためです。

ここで示した例では、デフォルトのフィルタモード設定である `WRAP` が使用されます。そのため、ピクセルが左方向にシフトしたとき、右側に生じる空白の領域は、イメージの左端から押し出されたピクセルで塗りつぶされます。この値を変更してみると、さまざまな効果を確認できます。たとえば、置き換えの各プロパティを設定しているコードの部分 (`loader.filters = [displacementMap]` と記述されている行の前) に次の行を追加すると、イメージは、ステージ上でにじんだように見えるようになります。

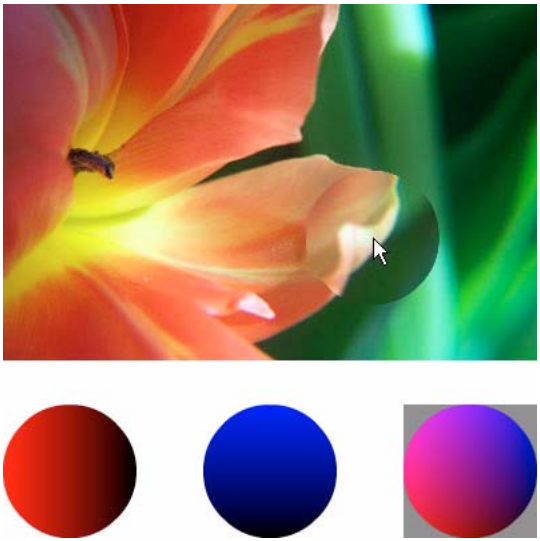
```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

さらに複雑な例を示します。次のリストでは、置き換えマップフィルタを使用して、イメージに拡大鏡の効果を付加します。

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// グラデーション塗りを適用した円を 2 つ作成します。両方とも
// 置き換えマップイメージの形成に使用されます。
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [ 0xFF0000, 0x000000 ];
var blueColors:Array = [ 0x0000FF, 0x000000 ];
var alphas:Array = [ 1, 1 ];
var ratios:Array = [ 0, 255 ];
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios, xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios, yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// 円を画面下部に配置します ( 参照用 )。
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;

// ステージ上にイメージをロードします。
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/
   image1.jpg");
```

```
loader.load(url);
this.addChild(loader);

// 2 つのグラデーション塗りを適用した円を合成してマップイメージを作成します。
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false,
  0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false,
  0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE,
  BitmapDataChannel.BLUE);
yMap.dispose();

// マップイメージをステージ上に表示します（参照用）。
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// この関数は、マウスの位置に置き換えマップフィルタを作成します。
function magnify():void
{
  // フィルタを配置します。
  var filterX:Number = (loader.mouseX) - (map.width / 2);
  var filterY:Number = (loader.mouseY) - (map.height / 2);
  var pt:Point = new Point(filterX, filterY);
  var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
  xyFilter.mapBitmap = map;
  xyFilter.mapPoint = pt;
  // x 置き換えはマップイメージ内の赤に基づいて制御されます。
  xyFilter.componentX = BitmapDataChannel.RED;
  // y 置き換えはマップイメージ内の青に基づいて制御されます。
  xyFilter.componentY = BitmapDataChannel.BLUE;
  xyFilter.scaleX = 35;
  xyFilter.scaleY = 35;
  xyFilter.mode = DisplacementMapFilterMode.IGNORE;
  loader.filters = [xyFilter];
}

// この関数は、マウスが移動すると呼び出されます。マウスが、
// ロードされたイメージの上にあれば、フィルタが適用されます。
function moveMagnifier(event:MouseEvent):void
{
  if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
  {
    magnify();
  }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

このコードでは、まずグラデーション塗りを適用した円が２つ生成されます。これらを合成することによって置き換えマップイメージが形成されます。赤色の円は x 軸方向の置き換えを実現するためのもので (xyFilter.componentX = BitmapDataChannel.RED)、青色の円は y 軸方向の置き換えを実現するためのものです (xyFilter.componentY = BitmapDataChannel.BLUE)。また、置き換えマップイメージがどのようなものかを容易に把握できるようにするため、元の円が、マップイメージとなる合成後の円と共に画面下部に追加されます。



このコードでは、さらにイメージがロードされ、マウスが移動するとマウスポインタの下にある部分のイメージに置き換えフィルタが適用されます。グラデーション塗りを適用した円が置き換えマップイメージとして使用されているため、置き換えが適用された領域は、マウスポインタを中心として周囲に広がります。置き換えマップイメージのグレーの領域では、置き換えは発生しません。このグレーのカラーは 0x7F7F7F です。この階調がグレーの場合、青チャンネルと赤チャンネルの値は、それぞれのカラーチャンネルの階調範囲の中央の値とちょうど一致します。そのため、マップイメージのグレーの領域では置き換えが発生しません。同様に、円の中心でも置き換えが発生しません。この位置のカラーはグレーではありません。しかし、このカラーの青チャンネルと赤チャンネルの値は、赤、緑、青のちょうど中間のグレーの青チャンネルおよび赤チャンネルの値と一致しています。置き換えを発生させるカラーは青と赤です。したがって、この位置では置き換えが発生しません。

例 : フィルタワークベンチ

フィルタワークベンチには簡単なユーザーインターフェイスが用意されており、各種フィルタをイメージに適用して、生成されるコードを確認することができます。そのコードを利用すれば、ActionScript を使用して同じ効果を生成できます。この例の詳細とソースコードのダウンロードについては、www.adobe.com/go/learn_fl_filters_jp を参照してください。

第 16 章

ムービークリップの操作

16

MovieClip クラスは、Adobe Flash CS3 Professional で作成されたアニメーションおよびムービークリップのシンボルのコアクラスです。このクラスには、表示オブジェクトの動作と機能がすべて備えられていますが、ムービークリップのタイムラインを制御するための追加プロパティとメソッドもあります。この章では、ActionScript を使用してムービークリップ再生を制御する方法と、ムービークリップを動的に作成する方法について説明します。

目次

ムービークリップの基礎 470
ムービークリップの再生の制御. 472
ActionScript での MovieClip オブジェクトの作成 475
外部の SWF ファイルのロード 479
例 : RuntimeAssetsExplorer 480

ムービークリップの基礎

ムービークリップの操作の概要

ムービークリップとは、Flash オーサリングツールを使用してアニメーションコンテンツを作成したり、そのコンテンツを ActionScript で操作するユーザー向けの主要エレメントです。Flash でムービークリップのシンボルを作成すると、その Flash ドキュメントのライブラリにシンボルが必ず追加されます。デフォルトでは、このシンボルは MovieClip クラスのインスタンスとなるため、MovieClip クラスのプロパティとメソッドを持ちます。

ムービークリップシンボルのインスタンスがステージ上に配置されると、1 フレーム以上ある場合、ムービークリップは自動的にタイムライン上を進みます。ただし、ActionScript を使用して再生が変更される場合を除きます。MovieClip クラスを識別するのは、このタイムラインです。MovieClip クラスを使用すると、Flash オーサリングツールを使用して、モーショントゥイーンやシェイプトゥイーンによるアニメーションの作成が可能になります。反対に、Sprite クラスのインスタンスである表示オブジェクトでは、アニメーションを作成するにはオブジェクトの値をプログラムによって変更するしか方法はありません。

以前のバージョンの ActionScript では、MovieClip クラスはステージ上のすべてのインスタンスの基本クラスとして機能していました。ActionScript 3.0 では、ムービークリップは画面に表示可能な多くの表示オブジェクトの 1 つにすぎません。表示オブジェクトの関数でタイムラインが必要でない場合、MovieClip クラスの代わりに Shape クラスまたは Sprite クラスを使用すると、レンダリングパフォーマンスが向上する可能性があります。タスクに適した表示オブジェクトの選択方法の詳細については、375 ページの「DisplayObject サブクラスの選択」を参照してください。

ムービークリップの一般的なタスク

この章では、次の一般的なムービークリップのタスクについて説明します。

- ムービークリップの再生と停止
- ムービークリップの逆方向再生
- ムービークリップのタイムラインにおける指定ポイントへの再生ヘッドの移動
- ActionScript のフレームラベルの操作
- ActionScript のシーン情報へのアクセス
- ActionScript を使用したライブラリムービークリップシンボルのインスタンスの作成
- 旧バージョンの Flash Player 向けに作成されたファイルを含む、外部 SWF ファイルのロードと制御
- 実行時にロードされ、使用されるグラフィカルアセットを作成する ActionScript システムの構築

重要な概念と用語

次の参照用の一覧には、この章で使用される重要な用語に関する説明が記載されています。

- AVM1 SWF: ActionScript 1.0 または ActionScript 2.0 によって作成される SWF ファイルです。通常は、Flash Player 8 以前を対象としています。
- AVM2 SWF: Flash Player 9 向けとして ActionScript 3.0 によって作成される SWF ファイルです。
- 外部 SWF : プロジェクト SWF ファイルとは別に作成される SWF ファイルです。プロジェクト SWF ファイルにロードされ、その SWF ファイル内で再生されます。
- フレーム : タイムライン上の時間の最小分割単位です。映画スライドと同じように、各フレームはアニメーションのスナップショットのようなものであり、フレームをすばやく連続して再生すると、アニメーションの効果を得ることができます。
- タイムライン : ムービークリップのアニメーションシーケンスを成す一連のフレームを、比喩的に表示するものです。MovieClip オブジェクトのタイムラインは、Flash オーサリングツールのタイムラインと同じです。
- 再生ヘッド : タイムライン上の現在時点 (フレーム) を示すマーカーです。

本章の例について

本章をお読みになる際は、いくつかのコード例を試すことができます。この章では ActionScript でのムービークリップの操作について扱っているため、この章にあるほぼすべてのコードは、ステージ上で作成されて配置されたムービークリップシンボルを操作するように作成されています。サンプルのテストには、Flash Player で結果を表示してシンボルに対するコードの効果を確認することが含まれます。この章のコードをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインでキーフレームを選択します。
3. [アクション] パネルを開き、[スクリプト] ペインにコードをコピーします。
4. ステージ上でムービークリップシンボルのインスタンスを作成します。たとえば、図形を描画して選択し、[変更]-[シンボルに変換] を選択して、そのシンボルに名前を付けます。
5. ムービークリップが選択された状態で、プロパティインスペクタを使用してそのクリップにインスタンス名をつけます。この名前は、コード例でムービークリップに使われている名前と一致させる必要があります。たとえば、コードで `myMovieClip` という名前のムービークリップを操作する場合は、ムービークリップのインスタンス名も `myMovieClip` にします。
6. [コントロール]-[ムービープレビュー] を使用して、プログラムを実行します。

   画面に、コードで指定したようにムービークリップを操作した結果が表示されます。

コード例をテストするその他の方法については、60 ページの「章内のコード例のテスト」で詳しく説明します。

MovieClip オブジェクトの操作

SWF ファイルをパブリッシュすると、Flash はステージ上のすべてのムービークリップシンボルインスタンスを MovieClip オブジェクトに変換します。プロパティインスペクタの [インスタンス名] フィールドにムービークリップシンボルのインスタンス名を入力すると、ActionScript でそのシンボルを使用できるようになります。SWF ファイルが作成されると、Flash はステージ上に MovieClip インスタンスを作成するコードを生成し、インスタンス名を使用して変数を宣言します。他の名前を持つムービークリップの中にネストされたムービークリップに名前を付けると、その子ムービークリップは親ムービークリップのプロパティとして扱われます。子ムービークリップにアクセスするには、ドットシンタックスを使用します。たとえば、インスタンス名が `childClip` のムービークリップが、インスタンス名が `parentClip` のムービークリップ内にネストされている場合、このコードを呼び出すことで子クリップのタイムラインのアニメーションを再生することができます。

```
parentClip.childClip.play()
```

ActionScript 2.0 の MovieClip クラスの古いメソッドおよびプロパティには、変更されているものと、変更されていないものがあります。アンダースコアで始まるプロパティの名前はすべて変更されています。たとえば、`_width` と `_height` プロパティは `width` と `height`、`_xscale` と `_yscale` は `scaleX` と `scaleY` になっています。MovieClip クラスのプロパティとメソッドの詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

ムービークリップの再生の制御

Flash では、タイムラインのメタファーを使用して、アニメーションまたは状態の変化を伝達します。タイムラインを使用するビジュアルエレメントは、MovieClip オブジェクトか、または MovieClip クラスの拡張である必要があります。ActionScript ではムービークリップの停止、再生、またはタイムラインの別のポイントへの移動を指示できますが、ActionScript を使用してタイムラインを動的に作成したり、特定のフレームにコンテンツを追加したりすることはできません。これには、Flash オーサリングツールを使用する必要があります。

MovieClip の再生は、SWF のフレームレートで指定された速度でタイムラインに沿って行われます。また、ActionScript で `Stage.frameRate` プロパティを設定すると、この設定をオーバーライドできます。

ムービークリップの再生と再生の停止

play() および stop() メソッドを使用すると、タイムライン内でムービークリップの基本的な制御を行うことができます。たとえば、画面上を走る自転車のアニメーションを含むステージ上にムービークリップシンボルがあり、そのインスタンス名が bicycle に設定されているとします。メインタイムラインのキーフレームに次のコードが関連付けられている場合、

```
bicycle.stop();
```

自転車は動きません。つまり、アニメーションは再生されません。自転車の動きは、他のユーザー操作によって開始することができます。たとえば、startButton という名前のボタンがある場合、ボタンをクリックするとアニメーションが再生されるように、メインタイムライン上のキーフレームのコードが実行されます。

```
// この関数は、ボタンをクリックしたときに呼び出されます。つまり、
// 自転車のアニメーションが再生されます。
function playAnimation(event:MouseEvent):void
{
  bicycle.play();
}
// この関数を、ボタンのリスナーとして登録します。
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

早送りと巻き戻し

ムービークリップで再生を制御する方法は、play() および stop() メソッドだけではありません。nextFrame() および prevFrame() メソッドを使用すると、再生ヘッドをタイムラインに沿って手動で前後に動かすことができます。これらのメソッドのいずれかを呼び出すと、再生が停止して、再生ヘッドが前方または後方に1フレームずつ動きます。

play() メソッドを使用することは、ムービークリップオブジェクトの enterFrame イベントがトリガーされるたびに nextFrame() を呼び出すのに似ています。次の例では、enterFrame イベントにイベントリスナーを作成して bicycle をリスナー関数の前のフレームに戻すことによって、bicycle のムービークリップを逆方向に動かすことができます。

```
// この関数は、enterFrame イベントがトリガーされたときに呼び出されます。つまり、
// フレームごとに呼び出されます。
function everyFrame(event:Event):void
{
  if (bicycle.currentFrame == 1)
  {
    bicycle.gotoAndStop(bicycle.totalFrames);
  }
  else
  {
    bicycle.prevFrame();
  }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

ムービークリップに複数のフレームが含まれている場合、ムービークリップの再生は無限に繰り返されます。つまり、最後のフレームの再生が終わると最初のフレームに戻ります。prevFrame() または nextFrame() を使用すると、この動作は自動的には行われません。つまり、再生ヘッドが最初のフレーム上にあるときに prevFrame() を呼び出しても、再生ヘッドは最後のフレームに移動しません。上記の例の if 条件では、逆方向の再生ヘッドが最初のフレームまで到達したかどうかを確認し、最初のフレームに到達した場合に再生ヘッドを最終フレームに設定することによって、ムービークリップの逆方向連続再生を実現しています。

異なるフレームへのジャンプとフレームラベルの使用

ムービークリップを新しいフレームに進めるのは簡単です。gotoAndPlay() または gotoAndStop() を呼び出すと、ムービークリップはパラメータで指定されたフレーム番号までジャンプします。あるいは、フレームラベルの名前と同じ文字列を渡すこともできます。タイムライン上の任意のフレームに、ラベルを割り当てることができます。ラベルを設定するには、タイムライン上のフレームを選択して、プロパティインスペクタの [フレームラベル] フィールドに名前を入力します。

複雑なムービークリップを作成する場合を考えると、番号の代わりにフレームラベルを使用する利点は明らかです。アニメーションのフレーム、レイヤー、およびトゥイーンの数が大きくなる場合は、重要なフレームにムービークリップの動作を表す説明として " オフ "、" ウォーキング "、または " ランニング " などを付けるようにしてください。ラベル付きフレームに移動する ActionScript 呼び出しは、指定のフレーム番号ではなく単一の参照、つまりラベルに対するポインタであるため、読みやすさと柔軟性が向上します。アニメーションの特定のセグメントを、後から別のフレームに移動する場合でも、移動先のフレームに対して同じラベルを使用する限りは、ActionScript コードを変更する必要はありません。

コードのフレームラベルを表示するため、ActionScript 3.0 には FrameLabel クラスがあります。このクラスの各インスタンスは単一のフレームラベルを表し、プロパティインスペクタで示されるフレームラベル名を表す name プロパティと、frame プロパティがあります。後者のプロパティは、タイムライン上でラベルが配置されているフレームの番号を示します。

ムービークリップインスタンスに関連付けられている FrameLabel インスタンスへのアクセスを可能にするため、MovieClip クラスには FrameLabel オブジェクトを直接返すプロパティが 2 つあります。currentLabels プロパティは、ムービークリップのタイムライン全体のすべての FrameLabel オブジェクトで構成される配列を返します。currentLabel プロパティはタイムラインで最後に到達したフレームラベルである単一の FrameLabel オブジェクトを返します。

ロボット (robot) という名前のムービークリップを作成し、そのアニメーションのさまざまな状態にラベルを付ける場合を考えます。次のコードでは、currentLabel プロパティをチェックする条件を設定して、ロボットの現在の状態を確認しています。

```
if (robot.currentLabel.name == "walking"
{
   // 何らかの処理を実行します。
}
```

シーンの操作

Flash のオーサリング環境では、シーンを使用して SWF ファイルで再生される一連のタイムラインを区切ることができます。`gotoAndPlay()` または `gotoAndStop()` メソッドの第 2 パラメータを使用すると、再生ヘッドを送るシーンを指定できます。FLA ファイルはすべて最初のシーンから開始されますが、新しいシーンを作成することもできます。

シーンにはいくつかの短所があるため、シーンの使用が必ずしも最適であるとは限りません。複数のシーンを含む Flash ドキュメントで、特に作成者が複数存在する場合には、管理が困難になる可能性があります。パブリッシュする際には 1 つのタイムラインにすべてのシーンがマージされるため、複数のシーンを使用すると、帯域幅の利用効率が悪くなる可能性もあります。再生されないシーンがある場合でも、すべてのシーンのプログレッシブダウンロードが行われます。このような理由から、複数の長いタイムラインベースアニメーションを整理するのに必要な場合を除いて、複数のシーンを使用することは推奨されません。

MovieClip クラスの `scenes` プロパティは、SWF ファイルのすべてのシーンを示す Scene オブジェクトの配列を返します。`currentScene` プロパティは Scene オブジェクトを返します。このオブジェクトは、再生中のシーンを示します。

Scene クラスには、シーンに関する情報を示すプロパティがいくつかあります。`labels` プロパティは、シーン内のフレームラベルを示す FrameLabel オブジェクトの配列を返します。`name` プロパティは、シーンの名前を文字列として返します。`numFrames` プロパティは、シーンのフレーム総数を示す整数を返します。

ActionScript での MovieClip オブジェクトの作成

ライブラリからステージ上にアセットをドラッグすると、Flash の画面にコンテンツを追加できますが、方法はこれだけではありません。複雑なプロジェクトの場合、一般的に、経験豊富な開発者はプログラムを使ってムービークリップを作成しようと考えます。この方法は、コードの再利用が容易でコンパイル時間を短縮できます。また、ActionScript のみで利用可能な高度な修正も可能になります。

ActionScript 3.0 の表示リスト API は、MovieClip オブジェクトを動的に作成することによって、プロセスを効率化します。表示リストに追加するプロセスとは別に、MovieClip インスタンスを直接インスタンス化できるので、コントロールを犠牲にせずに柔軟性と簡便性がもたらされます。

ActionScript 3.0 では、ムービークリップまたはその他の表示オブジェクトのインスタンスをプログラムによって作成すると、それが表示リストに追加されるまでは画面上には表示されません。追加するには、表示オブジェクトコンテナで `addChild()` または `add ChildAt()` メソッドを呼び出します。このため、ムービークリップが画面にレンダリングされる前にムービークリップを作成し、プロパティを設定し、メソッドを呼び出すことが可能です。表示リストの操作方法の詳細については、364 ページの「表示オブジェクトコンテナの操作」を参照してください。

ActionScript に対するライブラリシンボルの書き出し

デフォルトでは、Flash ドキュメントのライブラリのムービークリップシンボルのインスタンスは、動的に作成することはできません。つまり、ActionScript を使用してのみ作成可能です。これは、ActionScript で使用する目的で書き出されるシンボルはすべて SWF ファイルに追加されますが、シンボルの中にはステージで使用されないものもあるからです。このような理由から、ActionScript でシンボルを使用可能にするには、シンボルを ActionScript に書き出すよう指定する必要があります。

ActionScript 用にシンボルを書き出すには :

1. [ライブラリ] パネルでシンボルを選択し、[シンボルプロパティ] ダイアログボックスを開きます。
2. 必要であれば、詳細設定を開きます。
3. [リンケージ] セクションで、[AcrionScript に書き出し] チェックボックスを選択します。

   この操作を行うと [クラス] フィールドと [基本クラス] フィールドがアクティブ化されます。

   デフォルトでは、[クラス] フィールドにはスペースを消去したシンボル名が表示されます。たとえば、"Tree House" というシンボル名は、"TreeHouse" となります。シンボルの動作でカスタムクラスを使用するよう指定するには、パッケージも含むクラスのフルネームをこのフィールドに入力します。ActionScript にシンボルのインスタンスを作成できるようはしたいが、動作を追加する必要がない場合は、クラス名をそのまま残しておくことができます。

   [基本クラス] フィールドのデフォルト値は `flash.display.MovieClip` です。シンボルによって他のカスタムクラスの機能を拡張するには、そのクラスが Sprite または MovieClip クラスを拡張できれば、代わりにそのクラスの名前を指定することができます。
4. [OK] ボタンをクリックして変更内容を保存します。

   この段階で、シンボルに対して動作を追加する必要がなかった場合など、Flash が指定クラスの定義を含む外部 ActionScript ファイルを見つけることができなければ、警告が表示されます。

   クラスパスにこのクラスの定義が見つからなかったため、書き出す際に SWF ファイル内にクラスを自動生成します。

   ライブラリシンボルで MovieClip クラスの機能以外の独自の機能が必要ない場合は、この警告を無視できます。

シンボルに対してクラスを提供しなければ、Flash はシンボルに対し、次のようなクラスを作成します。

```
package
{
  import flash.display.MovieClip;

  public class ExampleMovieClip extends MovieClip
  {
    public function ExampleMovieClip()
    {
    }
  }
}
```

シンボルに対して ActionScript 機能を追加する場合は、この構造に適切なプロパティとメソッドを追加します。たとえば、幅 50 ピクセル、高さ 50 ピクセルの円を含むムービークリップシンボルがあり、そのシンボルは、クラス名 Circle として ActionScript に書き出されるよう設定されているとします。以下のコードを Circle.as ファイルに配置すると、MovieClip クラスが拡張されて、シンボルに getArea() と getCircumference() の追加メソッドが提供されます。

```
package
{
  import flash.display.MovieClip;

  public class Circle extends MovieClip
  {
    public function Circle()
    {
    }

    public function getArea():Number
    {
        // 公式は、π × 半径の 2 乗です。
        return Math.PI * Math.pow((width / 2), 2);
    }

    public function getCircumference():Number
    {
        // 公式は、π × 直径です。
        return Math.PI * width;
    }
  }
}
```

Flash ドキュメントの最初のフレームのキーフレームに配置される次のコードにより、シンボルのインスタンスが作成され、それが画面に表示されます。

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

次のコードでは、個々のアセットをステージ上にドラッグする代わりに、ActionScript ベースのインスタンス化を行っています。作成される円には、ムービークリップのすべてのプロパティと Circle クラスに定義した独自のメソッドが含まれます。これは非常に基本的な例です。ライブラリシンボルは、クラス内でプロパティとメソッドをいくつでも指定できます。

手作業で用意するのが困難な大量のインスタンスを動的に作成できるため、ActionScript ベースのインスタンス化は非常に有効です。また、作成時に各インスタンスのプロパティをカスタマイズできるため、柔軟性にも優れています。ループを使用して複数の Circle インスタンスを動的に作成すると、これらの利点を理解できます。Circle シンボルとクラスは Flash ドキュメントのライブラリであらかじめ記述されているので、次のコードを最初のフレームのキーフレーム上に配置します。

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
  // 新しい Circle インスタンスを作成します。
  var c:Circle = new Circle();
  // 新しい Circle インスタンスを、円のステージ上に均等な間隔で置く
  // X 座標上に配置します。
  c.x = (stage.stageWidth / totalCircles) * i;
  // Circle インスタンスを、ステージの垂直方向の中心に配置します。
  c.y = stage.stageHeight / 2;
  // Circle インスタンスをランダムなカラーに変更します。
  c.transform.colorTransform = getRandomColor();
  // Circle インスタンスを現在のタイムラインに追加します。
  addChild(c);
}

function getRandomColor():ColorTransform
{
  // 赤、緑、青のカラーチャンネルのランダム値を生成します。
  var red:Number = (Math.random() * 512) - 255;
  var green:Number = (Math.random() * 512) - 255;
  var blue:Number = (Math.random() * 512) - 255;

  // ランダムなカラーが付与された ColorTransform オブジェクトを作成して返します。
  return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

ここでは、コードを使用してシンボルの複数インスタンスをすばやく作成し、カスタマイズする方法を示します。各インスタンスはループ内で現在のカウントに基づいて配置され、各インスタンスには MovieClip クラスを拡張することによって Circle が継承する `transform` プロパティを設定してランダムなカラーが付与されます。

外部の SWF ファイルのロード

ActionScript 3.0 では、Loader クラスを使用して SWF ファイルをロードします。外部 SWF ファイルをロードするには、ActionScript で次の 4 つのタスクを実行する必要があります。

1. ファイルの url により、新規 URLRequest オブジェクトを作成します。
2. 新規 Loader オブジェクトを作成します。
3. URLRequest インスタンスをパラメータとして渡し、Loader オブジェクトの `load()` メソッドを呼び出します。
4. Flash ドキュメントのメインタイムラインなどの表示オブジェクトコンテナで `addChild()` メソッドを呼び出し、表示リストに Loader インスタンスを追加します。

最終的に、コードは次のようになります。

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

これと同じコードを使用し、JPEG、GIF、PNG イメージなどの外部イメージファイルをロードすることができます。その際、SWF ファイルの url ではなく、イメージファイルの url を指定します。イメージファイルとは異なり、SWF ファイルには ActionScript を含めることができます。このため、SWF ファイルをロードするプロセスがイメージのロードと同じでも、ActionScript を使用して外部の SWF ファイルと何らかの通信を行う場合は、外部の SWF ファイルをロードするときに、ロードする SWF ファイルとロードされる SWF ファイルの両方が同じセキュリティサンドボックスに属している必要があります。また、ロードされる SWF ファイル内のクラスと同じ名前空間を使用するクラスが外部の SWF ファイルに含まれる場合には、名前空間の競合を避けるために、ロードされる SWF ファイル用に新しいアプリケーションドメインを作成する必要があります。セキュリティとアプリケーションドメインの重要事項については、666 ページの「ApplicationDomain クラスの使用」および 738 ページの「SWF ファイルとイメージのロード」を参照してください。

正常にロードされた外部の SWF ファイルには、`Loader.content` プロパティを使用してアクセスできます。外部の SWF ファイルを ActionScript 3.0 用としてパブリッシュすると、拡張するクラスに基づいて、ムービークリップまたはスプライトのいずれかになります。

従来の SWF ファイルをロードする際の注意事項

以前のバージョンの ActionScript を使用して外部 SWF ファイルをパブリッシュする場合には、制約事項に注意する必要があります。AVM2 (ActionScript Virtual Machine 2) で実行される ActionScript 3.0 の SWF ファイルと異なり、ActionScript 1.0 または 2.0 用としてパブリッシュされる SWF ファイルは AVM1 (ActionScript Virtual Machine 1) で実行されます。

AVM1 SWF ファイルが正常にロードされると、ロードされたオブジェクト (`Loader.content` プロパティ) は、AVM1Movie オブジェクトになります。AVM1Movie インスタンスは、MovieClip インスタンスと同じではありません。これは表示オブジェクトで、ムービークリップとは異なり、タイムラインに関連するメソッドまたはプロパティを含んでいません。親 AVM2 SWF ファイルは、ロードされた AVM1Movie オブジェクトのプロパティ、メソッド、またはオブジェクトにアクセスすることはできません。

AVM2 SWF ファイルによってロードされた AVM1 SWF ファイルには、次の制約があります。詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の AVM1Movie クラスリストを参照してください。

例 : RuntimeAssetsExplorer

ActionScript に書き出す機能は、特にライブラリに有効で、複数のプロジェクトで役立つ場合があります。ActionScript に書き出されたシンボルは、その SWF ファイルだけでなく、SWF ファイルをロードする同じセキュリティ Sandbox 内のすべての SWF ファイルで利用できます。このようにすると、単一の Flash ドキュメントでグラフィカルアセット保持専用の SWF ファイルを生成できます。この方法は特に、大規模なプロジェクトで、ビジュアルアセットを担当する複数のデザイナーが " ラッパー " SWF ファイルを作成する複数の開発者と平行して作業を行い、実行時にグラフィカルアセットの SWF ファイルをロードする場合に役立ちます。この方法を使用すると、一連のバージョン化されたファイルの管理が可能です。この場合、グラフィカルアセットはプログラミング開発の進捗に依存しません。

RuntimeAssetsExplorer アプリケーションは RuntimeAsset のサブクラスであるすべての SWF ファイルをロードします。このため、SWF ファイルの利用可能なアセットを参照できます。この例では、次の内容について説明します。

- `Loader.load()` を使用した外部 SWF ファイルのロード
- ActionScript に書き出されるライブラリシンボルの動的な作成
- MovieClip 再生の ActionScript による制御

開始に先立ち、各 SWF ファイルは同じセキュリティサンドボックスに置く必要があることに注意してください。詳細については、732 ページの「セキュリティサンドボックス」を参照してください。

このサンプルのアプリケーションのファイルを入手するには、
www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。
RuntimeAssetsExplorer アプリケーションのファイルは、"Samples/Chapters/RuntimeAssetsExplorer" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
RuntimeAssetsExample.mxml または RuntimeAssetsExample.fla	Flex (MXML) または Flash (FLA) 用アプリケーションのユーザーインターフェイス。
GeometricAssets.as	RuntimeAsset インターフェイスを実装するサンプルクラス。
GeometricAssets.fla	ActionScript 用に書き出されるシンボルを含んでいる GeometricAssets クラス (FLA のドキュメントクラス) にリンクされている FLA ファイル。
com/example/programmingas3/runtimeassetsexplorer/RuntimeLibrary.as	エクスプローラコンテナにロードされるすべてのランタイムアセット SWF ファイルに必要なメソッドを定義するインターフェイス。
com/example/programmingas3/runtimeassetsexplorer/AnimatingBox.as	回転するボックス形のライブラリシンボルのクラス。
com/example/programmingas3/runtimeassetsexplorer/AnimatingStar.as	回転する星形のライブラリシンボルのクラス。

ランタイムライブラリインターフェイスの確立

エクスプローラで正しく SWF ライブラリを操作できるようにするには、ランタイムアセットライブラリの構造を形式化する必要があります。構造を形式化するには、インターフェイスを作成します。これはメソッドの設計図であり、目的の構造を定義するという点ではクラスに似ていますが、メソッド本体は提供しない点がクラスとは異なります。インターフェイスを使用すると、ランタイムライブラリとエクスプローラが相互にやり取りできます。ブラウザにロードされるランタイムアセットの SWF はそれぞれ、このインターフェイスを実装します。インターフェイスとその利便性については、148 ページの「インターフェイス」を参照してください。

RuntimeLibrary インターフェイスは非常にシンプルです。必要なのは、書き出されるシンボルのクラスパスの配列をエクスプローラに提供でき、ランタイムライブラリで使用可能な関数だけです。このため、インターフェイスには getAssets() というメソッドだけを含めます。

```
package com.example.programmingas3.runtimeassetexplorer
{
  public interface RuntimeLibrary
  {
    function getAssets():Array;
  }
}
```

アセットライブラリ SWF ファイルの作成

RuntimeLibrary インターフェイスを定義すると、別の SWF ファイルにロード可能な複数のアセットライブラリ SWF ファイルを作成できます。個々の SWF アセットライブラリを作成する際には、次の 4 つのタスクを実行します。

- アセットライブラリ SWF ファイルのクラスを作成する
- ライブラリを含む各アセットのクラスを作成する
- 実際のグラフィックアセットを作成する
- グラフィックエレメントにクラスを関連付けて、ライブラリ SWF をパブリッシュする

RuntimeLibrary インターフェイスを実装するクラスの作成

次に、RuntimeLibrary インターフェイスを実装する GeometricAssets クラスを作成します。これは FLA のドキュメントクラスになります。このクラスのコードは RuntimeLibrary インターフェイスによく似ています。両者の違いは、クラス定義では `getAssets()` メソッドにはメソッド本体がある点です。

```
package
{
  import flash.display.Sprite;
  import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

  public class GeometricAssets extends Sprite implements RuntimeLibrary
  {
    public function GeometricAssets() {

    }
    public function getAssets():Array {
      return [ "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",
             "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
    }
  }
}
```

2 番目のランタイムライブラリを作成する場合は、独自の `getAssets()` を実装する AnimationAssets などの別のクラスをベースにした別の FLA を作成できます。

各 MovieClip アセット用クラスの作成

この例では、カスタムアセットに機能を追加することなく、MovieClip クラスの拡張のみを行います。次の AnimatingStar のコードは、AnimatingBox のコードと類似しています。

```
package com.example.programmingas3.runtimeassetexplorer
{
  import flash.display.MovieClip;

  public class AnimatingStar extends MovieClip
  {
    public function AnimatingStar() {
    }
  }
}
```

ライブラリのパブリッシュ

次に、新しい FLA を作成し、プロパティインスペクタの [ドキュメントクラス] フィールドに GeometricAssets を入力して、MovieClip ベースのアセットと新しいクラスを接続します。この例では、タイムライントゥイーンを使用して、360 フレームを時計回りに 1 回転する 2 つの基本シェイプを作成します。`animatingBox` および `animatingStar` のシンボルはいずれも [ActionScript に書き出し] に設定され、[クラス] フィールドは `getAssets()` の実装で指定されたそれぞれのクラスパスに設定されます。標準の MovieClip メソッドをサブクラスにする必要があるため、デフォルトのベースクラスである `flash.display.MovieClip` は変更されません。

シンボルの書き出し設定が済んだら、FLA をパブリッシュします。これで、最初のランタイムライブラリが設定されます。この SWF は別の AVM2 SWF ファイルにロードできます。また、AnimatingBox および AnimatingStar のシンボルは新しい SWF ファイルでも使用できます。

別の SWF ファイルへのライブラリのロード

必要な最後の機能は、アセットエクスプローラのユーザーインターフェイスです。この例では、ランタイムライブラリのパスは、`ASSETS_PATH` という名前の変数としてハードコードされています。また、FileReference クラスを使用して、ハードドライブ上の特定の SWF ファイルを参照するインターフェイスの作成などを行うこともできます。

ランタイムライブラリが正常にロードされると、Flash Player は `runtimeAssetsLoadComplete()` メソッドを呼び出します。

```
private function runtimeAssetsLoadComplete(event:Event):void
{
  var rl:* = event.target.content;
  var assetList:Array = rl.getAssets();
  populateDropdown(assetList);
  stage.frameRate = 60;
}
```

このメソッドでは、rl 変数はロードされた SWF ファイルを示します。このコードは、ロードされた SWF ファイルの getAssets() メソッドを呼び出して使用可能なアセットのリストを取得します。その後、populateDropDown() メソッドを呼び出し、使用可能なアセットのリストを使用して ComboBox コンポーネントを指定します。このメソッドに各アセットの完全なクラスパスが保存されます。ユーザーインターフェイスの [追加] ボタンをクリックすると、addAsset() メソッドがトリガーされます。

```
private function addAsset():void
{
   var className:String = assetNameCbo.selectedItem.data;
   var AssetClass:Class = getDefinitionByName(className) as Class;
   var mc:MovieClip = new AssetClass();
   ...
}
```

これにより、ComboBox で選択中のアセットのクラスパスが取得され (assetNameCbo.selectedItem.data)、flash.utils パッケージより getDefinitionByName() 関数を使用して、アセットのクラスに対する実際の参照が入手され、そのアセットの新規インスタンスが作成されます。

第 17 章

テキストの操作

17

ActionScript 3.0 では、テキストは通常テキストフィールドに表示されますが、状況に応じて表示リストのアイテムのプロパティとして (たとえば、UI コンポーネントのラベルとして) 表示することもできます。この章では、テキストフィールドのスクリプト定義のコンテンツおよびユーザー入力、リモートファイルからのダイナミックテキスト、または Adobe Flash CS3 Professional で定義された静止テキストを操作する方法について説明します。ActionScript 3.0 プログラマとして、ユーザーはテキストフィールドの特定のコンテンツの設定、テキストのソースの指定、スタイルとフォーマットを使用したテキストの外観の設定を行うことができます。また、テキストの入力や、ハイパーリンクのクリックなどのユーザーイベントに応答することもできます。

目次

テキストの操作の基礎 486
テキストの表示 489
テキストの選択と操作 493
テキスト入力のキャプチャ 494
テキスト入力の制限 496
テキストのフォーマット 496
高度なテキストレンダリング 501
静止テキストの操作 504
例 : 新聞形式のテキストフォーマット 505

テキストの操作の基礎

テキストの操作の概要

Adobe Flash Player の画面にテキストを表示するには、TextField クラスのインスタンスを使用します。TextField クラスは、Adobe Flex Framework や Flash オーサリング環境に用意されている TextArea コンポーネントや TextInput コンポーネントなどの他のテキストベースコンポーネントの基盤になっています。Flash オーサリング環境でのテキストコンポーネントの使用の詳細については、『Flash ユーザーガイド』の「テキストコントロールについて」を参照してください。

テキストフィールドのコンテンツは、SWF ファイル内にあらかじめ指定することも、テキストファイルやデータベースなどの外部ソースからロードすることもできます。また、アプリケーションを操作するユーザーに入力してもらうこともできます。テキストフィールド内では、テキストはレンダリング済み HTML にイメージが埋め込まれたレンダリング済み HTML コンテンツとして表示されます。テキストフィールドのインスタンスを設定すると、TextFormat クラスおよび StyleSheet クラスなどの flash.text パッケージクラスを使用して、テキストの外観を制御することができます。flash.text パッケージには、ActionScript でのテキストの作成、管理、およびフォーマットに関連するほとんどすべてのクラスが含まれています。

テキストをフォーマットするには、TextFormat オブジェクトを使用してフォーマットを定義し、そのオブジェクトをテキストフィールドに割り当てます。テキストフィールドに HTML テキストが含まれる場合、テキストフィールドに StyleSheet オブジェクトを適用すると、テキストフィールドのコンテンツの特定の部分にスタイルを割り当てることができます。TextFormat オブジェクトまたは StyleSheet オブジェクトには、カラー、サイズ、太さなど、テキストの外観を定義するプロパティが含まれています。TextFormat オブジェクトは、テキストフィールド内のすべてのコンテンツ、または一部の範囲のテキストにこれらのプロパティを割り当てます。たとえば、同じテキストフィールド内で、ある文をボールドの赤いテキストとし、次の文をイタリックの青いテキストにすることができます。

テキストフォーマットの詳細については、497 ページの「テキストフォーマットの割り当て」を参照してください。

テキストフィールド内の HTML テキストの詳細については、490 ページの「HTML テキストの表示」を参照してください。

スタイルシートの詳細については、497 ページの「CSS の適用」を参照してください。

テキスト関連のユーザー操作への応答には、flash.text パッケージ内のクラスに加えて、flash.events.TextEvent クラスも使用することができます。

テキストの操作に関する一般的なタスク

この章では、以下の一般的なテキスト関連のタスクについて説明します。

- テキストフィールドのコンテンツの変更
- テキストフィールドでの HTML の使用
- テキストフィールドでのイメージの使用
- テキストの選択とユーザー選択テキストの操作
- テキスト入力のキャプチャ
- テキスト入力の制限
- フォーマットと CSS (カスケーディングスタイルシート) スタイルのテキストへの適用
- シャープネス、太さ、およびアンチエイリアスを利用したテキスト表示の微調整
- ActionScript からの静止テキストフィールドに対するアクセスと操作

重要な概念と用語

次の参照リストに、この章で使われる重要な用語を示します。

- CSS : XML (または HTML) 形式で構造化されたコンテンツのスタイルやフォーマットを指定するための標準的なシンタックスです。
- デバイスフォント : ユーザーのコンピュータにインストールされているフォントです。
- ダイナミックテキストフィールド : コンテンツを ActionScript で変更することはできるが、ユーザー入力で変更することはできないテキストフィールドです。
- 埋め込みフォント : 文字のアウトラインデータがアプリケーションの SWF ファイルに保存されるフォントです。
- HTML テキスト : 実際のテキストコンテンツと共に HTML フォーマットタグを含み、ActionScript によってテキストフィールドに入力されるテキストコンテンツです。
- 入力テキストフィールド : ユーザー入力と ActionScript のいずれでもコンテンツを変更できるテキストフィールドです。
- 静止テキストフィールド : Flash オーサリングツールで作成された、SWF ファイルの実行時にコンテンツを変更することができないテキストフィールドです。
- テキスト行のメトリック : テキストのベースライン、文字の最上部の高さ、ディセンダ (一部の英小文字でベースラインよりも下側にある部分) のサイズなどの、テキストフィールド内のテキストコンテンツのさまざまな部分のサイズの測定情報です。

本章の例について

本章をお読みになる際は、いくつかのコード例を試すことができます。この章では ActionScript でのテキストフィールドの操作について扱っているため、この章のほぼすべてのコードには TextField オブジェクトの操作が含まれています。操作の対象となる TextField オブジェクトは、Flash オーサリングツールのステージ上で作成されて配置されたオブジェクトか、ActionScript を使用して作成されたオブジェクトです。サンプルのテストには、Flash Player で結果を表示してテキストフィールドに対するコードの効果を確認することが含まれています。

この章の例は 2 つのグループに分類されます。一方のタイプの例では、明示的に作成せずに TextField オブジェクトを操作します。この章のコードをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインでキーフレームを選択します。
3. [アクション] パネルを開き、[スクリプト] ペインにコードをコピーします。
4. テキストツールを使用して、ステージ上でダイナミックテキストフィールドを作成します。
5. テキストフィールドが選択された状態で、プロパティインスペクタを使用してそのフィールドにインスタンス名をつけます。この名前は、コード例でテキストフィールドに使われている名前と一致させる必要があります。たとえば、コードで `myTextField` という名前のテキストフィールドを操作する場合は、テキストフィールドのインスタンス名も `myTextField` にします。
6. [コントロール]-[ムービープレビュー] を使用して、プログラムを実行します。

   画面に、コードで指定したようにテキストフィールドを操作した結果が表示されます。

本章のコード例のもう一方のタイプは、SWF のドキュメントクラスとして使用することを意図したクラス定義で構成されています。これらのコードでは、TextField インスタンスがコード例によって作成されるので、インスタンスを別個に作成する必要はありません。このタイプのコードをテストするには :

1. 新しい Flash ドキュメントを作成し、コンピュータに保存します。
2. 新しい ActionScript ファイルを作成し、Flash ドキュメントと同じディレクトリに保存します。ファイルの名前はコード内のクラス名と同じにする必要があります。たとえば、コードで TextFieldTest という名前のクラスが定義されている場合は、TextFieldTest.as という名前を使用して ActionScript ファイルを保存します。
3. ActionScript ファイル内にコードをコピーし、ファイルを保存します。
4. Flash ドキュメント内で、ステージまたはワークスペースの空白部分をクリックして、ドキュメントのプロパティインスペクタをアクティブにします。
5. プロパティインスペクタの [ドキュメントクラス] フィールドに、テキストからコピーした ActionScript クラスの名前を入力します。
6. [コントロール]-[ムービープレビュー] を使用して、プログラムを実行します。

   画面にコード例の結果が表示されます。

コード例をテストするその他の方法については、60 ページの「章内のコード例のテスト」で詳しく説明します。

テキストの表示

Adobe Flex Builder や Flash オーサリングツールなどのオーサリングツールには、テキスト関連コンポーネントやテキストツールなどのいくつかのオプションがテキスト表示のために用意されていますが、テキストをプログラムによって表示する主な方法はテキストフィールドを使用する方法です。

テキストのタイプ

テキストフィールド内のテキストのタイプは、そのソースによって次のように分類されます。

- ダイナミックテキスト

  ダイナミックテキストには、テキストファイル、XML ファイル、さらにはリモート Web サービスなどの外部ソースからロードされたコンテンツが含まれます。詳細については、489 ページの「テキストのタイプ」を参照してください。

- 入力テキスト

  入力テキストは、ユーザーが入力した任意のテキストまたはユーザーが編集可能なダイナミックテキストです。入力テキストをフォーマットするスタイルシートを設定するか、または flash.text.TextFormat クラスを使用して、入力コンテンツのテキストフィールドにプロパティを割り当てることができます。詳細については、494 ページの「テキスト入力のキャプチャ」を参照してください。

- 静止テキスト

  静止テキストは、Flash オーサリングツールを使用する場合にのみ作成されます。ActionScript 3.0 を使用して静止テキストのインスタンスを作成することはできません。ただし、StaticText および TextSnapshot などの ActionScript クラスを使用して、既存の静止テキストのインスタンスを操作することができます。詳細については、504 ページの「静止テキストの操作」を参照してください。

テキストフィールドのコンテンツの変更

`flash.text.TextField.text` プロパティにストリングを割り当てることにより、ダイナミックテキストを定義することができます。次のように、ストリングをプロパティに直接割り当てます。

```
myTextField.text = “Hello World”;
```

次の例に示すように、`text` プロパティにスクリプトで定義した変数の値を割り当てることもできます。

```
package
{
   import flash.display.Sprite;
   import flash.text.*;

   public class TextWithImage extends Sprite
   {
```

```
    private var myTextBox:TextField = new TextField();
    private var myText:String = "Hello World";

    public function TextWithImage()
    {
      addChild(myTextBox);
      myTextBox.text = myText;
    }
  }
}
```

または、text プロパティにリモート変数の値を割り当てることができます。リモートソースからテキストの値をロードする方法には、以下の 3 つのオプションがあります。

- flash.net.URLLoader および flash.net.URLRequest クラスは、ローカルまたはリモートロケーションからテキストの変数をロードします。
- FlashVars 属性は、SWF ファイルをホストする HTML ページに埋め込むもので、テキストの変数の値を含めることができます。
- flash.net.SharedObject クラスは値の永続ストレージを管理します。詳細については、635 ページの「ローカルデータの保存」を参照してください。

HTML テキストの表示

flash.text.TextField クラスには htmlText プロパティが含まれており、このプロパティを使用すると、コンテンツをフォーマットするための HTML タグが含まれているストリングとして、テキストストリングを指定することができます。次の例に示すように、Flash Player でテキストを HTML としてレンダリングするには、text プロパティではなく、htmlText プロパティにストリング値を割り当てる必要があります。

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u>
  text.</p>";
myTextBox.htmlText = myText;
```

Flash Player は、htmlText プロパティでは、一部の HTML タグと HTML エンティティをサポートします。『ActionScript 3.0 コンポーネントリファレンスガイド』の flash.text.TextField.htmlText プロパティの説明には、サポートされる HTML タグと HTML エンティティについての詳細情報が用意されています。

htmlText プロパティを使用してコンテンツを指定した後、スタイルシートまたは textformat タグを使用してコンテンツのフォーマットを管理することができます。詳細については、496 ページの「テキストのフォーマット」を参照してください。

テキストフィールドでのイメージの使用

コンテンツを HTML テキストとして表示することには、テキストフィールドにイメージを含めることができるという利点もあります。img タグを使用して、ローカルまたはリモートのイメージを参照し、関連付けられたテキストフィールドに表示させることができます。

次の例では、myTextBox という名前のテキストフィールドを作成し、SWF ファイルと同じディレクトリに格納されている目の JPG イメージを、表示テキスト内に埋め込みます。

```
package
{
  import flash.display.Sprite;
  import flash.text.*;

  public class TextWithImage extends Sprite
  {
    private var myTextBox:TextField;
    private var myText:String = "<p>This is <b>some</b> content to <i>test</i>
  and <i>see</i></p><p><img src='eye.jpg' width='20' height='20'></p><p>what can
  be rendered.</p><p>You should see an eye image and some <u>HTML</u> text.</p>";

    public function TextWithImage()
    {
      myTextBox.width = 200;
      myTextBox.height = 200;
      myTextBox.multiline = true;
      myTextBox.wordWrap = true;
      myTextBox.border = true;

      addChild(myTextBox);
      myTextBox.htmlText = myText;
    }
  }
}
```

img タグは、JPEG、GIF、PNG、および SWF ファイルをサポートします。

テキストフィールド内のテキストのスクロール

多くの場合、テキストは、テキストを表示するテキストフィールドよりも長くなります。または、一度に表示可能なテキストより長いテキストを入力できる入力フィールドを使用する場合があります。flash.text.TextField クラスのスクロール関連プロパティを使用して、長いコンテンツを垂直または水平方向に管理することができます。

スクロール関連のプロパティには、TextField.scrollV、TextField.scrollH、maxScrollV、および maxScrollH があります。これらのプロパティを使用して、マウスのクリックやキー押下などのイベントに応答します。

次の例では、フィールドで一度に表示可能な長さを超えるテキストを含むようにサイズが設定されたテキストフィールドを作成します。ユーザーがテキストフィールドをクリックすると、テキストが垂直方向にスクロールされます。

```
package
{
  import flash.display.Sprite;
  import flash.text.*;
  import flash.events.MouseEvent;

  public class TextScrollExample extends Sprite
  {
    private var myTextBox:TextField = new TextField();
    private var myText:String = "Hello world and welcome to the show. It's
  really nice to meet you. Take your coat off and stay a while. OK, show is over.
  Hope you had fun. You can go home now. Don't forget to tip your waiter. There
  are mints in the bowl by the door. Thank you. Please come again.";

    public function TextScrollExample()
    {
      myTextBox.text = myText;
      myTextBox.width = 200;
      myTextBox.height = 50;
      myTextBox.multiline = true;
      myTextBox.wordWrap = true;
      myTextBox.background = true;
      myTextBox.border = true;

      var format:TextFormat = new TextFormat();
      format.font = "Verdana";
      format.color = 0xFF0000;
      format.size = 10;

      myTextBox.defaultTextFormat = format;
      addChild(myTextBox);
      myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
    }

    public function mouseDownScroll(event:MouseEvent):void
    {
      myTextBox.scrollV++;
    }
  }
}
```

テキストの選択と操作

ダイナミックまたは入力テキストを選択できます。TextField クラスのテキスト選択プロパティとメソッドは、インデックス位置を使用して操作するテキストの範囲を設定するので、コンテンツがわからない場合でも、プログラムによってダイナミックまたは入力テキストを選択することができます。

メモ	Flash オーサリングツールでは、選択可能にするオプションを静止テキストフィールドで選択した場合、表示リストにエクスポートされて配置されるテキストフィールドは、通常のダイナミックテキストフィールドになります。

テキストの選択

flash.text.TextField.selectable プロパティはデフォルトで true になっています。したがって、setSelection() メソッドを使用してプログラムでテキストを選択することができます。

たとえば、ユーザーがテキストフィールドをクリックしたときにそのテキストフィールドの内部の特定のテキストが選択されるように設定することができます。

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL
  CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
  myTextField.setSelection(49, 65);
}
```

テキストフィールド内のテキストが最初に表示されるときにそのテキストが選択されるようにする場合も、同様にして、テキストフィールドが表示リストに追加されるときに呼び出されるイベントハンドラ関数を作成します。

ユーザー選択テキストのキャプチャ

TextField クラスの selectionBeginIndex プロパティと selectionEndIndex プロパティを使用すると、ユーザーが現在選択している内容をキャプチャできます。これらのプロパティは " 読み取り専用 " のため、プログラムに従ってテキストを選択するように設定することはできません。また、入力テキストフィールドでは caretIndex プロパティを使用できます。

たとえば、次のコードではユーザー選択テキストのインデックス値をトレースしています。

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index values
  for the first and last letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
  trace("First letter index position: " + myTextField.selectionBeginIndex);
  trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

選択範囲に TextFormat オブジェクトのプロパティのコレクションを適用すると、テキストの外観を変更することができます。選択されているテキストに TextFormat のプロパティのコレクションを適用する方法の詳細については、500 ページの「テキストフィールド内のテキストの範囲のフォーマット」を参照してください。

テキスト入力のキャプチャ

デフォルトでは、テキストフィールドの type プロパティは dynamic に設定されます。TextFieldType クラスを使用して type プロパティを input に設定すると、アプリケーションの他の部分で使用するために、ユーザー入力を収集し、値を保存することができます。入力テキストフィールドは、フォームで使用するのに便利です。また、テキスト値をユーザーに定義させてその値をプログラムの他の部分で使用するアプリケーションでも役に立ちます。

たとえば、次のコードは、myTextBox という名前の入力テキストフィールドを作成します。ユーザーがフィールドにテキストを入力すると、textInput イベントがトリガされます。textInputCapture という名前のイベントハンドラが、入力されたテキストのストリングをキャプチャして、変数を割り当てます。Flash Player は、myOutputBox という名前の別のテキストフィールドにその新しいテキストを表示します。

```
package
{
  import flash.display.Sprite;
  import flash.display.Stage;
  import flash.text.*;
  import flash.events.*;

  public class CaptureUserInput extends Sprite
  {
    private var myTextBox:TextField = new TextField();
    private var myOutputBox:TextField = new TextField();
    private var myText:String = "Type your text here.";

    public function CaptureUserInput()
    {
      captureText();
    }

    public function captureText():void
    {
      myTextBox.type = TextFieldType.INPUT;
      myTextBox.background = true;
      addChild(myTextBox);
      myTextBox.text = myText;
      myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
    }

    public function textInputCapture(event:TextEvent):void
    {
      var str:String = myTextBox.text;
      createOutputBox(str);
    }

    public function createOutputBox(str:String):void
    {
      myOutputBox.background = true;
      myOutputBox.x = 200;
      addChild(myOutputBox);
      myOutputBox.text = str;
    }

  }
}
```

テキスト入力の制限

通常、入力テキストフィールドはアプリケーションのフォームまたはダイアログボックスとして使用されるので、ユーザーがテキストフィールドに入力できる文字のタイプを制限する必要や、パスワードなどのテキストを非表示にする必要がある場合があります。flash.text.TextField クラスには、ユーザー入力を制御するために設定できる displayAsPassword プロパティと restrict プロパティが用意されています。

displayAsPassword プロパティは、単にユーザーが入力するテキストを非表示にします (一連のアスタリスクとして表示)。displayAsPassword を true に設定すると、[カット] コマンドと [コピー] コマンド、およびそれに対応するキーボードショートカットが機能しなくなります。次の例に示すように、displayAsPassword プロパティは、背景や色などの他のプロパティと同様の方法で割り当てます。

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

restrict プロパティは、ユーザーが入力テキストフィールドに入力できる文字を指定する必要があるので若干複雑です。特定の文字、数字、または文字、数字の範囲を指定することができます。次のコードでは、ユーザーはテキストフィールドに大文字のみ入力することができます (数字または特殊文字は入力できません)。

```
myTextBox.restrict = “A-Z”;
```

ActionScript 3.0 では、ハイフンを使用して範囲を定義し、キャレットを使用して除外する文字を定義します。入力テキストフィールドでの制限事項の定義の詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』の flash.text.TextField.restrict プロパティの項を参照してください。

テキストのフォーマット

テキストの表示をプログラムによってフォーマットする方法には、いくつかのオプションがあります。TextFIeld.thickness、TextField.textColor、および TextField.textHeight プロパティなどの TextField インスタンスで、直接プロパティを設定することができます。または、htmlText プロパティを使用してテキストフィールドのコンテンツを指定し、b、i、u などのサポートされる HTML タグを使用することができます。ただし、プレーンテキストを含むテキストフィールドに TextFormat オブジェクトを適用することや、htmlText プロパティを含むテキストフィールドに StyleSheet オブジェクトを適用することもできます。TextFormat と StyleSheet オブジェクトを使用することにより、アプリケーション全体を通じてテキストの外観を制御し、一貫性を保持することができます。TextFormat または StyleSheet オブジェクトを定義して、アプリケーションの大部分またはすべてのテキストフィールドに適用することができます。

テキストフォーマットの割り当て

TextFormat クラスを使用すると、さまざまなテキスト表示プロパティを設定して、TextField オブジェクト全体、または一部の範囲のテキストにそれらのプロパティを適用することができます。

次の例では、TextField オブジェクト全体に 1 つの TextFormat オブジェクトを適用し、その TextField オブジェクトに含まれるテキスト範囲に別の TextFormat オブジェクトを適用します。

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

`TextField.setTextFormat()` メソッドは、既にテキストフィールドに表示されているテキストにのみ影響します。TextField のコンテンツが変化する場合、`TextField.setTextFormat()` メソッドをアプリケーションからもう一度呼び出して、フォーマットを再適用しなければならないことがあります。また、TextField オブジェクトの `defaultTextFormat` プロパティを設定すると、ユーザー入力テキストに適用するフォーマットを指定することができます。

CSS の適用

テキストフィールドには、プレーンテキストまたは HTML 形式のテキストを含めることができます。プレーンテキストはインスタンスの `text` プロパティに格納し、HTML テキストは `htmlText` プロパティに格納します。

CSS スタイル宣言を使用すると、多数の異なるテキストフィールドに適用できるテキストスタイルを定義することができます。CSS スタイル宣言は、アプリケーションコード内で作成することも、実行時に外部 CSS ファイルからロードすることもできます。

CSS スタイルの処理には flash.text.StyleSheet クラスを使用します。StyleSheet クラスでは、一部の CSS プロパティに限り認識されます。StyleSheet クラスでサポートされるスタイルプロパティの詳細な一覧については、『ActionScript 3.0 コンポーネントリファレンスガイド』の flash.textStylesheet の項を参照してください。

次の例に示すように、StyleSheet オブジェクトを使用してコードで CSS を作成し、作成したスタイルを HTML テキストに適用することができます。

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

この例のコードでは、StyleSheet オブジェクトを作成した後、一連のスタイル宣言プロパティを保持する単純オブジェクトを作成します。続いて、`StyleSheet.setStyle()` メソッドを呼び出します。このメソッドによって、.darkRed という名前の新しいスタイルがスタイルシートに追加されます。その後、TextField オブジェクトの `styleSheet` プロパティに StyleSheet オブジェクトを割り当てて、スタイルシートによるフォーマットを適用します。

CSS スタイルを有効にするには、`htmlText` プロパティが設定される前に、TextField オブジェクトにスタイルシートを適用する必要があります。

設計上、スタイルシートを含むテキストフィールドは編集できません。入力テキストフィールドがあり、スタイルシートが割り当てられている場合、テキストフィールドにはスタイルシートのプロパティが表示されますが、ユーザーはテキストフィールドに新しいテキストを入力することはできません。また、スタイルシートが割り当てられたテキストフィールドで、次の ActionScript API を使用することはできません。

- `TextField.replaceText()` メソッド
- `TextField.replaceSelectedText()` メソッド
- `TextField.defaultTextFormat` プロパティ
- `TextField.setTextFormat()` メソッド

テキストフィールドにスタイルシートが割り当てられている場合に、後で `TextField.styleSheet` プロパティを `null` に設定すると、`TextField.text` プロパティと `TextField.htmlText` プロパティの両方のコンテンツがタグと属性をそれぞれのコンテンツに追加し、以前に割り当てられたスタイルシートのフォーマットを組み込みます。元の `htmlText` プロパティを維持するには、スタイルシートを `null` に設定する前に、このプロパティを変数に保存します。

外部 CSS ファイルのロード

CSS を使用してフォーマットを適用する手法は、実行時に外部ファイルから CSS 情報をロードできる場合にはいっそう強力なものになります。CSS データがアプリケーション自体の外部にある場合、ActionScript 3.0 のソースコードを変更しなくても、アプリケーションのテキストの視覚的なスタイルを変更することができます。アプリケーションをデプロイした後、外部 CSS ファイルを変更することによってアプリケーションの外観を変更できます。このとき、アプリケーションの SWF ファイルを再度デプロイする必要はありません。

`StyleSheet.parseCSS()` メソッドは、CSS データが含まれているストリングを変換して、StyleSheet オブジェクトにスタイル宣言として格納します。次の例は、外部 CSS ファイルを読み取り、そのファイルに含まれるスタイル宣言を TextField オブジェクトに適用する方法を示しています。

まず、ロードする CSS ファイルの内容を示します。このファイルには example.css という名前を付けます。

```
p {
  font-family: Times New Roman, Times, _serif;
  font-size: 14;
}

h1 {
  font-family: Arial, Helvetica, _sans;
  font-size: 20;
  font-weight: bold;
}

.bluetext {
  color: #0000CC;
}
```

次に、example.css ファイルをロードして TextField のコンテンツにスタイルを適用するクラスの ActionScript コードを示します。

```
package
{
  import flash.display.Sprite;
  import flash.events.Event;
  import flash.net.URLLoader;
  import flash.net.URLRequest;
  import flash.text.StyleSheet;
  import flash.text.TextField;
  import flash.text.TextFieldAutoSize;

  public class CSSFormattingExample extends Sprite
  {
    var loader:URLLoader;
    var field:TextField;
    var exampleText:String = "<h1>This is a headline</h1>" +
```

```
        “<p>This is a line of text. <span class='bluetext'>” +
        “This line of text is colored blue.</span></p>";

      public function CSSFormattingExample():void
      {
        field = new TextField();
        field.width = 300;
        field.autoSize = TextFieldAutoSize.LEFT;
        field.wordWrap = true;
        addChild(field);

        var req:URLRequest = new URLRequest("example.css");

        loader = new URLLoader();
        loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
        loader.load(req);
      }

      public function onCSSFileLoaded(event:Event):void
      {
        var sheet:StyleSheet = new StyleSheet();
        sheet.parseCSS(loader.data);
        field.styleSheet = sheet;
        field.htmlText = exampleText;
      }
    }
}
```

CSS データがロードされると、onCSSFileLoaded() メソッドが実行され、StyleSheet.parseCSS() メソッドが呼び出されて、スタイル宣言が StyleSheet オブジェクトに転送されます。

テキストフィールド内のテキストの範囲のフォーマット

flash.text.TextField クラスの特に便利なメソッドは、setTextFormat() メソッドです。setTextFormat() を使用すると、ユーザー入力に応答する形でテキストフィールドの一部のコンテンツに特定のプロパティを割り当てることができます。たとえば、特定の項目が入力必須であることをユーザーに通知する必要があるフォームや、ユーザーがテキストを部分的に選択したときにテキストフィールド内のテキストの一部の強調を変更する必要があるフォームで使用します。

次の例では、ユーザーがテキストフィールド myTextField をクリックすると一部の範囲の文字に対して TextField.setTextFormat() が使用され、このテキストフィールドの一部のコンテンツの外観が変更されます。

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL
  CAPS changes format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
```

```
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
  myTextField.setTextFormat(myformat, 49, 65);
}
```

高度なテキストレンダリング

ActionScript 3.0 では、埋め込みフォント、アンチエイリアス設定、アルファチャンネル制御、およびその他固有の設定を含む、表示するテキストのプロパティを制御するさまざまなクラスが flash.text パッケージに用意されています。CSMSettings、Font、TextRenderer の各クラスを始めとするこれらのクラスとプロパティについては、『ActionScript 3.0 コンポーネントリファレンスガイド』で詳細に説明しています。

埋め込みフォントの使用

アプリケーションで TextField に対して特定のフォントを指定すると、Flash Player は同じ名前のデバイスフォント (ユーザーのコンピュータに存在するフォント) を検索します。そのようなフォントがユーザーのシステムに見つからない場合や、名前が同じでバージョンが若干異なるフォントが存在している場合には、テキストの表示が、意図したものとはかなり異なるものになることがあります。

意図したとおりに適切なフォントがユーザーに表示されるようにするには、アプリケーションの SWF ファイルにそのフォントを埋め込みます。埋め込みフォントにはいくつかの利点があります。

- 埋め込みフォントの文字はアンチエイリアス処理されるので、エッジが滑らかに見えます。テキストのサイズが大きい場合に特によくわかります。
- 埋め込みフォントを使用しているテキストは回転することができます。
- 埋め込みフォントを使用しているテキストは透明や半透明にすることができます。
- 埋め込みフォントには CSS の `kerning` スタイルを使用することができます。

埋め込みフォントを使用する場合の最大の制約は、そのためにアプリケーションのファイルサイズまたはダウンロードサイズが増加することです。

アプリケーションの SWF ファイルにフォントを埋め込むための正確な方法は、開発環境によって異なります。

フォントを埋め込んだ後は、次の作業を行って、TextField で正しい埋め込みフォントが使用されるようにします。

- TextField の `embedFonts` プロパティを `true` に設定します。
- TextFormat オブジェクトを作成し、そのオブジェクトの `fontFamily` プロパティに、埋め込みフォントの名前を設定します。その後で、作成した TextFormat オブジェクトを TextField に適用します。埋め込みフォントを指定するとき、`fontFamily` プロパティに格納する名前は 1 つだけにする必要があります。このプロパティでは、複数のフォント名を含むカンマ区切りリストを使用することはできません。
- CSS スタイルを使用して TextField やコンポーネントのフォントを設定している場合は、CSS プロパティ `font-family` に、埋め込みフォントの名前を設定します。埋め込みフォントを指定する場合、`font-family` プロパティに格納する名前は 1 つだけにする必要があります。名前のリストを格納しないでください。

Flash でのフォントの埋め込み

Flash オーサリングツールでは、TrueType フォントや Type 1 PostScript フォントを含め、システムにインストールされているほとんどすべてのフォントを埋め込むことができます。

Flex アプリケーションにフォントを埋め込む方法は数多くあります。たとえば、次のような方法です。

- ステージ上にある TextField のフォントとスタイルのプロパティを設定し、[フォントを埋め込む] チェックボックスをクリックします。
- フォントシンボルの作成と参照
- 埋め込みフォントシンボルを含むランタイム共有ライブラリの作成と使用

Flash アプリケーションでフォントを埋め込む方法の詳細については、『Flash ユーザーガイド』の「ダイナミックテキストフィールド用またはテキスト入力フィールド用の埋め込みフォント」を参照してください。

シャープネス、太さ、およびアンチエイリアスの制御

Flash Player では、テキストのサイズ変更、色の変更、またはさまざまな背景でテキストを表示する場合に、シャープネス、太さ、アンチエイリアスなどのテキスト表示制御の設定がデフォルトで決定されます。非常に小さなテキスト、非常に大きなテキスト、さまざまな独自の背景を使用するテキストを使用する場合など、これらの設定を独自に制御したい場合があります。flash.text.TextRenderer クラスとその関連クラス (CSMSettings クラスなど) を使用して、Flash Player の設定をオーバーライドすることができます。これらのクラスにより、埋め込みテキストのレンダリング品質を正確に制御することができます。埋め込みフォントの詳細については、501 ページの「埋め込みフォントの使用」を参照してください。

メモ

シャープネス、太さ、または gridFitType プロパティを設定し、あるいは TextRenderer.setAdvancedAntiAliasingTable() メソッドを使用するには、flash.text.TextField.antiAliasType プロパティに値 AntiAliasType.ADVANCED (デフォルト値) を指定する必要があります。

次の例では、myFont という名前の埋め込みフォントを使用している表示テキストに、カスタム CSM (Continuous Stroke Modulation) プロパティとフォーマットを適用します。ユーザーが表示テキストをクリックすると、カスタム設定が Flash Player によって適用されます。

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
  var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
  var myAliasTable:Array = new Array(myAntiAliasSettings);
  TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC,
  TextColorType.DARK_COLOR, myAliasTable);
}
```

静止テキストの操作

静止テキストは、Flash オーサリングツール内でのみ作成されます。ActionScript を使用して、プログラムによって静止テキストをインスタンス化することはできません。静止テキストは、テキストが非常に短く、(ダイナミックテキストと異なり) 変更が意図されない場合に便利です。Flash オーサリングツールのステージで描画された円または四角形のようなグラフィックエレメントとしての静止テキストについて考えてください。静止テキストはダイナミックテキストよりも制約が大きいので、ActionScript 3.0 は flash.text.StaticText クラスを使用して、静止テキストのプロパティ値を読み取る機能をサポートします。さらに、flash.text.TextSnapshot クラスを使用して、静止テキスト外の値を読み取ることもできます。

StaticText クラスによる静止テキストフィールドへのアクセス

ステージ上に配置された静止テキストインスタンスを操作するには、通常は Flash オーサリングツールの [アクション] パネルで flash.text.StaticText クラスを使用します。静止テキストを含む SWF ファイルとやり取りする ActionScript ファイルを操作することもできます。いずれの場合も、プログラムを通じて静止テキストインスタンスをインスタンス化することはできません。静止テキストは Flash CS3 オーサリングツールで作成されます。

ActionScript 3.0 で既存の静止テキストフィールドへの参照を作成するには、表示リスト内のアイテムに対して反復処理を行い、変数を割り当てます。次に実際の使用例を示します。

```
for (var i = 0; i < this.numChildren; i++) {
    var displayitem:DisplayObject = this.getChildAt(i);
    if (displayitem instanceof StaticText) {
        trace("a static text field is item " + i + " on the display list");
     var myFieldLabel:StaticText = StaticText(displayitem);
     trace("and contains the text: " + myFieldLabel.text);
    }
}
```

静止テキストフィールドへの参照が存在すると、ActionScript 3.0 でそのフィールドのプロパティを使用できます。次のコードがタイムラインのフレームに関連付けられ、myFieldLabel という名前の変数が静止テキスト参照に割り当てられていると仮定します。この例では、myField という名前のダイナミックテキストフィールドが myFieldLabel の値 x と y を基準に配置され、myFieldLabel の値を再表示します。

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

TextSnapshot クラスの使用

既存の静止テキストインスタンスをプログラムで操作する必要がある場合は、flash.text.TextSnapshot クラスを使用して、flash.display.DisplayObjectContainer の `textSnapshot` プロパティを操作します。これは、`DisplayObjectContainer.textSnapshot` プロパティから TextSnapshot インスタンスを作成することになります。次に、このインスタンスにメソッドを適用して、値を取得するか、静止テキストの部分を選択することができます。

たとえば、テキスト "TextSnapshot Example" を含む静止テキストフィールドをステージ上に配置することができます。次の ActionScript をタイムラインのフレーム 1 に追加します。

```
var mySnap:TextSnapshot = this.getTextSnapshot();
var count:Number = mySnap.getCount();
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

TextSnapshot クラスは、テキストをアプリケーションの別の部分の値として使用する場合に、ロードされた SWF の静止テキストフィールドからテキストを取得するときに便利です。

例 : 新聞形式のテキストフォーマット

ここで示す News Layout という例では、紙面版の新聞の記事と同様に見えるようにテキストをフォーマットします。入力テキストには記事のヘッドライン、サブタイトル、および本文を含めることができます。この News Layout という例は、表示領域の幅と高さが指定されると、表示領域の幅いっぱいに表示されるようにヘッドラインとサブタイトルをフォーマットします。記事の本文テキストは 2 段組みまたは 3 段組みで配置されます。

この例では、ActionScript の以下のプログラミング手法について説明します。

- TextField クラスの拡張
- 外部 CSS ファイルのロードと適用
- CSS スタイルから TextFormat オブジェクトへの変換
- TextLineMetrics クラスによるテキスト表示サイズ情報の取得

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。News Layout アプリケーションのファイルは、"Samples/NewsLayout" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
NewsLayout.mxml または NewsLayout.fla	アプリケーションのユーザーインターフェイスです (Flex 用の MXML、または Flash 用の FLA)。
StoryLayout.as	新しい記事のすべてのコンポーネントを表示用に配置する、メインの ActionScript クラスです。
FormattedTextField.as	TextField クラスのサブクラスであり、専用の TextFormat オブジェクトを管理します。
HeadlineTextField.as	FormattedTextField クラスのサブクラスであり、指定の幅に合わせてフォントサイズを調整します。
MultiColumnTextField.as	テキストを複数の段に分割する ActionScript クラスです。
story.css	レイアウトのテキストスタイルが定義されている CSS ファイルです。
newsconfig.xml	記事のコンテンツが含まれている XML ファイルです。

外部 CSS ファイルの読み取り

News Layout アプリケーションは、最初にローカル XML ファイルから記事のテキストを読み取ります。次に、ヘッドライン、サブタイトル、および本文テキストのフォーマット情報が含まれている外部 CSS ファイルを読み取ります。

CSS ファイルには 3 つのスタイルが定義されています。記事本文用の標準段落スタイル、およびヘッドラインとサブタイトルにそれぞれ対応する h1 スタイルと h2 スタイルです。

```
p {
   font-family: Georgia, Times New Roman, Times, _serif;
   font-size: 12;
   leading: 2;
   text-align: justify;
}

h1 {
   font-family: Verdana, Arial, Helvetica, _sans;
   font-size: 20;
   font-weight: bold;
   color: #000099;
```

```
  text-align: left;
}

h2 {
  font-family: Verdana, Arial, Helvetica, _sans;
  font-size: 16;
  font-weight: normal;
  text-align: left;
}
```

外部 CSS ファイルの読み取りに使用される手法は、499 ページの「外部 CSS ファイルのロード」で説明している手法と同じです。CSS ファイルのロードが完了すると、アプリケーションでは、次に示す onCSSFileLoaded() メソッドが実行されます。

```
public function onCSSFileLoaded(event:Event):void
{
  this.sheet = new StyleSheet();
  this.sheet.parseCSS(loader.data);

  h1Format = getTextStyle("h1", this.sheet);
  if (h1Format == null)
  {
    h1Format = getDefaultHeadFormat();
  }
  h2Format = getTextStyle("h2", this.sheet);
  if (h2Format == null)
  {
    h2Format = getDefaultHeadFormat();
    h2Format.size = 16;
  }
  displayStory();
}
```

onCSSFileLoaded() メソッドによって新しい StyleSheet オブジェクトが作成され、そのオブジェクトを使用して入力 CSS データが解析されます。記事の本文テキストは MultiColumnTextField オブジェクトに表示されます。このオブジェクトでは StyleSheet オブジェクトを直接使用できます。一方、ヘッドラインフィールドには HeadlineTextField クラスが使用されます。このクラスではフォーマットに TextFormat オブジェクトを使用します。

onCSSFileLoaded() メソッドは、getTextStyle() メソッドを 2 回呼び出して CSS スタイル宣言を TextFormat オブジェクトに変換し、2 つの HeadlineTextField オブジェクトのそれぞれで TextFormat オブジェクトを使用できるようにします。getTextStyle() メソッドを次に示します。

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
  var format:TextFormat = null;

  var style:Object = ss.getStyle(styleName);
  if (style != null)
  {
```

```
    var colorStr:String = style.color;
    if (colorStr != null && colorStr.indexOf("#") == 0)
    {
      style.color = colorStr.substr(1);
    }
    format = new TextFormat(style.fontFamily,
                  style.fontSize,
                  style.color,
                  (style.fontWeight == "bold"),
                  (style.fontStyle == "italic"),
                  (style.textDecoration == "underline"),
                  style.url,
                  style.target,
                  style.textAlign,
                  style.marginLeft,
                  style.marginRight,
                  style.textIndent,
                  style.leading);

    if (style.hasOwnProperty("letterSpacing"))
    {
      format.letterSpacing = style.letterSpacing;
    }
  }
  return format;
}
```

プロパティ名とプロパティ値の意味は、CSS スタイル宣言と TextFormat オブジェクトでは異なります。getTextStyle() メソッドは、CSS のプロパティ値を、TextFormat オブジェクトで想定されている値に変換します。

ページへの記事要素の配置

StoryLayout クラスでは、ヘッドライン、サブタイトル、および本文テキストのフィールドが、フォーマットされ、レイアウトされて、新聞形式で配置されます。まず、displayText() メソッドによって各フィールドが作成され、配置されます。

```
public function displayText():void
{
  headlineTxt = new HeadlineTextField(h1Format);
  headlineTxt.wordWrap = true;
  this.addChild(headlineTxt);
  headlineTxt.width = 600;
  headlineTxt.height = 100;
  headlineTxt.fitText(this.headline, 1, true);

  subtitleTxt = new HeadlineTextField(h2Format);
  subtitleTxt.wordWrap = true;
  subtitleTxt.y = headlineTxt.y + headlineTxt.height;
```

```
  this.addChild(subtitleTxt);
  subtitleTxt.width = 600;
  subtitleTxt.height = 100;
  subtitleTxt.fitText(this.subtitle, 1, false);

  storyTxt = new MultiColumnTextField(2, 10, 600, 200);
  storyTxt.y = subtitleTxt.y + subtitleTxt.height + 4;
  this.addChild(storyTxt);
  storyTxt.styleSheet = this.sheet;
  storyTxt.htmlText = loremIpsum;
}
```

各フィールドは、単純にその直前のフィールドの下に配置されます。このように配置するため、各フィールドの y プロパティは、直前のフィールドの y プロパティに直前のフィールドの高さを加えた値と等しくなるように設定されます。このような動的配置のための計算が必要になるのは、HeadlineTextField オブジェクトと MultiColumnTextField オブジェクトの高さが、それぞれのコンテンツに合わせて変化するためです。

フィールドサイズに合わせるためのフォントサイズ変更

HeadlineTextField では、幅 (ピクセル単位) と最大表示行数が指定されると、テキストがフィールドいっぱいに表示されるようにフォントサイズが変更されます。テキストが短い場合には、フォントサイズが非常に大きくなるため、タブロイドスタイルのヘッドラインが作成されます。テキストが長い場合には、当然ながらフォントサイズが小さくなります。

次に示す HeadlineTextField.fitText() メソッドで、フォントサイズの変更処理が実行されます。

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean = false,
  targetWidth:Number = -1):uint
{
  this.text = toUpper ? msg.toUpperCase() : msg;

  if (targetWidth == -1)
  {
    targetWidth = this.width;
  }

  var pixelsPerChar:Number = targetWidth / msg.length;

  var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar * 1.8
  * maxLines));

  if (pointSize < 6)
  {
    // ポイントサイズが小さすぎます。
    return pointSize;
  }
```

```
    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(--pointSize, maxLines);
    }
    else
    {
        return growText(pointSize, maxLines);
    }
}

public function growText(pointSize:Number, maxLines:uint = 1):Number
{
    if (pointSize >= MAX_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
        // 直前のサイズに戻します。
        this.changeSize(pointSize);
        return pointSize;
    }
    else
    {
        return growText(pointSize + 1, maxLines);
    }
}

public function shrinkText(pointSize:Number, maxLines:uint=1):Number
{
    if (pointSize <= MIN_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
        return pointSize;
    }
}
```

HeadlineTextField.fitText() メソッドでは、単純な再帰的手法によってフォントサイズが変更されます。最初にテキストの1文字あたりの平均ピクセル数が推定され、それに基づいて開始ポイントサイズが算出されます。次に、テキストフィールドのフォントサイズが変更されます。その後、テキストが折り返されたされたためにテキスト行の最大数を超えて行が生成されていないかどうかが確認されます。行数が多すぎる場合には、shrinkText() メソッドが呼び出されて、フォントサイズを小さくする操作が何回か試行されます。最大行数を超えていない場合には、growText() メソッドが呼び出されて、フォントサイズを大きくする操作が何回か試行されます。フォントサイズを現在の値よりも1ポイント大きくすると最大行数を超えるような状態に達したときに、処理が終了します。

複数の段へのテキストの分割

MultiColumnTextField クラスは、複数の TextField オブジェクトにテキストを配分します。これらのオブジェクトは、後で新聞の段組みのように配置されます。

まず、MultiColumnTextField() コンストラクタによって、次のように TextField オブジェクトの配列が作成されます。1つのオブジェクトが1つの段に対応します。

```
for (var i:int = 0; i < cols; i++)
{
   var field:TextField = new TextField();
   field.autoSize = TextFieldAutoSize.NONE;
   field.wordWrap = true;
   field.styleSheet = this.styleSheet;

   this.fieldArray.push(field);
   this.addChild(field);
}
```

各 TextField オブジェクトは配列に追加され、さらに addChild() メソッドによって表示リストに追加されます。

StoryLayout オブジェクトの text プロパティまたは styleSheet プロパティが変更されると、layoutColumns() メソッドが呼び出されて、テキストが再表示されます。layoutColumns() メソッドは、次に示す getOptimalHeight() メソッドを呼び出して、指定されたレイアウト幅に合わせてすべてのテキストを配置するために必要な高さを正確にピクセル単位で算出します。

```
public function getOptimalHeight(str:String):int
{
   if (fieldArray.length == 0 || str == "" || str == null)
   {
      return this.preferredHeight;
   }
   else
   {
      var colWidth:int = Math.floor( (this.preferredWidth -
         ((this.numColumns - 1) * gutter)) / this.numColumns);
```

```
    var field:TextField = fieldArray[0] as TextField;
    field.width = colWidth;
    field.htmlText = str;

    var linesPerCol:int = Math.ceil(field.numLines / this.numColumns);
    var metrics:TextLineMetrics = field.getLineMetrics(0);
    var prefHeight:int = linesPerCol * metrics.height;
    return prefHeight + 4;
  }
}
```

getOptimalHeight() メソッドでは、まず各段の幅が計算されます。次に、配列の最初の TextField オブジェクトの幅と htmlText プロパティが設定されます。getOptimalHeight() メソッドでは、この最初の TextField オブジェクトを使用してテキストの折り返された総行数を調べ、その総行数に基づいて、各段に配分する行数を特定します。続いて、最初の行のテキストのサイズの詳細情報を含む TextLineMetrics オブジェクトを取得するために、TextField.getLineMetrics() メソッドが呼び出されます。TextLineMetrics.height プロパティは、アセント、ディセント、および行送りを含めたテキスト行全体の高さを表します (ピクセル単位)。MultiColumnTextField オブジェクトの最適な高さは、この行全体の高さと 1 段の行数を乗算し、さらに 4 ピクセルを加算したものになります。この 4 ピクセルは、TextField オブジェクトの上下それぞれに設ける 2 ピクセルの境界の分です。

layoutColumns() メソッド全体のコードを次に示します。

```
public function layoutColumns():void
{
  if (this._text == "" || this._text == null)
  {
    return;
  }

  if (this.fitToText)
  {
    this.preferredHeight = this.getOptimalHeight(this._text);
  }

  var colWidth:int = Math.floor( (this.preferredWidth -
    ((numColumns - 1) * gutter)) / numColumns);
  var field:TextField;
  var remainder:String = this._text;
  var fieldText:String = "";

  for (var i:int = 0; i < fieldArray.length; i++)
  {
    field = this.fieldArray[i] as TextField;
    field.width = colWidth;
    field.height = this.preferredHeight;
```

```
    field.x = i * (colWidth + gutter);
    field.y = 0;

    field.htmlText = "<p>" + remainder + "</p>";

    remainder = "";
    fieldText = "";

    var linesRemaining:int = field.numLines;
    var linesVisible:int = field.numLines - field.maxScrollV + 1;
    for (var j:int = 0; j < linesRemaining; j++)
    {
      if (j < linesVisible)
      {
        fieldText += field.getLineText(j);
      }
      else
      {
        remainder +=  field.getLineText(j);
      }
    }

    field.htmlText = "<p>" + fieldText + "</p>";
  }
}
```

getOptimalHeight() メソッドの呼び出しによって preferredHeight プロパティが設定された後、layoutColumns() メソッドでは TextField オブジェクトが反復処理され、各オブジェクトの高さには preferredHeight の値が設定されます。続いて、layoutColumns() メソッドによって、各フィールドにちょうど収まる行数のテキストが配分されます。このとき、個々のフィールドでスクロールが発生することがなく、また、各後続フィールドのテキストは直前のフィールドでテキストが終わった箇所から始まるように配分されます。

第 18 章

ビットマップの操作

18

ベクター描画機能に加え、ActionScript 3.0 にはビットマップイメージの作成や、SWF にロードされた外部ビットマップイメージのピクセルデータの操作を可能にする機能があります。個々のピクセル値にアクセスし、変更できる機能を利用すると、フィルタのような独自のイメージ効果を作成し、ビルトインのノイズ関数を使用してテクスチャやランダムノイズを作成することができます。これらの技法については、この章で解説します。

目次

ビットマップの操作の基礎 516
Bitmap および BitmapData クラス 519
ピクセルの操作 521
ビットマップデータのコピー 524
ノイズ関数によるテクスチャの作成 525
ビットマップのスクロール 527
例 : 画面外ビットマップを使用してスプライトをアニメーション化する 528

ビットマップの操作の基礎

ビットマップの操作の概要

デジタルイメージを操作する際、主なグラフィックタイプ、つまりビットマップとベクターを取り扱います。ビットマップグラフィックは、ラスターグラフィックとも呼ばれ、矩形のグリッドに配列された小さな正方形、つまりピクセルによって構成されます。ベクターグラフィックは、直線、曲線、ポリゴンなど、数学的に生成された幾何学図形によって構成されます。

ビットマップイメージはイメージの幅と高さによって定義され、ピクセル数と 1 ピクセルあたりのビット数、つまり 1 ピクセルあたりの色数によって測定されます。RGB カラーモデルを使用するビットマップイメージの場合、ピクセルは赤、緑、青の 3 バイトで構成されます。各バイトは、0 から 255 までの値になります。ピクセルの中でバイトを組み合わせることにより、画家が絵の具を混ぜ合わせるのと同じような形で色が生成されます。たとえば、ピクセルのバイト値が赤 255、緑 102、青 0 であれば、鮮やかなオレンジ色が生成されます。

ビットマップイメージの品質は、イメージの解像度と、カラー深度のビット値を組み合わせることによって決定されます。解像度とは、1 つのイメージに含まれるピクセル数を指します。ピクセル数が大きいほど解像度が高くなり、イメージが精細に表示されます。カラー深度とは、1 つのピクセルが格納できる情報量を指します。たとえば、カラー深度値がピクセルあたり 16 ビットのイメージは、カラー深度が 48 ビットのイメージと同じ数の色を表現することはできません。その結果、48 ビットのイメージの方が 16 ビットのイメージよりもスムーズな階調を示すことができます。

ビットマップグラフィックは解像度に依存するので、効果的に拡大 / 縮小することができません。これは、ビットマップを拡大すると特によく分かります。ビットマップイメージを拡大すると細部が失われ、品質が劣化します。

ビットマップファイル形式

ビットマップイメージは、いくつかの一般的なファイル形式に分けられます。これらの形式は、それぞれに異なる圧縮アルゴリズムによってファイルサイズを縮小すると共に、イメージの最終用途に応じて画質を最適化します。Adobe Flash Player が対応しているビットマップイメージ形式は GIF、JPG、および PNG です。

GIF

Graphics Interchange Format (GIF) は、256 色、つまり 8 ビットカラーのイメージを伝送するために、1987 年に CompuServe 社によって開発されました。この形式はファイルサイズが小さいので、Web ベースのイメージには最適です。GIF のカラーパレットには限界があるので、GIF イメージは通常は写真には向いていません。写真は、細かな階調やカラーグラデーションを必要とするからです。GIF イメージは単独ビットの透明度にも対応しているので、カラーを透明としてマッピングすることもできます。これにより、透明度がマッピングされたイメージを透過して Web ページの背景を表示することができます。

JPEG

Joint Photographic Experts Group (JPEG) によって開発された JPEG (または JPG) イメージ形式は非可逆圧縮アルゴリズムを使用し、小さいファイルサイズで 24 ビットカラー深度を実現することができます。非可逆圧縮とは、イメージが保存されるたびに品質とデータが失われますが、ファイルサイズを小さく抑えられることです。JPEG 形式では何百万もの色を表示できるので、写真に最適です。イメージに適用される圧縮の度合いを制御できるので、イメージの品質とファイルサイズを操作することができます。

PNG

Portable Network Graphics (PNG) 形式は、特許を取得した GIF ファイル形式のオープンソース代替形式として開発されました。PNG は最大で 64 ビットのカラー深度に対応しているので、最大で 1,600 万色を表示することができます。PNG は比較的新しい形式なので、古いブラウザの中には PNG ファイルをサポートしていないものもあります。JPG とは違い、PGN は可逆圧縮を使用します。つまり、イメージが保存されても、イメージデータは失われません。PNG ファイルは、最大で 256 レベルの透明度に対応する、アルファ透明度もサポートしています。

透明なビットマップと不透明なビットマップ

GIF または PNG 形式を使用するビットマップイメージでは、各ピクセルに追加バイトとしてアルファチャンネルが付与されます。この追加バイトは、ピクセルの透明度の値を表します。

GIF イメージは単独ビットの透明度に対応しているので、256 色のカラーパレットの中から 1 色を指定し、その色を透明にすることができます。一方 PNG イメージでは、最大で 256 レベルの透明度に対応しています。この機能は、イメージやテキストを背景と混ぜ合わせる場合に特に有効です。

ActionScript 3.0 は、追加された透明度ピクセルバイトを BitmapData クラス内に複製します。PNG 透明度モデルと同様に、`BitmapDataChannel.ALPHA` 定数は最大で 256 レベルの透明度を実現することができます。

ビットマップの操作に関する一般的なタスク

次のリストは、ActionScript でビットマップイメージを操作する際に実行するいくつかのタスクを示します。

- 画面へのビットマップの表示
- ピクセルカラー値の取得と設定
- ビットマップデータのコピー
  - ビットマップの同一コピーを作成します。
  - ビットマップの 1 カラーチャンネルのデータを、別のビットマップの 1 カラーチャンネルにコピーします。
  - 画面表示オブジェクトのスナップショットをビットマップにコピーします。
- ノイズとテクスチャのビットマップイメージへの作成
- ビットマップのスクロール

重要な概念と用語

次に、この章に記載される重要な用語を示します。

- アルファ : 色またはイメージの透明度 (正確には不透明度) のレベルです。アルファの量は、アルファチャンネル値として記されます。
- ARGB カラー : 各ピクセルの色を、赤、緑、青を混合したカラー値として示すカラースキームです。透明度はアルファ値として指定されます。
- カラーチャンネル : 通常、色は基本色を混ぜ合わせることで表示されます。コンピュータグラフィックでは、通常は赤、緑、青です。基本色はそれぞれカラーチャンネルとして見なされます。各カラーチャンネルが混ぜ合わされる量により、最終的な色が決定されます。
- カラー深度 : ビット深度とも呼ばれます。1 ピクセルに使用されるコンピュータメモリを示します。つまり、イメージで表示できる色の数を決定します。
- ピクセル : ビットマップイメージの最小情報単位、つまり色のドットです。
- 解像度 : イメージのピクセルサイズで、これによりイメージの詳細さが決定します。解像度は、ピクセルの幅と高さによって表現されます。
- RGB カラー : 各ピクセルの色が赤、緑、青のカラー値で表示されるカラースキームです。

本章の例について

本章をお読みの際は、コード例を試すことができます。この章では視覚的なコンテンツの作成と操作について扱っているため、コードのテストには、コードを実行し、作成された SWF の結果を表示することが含まれます。

この章のコード例をテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインでキーフレームを選択します。
3. [アクション] パネルを開き、[スクリプト] ペインにコードをコピーします。
4. [コントロール]-[ムービープレビュー] を使用して、プログラムを実行します。

   作成された SWF ファイルで、コードの結果が表示されます。

ほぼすべてのコード例にビットマップイメージを作成するコードが含まれています。そのため、それらのコードはビットマップコンテンツを提供しなくても直接テストできます。独自のイメージに対してコードをテストする場合は、そのイメージを Adobe Flash CS3 Professional にインポートするか、その外部イメージをテスト SWF にロードして、そのビットマップデータをコード例で使うことができます。外部イメージをロードする手順については、400 ページの「表示コンテンツの動的ロード」を参照してください。

Bitmap および BitmapData クラス

ビットマップイメージを操作するための主な ActionScript 3.0 クラスには、画面上にビットマップイメージを表示する Bitmap クラスと、ビットマップの生イメージデータにアクセスし、操作するための BitmapData クラスがあります。

Bitmap クラスについて

DisplayObject クラスのサブクラスである Bitmap クラスは、ビットマップイメージを表示するために使用される主な ActionScript 3.0 クラスです。これらのイメージは、flash.display.Loader クラスによって Flash にロードされているか、`Bitmap()` コンストラクタによって動的に作成されています。外部ソースからイメージをロードする場合、Bitmap オブジェクトは GIF、JPEG、または PNG 形式のイメージしか使用できません。Bitmap インスタンスは BitmapData オブジェクトのラッパーとして見なされ、ステージにレンダリングする必要があります。Bitmap インスタンスは表示オブジェクトなので、表示オブジェクトのすべての性質と機能を活用して Bitmap インスタンスを操作することもできます。表示オブジェクトの操作方法の詳細については、353 ページ、第 12 章の「表示のプログラミング」を参照してください。

ピクセルの吸着とスムージング

すべての表示オブジェクトに共通する機能に加え、Bitmap クラスはビットマップイメージに特有の機能も提供します。

Flash オーサリングツールのピクセルにスナップする機能と同様、Bitmap クラスの pixelSnapping プロパティは、Bitmap オブジェクトが最も近いピクセルに吸着されるかどうかを指定します。このプロパティは、PixelSnapping クラスで定義されている 3 つの定数、ALWAYS、AUTO、NEVER のいずれかを受け入れます。

ピクセルの吸着を適用するシンタックスは次のとおりです。

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

ビットマップイメージが拡大 / 縮小されると、ぼやけたり歪んだりすることが多々あります。この歪みを抑えるため、BitmapData クラスの smoothing プロパティを使用します。このブール型プロパティを true に設定すると、拡大 / 縮小されたイメージ内のピクセルをスムージングまたはアンチエイリアス化します。これにより、イメージをより鮮明に、より自然に表現できるようになります。

BitmapData クラスについて

BitmapData クラスは flash.display パッケージにあり、ロードされたビットマップイメージや動的に作成されたビットマップイメージ内のピクセルのスナップショットのようなものです。このスナップショットは、オブジェクト内のピクセルデータの配列として表されます。BitmapData クラスには、ピクセルデータの作成と操作に便利なビルトインメソッドもあります。

BitmapData オブジェクトをインスタンス化するには、次のコードを使用します。

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number,
  transparent:Boolean, fillColor:uinit);
```

width および height パラメータはビットマップのサイズを指定し、そのいずれも最大値は 2880 ピクセルです。transparent パラメータは、ビットマップデータにアルファチャンネルが含まれるかどうかを指定します。含まれる場合は true、含まれない場合は false です。fillColor パラメータは、背景色を指定する 32 ビットのカラー値です。true に設定されている場合は、透明度の値も指定します。次の例では、透明度が 50% に設定されたオレンジ色の背景の BitmapData オブジェクトを作成します。

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

新たに作成された BitmapData オブジェクトを画面にレンダリングするには、このオブジェクトを Bitmap インスタンスに割り当てるか、ラップします。そのためには、Bitmap オブジェクトのコストラクタのパラメータとして BitmapData オブジェクトを渡すか、BitmapData オブジェクトを既存の Bitmap インスタンスの bitmapData プロパティに割り当てます。また、Bitmap インスタンスを含む表示オブジェクトのコンテナの addChild() または addChildAt() メソッドを呼び出し、Bitmap インスタンスを表示リストに追加する必要もあります。表示リストの操作方法の詳細については、364 ページの「表示リストへの表示オブジェクトの追加」を参照してください。

次の例では、赤く塗りつぶした BitmapData オブジェクトを作成し、Bitmap インスタンスに表示します。

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

ピクセルの操作

BitmapData クラスには、ピクセルデータ値を操作するための一連のメソッドがあります。

個々のピクセルの操作

ピクセルレベルでビットマップイメージの外観を変更するには、最初に、操作対象の領域にあるピクセルのカラー値を取得する必要があります。これらのピクセルの値を読み取るには、getPixel() メソッドを使用します。

getPixel() メソッドは、パラメータとして渡される x, y (ピクセル) 座標から RGB 値を取得します。操作対象のピクセルのいずれかに透明度、つまりアルファチャンネルの情報が含まれている場合は、getPixel32() メソッドを使用する必要があります。このメソッドは RGB 値も取得しますが、getPixel() とは異なり、getPixel32() によって返される値には、選択されたピクセルのアルファチャンネル (透明度) 値を示す追加データがあります。

または、ビットマップに含まれるピクセルの色または透明度だけを変更するのであれば、setPixel() または setPixel32() メソッドを使用します。ピクセルの色を設定するには、いずれかのメソッドに x, y 座標とカラー値を渡すだけです。

次の例では、setPixel() を使用して、緑の BitmapData 背景に十字形を描きます。その後、getPixel() を使用して、座標値が 50, 50 のピクセルからカラー値を取得し、返された値をトレースします。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
   var red:uint = 0xFF0000;
   myBitmapData.setPixel(50, i, red);
   myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);
```

```
var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

単一ピクセルではなく、ピクセルのグループの値を読み取りたい場合は、getPixels() メソッドを使用します。このメソッドは、パラメータとして渡されるピクセルデータの矩形領域からバイト配列を生成します。バイト配列の各エレメント、つまりピクセル値は、符号なしの整数、つまり 32 ビットの乗算されていないピクセル値です。

反対に、ピクセルのグループの値を変更または設定するには、setPixels() メソッドを使用します。このメソッドは rect および inputByteArray の 2 つのパラメータを想定しています。これらのパラメータは、ピクセルデータ (inputByteArray) の矩形領域 (rect) を出力するために組み合わされます。

データが inputByteArray から読み取られ、書き込まれると、配列内の各ピクセルに対して ByteArray.readUnsignedInt() メソッドが呼び出されます。何らかの理由により、inputByteArray に矩形全体に相応するピクセルデータが含まれていない場合は、メソッドはその段階でイメージデータの処理を停止します。

重要な点は、ピクセルデータの取得と設定では、バイト配列は 32 ビットのアルファ、赤、緑、青 (ARGB) ピクセル値を想定していることです。

次の例では getPixels() および setPixels() メソッドを使用し、BitmapData オブジェクト間でピクセルのグループをコピーします。

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false, 0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false, 0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

ピクセルレベルの衝突検出

BitmapData.hitTest() メソッドは、ビットマップデータと別のオブジェクトまたはポイント間でピクセルレベルの衝突を検出します。

BitmapData.hitTest() メソッドは、次の 5 つのパラメータを受け入れます。

- firstPoint (ポイント): このパラメータは、最初の BitmapData の左上のピクセル位置であり、ヒットテストが実行される場所を指します。
- firstAlphaThreshold (uint) : このパラメータは、このヒットテストで不透明と見なされるアルファチャンネルの最大値です。
- secondObject (オブジェクト): このパラメータは、影響を受ける領域を示します。secondObject オブジェクトは、Rectangle、Point、Bitmap、または BitmapData オブジェクトです。このオブジェクトは、衝突検出が実行されるヒット領域を指します。
- secondBitmapDataPoint (ポイント): このパラメータはオプションであり、2 つめの BitmapData オブジェクトのピクセル位置を指定します。このパラメータは、secondObject の値が BitmapData オブジェクトである場合にのみ使用します。デフォルト値は null です。
- secondAlphaThreshold (uint) : このパラメータはオプションであり、2 つめの BitmapData オブジェクトで不透明と見なされるアルファチャンネルの最大値です。デフォルト値は 1 です。このパラメータは、secondObject が BitmapData オブジェクトであり、両方の BitmapData オブジェクトが透明である場合にのみ使用されます。

不透明なイメージで衝突を検出する場合、ActionScript はイメージを完全に不透明な矩形または境界ボックスとして扱います。または、透明なイメージでピクセルレベルのヒットテストを実行する場合は、両方のイメージが透明である必要があります。これに加え、ActionScript ではアルファしきい値パラメータを使用し、ピクセルがどの段階で透明から不透明に変わるかを決定します。

次の例では、3 つのビットマップイメージを作成し、false を返す衝突ポイントと true を返す衝突ポイントを使用してピクセル衝突を確認します。

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
```

```
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;

// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

ビットマップデータのコピー

イメージ間でビットマップデータをコピーするには、clone()、copyPixels()、copyChannel()、および draw() といったメソッドを使用できます。

その名のとおり、clone() メソッドを使用すると、BitmapData オブジェクト間でビットマップデータのクローンの作成やサンプリングを実行できます。このメソッドが呼び出されると、コピー元のインスタンスの同一クローンである新しい BitmapData オブジェクトが返されます。

次の例では、親であるオレンジ色の正方形のクローンを作成し、そのクローンを元の親正方形の横に配置します。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false,
  0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

copyPixels() メソッドは、BitmapData オブジェクト間でピクセルをすばやく、かつ簡単に作成できます。このメソッドは、sourceRect パラメータで定義された、元のイメージの矩形のスナップショットを取り出し、それを同一サイズの別の矩形領域にコピーします。新たに " ペーストされた " 矩形の場所は、destPoint パラメータ内で指定されます。

copyChannel() メソッドは、元の BitmapData オブジェクトから定義済みのカラーチャンネル値、つまりアルファ、赤、緑、または青をサンプリングし、それをターゲット BitmapData オブジェクトのチャンネルにコピーします。このメソッドを呼び出しても、ターゲットの BitmapData オブジェクトの他のチャンネルには影響しません。

draw() メソッドは、元のスプライト、ムービークリップ、またはその他の表示オブジェクトのグラフィックコンテンツを、新しいビットマップに描画またはレンダリングします。matrix、colorTransform、blendMode、およびターゲットの clipRect パラメータを使用すると、新しいビットマップのレンダリング方法を変更することができます。このメソッドでは、Flash Player ベクターレンダラを使用してデータを生成します。

draw() を呼び出すと、次のように、スプライト、ムービークリップ、またはその他の表示オブジェクトといったソースオブジェクトを最初のパラメータとして渡します。

```
myBitmap.draw(movieClip);
```

ソースオブジェクトが最初にロードされてから、色やマトリックスなどが変換されている場合は、変換内容は新しいオブジェクトにコピーされません。新しいビットマップに変換内容をコピーするには、元のオブジェクトの transform プロパティの値を、新しい BitmapData オブジェクトを使用する Bitmap オブジェクトの transform プロパティにコピーする必要があります。

ノイズ関数によるテクスチャの作成

ビットマップの外観を変えるには、noise() メソッドまたは perlinNoise() メソッドのいずれかを使用してノイズ効果を適用します。ノイズ効果とは、チューニングされていないテレビ画面に表示されるノイズのようなものです。

ビットマップにノイズ効果を適用するには、noise() メソッドを使用します。このメソッドは、ビットマップイメージの指定領域内のピクセルに対し、ランダムカラー値を適用します。

このメソッドは、次の 5 つのパラメータを受け入れます。

- randomSeed (int) : パターンを決定するランダムシード数です。その名に反し、同じ数が渡されると実際には同じ結果が生成されます。真にランダムな結果を得るには、Math.random() メソッドを使用し、このパラメータに乱数を渡します。
- low (uint) : このパラメータは、0 から 255 までの各ピクセルに対して生成される最小値を示します。デフォルト値は 0 です。この値を低く設定すると暗いノイズパターンになり、高い値に設定すると明るいパターンになります。
- high (uint) : このパラメータは、0 から 255 までの各ピクセルに対して生成される最高値を示します。デフォルト値は 255 です。この値を低く設定すると暗いノイズパターンになり、高い値に設定すると明るいパターンになります。

- channelOptions (uint) : このパラメータは、ノイズパターンを適用する、ビットマップオブジェクトのカラーチャンネルを指定します。値は、4 つの ARGB カラーチャンネル値の任意の組み合わせです。デフォルト値は 7 です。
- grayScale (ブール値) : true に設定されると、ビットマップピクセルに randomSeed 値が適用され、イメージからすべてのカラーが効果的にウォッシュアウトされます。このパラメータはアルファチャンネルに影響しません。デフォルト値は false です。

次の例では、ビットマップイメージを作成し、青いノイズパターンを適用します。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
var image:Bitmap = new Bitmap(myBitmap);
addChild(image);
```

より有機的な外観のテクスチャを作成するには、perlinNoise() メソッドを使用します。

perlinNoise() メソッドは、煙、雲、水、火、あるいは爆発といった効果に最適な、リアルで有機的なテクスチャを生成します。

perlinNoise() メソッドはアルゴリズムによって生成されるので、ビットマップベースのテクスチャよりもメモリ消費量を抑えられます。ただし、プロセッサの使用に影響を与えるので、Flash コンテンツを低速化し、特に古いコンピュータではフレームレートよりも画面の再描画速度が遅くなります。これは主に、Perlin ノイズアルゴリズムの処理に必要な浮動小数点計算によるものです。

このメソッドは、次の 9 つのパラメータを受け入れます。最初の 6 つのパラメータは必須です。

- baseX (数値) : 作成されるパターンの x (サイズ) 値を指定します。
- baseY (数値) : 作成されるパターンの y (サイズ) 値を指定します。
- numOctaves (unit) : このノイズを作成するために組み合わせるオクターブ (つまり個々のノイズ関数) の数です。オクターブ数を多くすると、よりきめ細かいイメージを作成できますが、処理時間も長くなります。
- randomSeed (int) : ランダムシードの数値の働きは、noise() 関数での働きと同じです。真にランダムな結果を得るには、Math.random() メソッドを使用し、このパラメータに乱数を渡します。
- stitch (ブール値) : true に設定されると、このメソッドはイメージのトランジションエッジをスティッチ (またはスムージング) してシームレスなテクスチャを作成し、ビットマップ塗りとして分割します。
- fractalNoise (ブール値) : このパラメータは、メソッドによって生成されたグラデーションのエッジに関連するものです。true に設定されていると、効果のエッジをスムージングするフラクタルノイズを生成します。false に設定すると、乱流を生成します。乱流があるイメージにはグラデーションに視覚的な不連続性があるので、炎や海の波のようなシャープな視覚効果に適している場合があります。

- channelOptions (uint) : channelOptions パラメータの働きは、noise() メソッドでの働きと同じです。ノイズパターンが適用される、ビットマップのカラーチャンネルを指定します。値は、4 つの ARGB カラーチャンネル値の任意の組み合わせです。デフォルト値は 7 です。
- grayScale (ブール値) : grayScale パラメータの働きは、noise() メソッドでの働きと同じです。true に設定されていると、ビットマップピクセルに randomSeed 値が適用され、イメージからすべての色が効果的にウォッシュアウトされます。デフォルト値は false です。
- offsets (配列) : 各オクターブの x オフセットと y オフセットに対応するポイントの配列です。オフセット値を操作することで、イメージのレイヤーをスムーズにスクロールできます。オフセット配列内の各ポイントは、特定のオクターブノイズ関数に影響を与えます。デフォルト値は null です。

次の例では、perlinNoise() メソッドを呼び出して緑と青の雲効果を生成する、150 x 150 ピクセルの BitmapData オブジェクトを作成します。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0x00FF0000);

var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false,
  null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myBitmap);
```

ビットマップのスクロール

ユーザーが地図を動かすたびに、その移動幅がわずか数ピクセルであったとしても、ビューを更新しなければならないようなストリートマッピングアプリケーションを作成したとします。

この機能を実現するには、ユーザーが地図を動かすたびに、更新された部分を含む新しいイメージを再レンダリングするという方法があります。または、1 つの大きなイメージと、scroll() メソッドを作成する方法もあります。

scroll() メソッドは画面上のビットマップをコピーし、それを x, y パラメータで指定する新しいオフセット位置にペーストします。ビットマップの一部がステージ外にあると、イメージがシフトされた効果をもたらします。タイマー制御関数または enterFrame イベントと組み合わせると、イメージがアニメーション化またはスクロールしているように見せることができます。

次の例では、前述の Perlin ノイズの例から大きなビットマップイメージを作成します。イメージの 4 分の 3 がステージ外でレンダリングされます。scroll() メソッドと、斜め下方向に 1 ピクセル分イメージをオフセットする enterFrame イベントリスナーが適用されます。このメソッドはフレームが入力されるたびに呼び出されるので、画面外にあるイメージは、イメージが下方向にスクロールされるとステージにレンダリングされます。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false,
  0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false,
  null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);

function scrollBitmap(event:Event):void
{
  myBitmapDataObject.scroll(1, 1);
}
```

例 : 画面外ビットマップを使用してスプライトをアニメーション化する

Flash ゲームでは、一度に数百ものイメージが画面上で動いています。このビットマップアニメーションの例では、数百個もの小さなビットマップまたはスプライトを、大きな画面外ビットマップに描画してから、その 1 枚のビットマップを画面に書き込みます。これによってアニメーションを大幅に高速化できます。この例に関する説明と、ソースコードのダウンロードについては、http://www.adobe.com/go/learn_fl_bitmaps_jp を参照してください。

第 19 章

ビデオの操作

19

Flash Video は、インターネットで使用可能な優れた技術の 1 つです。Flash アプリケーションのビデオでは、矩形の画面の下に、プログラスバーとコントロールボタンがついている、従来のビデオ画面以外の表示方法も可能です。ActionScript を使用すると、ビデオのロード、表示、再生を細部にわたって調節し、制御することができます。

目次

ビデオの基礎 530
FLV 形式について 533
Video クラスについて 534
ビデオファイルのロード 534
ビデオ再生の制御 535
ビデオファイルのストリーミング 538
キューポイントについて 538
onCuePoint および onMetaData のコールバックメソッドの記述 539
キューポイントの使用 545
ビデオメタデータの使用 546
カメラ入力のキャプチャ 549
高度なトピック 557
例 : ビデオジュークボックス 559

ビデオの基礎

ビデオの操作の概要

Adobe Flash Player には、イメージやアニメーション、テキストといったその他の視覚的なコンテンツと同様に、ActionScript でビデオ情報を表示し、操作することができるという重要な機能があります。

Adobe Flash CS3 Professional で FLV (Flash Vide) ファイルを作成する場合、共通の再生コントロールなど、ビデオのスキンを選択することができます。しかし、ここで提供されるオプショしか使えないわけではありません。ActionScript を使用すると、ビデオのロード、表示、および再生の制御を細部にわたって調節することができます。つまり、独自のビデオプレーヤースキンを作成、または新しい方法でビデオを使用することができます。

ActionScript でビデオを操作する場合は、いくつかのクラスを組み合わせます。

- Video クラス : ステージ上の実際のビデオコンテンツボックスは、Video クラスのインスタンスです。Video クラスは表示オブジェクトです。そのため、配置や変換、フィルターの適用、ブレンドモードなど、他の表示オブジェクトにも適用される技法を使用して操作することができます。
- NetStream クラス : ActionScript で制御するビデオファイルをロードするとき、NetStream インスタンスを使用し、ビデオコンテンツのソースを示します。この場合、ビデオデータのストリームです。NetStream インスタンスを使用すると、NetConnection オブジェクトも使用することになります。NetConnection オブジェクトとは、ビデオデータが送信されるトンネルのように、ビデオファイルに接続するために使用されます。
- Camera クラス : ユーザーのコンピュータに接続されたカメラからビデオデータを操作するとき、Camera インスタンスはビデオコンテンツのソース、つまり、ユーザーのカメラとビデオデータを示します。

外部ビデオをロードする場合、標準的な Web サーバーからファイルをロードしてプログレッシブダウンロード再生を行うか、アドビ システムズ社の Macromedia® Flash® Media Server などの専用のサーバーから配信されるストリーミングビデオを操作することができます。

ビデオに関する一般的なタスク

この章では、ビデオに関連して実行する次のタスクについて説明します。

- 画面上でのビデオの表示と制御
- 外部 FLV ファイルのロード
- ビデオファイルのメタデータとキューポイント情報の取り扱い
- ユーザーのカメラからのビデオ入力のキャプチャと表示

重要な概念と用語

- キューポイント：ビデオファイルの特定の時点に配置できるマーカーです。その時点を探すときのブックマークとして、またはその時点に関連するデータを追加するときなどに使用できます。
- エンコード : ある形式のビデオデータを、別のビデオデータ形式に変換することです。たとえば、高解像度のソースビデオを、インターネット配信に適した形式に変換するなどが挙げられます。
- フレーム : ビデオ情報のセグメント。1 枚のフレームは、ビデオの特定の時点のスナップショットを表す静止画のようなものです。フレームを連続して高速再生することで動きが生まれます。
- キーフレーム：フレームの完全な情報を格納したビデオフレームです。キーフレームの後に続くフレームには、フレーム全体に相当する情報ではなく、キーフレームとの違いに関する情報のみが含まれます。
- メタデータ：ビデオファイルに関する情報で、ビデオファイル内に埋め込むことができ、ビデオがロードされると取得することができるものです。
- プログレッシブダウンロード : ビデオファイルが標準的な Web サーバーから配信されると、ビデオデータはプログレッシブダウンロードによってロードされます。つまり、ビデオ情報は順番にロードされます。ファイル全体がダウンロードされる前にビデオの再生を開始できるという利点がありますが、ロードされていない部分にジャンプすることはできません。
- ストリーミング：プログレッシブダウンロードの代わりに、特別なビデオサーバーを使用し、ストリーミング (別名 " トゥルーストリーミング ") という技法により、インターネット上でビデオを配信することができます。ストリーミングを使用する場合、ユーザーのコンピュータは一度にビデオ全体をダウンロードしません。コンピュータではどの時点でも、ビデオ情報全体の一部しか必要としないため、ダウンロード時間を短縮することができます。特別なサーバーがビデオコンテンツの配信を制御するので、ビデオがダウンロードされるのを待たずに、ビデオのどの部分にも自由にアクセスすることができます。

本章の例について

本章をお読みになる際は、いくつかのコード例を試すことができます。この章では ActionScript でのビデオの操作について扱っているため、この章のコードの多くには Video オブジェクトの操作が含まれています。操作の対象となる Video オブジェクトは、Flash オーサリングツールのステージ上で作成されて配置されたオブジェクトか、ActionScript を使用して作成されたオブジェクトです。サンプルのテストには、Flash Player で結果を表示してビデオに対するコードの効果を確認することが含まれます。

ほとんどのコード例では、明示的に作成せずに Video オブジェクトを操作します。この章のコードをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインでキーフレームを選択します。

3. [アクション] パネルを開き、[スクリプト] ペインにコードをコピーします。
4. 必要であれば、[ライブラリ] パネルを開きます。
5. [ライブラリ] パネルのメニューから [新規ビデオ] を選択します。
6. [ビデオのプロパティ] ダイアログボックスで、新しいビデオシンボルの名前を入力し、[タイプ] フィールドで [ビデオ (ActionScript により制御)] を選択します。[OK] をクリックしてビデオシンボルを作成します。
7. ビデオシンボルのインスタンスを [ライブラリ] パネルからステージ上にドラッグします。
8. ビデオインスタンスが選択された状態で、プロパティインスペクタを使用してそのインスタンスにインスタンス名を付けます。この名前は、コード例でビデオインスタンスで使われている名前と一致させる必要があります。たとえば、コードで `vid` という名前の Video オブジェクトを操作する場合は、Stage インスタンスの名前も `vid` にします。
9. [コントロール]-[ムービープレビュー] を使用して、プログラムを実行します。

   画面に、コードで指定したようにビデオを操作した結果が表示されます。

この章の一部のコード例には、コード例に加えてクラス定義が含まれています。これらのコードでは、前の手順に加えて、SWF をテストする前に、例で使用されるクラスを作成する必要があります。コード例で定義されたクラスを作成するには :

1. テストに使用する FLA ファイルが保存されていることを確認します。
2. メインメニューから [ファイル]-[新規作成] を選択します。
3. [新規ドキュメント] ダイアログボックスの [タイプ] セクションで、ActionScript ファイルを選択します。[OK] をクリックし、新しい ActionScript ファイルを作成します。
4. クラス定義コードをコード例から ActionScript ドキュメントにコピーします。
5. メインメニューから [ファイル]-[保存] を選択します。ファイルを Flash ドキュメントと同じディレクトリに保存します。ファイルの名前はコード内のクラス名と同じにする必要があります。たとえば、コードリストで "VideoTest" という名前のクラスが定義されている場合は、ActionScript ファイルを "VideoTest.as" として保存します。
6. Flash ドキュメントに戻ります。
7. [コントロール]-[ムービープレビュー] を使用して、プログラムを実行します。

   画面にコード例の結果が表示されます。

コード例をテストするその他の方法については、60 ページの「章内のコード例のテスト」で詳しく説明します。

FLV 形式について

FLV ファイル形式には、Flash Player を使用してエンコードされた配信用のオーディオデータとビデオデータがあります。たとえば、QuickTime または Windows Media ビデオファイルがある場合、Flash Video Encoder または Sorenson™ Squeeze といったエンコーダーを使用し、ファイルを FLV ファイルに変換することができます。

Flash オーサリングツールにビデオを取り込み、それを FLV ファイルとして書き出すと、FLV ファイルを作成することができます。FLV Export プラグインを使用すると、サポートされるビデオ編集アプリケーションから FLV ファイルを書き出すことができます。

外部 FLV ファイルでは、取り込まれたビデオでは使用できない機能も使用することができます。

- 再生速度を遅くしなくても、Flash ドキュメントで長いビデオクリップを表示することができます。外部 FLV ファイルはキャッシュされたメモリによって再生されます。つまり、サイズの大きいファイルは小さく分割して格納され、動的にアクセスされるので、埋め込みビデオファイルよりもメモリを消費しません。
- 外部 FLV ファイルのフレームレートは、再生される Flash ドキュメントとは異なることもあります。たとえば、Flash ドキュメントのフレームレートを毎秒 30 フレーム (fps) に設定し、ビデオのフレームレートを 21 fps にするなどです。このように設定すると、埋め込みビデオよりも操作性が高くなり、ビデオをスムーズに再生することができます。また、既存の Flash コンテンツを変更しなくても、異なるフレームレートの FLV ファイルを再生することも可能になります。
- 外部 FLV ファイルを使用すると、ビデオファイルがロード中でも Flash ドキュメントの再生を中断する必要はありません。取り込まれたビデオファイルの場合、CD-ROM ドライブへのアクセスといった機能を実行するために、ドキュメントの再生が中断されることがあります。FLV ファイルは Flash ドキュメントとは関係なく機能を実行できるため、再生が中断されることはありません。
- また、ビデオコンテンツへのキャプション追加も、外部 FLV ファイルの方が簡単です。これは、イベントハンドラを使用してビデオのメタデータにアクセスできるからです。

メモ	Web サーバーから FLV ファイルをロードするために、Web サーバーにファイル拡張子と MIME タイプを登録しなければならないことがあります。Web サーバーのマニュアルを確認してください。FLV ファイルの MIME タイプは `video/x-flv` です。詳細については、557 ページの「サーバーでホスティングするための FLV ファイルの設定」を参照してください。

Video クラスについて

Video クラスを使用すると、SWF ファイルに埋め込むことなく、ライブストリーミングビデオをアプリケーションで表示することができます。`Camera.getCamera()` メソッドを使用して、ライブビデオをキャプチャして再生できます。また、Video クラスを使用して、HTTP 経由またはローカルファイルシステムから FLV ファイルを再生することもできます。プロジェクトで Video クラスを使用するには、いくつかの方法があります。

- NetConnection クラスおよび NetStream クラスを使用して FLV を動的にロードし、Video オブジェクトでビデオを表示します。
- ユーザーのカメラから入力をキャプチャします。
- FLVPlayback コンポーネントを使用します。

メモ	ステージ上の Video オブジェクトのインスタンスは、Video クラスのインスタンスです。

Video クラスは flash.media パッケージにありますが、flash.display.DisplayObject クラスから継承されるため、マトリックス変換やフィルタといった表示オブジェクトのすべての機能も Video インスタンスに適用されます。

詳細については、376 ページの「表示オブジェクトの操作」、411 ページの「ジオメトリの操作」、および 443 ページの「表示オブジェクトのフィルタ処理」を参照してください。

ビデオファイルのロード

NetStream および NetConnection クラスを使用したビデオのロードは、次に示す複数の手順で実行されます。

1. まず、NetConnection オブジェクトを作成します。NetConnection クラスを使用すると、Adobe Flash Media Server 2 や Adobe Flex などのサーバーを使用しないローカル FLV ファイルに接続している場合に、`null` 値を `connect()` メソッドに渡すことによって HTTP アドレスまたはローカルドライブからストリーミング FLV ファイルを再生できます。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2. 次に、NetConnection オブジェクトをパラメータとして取る NetStream オブジェクトを作成し、ロードする FLV ファイルを指定します。次のコード例では、NetStream オブジェクトを指定の NetConnection インスタンスに接続し、SWF ファイルと同じディレクトリに "video.flv" という名前の FLV をロードします。

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
   // エラーを無視します。
}
```

3. 次に、新しい Video オブジェクトを作成し、先に作成された NetStream オブジェクトを、Video クラスの attachNetStream() メソッドを使用して関連付けます。その後、次のコード例で示すように、addChild() メソッドを使用して表示リストに Video オブジェクトを追加します。

```
var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

前のコードを入力すると、Flash Player は SWF ファイルと同じディレクトリに video.flv ビデオファイルをロードしようとします。

ヒント

Web サーバーから FLV ファイルをロードするために、Web サーバーにファイル拡張子と MIME タイプを登録しなければならないことがあります。Web サーバーのマニュアルを確認してください。FLV ファイルの MIME タイプは video/x-flv です。詳細については、557 ページの「サーバーでホスティングするための FLV ファイルの設定」を参照してください。

ビデオ再生の制御

NetStream クラスは、ビデオ再生を制御する 4 つの主な方法を提供します。

pause() : ビデオストリームの再生を一時停止します。ビデオが既に停止している場合は、このメソッドを呼び出しても何も実行されません。

resume() : 再生を一時停止していたビデオストリームを再開します。ビデオが既に再生中である場合は、このメソッドを呼び出しても何も実行されません。

seek() : 指定の位置、すなわちストリームの先頭からのオフセット (秒単位) に最も近いキーフレームをシークします。

togglePause() : ストリームの再生を一時停止または再開します。

メモ

stop() メソッドはありません。ストリームを停止するには、再生を一時停止し、ビデオストリームの先頭にシークする必要があります。

メモ

play() メソッドは再生を再開しません。ビデオファイルのロードに使用されます。

次の例は、さまざまなボタンを使用してビデオを制御する方法を示します。次の例を実行するには、新しいドキュメントを作成し、ワークスペース内に pauseBtn、playBtn、stopBtn、および togglePauseBtn の 4 つのボタンインスタンスを追加します。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
```

```
  // エラーを無視します。
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
  ns.pause();
}
function playHandler(event:MouseEvent):void
{
  ns.resume();
}
function stopHandler(event:MouseEvent):void
{
  // ストリームを一時停止し、再生ヘッドを
  // ストリームの先頭に移動します。
  ns.pause();
  ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
  ns.togglePause();
}
```

ビデオが再生中に pauseBtn ボタンインスタンスをクリックすると、ビデオファイルが一時停止します。ビデオが既に一時停止状態にある場合、このボタンインスタンスをクリックしても何も効果がありません。再生が一時停止状態にある場合に playBtn ボタンインスタンスをクリックすると、ビデオ再生が再開されます。ビデオが既に再生中であれば、何も効果がありません。

ビデオストリームの末尾の検知

ビデオストリームの先頭と末尾を監視するには、netStatus イベントを監視するためのイベントリスナーを NetStream インスタンスに追加する必要があります。次のコードは、ビデオ再生全体において、さまざまなコードがどのように監視されるかを示します。

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
  trace(event.info.code)
}
```

上のコードを実行すると、次のように表示されます。

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

特に便利なのは、ビデオ再生の先頭と末尾を知らせる "NetStream.Play.Start" および "NetStream.Play.Stop" の 2 つのコードです。次のコード例では、switch ステートメントを使用してこれら 2 つのコードをフィルタリングし、メッセージをトレースします。

```
function statusHandler(event:NetStatusEvent):void
{
  switch (event.info.code)
  {
    case "NetStream.Play.Start":
      trace("Start [" + ns.time.toFixed(3) + " seconds]");
      break;
    case "NetStream.Play.Stop":
      trace("Stop [" + ns.time.toFixed(3) + " seconds]");
      break;
  }
}
```

netStatus イベント (NetStatusEvent.NET_STATUS) を監視すると、再生中のビデオが終了したときに、プレイリストの次のビデオをロードするビデオプレイヤーを作成することができます。

ビデオファイルのストリーミング

Flash Media Server からファイルをストリーミングするには、NetConnection および NetStream クラスを使用してリモートサーバーインスタンスに接続し、指定のストリームを再生することができます。RTMP (Real-Time Messaging Protocol) サーバーを指定するには、`null` 値ではなく "rtmp://localhost/appName/appInstance" といった RTMP URL を `NetConnection.connect()` メソッドに渡します。指定の Flash Media Server から特定のライブストリームまたは記録されているストリームを再生するには、`NetStream.publish()` によってパブリッシュされたライブデータの識別名か、記録されたファイルの名前を `NetStream.play()` メソッドに渡します。詳細については、Flash Media Server のマニュアルを参照してください。

キューポイントについて

すべての FLV ファイルにキューポイントがあるわけではありません。キューポイントは通常、FLV エンコード中に FLV ファイルに埋め込まれますが、既存の FLV ファイルにキューポイントを埋め込むためのツールもあります。

Flash Video では、数種類のキューポイントを使用することができます。ActionScript を使用し、FLV ファイルの作成時に FLV ファイルに埋め込んだキューポイントや、ActionScript で作成したキューポイントを操作することができます。

- ナビゲーションキューポイント : FLV ファイルをエンコードするときに、FLV ストリームと FLV メタデータパケットにナビゲーションキューポイントを埋め込みます。ナビゲーションキューポイントを使用すると、ユーザーはファイルの指定部分にシークすることができます。
- イベントキューポイント : FLV ファイルをエンコードするときに、FLV ストリームと FLV メタデータパケットにイベントキューポイントを埋め込みます。FLV の再生時に、指定のポイントでトリガーされるイベントに対して処理を実行するコードを作成することができます。
- ActionScript キューポイント : ActionScript コードによって作成される外部キューポイントです。ビデオの再生に関連付け、これらのキューポイントをトリガーするコードを作成することができます。ビデオプレイヤーは、これらのキューポイントを個別にトラッキングするため、埋め込まれるキューポイントに比べると精度が低くなります (最大で 10 分の 1 秒)。

ナビゲーションキューポイントは、指定のキューポイント位置でキーフレームを作成するため、コードを使用して、ビデオプレイヤーの再生ヘッドをその位置に移動することができます。ユーザーがシークする位置として、FLV ファイルにポイントを設定することもできます。たとえば、ビデオに複数のチャプターまたはセグメントがあるとすると、ビデオファイルに埋め込まれたナビゲーションキューポイントを使用してビデオを制御することができます。

ユーザーがキューポイントにナビゲートするようなアプリケーションを作成する場合は、ActionScript キューポイントを使用する代わりに、ファイルをエンコードするときにキューポイントを作成し、埋め込みます。キューポイントは、FLV ファイルに埋め込む必要があります。この方が、精度が高いからです。キューポイントと共に FLV ファイルをエンコードする方法の詳細については、『Flash ユーザーガイド』の「キューポイントの埋め込み」を参照してください。

ActionScript を記述することで、キューポイントパラメータにアクセスすることができます。キューポイントパラメータとは、onCuePoint コールバックハンドラから受け取るイベントオブジェクトの一部です。

ビデオが指定のキューポイントに到達したときに、コードで特定のアクションをトリガーするには、NetStream.onCuePoint イベントハンドラを使用します。詳細については、539 ページの「onCuePoint および onMetaData のコールバックメソッドの記述」を参照してください。

onCuePoint および onMetaData のコールバックメソッドの記述

指定のキューポイントに到達したとき、またはプレイヤーが指定のメタデータを受け取ったときに、アプリケーションでアクションをトリガーすることができます。このようなアクションをトリガーするには、onCuePoint および onMetaData イベントハンドラを使用します。これらのハンドラのコールバックメソッドも記述しておく必要があります。コールバックメソッドがなければ、Flash Player がエラーをスローします。たとえば、次のコードは、SWF ドキュメントと同じフォルダの video.flv という名前の FLV ファイルを再生します。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
  trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

前述のコードは、video.flv という名前のローカル FLV ファイルをロードし、送出される asyncError (AsyncErrorEvent.ASYNC_ERROR) を監視します。このイベントは、ネイティブ非同期コードから例外がスローされると送出されます。この場合、FLV にメタデータまたはキューポイント情報が含まれ、適切なリスナーが定義されていない場合に送出されます。前述のコードは、ユーザーがビデオファイルのメタデータまたはキューポイント情報に関心がなければ、asyncError イベントを処理し、エラーを無視します。メタデータと複数のキューポイントを持つ FLV がある場合、次の情報がトレースされます。

```
Error #2095: flash.net.NetStream was unable to invoke callback onMetaData.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
```

NetStream オブジェクトが onMetaData または onCuePoint コールバックメソッドを見つけられなかったため、エラーが発生しました。アプリケーション内でこれらのコールバックメソッドを定義するには、いくつかの方法があります。

- NetStream オブジェクトの client プロパティをオブジェクトに設定する
- カスタムクラスの作成とコールバックメソッドを処理するメソッドの定義
- NetStream クラスの拡張とコールバックメソッドを処理するメソッドの追加
- NetStream クラスの拡張と動的化
- NetStream オブジェクトの client プロパティに対する this の設定

NetStream オブジェクトの client プロパティをオブジェクトに設定する

client プロパティをオブジェクトまたは NetStream のサブクラスに設定すると、onMetaData および onCuePoint コールバックメソッドをリルートするか、完全に無視することができます。次の例は、空のオブジェクトを使用し、asyncError イベントを監視せずにコールバックメソッドを無視する方法を示します。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

onMetaData または onCuePoint コールバックメソッドのいずれかを監視するには、次のコード例で示すように、これらのコールバックメソッドを処理するメソッドを定義する必要があります。

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
  trace("metadata");
}
```

上記のコードは onMetaData コールバックメソッドを監視し、ストリングをトレースする metaDataHandler() メソッドを呼び出します。Flash Player がキューポイントに到達すると、onCuePoint コールバックメソッドが定義されていなくても、エラーは発生しません。

カスタムクラスの作成とコールバックメソッドを処理するメソッドの定義

次のコードは NetStream オブジェクトの client プロパティを、カスタムクラスである CustomClient に設定します。CustomClient は、コールバックメソッドのハンドラを定義します。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

CustomClient クラスを次に示します。

```
package
{
  public class CustomClient
  {
    public function onMetaData(infoObject:Object):void
    {
      trace("metadata");
    }
  }
}
```

CustomClient クラスは、onMetaData コールバックハンドラのハンドラを指定します。キューポイントに到達し、onCuePoint コールバックハンドラが呼び出されると、asyncError イベント (AsyncErrorEvent.ASYNC_ERROR) が送出され、"flash.net.NetStream was unable to invoke callback onCuePoint" (flash.net.NetStream は onCuePoint コールバックを開始できませんでした) と示されます。このエラーを回避するため、CustomClient クラスで onCuePoint コールバックメソッドを指定するか、asyncError イベントのイベントハンドラを指定します。

NetStream クラスの拡張とコールバックメソッドを処理するメソッドの追加

次のコードでは、後述するコードで定義されている CustomNetStream クラスのインスタンスを作成します。

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

次のコードは、NetStream クラスを拡張し、必要な NetConnection オブジェクトを作成し、onMetaData および onCuePoint コールバックハンドラメソッドを処理する CustomNetStream クラスを定義します。

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public class CustomNetStream extends NetStream
  {
    private var nc:NetConnection;
    public function CustomNetStream()
    {
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
    }
    public function onMetaData(infoObject:Object):void
    {
      trace("metadata");
    }
    public function onCuePoint(infoObject:Object):void
    {
      trace("cue point");
    }
  }
}
```

CustomNetStream クラスの onMetaData() および onCuePoint() メソッドの名前を変更するには、次のコードを使用します。

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public class CustomNetStream extends NetStream
  {
    private var nc:NetConnection;
    public var onMetaData:Function;
    public var onCuePoint:Function;
    public function CustomNetStream()
    {
      onMetaData = metaDataHandler;
      onCuePoint = cuePointHandler;
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
    }
    private function metaDataHandler(infoObject:Object):void
    {
      trace("metadata");
    }
    private function cuePointHandler(infoObject:Object):void
    {
      trace("cue point");
    }
  }
}
```

NetStream クラスの拡張と動的化

NetStream クラスを拡張し、サブクラスを動的にすると、onCuePoint および onMetaData コールバックハンドラを動的に追加することができます。例を次のコードに示します。

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

DynamicCustomNetStream クラスを次に示します。

```
package
{
  import flash.net.NetConnection;
  import flash.net.NetStream;
  public dynamic class DynamicCustomNetStream extends NetStream
```

```
  {
    private var nc:NetConnection;
    public function DynamicCustomNetStream()
    {
      nc = new NetConnection();
      nc.connect(null);
      super(nc);
    }
  }
}
```

onMetaData および onCuePoint コールバックハンドラのハンドラがない場合でも、DynamicCustomNetStream クラスは動的なのでエラーはスローされません。onMetaData および onCuePoint コールバックハンドラのメソッドを定義するには、次のコードを使用します。

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
  trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
  trace("cue point");
}
```

NetStream オブジェクトの client プロパティに対する this の設定

client プロパティを this に設定すると、Flash Player は onMetaData() および onCuePoint() メソッドの現在の範囲を探します。これを次の例に示します。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

onMetaData または onCuePoint コールバックハンドラが呼び出されず、コールバックを処理するメソッドがない場合は、エラーは発生しません。これらのコールバックハンドラを処理するには、次のコード例に示すように、コードに onMetaData( ) および onCuePoint( ) メソッドを作成します。

```
function onMetaData(infoObject:Object):void
{
  trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
  trace("cue point");
}
```

キューポイントの使用

次の例は、シンプルな for..in ループを onCuePoint コールバックハンドラの infoObject パラメータのプロパティで繰り返し、キューポイントデータを受け取ったときにメッセージをトレースします。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onCuePoint(infoObject:Object):void
{
  var key:String;
  for (key in infoObject)
  {
    trace(key + ": " + infoObject[key]);
  }
}
```

出力は次のとおりです。

```
parameters:
name: point1
time: 0.418
type: navigation
```

このコードでは、コールバックメソッドを開始するオブジェクトを設定する技法の 1 つを使用しています。他の技法もあります。詳細については onCuePoint および onMetaData のコールバックメソッドの記述を参照してください。

ビデオメタデータの使用

onMetaData コールバックハンドラを使用し、FLV ファイルのメタデータ情報を表示することができます。メタデータには、長さ、幅、高さ、フレームレートなど FLV ファイルに関する情報が含まれています。FLV ファイルに追加されるメタデータ情報は、FLV ファイルをエンコードするためのソフトウェア、またはメタデータ情報を追加するためのソフトウェアによって異なります。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
  var key:String;
  for (key in infoObject)
  {
    trace(key + ": " + infoObject[key]);
  }
}
```

前述のコードでは、FLV ファイルにキューポイントとオーディオが含まれていると仮定すると、次のようなコードを生成します。

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

ヒント

ビデオにオーディオがない場合、エンコード中にメタデータにオーディオ情報が追加されないため、audiodatarate などのオーディオ関連のメタデータ情報は undefined を返します。

前述のコードでは、キューポイント情報は表示されていません。キューポイントメタデータを表示するには、オブジェクトのアイテムを再帰的に表示する次の関数を使用することができます。

```
function traceObject(obj:Object, indent:uint = 0):void
{
  var indentString:String = "";
  var i:uint;
  var prop:String;
  var val:*;
  for (i = 0; i < indent; i++)
  {
    indentString += "\t";
  }
  for (prop in obj)
  {
    val = obj[prop];
    if (typeof(val) == "object")
    {
      trace(indentString + " " + j + ": [Object]");
      traceObject(val, indent + 1);
    }
    else
    {
      trace(indentString + " " + prop + ": " + val);
    }
  }
}
```

前述のコード例を使用し、onMetaData() メソッドの infoObject パラメータをトレースすると、出力は次のようになります。

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
height: 213
framerate: 15
cuePoints: [Object]
  0: [Object]
    parameters: [Object]
      lights: beginning
    name: point1
    time: 0.418
    type: navigation
  1: [Object]
    parameters: [Object]
      lights: middle
```

```
    name: point2
    time: 7.748
    type: navigation
  2: [Object]
    parameters: [Object]
      lights: end
    name: point3
    time: 16.02
    type: navigation
```

onMetaData の Info オブジェクト

次の表に、ビデオメタデータで使用可能な値をまとめます。

パラメータ	説明
audiocodecid	使用されるオーディオコーデック (コード / デコード技法) を示す数値です。
audiodatarate	オーディオがエンコードされるレートをキロバイト秒で示す数値です。
audiodelay	元の FLV ファイルの "time 0" の時間を示す数値です。オーディオと正しく同期化するには、ビデオコンテンツを少し遅延させる必要があります。
canSeekToEnd	FLV ファイルが最終フレームのキーフレームによってエンコードされている場合は `true` となるブール値です。これにより、プログレッシブダウンロードのムービークリップの末尾をシークすることができます。FLV ファイルが最終フレームのキーフレームでエンコードされていない場合は `false` となります。
cuePoints	オブジェクトの配列。FLV ファイルに埋め込まれたそれぞれのキューポイントに 1 つずつあります。FLV ファイルにキューポイントがない場合は、値は未定義となります。各オブジェクトには次のプロパティがあります。 ■ `type` : "navigation" または "event" としてキューポイントのタイプを指定するストリングです。 ■ `name` : キューポイントの名前を表すストリングです。 ■ `time` : キューポイントを秒単位で示す数値です。小数第 3 位 (ミリ秒) まで表示できます。 ■ `parameters` : キューポイントの作成時にユーザーによって指定される、名前と値のペアとして示されるオプションのオブジェクトです。
duration	FLV ファイルの長さを秒単位で示す数値です。
framerate	FLV ファイルのフレームレートを示す数値です。
height	FLV ファイルの高さをピクセル単位で示す数値です。
videocodecid	ビデオのエンコードに使用されたコーデックのバージョンを表す数値です。

パラメータ	説明
videodatarate	FLV ファイルのビデオデータレートを示す数値です。
width	FLV ファイルの幅をピクセル単位で示す数値です。

次の表に、`videocodecid` パラメータで使用可能な値をまとめます。

videocodecid	コーデック名
2	Sorenson H.263
3	Screen Video (SWF 7 以降のみ)
4	VP6 (SWF 8 以降のみ)
5	アルファチャンネルを使用する VP6 Video (SWF 8 以降のみ)

次の表に、`audiocodecid` パラメータで使用可能な値をまとめます。

audiocodecid	コーデック名
0	uncompressed
1	ADPCM
2	MP3
5	Nellymoser 8kHz モノラル
6	Nellymoser

カメラ入力のキャプチャ

外部ビデオファイルに加え、ユーザーのコンピュータに接続されたカメラをビデオデータのソースとして使用し、ActionScript で表示および操作することができます。Camera クラスは、コンピュータのカメラを操作するために ActionScript に組み込まれたメカニズムです。

Camera クラスについて

Camera オブジェクトを使用すると、ユーザーのローカルカメラに接続し、ビデオをローカルでブロードキャスト (ユーザーに表示)、またはリモートで Flash Media Server などのサーバーに表示することができます。

Camera クラスにより、ユーザーのカメラに関する次の情報を入手することができます。

- ユーザーのコンピュータにインストールされたカメラのうち、Flash Player で使用可能なもの
- カメラがインストールされているかどうか

- Flash Player がユーザーのカメラを使用できるかどうか
- どのカメラがアクティブか
- キャプチャされるビデオの幅と高さ

Camera クラスには、Camera オブジェクトの操作で使用できる便利なメソッドとプロパティがいくつかあります。たとえば、静的な `Camera.names` プロパティには、ユーザーのコンピュータにインストールされているカメラ名の配列があります。また、`name` プロパティを使用し、アクティブなカメラの名前を表示することもできます。

画面上でのカメラの表示

カメラに接続すると、NetConnection および NetStream クラスで FLV をロードするよりもコード数が少なくて済みます。同時に、Camera クラスはすぐに扱いにくくなることもあります。それは、カメラへのアクセスにの前に、Flash Player をカメラに接続するための許可をユーザーから得る必要があるからです。

次のコードでは、Camera クラスを使用してユーザーのローカルカメラに接続する方法を示します。

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

メモ

Camera クラスには、コンストラクタメソッドはありません。新しい Camera インスタンスを作成するには、静的な `Camera.getCamera()` メソッドを使用します。

カメラアプリケーションの設計

ユーザーのカメラに接続されるアプリケーションを設計するには、コードで次の点を考慮する必要があります。

- カメラがインストールされているかどうかを確認します。
- Flash Player がカメラにアクセスできるよう、明示的に許可されているかどうかを確認します。セキュリティの理由から、カメラへのアクセスを許可するか拒否するかを選択するため、Flash Player の [設定] ダイアログボックスが表示されます。これにより、Flash Player がユーザーの許可なくカメラに接続し、ビデオストリームをブロードキャストできなくなります。ユーザーの許可がある場合は、アプリケーションはユーザーのカメラに接続することができます。ユーザーが拒否した場合、アプリケーションはユーザーのカメラにアクセスすることはできません。いずれの場合でも、適切に対応する必要があります。

ユーザーのカメラへの接続

ユーザーのカメラに接続するには、最初に、Camera タイプの変数を作成して新しい Camera インスタンスを作成し、静的な `Camera.getCamera()` メソッドの値を返すよう初期化する必要があります。

次に、新しい Video オブジェクトを作成し、それに Camera オブジェクトを関連付けます。

次に、表示リストに Video オブジェクトを追加します。Camera クラスは DisplayObject クラスを拡張せず、表示リストに直接追加することができないため、手順 2 および 3 を実行する必要があります。カメラでキャプチャしたビデオを表示するには、新しい Video オブジェクトを作成し、`attachCamera()` メソッドを呼び出します。

次のコードは、上記の 3 つの手順を示します。

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

カメラがインストールされていない場合、Flash Player は何も表示しません。

実際には、アプリケーションで追加手順を実行する必要があります。詳細については、カメラのインストールの確認およびカメラのアクセス許可の検知を参照してください。

カメラのインストールの確認

Camera インスタンスでメソッドまたはプロパティを使用する前に、カメラがインストールされているかどうかを確認します。そのためには、次の 2 つの方法があります。

- 使用可能なカメラ名の配列を含む、静的な `Camera.names` プロパティを確認します。一度に複数のカメラをインストールしているユーザーはあまりいないため、この配列には通常は少数のストリングしか含まれていません。次のコードでは、使用可能なカメラの有無を確認するため `Camera.names` プロパティをチェックする方法を示します。

  ```
  if (Camera.names.length > 0)
  {
    trace("User has no cameras installed.");
  }
  else
  {
    var cam:Camera = Camera.getCamera(); // デフォルトカメラを取得します。
  }
  ```

- 静的な Camera.getCamera() メソッドの戻り値を確認します。使用可能またはインストールされているカメラがない場合は、このメソッドは null を返します。それ以外の場合は、Camera オブジェクトへの参照を返します。次のコードでは、使用可能なカメラの有無を確認するために Camera.getCamera() メソッドをチェックする方法を示します。

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
  trace("User has no cameras installed.");
}
else
{
  trace("User has at least 1 camera installed.");
}
```

Camera クラスは DisplayObject クラスを拡張しないため、addChild() メソッドを使用して表示リストに直接追加することはできません。カメラでキャプチャしたビデオを表示するには、新しい Video オブジェクトを作成し、Video インスタンスで attachCamera() メソッドを呼び出す必要があります。

このコード例では、カメラがある場合はそれを付加する方法を示します。カメラがない場合は、Flash Player には何も表示されません。

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  var vid:Video = new Video();
  vid.attachCamera(cam);
  addChild(vid);
}
```

カメラのアクセス許可の検知

カメラの出力を表示する前に、ユーザーは Flash Player がカメラにアクセスできるよう明示的に許可する必要があります。attachCamera() メソッドが呼び出されると、Flash Player は [設定] ダイアログボックスを表示します。ここでは、Flash Player がカメラとマイクにアクセスできるかどうかを指定します。ユーザーが [許可] ボタンをクリックすると、カメラからの出力はステージ上の Video インスタンスに表示されます。[拒否] ボタンをクリックした場合は、Flash Player はカメラに接続できないため、Video オブジェクトは何も表示しません。

カメラがインストールされていない場合は、Flash Player は何も表示しません。カメラがインストールされている場合は、Flash Player は [設定] ダイアログボックスを表示します。このダイアログボックスで、ユーザーは Flash Player によるカメラへのアクセスの可否を指定します。ユーザーがカメラへのアクセスを許可した場合は、ビデオが表示されます。拒否した場合は何も表示されません。

カメラへのアクセスへの可否を確認するには、次のコードで示すように、カメラの status イベント (StatusEvent.STATUS) を監視します。

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  var vid:Video = new Video();
  vid.attachCamera(cam);
  addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
  // このイベントは、Flash Player の [ 設定 ] ダイアログで
  // [ 許可 ] または [ 拒否 ] ボタンがクリックされると送出されます。
  trace(event.code); // "Camera.Muted" または "Camera.Unmuted" です。
}
```

[許可] または [拒否] のいずれかがクリックされると同時に、statusHandler( ) 関数が呼び出されます。次のいずれかのメソッドを使用し、どのボタンがクリックされたかを検知することができます。

- statusHandler( ) 関数の event パラメータには、"Camera.Muted" または "Camera.Unmuted" というストリングを含むコードプロパティがあります。値が "Camera.Muted" であれば、[拒否] ボタンがクリックされているので、Flash Player はカメラにアクセスできません。次に例を示します。

```
function statusHandler(event:StatusEvent):void
{
  switch (event.code)
  {
    case "Camera.Muted":
      trace("User clicked Deny.");
      break;
    case "Camera.Unmuted":
      trace("User clicked Accept.");
      break;
  }
}
```

- Camera クラスには muted という名前の読み取り専用のプロパティがあり、ユーザーがカメラへのアクセスを拒否したか (true) または許可したか (false) が、[Macromedia Flash Player 設定] の [プライバシー] パネルに表示されます。次に例を示します。

```
function statusHandler(event:StatusEvent):void
{
  if (cam.muted)
  {
    trace("User clicked Deny.");
  }
  else
  {
    trace("User clicked Accept.");
```

```
        }
    }
```

送出されるステータスイベントを確認すると、カメラへのアクセスの可否を処理するコードを記述し、適切にクリーンアップすることができます。たとえば、[拒否] ボタンがクリックされた場合、ビデオチャットに参加するには [許可] ボタンをクリックする必要があるというメッセージを表示することができます。または、表示リストの Video オブジェクトが削除され、システムリソースが解放されていることを確認します。

ビデオ品質の最大化

デフォルトでは、Video クラスの新規インスタンスは幅 320 ピクセル、高さ 240 ピクセルです。ビデオ品質を最大化するには、Video オブジェクトと、Camera オブジェクトによって返されるビデオのサイズを一致させる必要があります。Camera クラスの `width` および `height` プロパティを使用して Camera オジェクトの幅と高さを取得し、Video オブジェクトの `width` および `height` プロパティを Camera オブジェクトと同じサイズに設定するか、次に示すように、カメラの幅と高さを Video クラスのコンストラクタメソッドに渡します。

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
  var vid:Video = new Video(cam.width, cam.height);
  vid.attachCamera(cam);
  addChild(vid);
}
```

`getCamera()` メソッドは Camera オブジェクトへの参照を返すので (カメラがない場合は `null`)、ユーザーがカメラへのアクセスを拒否している場合でも、カメラのメソッドとプロパティを使用することができます。これにより、カメラのネイティブの高さと幅を使用してビデオインスタンスのサイズを設定することができます。

```
var vid:Video;
var cam:Camera = Camera.getCamera();

if (cam == null)
{
  trace("Unable to locate available cameras.");
}
else
{
  trace("Found camera: " + cam.name);
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  vid = new Video();
  vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
```

```
  if (cam.muted)
  {
    trace("Unable to connect to active camera.");
  }
  else
  {
    // カメラの設定に一致するように Video オブジェクトのサイズを変更し
    // ビデオを表示リストに追加します。
    vid.width = cam.width;
    vid.height = cam.height;
    addChild(vid);
  }
  // ステータスイベントリスナーを削除します。
  cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

再生状況の監視

カメラクラスには、Camera オブジェクトの現在のステータスを監視するためのプロパティがいくつかあります。たとえば次のコードでは、表示リストで Timer オブジェクトとテキストフィールドインスタンスを使用して、カメラのプロパティをいくつか表示します。

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
  cam.addEventListener(StatusEvent.STATUS, statusHandler);
  vid = new Video();
  vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
  if (!cam.muted)
  {
    vid.width = cam.width;
    vid.height = cam.height;
    addChild(vid);
    t.start();
  }
  cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
```

```
function timerHandler(event:TimerEvent):void
{
  tf.text = "";
  tf.appendText("activityLevel: " + cam.activityLevel + "\n");
  tf.appendText("bandwidth: " + cam.bandwidth + "\n");
  tf.appendText("currentFPS: " + cam.currentFPS + "\n");
  tf.appendText("fps: " + cam.fps + "\n");
  tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
  tf.appendText("loopback: " + cam.loopback + "\n");
  tf.appendText("motionLevel: " + cam.motionLevel + "\n");
  tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
  tf.appendText("quality: " + cam.quality + "\n");
}
```

Timer オブジェクトの `timer` イベントは 10 分の 1 秒、つまり 100 ミリ秒ごとに送出され、`timerHandler()` 関数は表示リストのテキストフィールドを更新します。

サーバーへのビデオ送信

Video オブジェクトまたは Camera オブジェクトを使用する、より複雑なアプリケーションを構築するため、Flash Media Server ではメディアアプリケーションを幅広いユーザー向けに作成して配信するためのストリーミングメディア機能と開発環境を提供します。これらがあると、開発者はビデオオンデマンド、ライブ Web イベントブロードキャスト、MP3 ストリーミング、そしてビデオブログ、ビデオメッセージング、マルチメディアチャット環境などのアプリケーションを構築することができます。詳細については、オンラインの Flash Media Server ドキュメント、http://livedocs.macromedia.com/fms/2/docs/ を参照してください。

高度なトピック

次のトピックでは、ビデオを操作する上での特別な問題点について取り上げます。

Flash Player とエンコードされた FLV ファイルとの互換性

Flash Player 7 は、Sorenson™ Spark™ ビデオコーデックによってエンコードされた FLV ファイルをサポートします。Flash Player 8 は、Sorenson Spark または Flash Professional 8 の On2 VP6 エンコーダーによってエンコードされた FLV ファイルをサポートします。On2 VP6 ビデオコーデックはアルファチャンネルに対応しています。Flash Player のバージョンによって、FLV のサポート方法が異なります。詳細については、次の表を参照してください。

コーデック	SWF ファイルバージョン (パブリッシュバージョン)	再生に必要な Flash Player のバージョン
Sorenson Spark	6	6、7、または 8
	7	7, 8
On2 VP6	6	8*
	7	8
	8	8

* SWF ファイルが FLV ファイルをロードする場合、SWF ファイルの表示に Flash Player 8 が使用されている限り、SWF ファイルを Flash Plaher 8 に再パブリッシュすることなく On2 VP6 ビデオを使用することができます。Flash Player 8 のみが On2 VP6 ビデオのパブリッシュと再生をサポートしています。

サーバーでホスティングするための FLV ファイルの設定

FLV ファイルを操作するとき、FLV ファイル形式を扱えるようサーバーを設定する必要が生じることがあります。MIME は標準的なデータ仕様であり、非 ASCII ファイルをインターネット上で送信することができます。Web ブラウザや電子メールクライアントは、ビデオやオーディオ、グラフィック、およびフォーマット済みテキストを送信できるよう、さまざまな MIME タイプを解釈できるしくみになっています。Web サーバーから FLV ファイルをロードするには、ファイル拡張子と MIME タイプを Web サーバーに登録する必要があります。そのため、Web サーバーのマニュアルを確認する必要があります。FLV ファイルの MIME タイプは `video/x-flv` です。FLV ファイルタイプの詳細は次のとおりです。

- MIME タイプ : video/x-flv
- ファイル拡張子 : .flv
- 必要なパラメータ : なし

- オプションのパラメータ : なし
- エンコード上の考慮事項 : FLV ファイルはバイナリファイルであること。アプリケーションの中には、application/octet-stream サブタイプを設定する必要があるものもあります。
- セキュリティ上の問題点 : なし
- パブリッシュされた仕様 : http://www.adobe.com/go/flashfileformat_jp

Microsoft Microsoft Internet Information Services (IIS) 6.0 Web サーバーのストリーミングメディアの処理方法が、旧バージョンの処理方法から変更されました。旧バージョンの IIS では、Flash Video をストリーミングするために何も修正する必要はありませんでした。しかし、Windows 2003 のデフォルト Web サーバーとして付属している IIS 6.0 では、FLV ファイルをストリーミングメディアとして認識させるには、MIME タイプが必要となります。

外部 FLV ファイルをストリーミングする SWF ファイルが Microsoft Windows Server® 2003 に配置され、ブラウザで表示されると、SWF ファイルは正常に再生されますが、FLV ビデオはストリーミングされません。この問題は、アドビ システムズ社の旧バージョンの Flash オーサリングツールである Macromedia Flash Video Kit for Dreamweaver MX 2004 で作成されたファイルを含め、Windows Server 2003 上に置かれたすべての FLV ファイルに影響を与えます。これらのファイルは、他のオペレーティングシステムでテストした場合は正常に動作します。

Microsoft Windows 2003 および Microsoft IIS Server 6.0 で FLV ビデオをストリーミングするよう設定する方法については、www.adobe.com/go/tn_19439_jp を参照してください。

Macintosh のローカル FLV ファイルのターゲット化

スラッシュ (/) を使用する相対パスにより、Apple® Macintosh® の非システムドライブからローカル FLV を再生しようとしても、ビデオは再生されません。ここで、非システムドライブとは、CD-ROM、パーティション分割されたハードディスク、リムーバブルストレージメディア、外部ストレージメディアなどがありますが、これらに限定されるわけではありません。

メモ	再生されない理由は、Flash Player 上の制限ではなく、オペレーティングシステムの制限です。

Macintosh の非システムドライブから FLV ファイルを再生するには、スラッシュ (/) ベースではなくコロン (:) ベースの表記による絶対パスにより、そのファイルを参照します。次に、2 つの表記法の違いを示します。

- スラッシュベース表記 : myDrive/myFolder/myFLV.flv
- コロンベース表記 : (Mac OS®) myDrive:myFolder:myFLV.flv

また、CD-ROM 向けのプロジェクタファイルを作成し、Macintosh での再生で使用することもできます。Mac OS CD-ROM および FLV ファイルに関する最新情報については、www.adobe.com/go/3121b301_jp を参照してください。

例 : ビデオジュークボックス

次の例では、順番に再生されるビデオのリストを動的にロードする、シンプルなビデオジュークボックスを構築します。こうすると、一連のビデオチュートリアルを閲覧、またはユーザーが要求したビデオの再生前にどの広告を再生するかを指定できるようなアプリケーションを構築できるようになります。この例は、ActionScript 3.0 の次のような機能を示しています。

- ビデオファイルの再生の進行状況に基づく再生ヘッドの更新
- ビデオファイルのメタデータの監視と解析
- ネットストリームの特定のコーデックの処理
- 動的にロードされた FLV のロード、再生、一時停止、停止
- ネットストリームのメタデータに基づく、表示リストの Video オブジェクトのサイズ変更

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。ビデオジュークボックスアプリケーションのファイルは、"Samples/VideoJukebox" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
VideoJukebox.as	このクラスは、アプリケーションの主な機能を提供します。
VideoJukebox.fla	Flash の主要なアプリケーションです。
playlist.xml	ビデオジュークボックスにロードされるビデオファイルをリストするファイルです。

ビデオプレイリストファイルのロード

外部の playlist.xml ファイルは、ロードするビデオ、および再生の順序を指定します。XML ファイルをロードするには、次に示すように、URLLoader オブジェクトと URLRequest オブジェクトを使用する必要があります。

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

このコードは、VideoJukebox クラスのコンストラクタに配置されるので、ファイルはコードが実行される前にロードされます。XML ファイルのロードが完了するとすぐに `xmlCompleteHandler()` メソッドが呼び出され、次のコードに示すように、外部ファイルを XML オブジェクトに解析します。

```
private function xmlCompleteHandler(event:Event):void
{
  playlist = XML(event.target.data);
  videosXML = playlist.video;
```

```
  main();
}
```

プレイリストの XML オブジェクトには、外部ファイルから取得された未処理 XML がありますが、videosXML は、ビデオノードしか含まない XMLList オブジェクトです。サンプルの playlist.xml ファイルを次のコード例に示します。

```
<videos>
  <video url="video/caption_video.flv" />
  <video url="video/cuepoints.flv" />
  <video url="video/water.flv" />
</videos>
```

最後に xmlCompleteHandler() メソッドが main() メソッドを呼び出し、表示リスト上のさまざまなコンポーネントインスタンスに加え、外部 FLV ファイルをロードするために使用される NetConnection および NetStream オブジェクトを設定します。

ユーザーインターフェイスの作成

ユーザーインターフェイスを構築するには、表示リストに 5 つの Button インスタンスをドラッグし、それぞれ playButton、pauseButton、stopButton、backButton、forwardButton というインスタンス名を与える必要があります。

これらの Button インスタンスのそれぞれに対し、次のコード例に示すように、click イベントのハンドラを割り当てる必要があります。

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

次のコードで示すように、buttonClickHandler() メソッドでは、どのボタンインスタンスがクリックされたかを判断するために switch ステートメントを使用します。

```
private function buttonClickHandler(event:MouseEvent):void
{
  switch (event.currentTarget)
  {
    case playButton:
      ns.resume();
      break;
    case pauseButton:
      ns.togglePause();
      break;
    case stopButton:
      ns.pause();
      ns.seek(0);
      break;
    case backButton:
```

```
        playPreviousVideo();
        break;
      case forwardButton:
        playNextVideo();
        break;
    }
}
```

次に、Slider インスタンスを表示リストに追加し、volumeSlider というインスタンス名を与えます。次のコードは、Slider インスタンスの liveDragging プロパティを true に設定し、Slider インスタンスの change イベントのイベントリスナーを定義します。

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

表示リストに ProgressBar インスタンスを追加し、positionBar というインスタンス名を与えます。次のコード例で示すように、mode プロパティを manual に設定します。

```
positionBar.mode = ProgressBarMode.MANUAL;
```

最後に、表示リストに Label インスタンスを追加し、positionLabel というインスタンス名を与えます。Label インスタンスの値は、Timer インスタンスによって設定されます。

Video オブジェクトのメタデータの監視

Flash Player がロードされたビデオのメタデータを検出すると、NetStream オブジェクトの client プロパティで onMetaData() コールバックハンドラが呼び出されます。次のコードではオブジェクトを初期化し、指定のコールバックハンドラを設定します。

```
client = new Object();
client.onMetaData = metadataHandler;
```

metadataHandler() メソッドは、コードの前部で指定されたメタプロパティにデータをコピーします。これにより、アプリケーション全体で、いつでも現在のビデオのメタデータにアクセスすることが可能になります。次に、ステージ上の Video オブジェクトは、メタデータから返されたサイズに合わせてサイズ変更されます。最後に、再生中のビデオのサイズに合わせて positionBar プログレスバーインスタンスが移動、サイズ変更されます。次のコードは、metadataHandler() メソッドを示します。

```
private function metadataHandler(metadataObj:Object):void
{
  meta = metadataObj;
  vid.width = meta.width;
  vid.height = meta.height;
  positionBar.move(vid.x, vid.y + vid.height);
  positionBar.width = vid.width;
}
```

Flash Video の動的なロード

Flash Video を動的にロードするには、アプリケーションでは NetConnection および NetStream オブジェクトを使用する必要があります。次のコードは NetConnection オブジェクトを作成し、connect() メソッドに null を渡します。null を指定すると、Flash Player は、Flash Media Server などのサーバーではなく、ローカルサーバー上のビデオに接続します。

次のコードは、NetConnection および NetStream インスタンスの両方を作成し、netStatus イベントのイベントリスナーを定義し、client プロパティに client オブジェクトを割り当てます。

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

ビデオのステータスが変更されるたびに netStatusHandler() メソッドが呼び出されます。ステータスとは、ビデオの再生の開始または停止、バッファリング、またはビデオストリームが見つからないなどの状態を指します。次のコードは netStatusHandler() イベントをリストします。

```
private function netStatusHandler(event:NetStatusEvent):void
{
  try
  {
    switch (event.info.code)
    {
      case "NetStream.Play.Start":
        t.start();
        break;
      case "NetStream.Play.StreamNotFound":
      case "NetStream.Play.Stop":
        t.stop();
        playNextVideo();
        break;
    }
  }
  catch (error:TypeError)
  {
    // エラーを無視します。
  }
}
```

前述のコードは、info オブジェクトのコードプロパティを評価し、コードが "NetStream.Play.Start"、"NetStream.Play.StreamNotFound"、または "NetStream.Play.Stop" のどれであるかをフィルタリングします。他のすべてのコードは無視されます。ネットストリームが開始すると、再生ヘッドを更新する Timer インスタンスがコードによって開始します。ネットストリームが見つからない場合、または停止している場合は、Timer インスタンスは停止し、アプリケーションはプレイリスト内の次のビデオの再生を試行します。

Timer が実行されるたびに、ProgressBar クラスの setProgress() メソッドを呼び出すことで positionBar プログレスバーインスタンスの現在の位置が更新され、Label インスタンスである positionLabel は、現在のビデオの経過時間と合計時間によって更新されます。

```
private function timerHandler(event:TimerEvent):void
{
  try
  {
    positionBar.setProgress(ns.time, meta.duration);
    positionLabel.text = ns.time.toFixed(1) + " of " meta.duration.toFixed(1) +
  " seconds";
  }
  catch (error:Error)
  {
    // このエラーを無視します。
  }
}
```

ビデオのボリュームの制御

動的にロードされたビデオのボリュームを制御するには、NetStream オブジェクトで soundTransform プロパティを設定します。ビデオジュークボックスアプリケーションでは、Slider インスタンスである volumeSlider の値を変更することでボリュームレベルを変更することができます。次のコードは、NetStream オブジェクトの soundTransform プロパティに設定されている SoundTransform オブジェクトに Slider コンポーネントの値を割り当てることで、ボリュームレベルを変更する方法を示します。

```
private function volumeChangeHandler(event:SliderEvent):void
{
  volumeTransform.volume = event.value;
  ns.soundTransform = volumeTransform;
}
```

ビデオ再生の制御

アプリケーションの残りの部分では、ビデオがビデオストリームの末尾に到達したとき、またはユーザーが前のビデオまたは次のビデオにスキップしたときに、ビデオの再生を制御します。

次のメソッドは、選択中のインデックスの XMLList からビデオ URL を取得します。

```
private function getVideo():String
{
  return videosXML[idx].@url;
}
```

playVideo() メソッドは、NetStream オブジェクトの play() メソッドを呼び出し、選択中のビデオをロードします。

```
private function playVideo():void
{
  var url:String = getVideo();
  ns.play(url);
}
```

playPreviousVideo() メソッドは、現在のビデオインデックスをデクリメントし、playVideo() メソッドを呼び出して新しいビデオファイルをロードし、プログレスバーを表示可能な状態に設定します。

```
private function playPreviousVideo():void
{
  if (idx > 0)
  {
    idx--;
    playVideo();
    positionBar.visible = true;
  }
}
```

最後のメソッド、playNextVideo() はビデオインデックスをインクリメントし、playVideo() メソッドを呼び出します。現在のビデオがプレイリストで最後のビデオであれば、Video オブジェクトで clear() メソッドが呼び出され、プログレスバーインスタンスの visible プロパティは false に設定されます。

```
private function playNextVideo():void
{
  if (idx < (videosXML.length() - 1))
  {
    idx++;
    playVideo();
    positionBar.visible = true;
  }
  else
  {
    idx++;
    vid.clear();
    positionBar.visible = false;
  }
}
```

第 20 章

サウンドの操作

20

ActionScript は没入型のインタラクティブなアプリケーション向けに作成されています。そして、没入性の高いアプリケーションの中でも見過ごされがちな要素と言えばサウンドです。サウンドをアプリケーションの中核として、ビデオゲームにサウンド効果を追加、アプリケーションユーザーインターフェイスにオーディオフィードバックを追加、またはインターネット上でロードされた MP3 ファイルを分析するプログラムを作成することができます。

この章では、外部オーディオファイルをロードし、SWF に埋め込まれたオーディオの操作方法について説明します。サウンド情報の視覚的表現を作成したりユーザーのマイクからサウンドをキャプチャするためのオーディオの操作方法を説明します。

目次

サウンドの操作の基礎 566
サウンドアーキテクチャについて 568
外部サウンドファイルのロード 570
埋め込みサウンドの操作 573
ストリーミングサウンドファイルの操作 574
サウンドの再生 575
サウンドのロード時および再生時のセキュリティ重要事項 578
サウンドボリュームとパンの制御 580
サウンドメタデータの操作 582
生のサウンドデータへのアクセス 583
サウンド入力のキャプチャ 586
例 : Podcast Player 591

サウンドの操作の基礎

サウンドの操作の概要

コンピュータではイメージをデジタル形式でエンコードし、コンピュータに保存し、さらに取得して画面上に表示することができます。それと同じように、サウンド情報をコンピュータで表示する形式であるデジタルオーディオをキャプチャしてエンコードし、それを保存したり、コンピュータに接続されたスピーカーで再生することもできます。再生方法の 1 つとして挙げられるのが、Adobe Flash Player と ActionScript の使用です。

サウンドデータがデジタル形式に変換されると、サウンドボリュームや、サウンドがステレオであるかモノラルサウンドであるかなどのさまざまな性質があります。ActionSctipt でサウンドを再生すると、それらの性質も調節ができます。つまり、サウンドを大きくしたり、一定の方向から聞こえるようにすることができます。

ActionScript でサウンドを制御する前に、Flash Player にサウンド情報をロードする必要があります。オーディオデータを Flash Player に取り入れ、ActionScript で操作するには 4 とおりの方法があります。MP3 などの外部サウンドファイルを SWF にロードする方法、SWF ファイルの作成時にそのファイルにサウンド情報を直接埋め込む方法、ユーザーのコンピュータに接続されたマイクからオーディオを入力する方法、そしてサーバーからストリーミングされたサウンドデータにアクセスする方法です。

外部サウンドファイルからサウンドデータをロードする場合、ロードが完全に終了していなくても、サウンドファイルを最初から再生することができます。

デジタルオーディオをエンコードするサウンドファイルにはさまざまな形式がありますが、ActionScript 3.0 および Flash Player では、MP3 形式で保存されるサウンドファイルをサポートしています。WAV や AIFF といった、他の形式のサウンドファイルを直接ロードまたは再生することはできません。

ActionScript でサウンドを操作していると、flash.media パッケージのいくつかのクラスを扱うことになります。Sound クラスは、サウンドファイルをロードし、再生を開始することでオーディオ情報にアクセスするために使用されるクラスです。サウンドの再生を開始すると、Flash Player は SoundChannel オブジェクトへのアクセスを可能にします。ロードされたオーディオファイルは、ユーザーのコンピュータで再生される複数のサウンドのうちの 1 つでしかないこともあるので、再生されるそれぞれのサウンドは独自の SoundChannel オブジェクトを使用します。すべての SoundChannel オブジェクトを合成して出力すると、そのサウンドがコンピュータのスピーカーから実際に流れます。この SoundChannel インスタンスを使用してサウンドのプロパティを制御し、再生を停止します。最後に、組み合わせられたオーディオを制御する場合、SoundMixer クラスを使用すると合成出力を制御することができます。

また、ActionScript でサウンドを操作しているときには、他のクラスも使用してより特定なタスクを実行することもできます。サウンド関連のすべてのクラスについては、568 ページの「サウンドアーキテクチャについて」を参照してください。

サウンドの操作に関する一般的なタスク

この章では、サウンドに関連して実行する次のタスクについて説明します。

- 外部 MP3 ファイルのロードと、ロードの進行状況のトラッキング
- サウンドの再生、一時停止、再開、および停止
- ロード中のストリーミングサウンドの再生
- サウンドボリュームとパンの操作
- MP3 ファイルからの ID3 メタデータの取得
- 生のサウンドの波形データの使用
- マイクからのサウンド入力のキャプチャと再生

重要な概念と用語

次の参照用の一覧に、この章で使用される重要な用語を示します。

- 振幅 : ゼロまたは平衡の状態から、サウンド波形上のポイントまでの距離です。
- ビットレート : サウンドファイルで 1 秒あたりにエンコードまたはストリーミングされるデータ量です。MP3 ファイルでは、ビットレートは千ビット / 秒 (kbps) で示されます。ビットレートが高くなると、通常は高品質なサウンド波形が生まれます。
- バッファリング : サウンドデータを再生する前に受信し、保存することです。
- MP3 : MPEG-1 オーディオレイヤー 3、つまり MP3 は、広く使用されているサウンド圧縮形式です。
- パン : ステレオ音場の左チャンネルと右チャンネルの間にオーディオ信号を配置することです。
- ピーク : 波形で一番高い地点です。
- サンプリングレート : デジタル信号を生成するために、アナログオーディオ信号から取り出した 1 秒あたりのサンプル数を示すものです。標準的なコンパクトディスクオーディオのサンプリングレートは、毎秒 44.1 kHz、つまり毎秒 44,100 サンプルです。
- ストリーミング : サウンドファイルまたはビデオファイルの後の方の部分をサーバーからロードしながら、ファイルの前の方の部分を再生することです。
- ボリューム : サウンドの音量です。
- 波形 : サウンド信号の振幅の推移を示すグラフです。

本章の例について

本章をお読みになる際は、いくつかのコード例を試すことができます。この章では ActionScript でのサウンドの操作について扱っているため、多くの例では、サウンドの再生、再生の停止、サウンドの調節といった、サウンドファイルの操作を実行します。この章の例をテストするには :

1. 新しい Flash ドキュメントを作成し、コンピュータに保存します。
2. タイムラインで最初のキーフレームを選択し、[アクション] パネルを開きます。
3. コード例のリストを [スクリプト] ペインにコピーします。
4. 外部サウンドファイルのロードが行われる場合は、コードに次のような行が含まれています。

   ```
   var req:URLRequest = new URLRequest("click.mp3");
   var s:Sound = new Sound(req);
   ```

   "click.mp3" はロードされているサウンドファイルの名前です。これらの例をテストするには、MP3 ファイルが必要です。MP3 ファイルは Flash ドキュメントと同じフォルダに入っている必要があります。その際、コードを変更して、コード内の名前の代わりに MP3 ファイルの名前を使用する必要があります。たとえば、上のコードでは "click.mp3" を mp3 ファイルの名前に変更します。
5. メインメニューから [コントロール]-[ムービープレビュー] を選択し、SWF ファイルを作成して例の出力をプレビュー (オーディオも再生) します。

オーディオの再生に加えて、一部の例は `trace()` 関数を使って値を表示します。これらの例をテストすると、[出力] パネルにその値の結果が表示されます。一部の例では画面にコンテンツが描画されるため、Flash Player のウィンドウにもコンテンツが表示されます。

本マニュアルのコード例のテストの詳細については、60 ページの「章内のコード例のテスト」を参照してください。

サウンドアーキテクチャについて

アプリケーションでは、次の 4 つのメインソースからサウンドデータをロードすることができます。

- 実行時にロードされた外部サウンドファイル
- アプリケーションの SWF ファイル内に埋め込まれたサウンドリソース
- ユーザーのシステムに接続されたマイクから出力されるサウンドデータ
- Flash Media Server などのリモートメディアサーバーからストリーミングされたサウンドデータ

サウンドデータを再生する前に完全にロードすることもできますが、ストリーミングすることもできます。後者の場合、データがロードされている間に再生が開始します。

ActionScript 3.0 および Flash Player は、MP3 形式で保存されたサウンドファイルに対応しています。WAV や AIFF といった、他の形式のサウンドファイルを直接ロードまたは再生することはできません。

Adobe Flash CS3 Professional を使うと、WAV または AIFF サウンドファイルをインポートして、アプリケーションの SWF ファイルに MP3 形式で埋め込むことができます。Flash オーサリングツールを使うと、埋め込まれたサウンドファイルを圧縮してファイルサイズを小さくすることができます。ただし、サイズが減少する代わりにサウンドの品質は低下します。詳細については、『Flash ユーザーガイド』の「サウンドの読み込み」を参照してください。

ActionScript 3.0 サウンドアーキテクチャは、flash.media パッケージの次のクラスを利用します。

クラス	説明
flash.media.Sound	Sound クラスは、サウンドのロード、基本的なサウンドプロパティの管理、サウンドの再生の開始を行います。
flash.media.SoundChannel	アプリケーションが Sound オブジェクトを再生すると、新しい SoundChannel オブジェクトが作成され、再生を制御します。SoundChannel オブジェクトは、サウンドの左再生チャンネルと右再生チャンネルのボリュームを制御します。再生されるそれぞれのサウンドには、独自の SoundChannel オブジェクトがあります。
flash.media.SoundLoaderContext	SoundLoaderContext クラスは、サウンドのロード中に何秒間バッファリングするか、また、ロード中に Flash Player がサーバーからクロスドメインポリシーファイルを探すかどうかを決定します。SoundLoaderContext オブジェクトは、`Sound.load()` メソッドのパラメータとして使用されます。
flash.media.SoundMixer	SoundMixer クラスは、アプリケーション内のすべてのサウンドに関連する再生およびセキュリティのプロパティを制御します。実際、共通の SoundMixer オブジェクトを通じて複数のサウンドチャンネルが合成されるので、SoundMixer オブジェクトのプロパティ値は、再生中のすべての SoundChannel オブジェクトに影響を与えます。
flash.media.SoundTransform	SoundTransform クラスには、サウンドボリュームとパンを制御する値があります。SoundTransform オブジェクトは、個々の SoundChannel オブジェクト、グローバルな SoundMixer オブジェクト、Microphone オブジェクトなどに適用されます。

クラス	説明
flash.media.ID3Info	ID3Info オブジェクトには、通常は MP3 サウンドファイルに保存される ID3 メタデータ情報を表すプロパティが含まれています。
flash.media.Microphone	Microphone クラスは、ユーザーのコンピュータに接続されたマイクやその他のサウンド入力デバイスを表します。マイクからのオーディオ入力はローカルのスピーカーに転送されるか、リモートサーバーに送信されます。Microphone オブジェクトは、独自のサウンドストリームのゲイン、サンプリングレート、およびその他の性質を制御します。

ロードされ、再生される各サウンドには、Sound クラスと SoundChannel クラスの独自のインスタンスが必要です。複数の SoundChannel インスタンスからの出力は、再生中にグローバルな SoundMixer クラスによって合成されます。

Sound、SoundChannel、および SoundMixer クラスは、マイクから拾われたサウンドデータ、または Flash Media Server などのストリーミングメディアサーバーから取得されたサウンドデータには使用されません。

外部サウンドファイルのロード

Sound クラスの各インスタンスは、特定のサウンドリソースをロードし、再生をトリガーするために使用されるものです。アプリケーションでは、Sound オブジェクトを再利用してサウンドをロードすることはできません。新しいサウンドリソースをロードする必要がある場合は、新しい Sound オブジェクトを作成する必要があります。

ボタンに関連付けるクリック音など、小さなサウンドファイルをロードする場合、アプリケーションは新しい Sound オブジェクトを作成し、次のようにサウンドファイルを自動的にロードできます。

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

`Sound()` コンストラクタは、最初のパラメータとして URLRequest オブジェクトを受け入れます。URLRequest パラメータの値が与えられると、新しい Sound オブジェクトは指定のサウンドリソースのロードを自動的に開始します。

どのような場合でも、アプリケーションはサウンドのロードの進行状況に注意し、ロード中のエラーを確認する必要があります。たとえば、クリックサウンドが比較的大きい場合、サウンドをトリガーするボタンがクリックされても、サウンドが完全にロードされていないこともあります。ロードされていないサウンドを再生しようとすると、ランタイムエラーが発生する可能性があります。そのため、サウンドが完全にロードされてから、サウンドの再生を開始するアクションを実行する方が安全です。

Sound オブジェクトは、サウンドのロードプロセス中にいくつかのイベントを送出します。アプリケーションはこれらのイベントを監視してロードの進行状況をトラッキングし、サウンドが再生される前に完全にロードするようにします。次の表は、Sound オブジェクトによって送出されるイベントをまとめたものです。

イベント	説明
open (`Event.OPEN`)	サウンドのロード処理が開始する直前に送出されます。
progress (`ProgressEvent.PROGRESS`)	データがファイルまたはストリームから受信されると、サウンドのロードプロセス中に定期的に送出されます。
id3 (`Event.ID3`)	ID3 データが MP3 サウンドで使用できるようになると送出されます。
complete (`Event.COMPLETE`)	すべてのサウンドリソースのデータがロードされると送出されます。
ioError (`IOErrorEvent.IO_ERROR`)	サウンドファイルが見つからない場合、またはすべてのサウンドデータが受信される前にロードプロセスが中断されると送出されます。

次のコードは、サウンドのロードが完了した後のサウンドの再生方法を示します。

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

このコード例では、まず、URLRequest パラメータの初期値を与えずに新しい Sound オブジェクトを作成します。その後、Sound オブジェクトから `Event.COMPLETE` イベントを監視し、すべてのサウンドデータがロードされると `onSoundLoaded()` メソッドが実行されます。次に、サウンドファイルの新しい URLRequest 値で `Sound.load()` メソッドを呼び出します。

サウンドのロードが完了すると `onSoundLoaded()` メソッドが実行されます。Event オブジェクトのターゲットプロパティは、Sound オブジェクトに対する参照です。Sound オブジェクトの `play()` メソッドを呼び出すと、サウンドの再生が開始します。

サウンドのロードプロセスの監視

サウンドファイルのサイズが大きく、ロードに時間がかかることがあります。Flash Player では、サウンドを完全にロードしなくてもアプリケーションで再生させることができます。その際に、サウンドデータがどれだけロード済みであるか、また、どれだけのサウンドが再生済みであるかを示すこともできます。

Sound クラスは、ProgressEvent.PROGRESS および Event.COMPLETE の 2 つのイベントを送出するので、サウンドのロードの進行状況を比較的簡単に表示することができます。次の例では、これらのイベントを使用し、ロード中のサウンドの進行情報を表示する方法を示します。

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
  var loadedPct:uint =
    Math.round(100 * (event.bytesLoaded / event.bytesTotal));
  trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event:Event):void
{
  var localSound:Sound = event.target as Sound;
  localSound.play();
}
function onIOError(event:IOErrorEvent)
{
  trace("The sound could not be loaded: " + event.text);
}
```

このコードは、まず Sound オブジェクトを作成し、次に ProgressEvent.PROGRESS および Event.COMPLETE イベントのリスナーをこのオブジェクトに追加します。Sound.load() メソッドが呼び出され、サウンドファイルから最初のデータが受信されると、ProgressEvent.PROGRESS イベントが発生し、onSoundLoadProgress() メソッドがトリガーされます。

ロード済みのサウンドデータの割合は、ProgressEvent オブジェクトの bytesLoaded プロパティの値を、bytesTotal プロパティの値で除算した数に相当します。同じ bytesLoaded および bytesTotal プロパティは Sound オブジェクトにもあります。上記の例は、サウンドのロードの進行状況に関するメッセージを表示するだけですが、bytesLoaded および bytesTotal 値を使用し、Adobe Flex 2 Framework または Flash オーサリングツールに付属しているようなプログレスバーコンポーネントを簡単に更新することもできます。

この例では、サウンドファイルのロード中に、アプリケーションがどのようにエラーを認識し、対処するかも示します。たとえば、指定のファイル名を持つサウンドファイルが見つからない場合は、Sound オブジェクトによって Event.IO_ERROR イベントが送出されます。前述のコードでは、onIOError() メソッドが実行され、エラー発生時に簡単なエラーメッセージを表示します。

埋め込みサウンドの操作

外部ファイルからサウンドをロードする代わりに埋め込みサウンドを使用することは、ボタンがクリックされたときに再生されるサウンドなど、アプリケーションのユーザーインターフェイス内のインジケータとして使用される小さなファイルにおいて非常に便利です。

アプリケーションにサウンドファイルを埋め込むと、その結果として生成される SWF ファイルのサイズが、サウンドファイルの分だけ増加します。つまり、アプリケーションに大きなサウンドファイルを埋め込むと、SWF ファイルが望ましくないサイズに増加する可能性があるということです。

サウンドファイルをアプリケーションの SWF ファイルに埋め込む方法は、開発環境によって異なります。

Flash での埋め込みサウンドファイルの使用

Flash オーサリングツールでは、複数のサウンド形式のサウンドをインポートして、ライブラリにシンボルとして保存できます。保存したサウンドは、タイムラインのフレームやボタン状態のフレームに割り当てる、ビヘイビアで使用する、ActionScript コードで直接使用する、というように使用できます。このセクションでは、Flash オーサリングツールで ActionScript コードを処理する際に埋め込みサウンドを使用する方法について説明します。Flash で埋め込みサウンドを使用するその他の方法については、『Flash ユーザーガイド』の「サウンドの読み込み」を参照してください。

Flash ムービーにサウンドファイルを埋め込むには :

1. [ファイル]-[読み込み]-[ライブラリに読み込み] を選択し、サウンドファイルを選択して読み込みます。
2. [ライブラリ] パネルで読み込んだファイルの名前を右クリックし、[プロパティ] を選択します。[ActionScript に書き出し] チェックボックスをクリックします。

3. [クラス] フィールドに、ActionScript でこの埋め込みサウンドを参照する際に使う名前を入力します。デフォルトでは、このフィールド内のサウンドファイル名が使用されます。"DrumSound.mp3" のようにファイル名にピリオドが含まれている場合は、"DrumSound" のように変更する必要があります。ActionScript では、クラス名でピリオドを使用できません。[基本クラス] フィールドには引き続き flash.media.Sound と表示されます。
4. [OK] をクリックします。このクラスの定義がクラスパスで見つからなかったことを示すダイアログボックスが表示される場合もあります。[OK] をクリックして続行してください。アプリケーションのクラスパスに含まれるいずれのクラス名とも一致しないクラス名を入力した場合は、flash.media.Sound から継承する新しいクラスが自動的に生成されます。
5. 埋め込みサウンドを使用するには、ActionScript でそのサウンドのクラス名を参照します。例えば、次のコードでは最初に、自動生成された DrumSound クラスの新しいインスタンスが作成されます。

```
var drum:DrumSound = new DrumSound();
var channel:SoundChannel = drum.play();
```

DrumSound は、flash.media.Sound クラスのサブクラスなので、上記のとおり play() メソッドを含む Sound クラスのメソッドとプロパティを継承します。

ストリーミングサウンドファイルの操作

サウンドファイルまたはビデオファイルのデータをロード中にデータを再生することを、ストリーミングと呼びます。データが完全にロードされなくてもサウンドを聴くことができるので、リモートサーバーからロードされる外部サウンドファイルの多くはストリーミングされます。

SoundMixer.bufferTime プロパティは、サウンドが再生される前に Flash Player が収集するサウンドデータの長さを、ミリ秒単位で示します。つまり、bufferTime プロパティが 5000 に設定されていると、Flash Player はサウンドが再生されるまでに、少なくとも 5000 ミリ秒分のデータをサウンドファイルからロードします。デフォルトの SoundMixer.bufferTime 値は 1000 です。

アプリケーションは、サウンドのロード中に新しい bufferTime 値を明示的に指定することで、個々のサウンドのグローバルな SoundMixer.bufferTime 値をオーバーライドすることができます。デフォルトのバッファ時間をオーバーライドするには、次のように、まず SoundLoaderContext クラスの新しいインスタンスを作成し、その bufferTime プロパティを設定し、それを Sound.load() メソッドのパラメータとして渡します。

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

サウンドの再生時において、Flash Player はサウンドバッファを同じサイズか、それ以上のサイズに維持します。サウンドデータが再生速度よりも速くロードされると、再生は中断することなく続行します。しかし、ネットワークの制限によりデータのロード速度が遅くなると、再生ヘッドがサウンドバッファの最後に到達することがあります。このような場合、再生は中断されますが、サウンドデータがさらにロードされると自動的に再開されます。

Flash Player がデータのロードを待機しているために再生が中断しているのかどうかを確認するには、`Sound.isBuffering` プロパティを使用します。

サウンドの再生

ロードされたサウンドを再生するには、次のように、Sound オブジェクトの `Sound.play()` メソッドを呼び出すだけです。

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

ActionScript 3.0 を使用してサウンドを再生しているときに、次の操作を実行することができます。

- 指定の開始位置からサウンドを再生します。
- サウンドを一時停止し、後で同じ位置から再生を再開します。
- サウンドの再生がいつ終了するかを正確に把握します。
- サウンド再生の進行状況をトラッキングします。
- サウンドの再生中にボリュームまたはパンを変更します。

再生中にこれらの操作を実行するには、SoundChannel、SoundMixer、および SoundTransform クラスを使用します。

SoundChannel クラスは、単独のサウンドの再生を制御します。`SoundChannel.position` プロパティは、再生中のサウンドデータの現在位置を示す再生ヘッドと考えることができます。

アプリケーションが `Sound.play()` メソッドを呼び出すと、再生を制御するための SoundChannel クラスの新しいインスタンスが作成されます。

ミリ秒単位で指定した開始位置を `Sound.play()` メソッドの `startTime` パラメータとして渡すと、その位置からサウンドを再生できます。また、`Sound.play()` メソッドの loops パラメータに数値を渡すことで、サウンドの連続繰り返し回数を指定することもできます。

`startTime` パラメータと `loops` パラメータの両方と共に `Sound.play()` メソッドが呼び出されると、次のコードに示すように、同じ位置からサウンドが繰り返し再生されます。

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

この例では、サウンド開始の 1 秒後から、3 回連続で再生されます。

サウンドの一時停止と再開

曲やポッドキャストなどの長いサウンドを再生していると、ユーザーはサウンドの再生を一時停止し、再開することがあります。サウンドは、ActionScript での再生中は文字どおり一時停止することはできません。サウンドは停止されるだけです。サウンドの再生はどの時点からでも再開できます。サウンドが停止された位置を記録し、後でその位置からサウンドの再生を再開することができます。

たとえば、次のようなサウンドファイルをロードし、再生するコードがあるとします。

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

サウンドが再生中に、SoundChannel.position プロパティは再生中のサウンドファイルの位置を示します。次に示すように、アプリケーションは、サウンドの再生が停止したときの位置の値を保存することができます。

```
var pausePosition:int = channel.position;
channel.stop();
```

サウンドの再生を再開するには、前に保存された位置の値を渡します。

```
channel = snd.play(pausePosition);
```

再生の監視

アプリケーションでは、別のサウンドを再生、または前回の再生で使用されたリソースの一部をクリーンアップするため、サウンドの再生がいつ停止されるかを把握する必要がある場合もあります。SoundChannel クラスは、サウンドが再生を終了したときに Event.SOUND_COMPLETE イベントを送出します。アプリケーションは、このイベントを監視し、次のような適切なアクションを実行することができます。

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound("smallSound.mp3");
var channel:SoundChannel = snd.play();
s.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
  trace("The sound has finished playing.");
}
```

SoundChannel クラスは、再生中に progress イベントを送出しません。再生の進行状況をレポートするには、アプリケーションで独自のタイミング制御メカニズムを設定し、サウンド再生ヘッドの位置をトラッキングすることができます。

再生済みのサウンドの割合を算出するには、SoundChannel.position プロパティの値を、再生中のサウンドデータの長さで除算します。

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

ただし、このデータが正確な再生率をレポートできるのは、再生が開始する前にサウンドデータが完全にロードされていた場合のみです。Sound.length プロパティは、サウンドファイルの最終的なサイズではなく、ロードされたサウンドデータの長さを示します。ロード中のストリーミングサウンドの再生の進行状況をトラッキングするには、アプリケーションでサウンドファイル全体の最終的なサイズを予測し、その値を計算に使用する必要があります。サウンドデータの最終的な長さを予測するには、次のように Sound オブジェクトの bytesLoaded プロパティおよび bytesTotal プロパティを使用します。

```
var estimatedLength:int =
  Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

次のコードは大きなサウンドファイルをロードし、再生の進行状況を示すタイミング制御メカニズムとして Event.ENTER_FRAME イベントを使用します。ここでは再生率を定期的にレポートします。再生率は、現在位置の値をサウンドデータの合計長で除算することで算出されます。

このコードでは、

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
  URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

function onEnterFrame(event:Event):void
{
  var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
  var playbackPercent:uint =
    Math.round(100 * (channel.position / estimatedLength));
  trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
  trace("The sound has finished playing.");
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

サウンドデータのロードが開始すると、snd.play() メソッドを呼び出し、生成される SoundChannel オブジェクトを channel 変数に保存します。次に、Event.ENTER_FRAME イベントのリスナーをメインアプリケーションに追加し、さらに、再生が完了したときに発生する Event.SOUND_COMPLETE イベントのリスナーを SoundChannel オブジェクトに追加します。

アプリケーションがアニメーションの新しいフレームに到達するたびに、onEnterFrame() メソッドが呼び出されます。onEnterFrame() メソッドは、ロード済みのデータ量に基づいてサウンドファイルの合計長を予測し、現在の再生率を算出して表示します。

サウンド全体が再生されると onPlaybackComplete() メソッドが実行され、Event.ENTER_FRAME イベントのリスナーが削除されるので、再生が完了すると進行状況の更新情報は表示されなくなります。

Event.ENTER_FRAME イベントは、1 秒間に何度も送出することができます。場合によっては、頻繁に再生の進行状況を表示したくないこともあります。そのような場合は、flash.util.Timer クラスを使用して独自のタイミング制御メカニズムを設定することができます。詳しくは 179 ページの「日付と時刻の操作」を参照してください。

ストリーミングサウンドの停止

ストリーミング中のサウンドの再生処理、つまり、再生中にロードされているサウンドの再生処理では、予測できないことが起こる場合があります。ストリーミングサウンドを再生している SoundChannel インスタンスで SoundChannel.stop() メソッドが呼び出されると、サウンドの再生は 1 フレーム分停止し、次のフレームでサウンドを最初から再開します。これは、サウンドのロード処理が実行中であるために発生する現象です。ストリーミングサウンドのロードと再生の両方を停止するには、Sound.close() メソッドを呼び出します。

サウンドのロード時および再生時のセキュリティ重要事項

アプリケーションによるサウンドデータへのアクセスは、Flash Player のセキュリティモデルによって制限されることがあります。各サウンドは、2 つの異なるセキュリティサンドボックスの制限を受けます。コンテンツそのもののサンドボックス (" コンテンツサンドボックス ")、そしてサウンドをロードおよび再生するアプリケーションまたはオブジェクトのサンドボックス ("オーナーサンドボックス ") です。Flash Player セキュリティモデルの一般的な情報と、サンドボックスの定義については 719 ページの「Flash Player セキュリティ」を参照してください。

コンテンツサンドボックスは、詳細なサウンドデータを id3 プロパティから抽出するか、または SoundMixer.computeSpectrum() メソッドから抽出するかを制御します。サウンドファイルそのもののロードまたは再生を制限するわけではありません。

サウンドファイルの発生ドメインによってコンテンツサンドボックスのセキュリティ制限が決定します。一般的に、サウンドファイルがアプリケーションの SWF ファイル、またはサウンドファイルをロードするオブジェクトと同じドメインまたはフォルダにある場合、アプリケーションまたはオブジェクトには、そのサウンドファイルに対するフルアクセス権が与えられます。サウンドがアプリケーションとは異なるドメインから発生していても、クロスドメインポリシーファイルを使用することで、コンテンツサンドボックス内に取り入れることができます。

アプリケーションは、`Sound.load()` メソッドのパラメータとして `checkPolicyFile` プロパティを使用することで、SoundLoaderContext オブジェクトを渡すことができます。`checkPolicyFile` プロパティを `true` に設定すると、Flash Player は、サウンドがロードされているサーバーにあるクロスドメインポリシーファイルを探します。クロスドメインポリシーファイルが存在し、ロードされる SWF ファイルのドメインへのアクセス権が付与される場合は、SWF は、サウンドファイルのロード、Sound オブジェクトの `id3` プロパティへのアクセス、およびロードされたサウンドの `SoundMixer.computeSpectrum()` メソッドの呼び出しができます。

オーナーサンドボックスは、サウンドのローカル再生を制御します。サウンドの再生を開始するアプリケーションまたはオブジェクトによってオーナーサンドボックスが定義されます。

`SoundMixer.stopAll()` メソッドは、次の基準を満たしている場合、再生中のすべての SoundChannel オブジェクトのサウンドを無音にします。

- サウンドは、同じオーナーサンドボックス内のオブジェクトによって開始されています。
- サウンドは、アプリケーションのドメインにアクセス権を付与するクロスドメインポリシーファイルか、`SoundMixer.stopAll()` メソッドを呼び出すメソッドを持つソースから発生しています。

`SoundMixer.stopAll()` メソッドが再生中のすべてのサウンドを実際に停止するかどうかを確認するために、アプリケーションで `SoundMixer.areSoundsInaccessible()` メソッドを呼び出すことができます。このメソッドが `true` の値を返すと、再生中のサウンドの一部は現在のオーナーサンドボックスの制御の対象外にあるので、`SoundMixer.stopAll()` メソッドによって停止することはできません。

`SoundMixer.stopAll()` メソッドは、外部ファイルからロードされたすべてのサウンドで、再生ヘッドの動作を停止します。ただし、FLA ファイルに埋め込まれ、Flash オーサリングツールによってタイムラインのフレームに関連付けられたサウンドは、アニメーションが新しいフレームに移動すると再生を再開することがあります。

サウンドボリュームとパンの制御

各 SoundChannel オブジェクトは、サウンドの左右のステレオチャンネルを制御します。MP3 サウンドがモノラルサウンドである場合、SoundChannel オブジェクトの左右のステレオチャンネルの波形は同一になります。

SoundChannel オブジェクトの `leftPeak` プロパティおよび `rightPeak` プロパティを使用すると、再生中のサウンドの各ステレオチャンネルの振幅を確認することができます。これらのプロパティは、サウンド波形そのもののピーク振幅を示します。実際の再生ボリュームを表すわけではありません。実際の再生ボリュームは、サウンド波形の振幅と、SoundChannel オブジェクトと SoundMixer クラスに設定されたボリューム値の関数です。

SoundChannel オブジェクトの pan プロパティは、再生中に左右それぞれに異なるボリュームレベルを指定するために使用されます。pan プロパティには、-1 から 1 までの値を指定することができます。ここで -1 とは、左チャンネルが最大ボリュームで再生され、右チャンネルが無音になること、そして 1 は右チャンネルが最大ボリュームで再生され、左チャンネルが無音であることを意味します。-1 から 1 までの数値は、左右のチャンネル値に対して相対値を設定します。値が 0 であると、両方のチャンネルで均衡が取れた中程度のボリュームレベルで再生されます。

次のコード例では、ボリューム値 0.6、パン値 -1 で SoundTransform オブジェクトを作成します。つまり、左チャンネルのボリュームが最大、右チャンネルのボリュームは無音です。SoundTransform オブジェクトが `play()` メソッドのパラメータとして渡されます。そうすると SoundTransform オブジェクトは、再生を制御するための新しい SoundChannel オブジェクトに適用されます。

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

SoundTransform オブジェクトの `pan` または `volume` プロパティを設定し、このオブジェクトを SoundChannel オブジェクトの `soundTransform` プロパティとして適用することで、サウンドの再生中にボリュームとパンを変更することができます。

また、次の例で示すように、SoundMixer クラスの `soundTransform` プロパティを使用することで、すべてのサウンドのグローバルボリューム値とパン値を一度に設定することもできます。

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

SoundTransform オブジェクトを使用し、Microphone オブジェクト (586 ページの「サウンド入力のキャプチャ」を参照)、Sprite オブジェクト、および SimpleButton オブジェクトのボリューム値とパン値を設定することも可能です。

次の例では、サウンドの再生中に左右のチャンネル間でサウンドを交互にパンする方法を示します。

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;
```

```
var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;

var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
  trans.pan = Math.sin(panCounter);
  channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
  panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

このコードは、まずサウンドファイルをロードし、次に新しい SoundTransform オブジェクトを作成します。このとき、ボリュームを 1 (フルボリューム) に、パンを 0 (左右で均衡) に設定します。次に SoundTransform オブジェクトをパラメータとして渡し、`snd.play()` メソッドを呼び出します。

サウンドの再生中は、`onEnterFrame()` メソッドが繰り返し実行されます。`onEnterFrame()` メソッドは、`Math.sin()` 関数を使用して -1 から 1 までの値を生成します。この値の範囲は、`SoundTransform.pan` プロパティで許容される値の範囲となります。SoundTransform オブジェクトの `pan` プロパティは新しい値に設定され、チャンネルの `soundTransform` プロパティは、変更された SoundTransform オブジェクトを使用するよう設定されます。

この例を実行する場合は、ファイル名 bigSound.mp3 をローカルの MP3 ファイルの名前に変更します。その後、この例を実行します。左チャンネルのボリュームが大きくなると右チャンネルのボリュームが小さくなり、反対に、左チャンネルのボリュームが小さくなると右チャンネルのボリュームが大きくなります。

この例では、SoundMixer クラスの `soundTransform` プロパティを設定することで同じ効果を得ることができます。ただし、これにより、SoundChannel オブジェクトによって再生されている単独のサウンドだけではなく、再生中のすべてのサウンドのパンが影響を受けます。

サウンドメタデータの操作

MP3 形式を使用するサウンドファイルには、サウンドに関する追加データが ID3 タグの形式で含まれることがあります。

すべての MP3 ファイルに ID3 メタデータが含まれるわけではありません。Sound オブジェクトが MP3 サウンドファイルをロードすると、サウンドファイルに ID3 メタデータが含まれる場合でも、Event.ID3 イベントが送出されます。ランタイムエラーを回避するために、アプリケーションは、ロードされたサウンドの Sound.id3 プロパティにアクセスする前に Event.ID3 イベントが受信されるまで待機する必要があります。

次のコードは、サウンドファイルの ID3 メタデータがロードされたときにどのように認識されるかを示します。

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
  var id3:ID3Info = event.target.id3;

  trace("Received ID3 Info:");
  for (var propName:String in id3)
  {
    trace(propName + " = " + id3[propName]);
  }
}
```

このコードは、まず Sound オブジェクトを作成し、Event.ID3 イベントを監視するよう指示します。サウンドファイルの ID3 メタデータがロードされると、onID3InfoReceived() メソッドが呼び出されます。onID3InfoReceived() メソッドに渡される Event オブジェクトのターゲットは元の Sound オブジェクトなので、メソッドは Sound オブジェクトの id3 プロパティを取得し、指定のすべてのプロパティで反復処理を実行して値をトレースします。

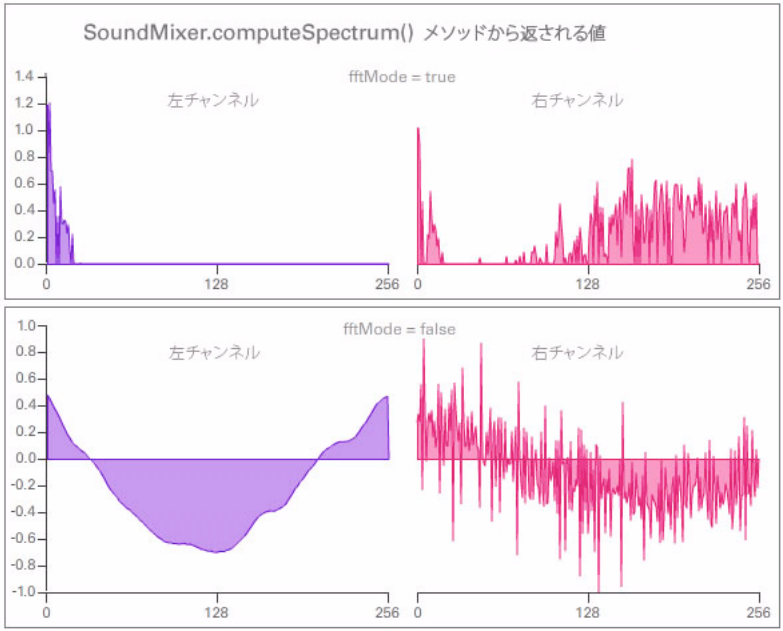
生のサウンドデータへのアクセス

SoundMixer.computeSpectrum() メソッドを使用すると、アプリケーションは再生中の波形の生のサウンドデータを読み取ることができます。SoundChannel オブジェクトが再生中であれば、SoundMixer.computeSpectrum() メソッドはすべての SoundChannel オブジェクトを合成したサウンドデータを示します。

サウンドデータは、512 バイトのデータを含む ByteArray オブジェクトとして返されます。このデータはそれぞれ -1 から 1 までの浮動小数点値を持ちます。これらの値は、再生中のサウンド波形のポイントの振幅を示します。値は 256 バイトずつ 2 グループに分けて送信されます。うち 1 つめのグループは左ステレオチャンネル用、そしてもう 1 つのグループは右ステレオチャンネル用です。

FFTMode パラメータが true に設定されている場合は、SoundMixer.computeSpectrum() メソッドは、波形データではなく周波数スペクトルを返します。周波数スペクトルとは、最も低い周波数から最も高い周波数までのサウンド周波数で配列した振幅を示します。波形データを周波数スペクトルデータに変換するために、高速フーリエ変換 (FFT) が使用されます。その結果として生成される周波数スペクトルの値の範囲は、0 からおよそ 1.414 (2 の平方根) までです。

次の図は、FFTMode パラメータが true に設定されているときと false に設定されているときに computeSpectrum() メソッドから返されるデータを比較します。この図でデータが使用されたサウンドでは、左チャンネルには大音量のベースサウンドが、右チャンネルにはドラム音が流れています。



computeSpectrum() メソッドは、低いビットレートで再サンプリングされたデータも返すことができます。通常、これによって細部が失われますが、よりスムーズな波形データまたは周波数データを得ることができます。stretchFactor パラメータは、computeSpectrum() メソッドデータがサンプリングされるレートを制御します。stretchFactor パラメータがデフォルトの 0 に設定されていると、サウンドデータは 44.1 KHz のレートでサンプリングされます。stretchFactor パラメータの値が増えるに従い、レートは半割になります。つまり、値が 1 であればレートは 22.05 KHz、値が 2 であればレートは 11.025 KHz となり、以下同様です。computeSpectrum() メソッドは、stretchFactor 値が高ければ、ステレオチャンネルごとに 256 バイトを返します。

SoundMixer.computeSpectrum() メソッドにもいくつか制限があります。

- マイクまたは RTMP ストリームからのサウンドデータはグローバルな SoundMixer オブジェクトを通過しないので、SoundMixer.computeSpectrum() メソッドはこれらのソースからデータを返しません。
- 再生中のサウンドが現在のコンテンツサンドボックス外から発生している場合は、セキュリティ制限により、SoundMixer.computeSpectrum() メソッドはエラーをスローします。SoundMixer.computeSpectrum() メソッドのセキュリティ制限については、578 ページの「サウンドのロード時および再生時のセキュリティ重要事項」および 746 ページの「ロードされたメディアへのデータとしてのアクセス」を参照してください。

シンプルなサウンドビジュアライザの構築

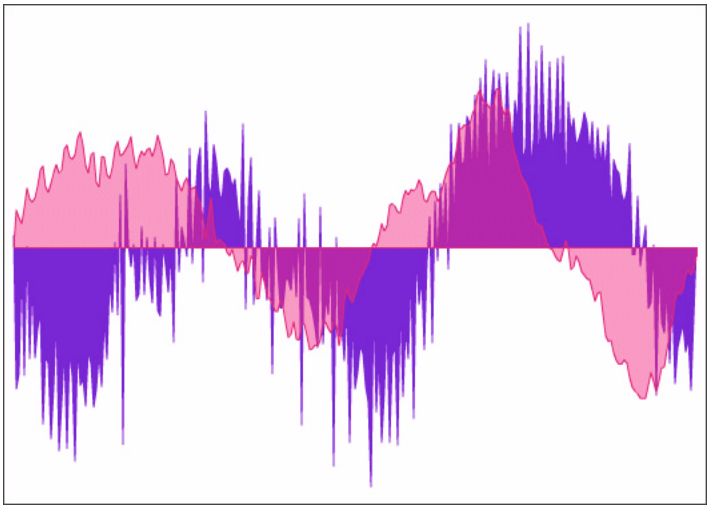
次の例は、SoundMixer.computeSpectrum() メソッドを使用し、各フレームでアニメーション化されるサウンド波形のチャートを示します。

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();
```

```
function onEnterFrame(event:Event):void
{
  SoundMixer.computeSpectrum(bytes, false, 0);

  var g:Graphics = this.graphics;

  g.clear();
  g.lineStyle(0, 0x6600CC);
  g.beginFill(0x6600CC);
  g.moveTo(0, PLOT_HEIGHT);

  var n:Number = 0;

  // left channel
  for (var i:int = 0; i < CHANNEL_LENGTH; i++)
  {
    n = (bytes.readFloat() * PLOT_HEIGHT);
    g.lineTo(i * 2, PLOT_HEIGHT - n);
  }
  g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
  g.endFill();

  // right channel
  g.lineStyle(0, 0xCC0066);
  g.beginFill(0xCC0066, 0.5);
  g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

  for (i = CHANNEL_LENGTH; i > 0; i--)
  {
    n = (bytes.readFloat() * PLOT_HEIGHT);
    g.lineTo(i * 2, PLOT_HEIGHT - n);
  }
  g.lineTo(0, PLOT_HEIGHT);
  g.endFill();
}

function onPlaybackComplete(event:Event)
{
  removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

この例では、まずサウンドファイルをロードして再生し、次にサウンドの再生中に onEnterFrame() メソッドをトリガーする Event.ENTER_FRAME イベントを監視します。onEnterFrame() メソッドは、サウンド波形データを bytes ByteArray オブジェクトに保存する SoundMixer.computeSpectrum() メソッドを呼び出します。

サウンド波形は、ベクター描画 API によってプロットされます。for ループは、左ステレオチャンネルを表す最初の 256 バイトのデータ値を順次実行し、Graphics.lineTo() メソッドを使用してポイントとポイントの間に線を描画します。2 つめの for ループは、次の 256 バイトのデータ値を順次実行し、今回は右から左へと逆順にプロットします。この結果として生成される波形プロットから、次に示すような面白い鏡像効果が生まれます。



サウンド入力のキャプチャ

Microphone クラスにより、アプリケーションはユーザーのシステム上にあるマイクまたはその他のサウンド入力デバイスに接続し、入力されたオーディオをシステムのスピーカーにブロードキャストしたり、オーディオデータを Flash Media Server などのリモートサーバーに送信できます。

マイクへのアクセス

Microphone クラスには、コンストラクタメソッドはありません。代わりに、次のように静的な Microphone.getMicrophone() メソッドを使用し、新しい Microphone インスタンスを取得します。

```
var mic:Microphone = Microphone.getMicrophone();
```

パラメータなしで Microphone.getMicrophone() メソッドを呼び出すと、ユーザーのシステムで検出された最初のサウンド入力デバイスが返されます。

システムには複数のサウンド入力デバイスを接続することができます。アプリケーションで Microphone.names プロパティを使用すると、使用可能なすべてのサウンド入力デバイスの名前の配列を取得することができます。その後、配列内のデバイス名のインデックス値を一致させる index パラメータにより Microphone.getMicrophone() メソッドを呼び出します。

システムには、マイクまたはその他のサウンド入力デバイスが接続されていないこともあります。Microphone.names プロパティまたは Microphone.getMicrophone() メソッドを使用すると、サウンド入力デバイスがインストールされているかどうかを確認することができます。サウンド入力デバイスがインストールされていない場合は、names 配列の長さはゼロになり、getMicrophone() メソッドは null 値を返します。

アプリケーションが Microphone.getMicrophone() メソッドを呼び出すと、Flash Player は [設定] ダイアログボックスを表示します。ここでは、Flash Player がシステムのカメラおよびマイクにアクセスできるかどうかを指定することができます。このダイアログの [許可] または [拒否] ボタンをクリックすると、StatusEvent が送出されます。その StatusEvent インスタンスの code プロパティは、次のように、マイクのアクセスが許可または拒否されているかを示します。

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
  if (event.code == "Microphone.Unmuted")
  {
    trace("Microphone access was allowed.");
  }
  else if (event.code == "Microphone.Muted")
  (
    trace("Microphone access was denied.");
  }
}
```

アクセスが許可されている場合は、StatusEvent.code プロパティに "Microphone.Unmuted" があり、アクセスが拒否されている場合は "Microphone.Muted" があります。

メモ

Microphone.muted プロパティは、ユーザーがマイクのアクセスを許可した場合は true、拒否している場合は false にそれぞれ設定されます。ただし、muted プロパティは、StatusEvent が送出されるまでは Microphone インスタンスに設定されないので、Microphone.muted プロパティを確認する前に StatusEvent.STATUS イベントが送出されるまで待機する必要があります。

マイクオーディオのローカルスピーカーへのルーティング

マイクからのオーディオ入力をローカルシステムスピーカーに転送するには、Microphone.setLoopback() メソッドを true のパラメータ値で呼び出します。

ローカルマイクからのサウンドがローカルスピーカーに転送されると、オーディオフィードバックループが発生する恐れがあります。これにより、きしんだような音が鳴り、サウンドハードウェアが損傷を受ける可能性もあります。Microphone.setUseEchoSuppression() メソッドを、パラメータ値 true で呼び出すとオーディオフィードバック発生のリスクが少なくなりますが、完全になくなるわけではありません。ヘッドホンなど、スピーカー以外のものからサウンドが出力されていることが確実に分かっている場合を除き、Microphone.setLoopback(true) を呼び出す前に必ず Microphone.setUseEchoSuppression(true) を呼び出すことをお勧めします。

次のコードは、ローカルマイクからローカルシステムスピーカーにオーディオを転送する方法を示します。

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

マイクオーディオの変更

アプリケーションは、マイクから入力されるオーディオデータを 2 つの方法で変更することができます。1 つ目は、入力サウンドのゲインを変更することです。これにより、入力値が指定量の分だけ効果的に増加するので、サウンドを大きくまたは小さくすることができます。Microphone.gain プロパティは、0 以上 100 以下の数値を受け入れます。この値が 50 であれば、1 の乗数のような役割を果たし、標準的なボリュームとなります。ゼロは、ゼロの乗数のような役割を果たし、入力オーディオは無音になります。値が 50 を超えると、標準以上のボリュームになります。

アプリケーションでは、入力オーディオのサンプルレートも変更することができます。サンプルレートが高くなるとサウンドの品質も高くなりますが、その場合データストリームの密度が上がり、伝送と記憶域のリソースがより多く必要になります。Microphone.rate プロパティは、サンプルレートをキロヘルツ (kHz) 単位で表します。デフォルトのサンプルレートは 8 kHz です。マイクが高いレートをサポートしている場合は、8 kHz 以上の値を Microphone.rate プロパティに設定することができます。たとえば、Microphone.rate プロパティの値を 11 に設定すると、サンプルレートは 11 kHz になります。22 に設定すると、22 kHz になります。

マイクのアクティビティの検知

帯域幅と処理のリソースを節約するため、Flash Player は、マイクからサウンドが伝送されていない状態を検知しようとします。一定期間にわたり、マイクのアクティビティレベルが無音レベルのしきい値を下回っていると、Flash Player はオーディオ入力の伝送を停止し、代わりにシンプルな ActivityEvent を送出します。

Microphone クラスの次の 3 つのプロパティがアクティビティを監視し、検知します。

- 読み取り専用の `activityLevel` プロパティは、マイクが検知するサウンドの量を 0 から 100 までのスケールで示します。
- `silenceLevel` プロパティは、マイクを有効にするために必要なサウンド量を指定し、`ActivityEvent.ACTIVITY` イベントを送出します。`silenceLevel` プロパティも 0 から 100 までのスケールを使用します。デフォルト値は 10 です。
- `silenceTimeout` プロパティは、`ActivityEvent.ACTIVITY` イベントが送出されてマイクが無音状態であると判断されるまでの、アクティビティレベルが無音レベルを下回る時間をミリ秒単位で記述します。デフォルトの `silenceTimeout` 値は 2000 です。

`Microphone.silenceLevel` プロパティと `Microphone.silenceTimeout` プロパティは読み取り専用ですが、`Microphone.setSilenceLevel()` メソッドを使用すると値を変更することができます。

場合によっては、新しいアクティビティが検知されてマイクが有効になるまで、若干の遅延が発生することがあります。マイクを常に有効にしておくと、このような遅延をなくすことができます。`silenceLevel` パラメータをゼロに設定して `Microphone.setSilenceLevel()` メソッドを呼び出すと、サウンドが検知されていなくてもマイクを常に有効にし、オーディオデータを継続的に収集するよう Flash Player に指示することができます。反対に、`silenceLevel` パラメータを 100 に設定すると、マイクは有効になりません。

次の例は、マイクに関する情報と、Microphone オブジェクトが送出するアクティビティイベントおよびステータスイベントのレポートを表示します。

```
import flash,events.ActivityEvent;
import flash,events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
  trace("   " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
```

```
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);

function onMicActivity(event:ActivityEvent):void
{
  trace("activating=" + event.activating + ", activityLevel=" +
    mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
  trace("status: level=" + event.level + ", code=" + event.code);
}
```

上記の例を実行すると、システムのマイクに向かって発話するか音を出し、コンソールまたはデバッグウィンドウに表示されるトレースステートメントを確認します。

メディアサーバーとのオーディオの送受信

Flash Media Server (FMS) のようなストリーミングメディアサーバーと共に ActionScript を使用すると、オーディオ機能が追加で提供されます。

具体的には、Microphone オブジェクトを NetStream オブジェクトに付加し、ユーザーのマイクからサーバーに直接データを送信することができます。オーディオデータはサーバーから Flash または Flex アプリケーションにストリーミングされ、MovieClip の一部として、または Video オブジェクトを使用して再生されます。

詳細については、Flash Media Server のオンラインマニュアル、http://livedocs.macromedia.com を参照してください。

例 : Podcast Player

ポッドキャストとは、オンデマンドで、またはサブスクリプションによってインターネット配信されるサウンドファイルです。通常、ポッドキャストは、ポッドキャストチャンネルとも呼ばれるシリーズの一部としてパブリッシュされています。ポッドキャストエピソードは、1分で終わるものから数時間かかるものまであるため、再生しながらストリーミングされます。ポッドキャストエピソードは別名アイテムとも呼ばれ、多くの場合は MP3 ファイル形式で配信されます。ビデオポッドキャストも一般的ではありますが、このサンプルアプリケーションでは MP3 ファイルを使用するオーディオポッドキャストのみを再生します。

この例は、フル機能を備えたポッドキャストアグリゲーターアプリケーションではありません。たとえば、このアプリケーションは特定のポッドキャストへのサブスクリプションを管理したり、アプリケーションの次回実行時のために、ユーザーがどのポッドキャストを聴いていたかを覚えるということはありません。だたし、より多くの機能を備えたポッドキャストアグリゲーターのたたき台として使用することができます。

Podcast Player の例では、次の ActionScript プログラミング手法を示します。

- 外部 RSS フィードの読み取りと、XML コンテンツの解析
- SoundFacade クラスを作成することによる、サウンドファイルのロードと再生の簡便化
- サウンド再生の進行状況の表示
- サウンド再生の一時停止と再開

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。Podcast Player アプリケーションのファイルは、"Samples/PodcastPlayer" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
PodcastPlayer.mxml または PodcastPlayer.fla	Flex (MXML) または Flash (FLA) 用アプリケーションのユーザーインターフェイスです。
RSSBase.as	RSSChannel クラスと RSSItem クラスに共通のプロパティとメソッドを提供する基本クラスです。
RSSChannel.as	RSS チャンネルに関するデータを保有する ActionScript クラスです。
RSSItem.as	RSS アイテムに関するデータを保有する ActionScript クラスです。

ファイル	説明
SoundFacade.as	アプリケーションのメイン ActionScript クラスです。Sound クラスおよび SoundChannel クラスのメソッドとイベントをカプセル化し、再生の一時停止と再開をサポートします。
URLService.as	リモート URL からデータを取得する ActionScript クラスです。
playerconfig.xml	ポッドキャストチャンネルを示す RSS フィードのリストが記された XML ファイルです。

ポッドキャストチャンネルの RSS データの読み取り

Podcast Player アプリケーションは、最初にいくつかのポッドキャストチャンネルとエピソードに関する情報を読み取ります。

1. アプリケーションはまず、ポッドキャストチャンネルのリストを含む XML 設定ファイルを読み取り、チャンネルのリストを表示します。

2. ユーザーがいずれかのポッドキャストチャンネルを選択すると、チャンネルの RSS フィードが読み取られ、チャンネルエピソードのリストが表示されます。

この例では、URLLoader ユーティリティクラスを使用し、テキストベースのデータをリモートまたはローカルのファイルから取得します。Podcast Player はまず URLLoader オブジェクトを作成し、playerconfig.xml ファイルから XML 形式の RSS フィードのリストを取得します。次に、ユーザーがリストから指定のフィードを選択すると、新しい URLLoader オブジェクトが作成され、フィードの URL から RSS データが読み取られます。

SoundFacade クラスを使用した、サウンドのロードと再生の簡便化

ActionScript 3.0 のサウンドアーキテクチャは強力ですが、複雑でもあります。サウンドの基本的なロード機能と再生機能しか必要としないアプリケーションでは、シンプルなメソッド呼び出しとイベントによって複雑さを隠すことができるクラスを使用することができます。ソフトウェア設計パターンの世界では、このようなクラスはファサードと呼ばれます。

SoundFacade クラスは、次のタスクを実行するために単一のインターフェイスを提供します。

- Sound オブジェクト、SoundLoaderContext オブジェクト、および SoundMixer クラスを使用したサウンドファイルのロード
- Sound オブジェクトと SoundChannel オブジェクトを使用したサウンドファイルの再生
- playback progress イベントの送出
- Sound オブジェクトと SoundChannel オブジェクトを使用したサウンド再生の一時停止と再開

SoundFacade クラスは、ActionScript サウンドクラスの大半の機能を、複雑さを軽減した形で提供します。

次のコードは、クラス宣言、クラスプロパティ、および SoundFacade() コンストラクタメソッドを示します。

```
public class SoundFacade extends EventDispatcher
{
  public var s:Sound;
  public var sc:SoundChannel;
  public var url:String;
  public var bufferTime:int = 1000;

  public var isLoaded:Boolean = false;
  public var isReadyToPlay:Boolean = false;
  public var isPlaying:Boolean = false;
  public var isStreaming:Boolean = true;
  public var autoLoad:Boolean = true;
  public var autoPlay:Boolean = true;

  public var pausePosition:int = 0;

  public static const PLAY_PROGRESS:String = "playProgress";
  public var progressInterval:int = 1000;
  public var playTimer:Timer;

  public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                    autoPlay:Boolean = true, streaming:Boolean = true,
                    bufferTime:int = -1):void
  {
    this.url = soundUrl;

    // このオブジェクトの動作を決定するブール値を設定します。
    this.autoLoad = autoLoad;
    this.autoPlay = autoPlay;
    this.isStreaming = streaming;

    // デフォルトでは、グローバルな bufferTime 値です。
    if (bufferTime < 0)
    {
      bufferTime = SoundMixer.bufferTime;
    }

    // バッファ時間を 0 から 30 秒までの妥当な時間にします。
    this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

    if (autoLoad)
    {
      load();
    }
  }
```

SoundFacade クラスは、独自のイベントを送出できるよう EventDispatcher クラスを拡張します。クラスコードは、まず Sound オブジェクトと SoundChannel オブジェクトのプロパティを宣言します。このクラスは、サウンドをストリーミングするときに使用する、サウンドファイルの URL の値と bufferTime プロパティも保存します。さらに、ロードおよび再生の動作に影響するブールパラメータ値のいくつかを受け取ります。

- autoLoad パラメータは、オブジェクトが作成されると同時にサウンドのロードを開始するようオブジェクトに指示します。
- autoPlay パラメータは、十分な長さのサウンドデータがロードされると同時に、サウンドの再生を開始するよう指示します。ストリーミングサウンドであれば、bufferTime プロパティで指定されるデータ量がロードされると再生が開始されます。
- streaming パラメータは、ロードが完了する前にサウンドファイルの再生を開始できることを示します。

bufferTime パラメータのデフォルト値は -1 です。コンストラクタメソッドが bufferTime パラメータに負の値を検知した場合、bufferTime プロパティの値を SoundMixer.bufferTime の値に設定します。これにより、アプリケーションのデフォルト値は、必要に応じてグローバルな SoundMixer.bufferTime の値になります。

autoLoad パラメータが true に設定されている場合、コンストラクタメソッドはすぐに次の load() メソッドを呼び出し、サウンドファイルのロードを開始します。

```
public function load():void
{
  if (this.isPlaying)
  {
    this.stop();
    this.s.close();
  }
  this.isLoaded = false;

  this.s = new Sound();

  this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
  this.s.addEventListener(Event.OPEN, onLoadOpen);
  this.s.addEventListener(Event.COMPLETE, onLoadComplete);
  this.s.addEventListener(Event.ID3, onID3);
  this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
  this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

  var req:URLRequest = new URLRequest(this.url);

  var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime,
                                                          true);
  this.s.load(req, this.slc);
}
```

load() メソッドは新しい Sound オブジェクトを作成し、すべての重要なサウンドイベントのリスナーを追加します。その後、bufferTime 値に渡す SoundLoaderContext オブジェクトを使用し、Sound オブジェクトにサウンドファイルをロードするよう指示します。

url プロパティは変更することができるので、SoundFacade インスタンスを使用して異なるサウンドファイルを連続して再生することができます。そうするには、url プロパティを変更して load() メソッドを呼び出します。新しいサウンドファイルがロードされます。

次の 3 つのイベントリスナーメソッドは、SoundFacade オブジェクトがどのようにロードの進行状況をトラッキングし、サウンドを再生するタイミングを決定するかを示します。

```
public function onLoadOpen(event:Event):void
{
  if (this.isStreaming)
  {
    this.isReadyToPlay = true;
    if (autoPlay)
    {
      this.play();
    }
  }
  this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
  this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
  this.isReadyToPlay = true;
  this.isLoaded = true;
  this.dispatchEvent(evt.clone());

  if (autoPlay && !isPlaying)
  {
    play();
  }
}
```

サウンドのロードが開始すると、onLoadOpen() メソッドが実行されます。サウンドをストリーミングモードで再生できる場合、onLoadComplete() メソッドは isReadyToPlay フラグをすぐに true に設定します。isReadyToPlay フラグは、[再生] ボタンをクリックするなどのユーザーアクションに反応して、アプリケーションがサウンドを再生できるかどうかを決定します。SoundChannel クラスはサウンドデータのバッファリングを管理するので、play() メソッドを呼び出す前に十分なデータ量がロードされたかどうか明示的に確認する必要はありません。

ロードプロセス中は、onLoadProgress() メソッドが定期的に実行されます。このメソッドは、SoundFacade オブジェクトを使用するコードによって使用される ProgressEvent オブジェクトのクローンを送出するだけです。

サウンドデータが完全にロードされると、onLoadComplete() メソッドが実行されます。必要であれば、非ストリーミングサウンドのために play() メソッドを呼び出します。play() メソッドを以下に示します。

```
public function play(pos:int = 0):void
{
  if (!this.isPlaying)
  {
    if (this.isReadyToPlay)
    {
      this.sc = this.s.play(pos);
      this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
      this.isPlaying = true;

      this.playTimer = new Timer(this.progressInterval);
      this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
      this.playTimer.start();
    }
  }
}
```

サウンドが再生できる状態になると、play() メソッドは Sound.play() メソッドを呼び出します。生成される SoundChannel オブジェクトは、sc プロパティに保存されます。その後、play() メソッドは Timer オブジェクトを作成します。これを使用すると、playback progress イベントを定期的に送出することができます。

再生の進行状況の表示

Timer オブジェクトを作成して再生を監視させる処理は複雑なので、コードの実行は一度きりにする必要があります。この Timer ロジックを、SoundFacade クラスのような再利用可能なクラスにカプセル化することで、アプリケーションはサウンドのロード時や再生時に同様の progress イベントを監視することができます。

SoundFacade.play() メソッドによって作成される Timer オブジェクトは、TimerEvent インスタンスを毎秒送出します。以下に示す onPlayTimer() メソッドは、新しい TimerEvent が受信されるたびに実行されます。

```
public function onPlayTimer(event:TimerEvent):void
{
  var estimatedLength:int =
    Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
  var progEvent:ProgressEvent =
    new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position,
  estimatedLength);
  this.dispatchEvent(progEvent);
}
```

onPlayTimer() メソッドは、576 ページの「再生の監視」のセクションで解説するサイズ予測方法を実装します。イベントタイプ SoundFacade.PLAY_PROGRESS で新しい ProgressEvent インスタンスが作成されます。そのとき、bytesLoaded プロパティは SoundChannel オブジェクトの現在位置に設定され、bytesTotal プロパティはサウンドデータの予測長に設定されます。

再生の一時停止と再開

前述の SoundFacade.play() メソッドは、サウンドデータの開始位置に対応する pos パラメータを受け取ります。pos 値がゼロの場合、サウンドは最初から再生されます。

SoundFacade.stop() メソッドは、次に示す pos パラメータも受け取ります。

```
public function stop(pos:int = 0):void
{
  if (this.isPlaying)
  {
    this.pausePosition = pos;
    this.sc.stop();
    this.playTimer.stop();
    this.isPlaying = false;
  }
}
```

SoundFacade.stop() メソッドが呼び出されると pausePosition プロパティが設定されるので、ユーザーが同じサウンドの再生を再開する場合でも、再生ヘッドをどこに配置すれば良いかが分かります。

以下に示す SoundFacade.pause() メソッドおよび SoundFacade.resume() メソッドは、それぞれ SoundFacade.stop() メソッドおよび SoundFacade.play() メソッドを呼び出します。呼び出しのたびに pos パラメータ値が渡されます。

```
public function pause():void
{
   stop(this.sc.position);
}

public function resume():void
{
   play(this.pausePosition);
}
```

pause() メソッドは、現在の SoundChannel.position 値を play() メソッドに渡します。このメソッドは、この値を pausePosition プロパティに保存します。resume() メソッドは、開始位置として pausePosition 値を使用し、同じサウンドを再生します。

Podcast Player の拡張例

この例では、再利用可能な SoundFacade クラスを主に使用する、必要最小限の機能だけを備えた Podcast Player を示します。次のような機能を追加して、アプリケーションの利便性を高めることもできます。

- フィードのリストと、各エピソードに関する使用情報を SharedObject インスタンスに保存します。このインスタンスは、アプリケーションの次回実行時に使用することができます。
- ユーザー独自の RSS フィードを、ポッドキャストチャンネルのリストに追加します。
- ユーザーがエピソードを停止または終了したときの再生ヘッドの位置を記憶します。そうすると、アプリケーションの次回実行時に同じ場所から再開することができます。
- ユーザーがインターネット接続されていなくても、オフライン状態でエピソードを聞くために、MP3 ファイルをダウンロードします。
- ポッドキャストチャンネルの新しいエピソードを定期的に確認してエピソードリストを自動的に更新する、サブスクリプション機能を追加します。
- Odeo.com などのポッドキャストホスティングサービスの API を使用し、ポッドキャスト検索および閲覧機能を追加します。

第 21 章

ユーザー入力のキャプチャ

21

この章では、ActionScript 3.0 を使用して、ユーザー操作に応答するインタラクティブ機能を作成する方法について説明します。まずキーボードイベントおよびマウスイベントについて説明し、次に、コンテキストメニューやフォーカス管理のカスタマイズといった高度なトピックに進みます。最後に、マウス入力に応答するアプリケーションの例として、WordSearch を取り上げます。

この章では、ActionScript 3.0 イベントモデルについての知識を前提としています。詳細については、299 ページ、第 10 章の「イベントの処理」を参照してください。

目次

ユーザー入力の基礎 599
キーボード入力のキャプチャ 601
マウス入力のキャプチャ 603
例 : WordSearch 608

ユーザー入力の基礎

ユーザー入力のキャプチャの概要

キーボード、マウス、カメラ、またはこれらのデバイスの組み合わせによるユーザー操作は、インタラクティブ機能の基盤となります。ActionScript 3.0 では、ユーザー操作を識別しそれに応答するには、まずイベントを監視する必要があります。

InteractiveObject クラスは、DisplayObject クラスのサブクラスであり、ユーザー操作の処理に必要なイベントおよび機能に共通の構造を提供します。InteractiveObject クラスのインスタンスは直接作成しません。代わりに、SimpleButton、Sprite、TextField などの表示オブジェクトとさまざまな Flash および Flex コンポーネントは、このクラスからユーザー操作モデルを継承し、共通の構造を共有します。つまり、InteractiveObject から派生した 1 つのオブジェクトでユーザー操作を処理するために使用する方法およびコードは、派生した他のオブジェクトにも適用できます。

この章では、次の一般的なユーザー操作タスクについて説明します。

- アプリケーション全体でのキーボード入力のキャプチャ
- 特定の表示オブジェクトに対するキーボード入力のキャプチャ
- アプリケーション全体でのマウス操作のキャプチャ
- 特定の表示オブジェクトに対するマウス入力のキャプチャ
- ドラッグ & ドロップのインタラクティブ機能の作成
- カスタムマウスカーソル (マウスポインタ) の作成
- コンテキストメニューへの新しい動作の追加
- フォーカスの管理

重要な概念と用語

先に進む前に、次の主要なユーザー操作の用語について理解しておくことが重要です。

- 文字コード：現在の文字セットの文字を表す数値コード。文字セットは、キーボードで押されるキーに関連付けられています。たとえば、“D” と “d” は米国英語キーボードでは同じキーで入力されますが、それぞれ異なる文字コードを持っています。
- コンテキストメニュー：ユーザーが右クリックした場合、またはキーボードとマウスの特定の組み合わせを使用した場合に表示されるメニュー。コンテキストメニューのコマンドは、通常、クリックしたエレメントに対して直接適用されます。たとえば、イメージのコンテキストメニューにはイメージを別のウィンドウで表示するコマンドや、イメージをダウンロードするコマンドがあります。
- フォーカス：選択されたエレメントがアクティブ状態で、キーボード操作またはマウス操作の対象となっていることの表示。
- キーコード : キーボード上の物理的なキーに対応する数値コード。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章は ActionScript を使用したユーザー入力の操作に関するものであり、基本的にこの章のすべてのコードリストは、一部の種類の表示オブジェクト (通常はテキストフィールドまたは InteractiveObject サブクラス) を操作することが関連しています。例の目的に対応して、表示オブジェクトは Adobe Flash CS3 Professional で作成されてステージに配置されたものか、ActionScript を使用して作成されたものです。サンプルをテストするには、Flash Player で結果を表示し、サンプルを操作してコードの効果を確認する必要があります。

この章のコードリストをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. ステージ上にインスタンスを作成します。
   - コードがテキストフィールドを参照する場合は、テキストツールを使用してステージ上にダイナミックテキストフィールドを作成します。
   - それ以外の場合は、ステージ上にボタンまたはムービークリップシンボルのインスタンスを作成します。
5. テキストフィールド、ボタン、またはムービークリップを選択して、プロパティインスペクタでインスタンス名を付けます。名前は、サンプルコードの表示オブジェクトと同じ名前にします。たとえば、コードで `myDisplayObject` という名前のオブジェクトを操作する場合は、ステージオブジェクトにも `myDisplayObject` という名前を付けます。
6. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   画面上で、表示オブジェクトはコードで指定されたとおりに操作されます。

キーボード入力のキャプチャ

InteractiveObject クラスから操作モデルを継承する表示オブジェクトは、イベントリスナーを使用してキーボードイベントに応答できます。たとえば、ステージにイベントリスナーを配置してキーボード入力を監視し、それに応答できます。次のコードでは、イベントリスナーでキーの押し下げをキャプチャして、キー名とキーコードのプロパティが表示されます。

```
function reportKeyDown(event:KeyboardEvent):void
{
  trace("Key Pressed: " + String.fromCharCode(event.charCode) +
    " (character code: " + event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Ctrl キーなどの一部のキーには字形表現がありませんが、キーを押すとイベントが生成されます。

上記のコード例では、キーボードイベントリスナーがステージ全体のキーボード入力をキャプチャします。ステージ上の特定の表示オブジェクトに対してイベントリスナーを記述することもできます。このイベントリスナーは、オブジェクトにフォーカスがあるとトリガされます。

次の例では、ユーザーが TextField インスタンス内に入力した場合のみ、キーストロークが [出力] パネルに表示されます。Shift キーを押したままにすると、TextField の境界線の色が一時的に赤に変わります。

このコードは、ステージ上に tf という名前の TextField インスタンスがあることを前提としています。

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
  trace("Key Pressed: " + String.fromCharCode(event.charCode) +
    " (key code: " + event.keyCode + " character code: "
    + event.charCode + ")");
  if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
  trace("Key Released: " + String.fromCharCode(event.charCode) +
    " (key code: " + event.keyCode + " character code: " +
    event.charCode + ")");
  if (event.keyCode == Keyboard.SHIFT)
  {
    tf.borderColor = 0x000000;
  }
}
```

TextField クラスは、ユーザーがテキストを入力したときに監視できる textInput イベントも報告します。詳細については、494 ページの「テキスト入力のキャプチャ」を参照してください。

キーコードおよび文字コードについて

キーボードイベントの keyCode プロパティと charCode プロパティを使用することで、どのキーが押されたかを判断し、その結果として他のアクションをトリガできます。keyCode プロパティは、キーボードのキーの値に対応する数値です。charCode プロパティは、現在の文字セットにおけるそのキーの数値です。デフォルトの文字セットは UTF-8 で、ASCII をサポートしています。

キーコード値と文字値の主な違いは、キーコード値がキーボードの特定のキーを表すのに対し、文字値は特定の文字を表すことです。キーコード値では、キーパッドの 1 はキーボードの一番上の行の 1 とは異なりますが、"1" を生成するキーと "!" を生成するキーは同じです。文字値では、R と r は異なります。

メモ

ASCII でのキーとその文字コード値の間のマッピングについては、693 ページ、付録 C の「キーボードのキーとキーコードの値」を参照してください。

キーとキーコード間のマッピングは、デバイスやオペレーティングシステムによって異なります。このため、キーマッピングを使用してアクションをトリガしないでください。代わりに、Keyboard クラスによって提供される事前定義された定数値を使って、適切な keyCode プロパティを参照します。たとえば、Shift キーのキーマッピングを使用する代わりに、Keyboard.SHIFT 定数を使用します (前のサンプルコードを参照)。

KeyboardEvent の優先度について

他のイベントの場合と同様、キーボードイベントシーケンスは、コードで addEventListener() メソッドが割り当てられた順番ではなく、表示オブジェクト階層によって決定されます。

たとえば、次の例に示すように、container というムービークリップ内に tf テキストフィールドを配置し、キーボードイベント用のイベントリスナーを両方のインスタンスに追加すると仮定します。

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
  trace(event.currentTarget.name + " hears key press: " +
    String.fromCharCode(event.charCode) + " (key code: " +
    event.keyCode + " character code: " + event.charCode + ")");
}
```

テキストフィールドとその親コンテナの両方にリスナーがあるため、reportKeyDown() 関数は TextField 内でのキーストロークごとに 2 回呼び出されます。キーが押されるたびに、テキストフィールドの方が container ムービークリップより先にイベントを送出します。

オペレーティングシステムや Web ブラウザは、Adobe Flash Player より先にキーボードイベントを処理します。たとえば、Microsoft Internet Explorer では、Ctrl + W を押すと、表示されている SWF ファイルがキーボードイベントを送出する前に、ブラウザウィンドウが閉じます。

マウス入力のキャプチャ

マウスクリックにより、インタラクティブ機能のトリガに使用できるマウスイベントが作成されます。イベントリスナーをステージに追加して、SWF ファイル内で発生するマウスイベントを監視できます。InteractiveObject クラスを継承するステージ上のオブジェクト (Sprite や MovieClip など) にイベントリスナーを追加することもできます。これらのリスナーはオブジェクトがクリックされるとトリガされます。

キーボードイベントと同様に、マウスイベントはバブルします。次の例では、square はステージの子であるため、四角形がクリックされると、スプライト square および Stage オブジェクトの両方によってイベントが送出されます。

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
  trace(event.currentTarget.toString() +
    " MouseEvent を送出します。Local coords [" +
    event.localX + "," + event.localY + "] Stage coords [" +
    event.stageX + "," + event.stageY + "]");
}
```

上記の例では、マウスイベントには、クリックに関する位置情報が含まれます。localX プロパティと localY プロパティには、表示チェーン内の最下位の子上でのクリックの場所に関する位置情報が含まれます。たとえば、square の左上隅をクリックすると、この位置は square の基準点であるため、ローカル座標の [0,0] が表示されます。一方、stageX プロパティと stageY プロパティでは、ステージ上でのクリックのグローバル座標が参照されます。同じ場所をクリックすると、グローバル座標の [50,50] が表示されます。これは、square がこの座標に移動されてたためです。ユーザー操作に応答する方法に応じて、これらの座標ペアの両方を使用できます。

MouseEvent オブジェクトには、altKey プロパティ、ctrlKey プロパティ、shiftKey Boolean プロパティがあります。これらのプロパティを使って、Alt キー、Ctrl キー、または Shift キーがマウスクリックと同時に押されていたかどうかを確認することができます。

ドラッグ & ドロップ機能の作成

ドラッグ & ドロップ機能を使用すると、ユーザーは左マウスボタンを押してオブジェクトを選択し、ボタンを押したままそのオブジェクトを画面上の新しい場所に移動し、そこで左マウスボタンを離すことによってオブジェクトをその場所にドロップすることができます。その例を次のコードに示します。

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

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);
```

```
var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
  circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
  circle.stopDrag();
  trace(circle.dropTarget.name);
}
```

詳細については、378 ページの「ドラッグ & ドロップ操作の作成」を参照してください。

マウスカーソルのカスタマイズ

マウスカーソル (マウスポインタ) を非表示にしたり、ステージ上の任意の表示オブジェクトと入れ替えたりすることができます。マウスカーソルを非表示にするには、Mouse.hide() メソッドを呼び出します。カーソルをカスタマイズするには、Mouse.hide() を呼び出し、MouseEvent.MOUSE_MOVE イベントについてステージを監視し、表示オブジェクト (カスタムカーソル) の座標をイベントの stageX プロパティと stageY プロパティに設定します。次の例は、この操作の基本的な実行方法を示したものです。

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();
```

```
function redrawCursor(event:MouseEvent):void
{
  cursor.x = event.stageX;
  cursor.y = event.stageY;
}
```

コンテキストメニューのカスタマイズ

InteractiveObject クラスを継承している各オブジェクトには、SWF ファイル内で右クリックすると表示される独自のコンテキストメニューを設定できます。[進む]、[戻る]、[印刷]、[品質]、[ズーム] などのいくつかのコマンドは、デフォルトで組み込まれています。

[設定] および [Flash Player について] コマンド以外のデフォルトのコマンドは、すべてメニューから削除できます。そうするには、Stage プロパティ `showDefaultContextMenu` を `false` に設定します。

特定の表示オブジェクトに対してカスタマイズしたコンテキストメニューを作成する場合、ContextMenu クラスの新しいインスタンスを作成して `hideBuiltInItems()` メソッドを呼び出し、そのインスタンスを該当の DisplayObject インスタンスの `contextMenu` プロパティに割り当てます。次の例では、動的に描画される四角形に、色をランダムに変更するコンテキストメニューコマンドを設定します。

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
  square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
  return new ColorTransform(Math.random(), Math.random(),
    Math.random(),1,(Math.random() * 512) - 255,
    (Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

フォーカスの管理

インタラクティブオブジェクトは、プログラムまたはユーザー操作のいずれかによってフォーカスを取得できます。どちらの場合も、フォーカスを設定すると、オブジェクトの focus プロパティが true に変更されます。また、tabEnabled プロパティが true に設定されている場合に Tab キーを押すと、オブジェクトのフォーカスを別のオブジェクトに渡すことができます。tabEnabled 値のデフォルトは false です。ただし、次の場合を除きます。

- SimpleButton オブジェクトの場合は true。
- テキスト入力フィールドの場合は true。
- buttonMode が true に設定されている Sprite オブジェクトまたは MovieClip オブジェクトの場合は true。

これらの各シナリオでは、FocusEvent.FOCUS_IN または FocusEvent.FOCUS_OUT に対するリスナーを追加して、フォーカスを変更したときの追加動作を設定できます。これはテキストフィールドおよびフォームで特に便利ですが、スプライトやムービークリップ、つまり InteractiveObject クラスを継承する任意のオブジェクトでも使用できます。次の例では、Tab キーを使用してフォーカスの順次移動を有効にする方法と、それ以降のフォーカスイベントに応答する方法を示しています。この場合、各四角形はフォーカスを取得すると色が変わります。

メモ	Flash オーサリングツールは、フォーカスの管理にキーボードショートカットを使用するため、フォーカス管理を正確にシミュレートするには、Flash ではなくブラウザで SWF ファイルをテストする必要があります。

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
  for (j = 0; j < cols; j++)
  {
    createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
  }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
  var square:Sprite = new Sprite();
  square.graphics.beginFill(0x000000);
  square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
  square.graphics.endFill();
  square.x = startX;
  square.y = startY;
  square.tabEnabled = true;
```

```
    square.tabIndex = tabNumber;
    square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
    addChild(square);
}
function changeColor(event:FocusEvent):void
{
    e.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    // 赤、緑、青のカラーチャンネルのランダム値を生成します。
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // ランダムなカラーが付与された ColorTransform オブジェクトを作成して返します。
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

例 : WordSearch

この例では、マウスイベントの処理によるユーザーとの対話を説明します。ユーザーはマス目にランダムに配置された文字を縦横につないで、できる限り多くの単語を作ります。ただし、同じ文字を 2 回使用することはできません。この例は、ActionScript 3.0 の次のような機能を使用します。

- コンポーネントのグリッドの動的な作成
- マウスイベントへの応答
- ユーザー操作に基づくスコアの管理

このサンプルのアプリケーションファイルを入手するには、
www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。
WordSearch アプリケーションのファイルは、"Samples/WordSearch" フォルダにあります。
アプリケーションは、次のファイルで構成されています。

ファイル	説明
WordSearch.as	アプリケーションの主要な機能を提供するクラス
WordSearch.fla	Flash のメインアプリケーションファイル
dictionary.txt	作成された単語のスペルが正しく、スコアを加算できるかどうかを判断するファイル

辞書のロード

単語検索を利用するゲームを作成するには、辞書が必要になります。この例では、復帰文字で区切られた単語のリストを格納する dictionary.txt というテキストファイルを使用しています。words という配列を作成した後、loadDictionary() 関数がこのファイルを要求します。ロードに成功すると、このファイルは長い1つのストリングになります。split() メソッドを使用し、復帰文字 (文字コード 10) の各インスタンスで分割することにより、このストリングを単語の配列に変換して解析することができます。この解析は、dictionaryLoaded() 関数で発生します。

```
words = dictionaryText.split(String.fromCharCode(10));
```

ユーザーインターフェイスの作成

単語を配列に格納したら、ユーザーインターフェイスを設定します。2 つの Button インスタンスを作成します。1つは単語の送信用、もう1つは、ユーザーが現在作成している単語のクリア用です。どちらの場合も、MouseEvent.CLICK イベントを監視して、関数を呼び出し、ユーザー入力に応答する必要があります。このイベントは、ボタンが押されるとブロードキャストされます。setupUI() 関数内で、このコードは 2 つのボタンに対するリスナーを作成します。

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

ゲームボードの作成

ゲームボードはランダムな文字のマス目です。generateBoard() 関数内で、1つのループを別のループ内にネストすることによって 2 次元のマス目を作成できます。最初のループが行を増やし、2 番目のループが行ごとの列の総数を増やします。これらの行と列で作成された各セルには、ボード上の文字を表すボタンを配置します。

```
private function generateBoard(startX:Number,
  startY:Number,
  totalRows:Number,
  totalCols:Number,
  buttonSize:Number):void
{
  buttons = new Array();
  var colCounter:uint;
  var rowCounter:uint;
  for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
  {
    for (colCounter = 0; colCounter < totalCols; colCounter++)
    {
      var b:Button = new Button();
      b.x = startX + (colCounter*buttonSize);
      b.y = startY + (rowCounter*buttonSize);
      b.addEventListener(MouseEvent.CLICK, letterClicked);
      b.label = getRandomLetter().toUpperCase();
      b.setSize(buttonSize,buttonSize);
```

```
        b.name = "buttonRow"+rowCounter+"Col"+colCounter;
        addChild(b);

        buttons.push(b);
      }
    }
}
```

MouseEvent.CLICK イベントに対するリスナーを追加するコードを1行記述するだけで、これは for ループ内の処理であるため、リスナーが各 Button インスタンスに割り当てられます。さらに、その行と列の位置を使用した名前を各ボタンに割り当てます。これにより、後からコード内で各ボタンの行と列を容易に参照できるようになります。

ユーザー入力からの単語構築

単語は横または縦に隣り合う文字を選択することで作成できますが、同じ文字を2度使用することはできません。文字をクリックすると、マウスイベントが生成されます。このときに、ユーザーが作成している単語をチェックし、前にクリックした文字の続きとして適切かどうかを確認する必要があります。適切でない場合は、前に作成していた単語を削除して新しい単語を開始します。このチェックは、isLegalContinuation() メソッドで発生します。

```
private function isLegalContinuation(prevButton:Button,
  currButton:Button):Boolean
{
  var currButtonRow:Number = Number(currButton.name.charAt(currButton.name.
  indexOf("Row") + 3));
  var currButtonCol:Number =
  Number(currButton.name.charAt(currButton.name.indexOf("Col") + 3));
  var prevButtonRow:Number =
  Number(prevButton.name.charAt(prevButton.name.indexOf("Row") + 3));
  var prevButtonCol:Number =
  Number(prevButton.name.charAt(prevButton.name.indexOf("Col") + 3));

  return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow -
  currButtonRow) <= 1) ||
        (prevButtonRow == currButtonRow && Math.abs(prevButtonCol -
  currButtonCol) <= 1));
}
```

String クラスの charAt() メソッドと indexOf() メソッドは、現在クリックされているボタンと前にクリックされたボタンの両方から適切な行と列を取得します。行または列の値が変更されていない場合、あるいは、変更された行または列の値が前のクリックの値よりも1だけ大きい場合、isLegalContinuation() メソッドは true を返します。ゲームのルールを変えて縦横の他に斜めのスペリングも許可したい場合、変更されていない行または列の値のチェックを削除します。この場合、最終行は次のようになります。

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol
  - currButtonCol) <= 1));
```

単語送信のチェック

ゲームのコードを完成するには、単語送信とスコア計算のチェック機能が必要です。searchForWord() メソッドにこの両方の処理が含まれます。

```
private function searchForWord(str:String):Number
{
  if (words && str)
  {
    var i:uint = 0
    for (i = 0; i < words.length; i++)
    {
      var thisWord:String = words[i];
      if (str == words[i])
      {
        return i;
      }
    }
    return -1;
  }
  else
  {
    trace("WARNING: cannot find words, or string supplied is null");
  }
  return -1;
}
```

この関数は、辞書内にあるすべての単語をループ処理します。ユーザーの作成した単語が辞書内の単語と一致した場合、この単語の辞書内での位置が返されます。次に、submitWord() メソッドがこの応答をチェックし、位置が有効であればスコアを更新します。

カスタマイズ

クラスの先頭にはいくつかの定数があります。これらの変数を変更してこのゲームを変更することができます。たとえば、TOTAL_TIME 変数の値を大きくすることにより、プレイ時間を変更できます。また、PERCENT_VOWELS 変数の値を少しだけ大きくすると、単語が見つかる可能性が高くなります。

第 22 章

ネットワーキングとコミュニケーション

22

この章では、SWF ファイルで外部ファイルおよび他の Adobe Flash Player 9 インスタンスと通信できるようにする方法について説明します。また、外部ソースからデータをロードする方法、Java サーバーと Flash Player の間でメッセージを送信する方法、FileReference および FileReferenceList クラスを使用してファイルのアップロードとダウンロードを行う方法についても説明します。

目次

ネットワーキングとコミュニケーションの基本 614
外部データの操作 617
他の Flash Player インスタンスへの接続 623
ソケットの接続 629
ローカルデータの保存 635
ファイルのアップロードおよびダウンロード操作 638
例 : Telnet クライアントの構築 649
例 : ファイルのアップロードとダウンロード 652

ネットワーキングとコミュニケーションの基本

ネットワーキングとコミュニケーションの概要

より複雑な ActionScript アプリケーションを構築する場合、サーバーサイドスクリプトとの通信や、外部 XML または外部テキストファイルからのデータのロードが必要になる場合が多くあります。flash.net パッケージには、リモート URL からのコンテンツのロード、他の Flash Player インスタンスとの通信、リモート Web サイトへの接続を行う場合など、インターネットを介してデータを送受信するクラスが含まれています。

ActionScript 3.0 では、外部ファイルをロードするには URLLoader クラスと URLRequest クラスを使用します。その後、ロードしたデータのタイプに応じて、特定のクラスを使用してデータにアクセスします。たとえば、リモートコンテンツが名前 / 値のペアの形式になっている場合は、URLVariables クラスを使用してサーバーの結果を解析できます。あるいは、URLLoader クラスおよび URLRequest クラスを使用してロードしたファイルがリモート XML ドキュメントである場合は、XML クラスのコントラクタ、XMLDocument クラスのコントラクタ、または `XMLDocument.parseXML()` メソッドを使用して、XML ドキュメントを解析できます。これにより、URLVariables クラス、XML クラス、またはリモートデータを解析および処理する他のクラスのいずれを使用する場合でも、外部ファイルのロードに関するコードを同じにすることができ、ActionScript コードが簡潔になります。

flash.net パッケージには、他のタイプのリモート通信用クラスも含まれています。たとえば、サーバーとのファイルのアップロードおよびダウンロード用の FileReference クラス、ソケット接続を介してリモートコンピュータと直接通信ができる Socket クラスおよび XMLSocket クラス、Flash 固有のサーバーリソース (Flash Media Server や Flash Remoting サーバーなど) との通信やビデオファイルのロードに使用する NetConnection クラスおよび NetStream クラスなどがあります。

また、flash.net パッケージにはユーザーのローカルコンピュータ通信用のクラスが含まれています。1 台のコンピュータ上で実行中の 2 つ以上の SWF ファイル間で通信が可能になる LocalConnection クラス、ユーザーのコンピュータにデータを格納し、後でアプリケーションに戻ったときにそのデータを取得できる SharedObject クラスなどがこれに該当します。

一般的なネットワーキングとコミュニケーションのタスク

次のリストは、ActionScript からの外部通信に関して最も一般的に行われるタスクを示しています。これらについては、この章で説明します。

- 外部のファイルまたはサーバースクリプトからのデータのロード
- サーバースクリプトへのデータの送信
- 他のローカル SWF ファイルとの通信
- バイナリソケット接続の操作
- XML ソケットとの通信

- ローカルの永続データの保存
- サーバーへのファイルのアップロード
- サーバーからユーザーのマシンへのファイルのダウンロード

重要な概念と用語

次の参照リストに、この章で使われる重要な用語を示します。

- 外部データ : SWF ファイルの外部に何らかの形式で格納され、必要なときに SWF ファイル内にロードされるデータ。このデータは、直接ロードされるファイル内に格納したり、データベース内、またはサーバー上で実行中のプログラムやスクリプトを呼び出すことで取得可能なその他の形式で格納することができます。
- URL エンコード変数 : URL エンコード形式により、1 つのテキストストリング内でいくつかの変数 (変数の名前と値のペア) を表現する方法が提供されます。個々の変数は、`name=value` という形式で記述されます。各変数 (名前と値の各ペア) は、アンパサンド文字で `variable1=value1&variable2=value2` のように区切られます。これにより、多くの変数を 1 つのメッセージとして送信できます。
- MIME タイプ : インターネット通信でファイルのタイプを特定するために使用される標準コード。指定するすべてのファイルタイプに、それを識別するための固有のコードが設定されています。ファイルやメッセージの送信時、Web サーバーやユーザーの Flash Player インスタンスなどのコンピュータが、送信するファイルのタイプを指定します。
- HTTP: Hypertext Transfer Protocol (ハイパーテキスト転送プロトコル)。インターネット上で送信される Web ページおよびその他のさまざまなタイプのコンテンツを提供するための標準形式です。
- リクエストメソッド : Flash Player などのプログラムや Web ブラウザが HTTP リクエストと呼ばれるメッセージを Web サーバーに送信するとき、送信されるデータはすべて 2 つの方法のいずれかでリクエスト内に埋め込まれます。この 2 つのリクエストメソッドとは、GET および POST です。サーバー側で、リクエストを受信するプログラムはリクエストの該当箇所を確認してデータを探す必要があります。そのため、ActionScript からのデータの送信に使用するリクエストメソッドは、サーバーでそのデータの読み取りに使用するリクエストメソッドと一致している必要があります。
- ソケット接続 : 2 台のコンピュータ間での通信に使用される永続的な接続
- アップロード : 他のコンピュータにファイルを送信すること
- ダウンロード : 他のコンピュータからファイルを取得すること

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章の一部のコードリストでは、外部データを読み込んだり、その他の通信を行います。多くの場合、そのようなサンプルには `trace()` 関数が含まれており、サンプルの実行結果が [出力] パネルに表示されます。その他の例では、サーバーへのファイルのアップロードなど、特定の機能を実際に実行します。そのような例をテストするには、SWF とやり取りして、目的のアクションが実際に実行されることを確認する必要があります。

コード例は 2 つのカテゴリに分けられます。一部のサンプルコードリストは、Flash ドキュメントのキーフレームにアタッチされるコードのように、コードがスタンドアロンスクリプト内に置かれるという前提で作成されています。これらの例をテストするには :

1. 新しい Flash ドキュメントを作成します。
2. タイムラインのフレーム 1 のキーフレームを選択して、[アクション] パネルを開きます。
3. コードリストを [スクリプト] ペインにコピーします。
4. メインメニューから [制御]-[ムービープレビュー] を選択して SWF ファイルを作成し、例をテストします。

その他のサンプルコードリストは、クラスとして作成されています。サンプルクラスは Flash ドキュメントのドキュメントクラスとして使用することを前提としています。これらの例をテストするには :

1. 空の Flash ドキュメントを作成して、コンピュータに保存します。
2. 新しい ActionScript ファイルを作成して、Flash ドキュメントと同じディレクトリに保存します。ファイル名は、コードリスト内のクラスの名前と同じ名前にしてください。たとえば、コードリストで "UploadTest" という名前のクラスが定義されている場合、ActionScript ファイルを "UploadTest.as" という名前で保存します。
3. コードリストを ActionScript ファイルにコピーして、ファイルを保存します。
4. Flash ドキュメントで、ステージの何もない部分またはペーストボードをクリックして、ドキュメントのプロパティインスペクタを起動します。
5. プロパティインスペクタの [ドキュメントクラス] フィールドに、テキストからコピーした ActionScript クラスの名前を入力します。
6. [制御]-[ムービープレビュー] を使用してプログラムを実行し、例をテストします。

最後に、この章の一部の例では、サーバー上で実行しているプログラムとのやり取りが含まれています。このような例には、例をテストするのに必要なサーバープログラムを作成するためのコードが含まれています。このような例をテストするには、Web サーバーコンピュータ上に適切なアプリケーションをセットアップする必要があります。

外部データの操作

ActionScript 3.0 には、外部ソースからデータをロードするメカニズムが含まれます。ソースは、テキストファイルなどの静的コンテンツや、データベースからデータを取得する Web スクリプトなどの動的コンテンツである場合があります。このデータはさまざまな方法でフォーマットすることができ、ActionScript の機能によりデータのデコードやアクセスが可能になります。また、データ取得のプロセスの一部として、外部サーバーにデータを送信することもできます。

URLLoader および URLVariables クラスの使用

ActionScript 3.0 では、URLLoader クラスおよび URLVariables クラスを使用して外部データをロードします。URLLoader クラスは、指定した URL からテキスト、バイナリデータ、または URL エンコード形式の変数をダウンロードする際に使用します。データ駆動の動的な ActionScript アプリケーションで使用するテキストファイル、XML、その他の情報をダウンロードする場合には URLLoader クラスが便利です。URLLoader クラスでは ActionScript 3.0 の高度なイベント処理モデルを利用しているため、`complete`、`httpStatus`、`ioError`、`open`、`progress`、`securityError` などのイベントを受け付けて処理できます。新しいイベント処理モデルはエラーやイベントをより効率よく扱うことができ、ActionScript 2.0 における `LoadVars.onData`、`LoadVars.onHTTPStatus`、`LoadVars.onLoad` 各イベントハンドラのサポートに比べて大きく改良されています。イベント処理の詳細については、第 10 章の「イベントの処理」を参照してください。

ActionScript の以前のバージョンにおける XML クラスおよび LoadVars クラスの場合と同様に、URLLoader でも、指定した URL のデータはダウンロードが完了するまで利用可能となりません。ダウンロードの進捗状況 (ロード済みバイト数と合計バイト数) は、`flash.events.ProgressEvent.PROGRESS` イベントを監視することにより監視できますが、ファイルのロードが非常に早く完了した場合は `ProgressEvent.PROGRESS` イベントが送出されないことがあります。ファイルのダウンロードが正常に完了すると、`flash.events.Event.COMPLETE` イベントが送出されます。ロードされたデータは、UTF-8 または UTF-16 のエンコード形式からストリングにデコードされます。

メモ

`URLRequest.contentType` が設定されていない場合、値は `application/x-www-form-urlencoded` 形式で送信されます。

`URLLoader.load()` メソッドのパラメータは、URLRequest インスタンスを指定する `request` パラメータの 1 つだけです (URLLoader クラスのコンストラクタにも、必要に応じて同じパラメータを指定できます)。URLRequest インスタンスには、1 件の HTTP リクエストに関して、ターゲット URL、リクエストメソッド (`GET` または `POST`)、付加的なヘッダ情報、MIME タイプ (XML コンテンツのアップロード時など)、その他すべての情報が格納されます。

たとえば、XML パケットをサーバーサイドスクリプトにアップロードする場合の ActionScript 3.0 コードは次のようになります。

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
  <login>
    <time>{secondsUTC}</time>
    <username>Ernie</username>
    <password>guru</password>
  </login>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/login.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
try
{
  loader.load(request);
}
catch (error:ArgumentError)
{
  trace("An ArgumentError has occurred.");
}
catch (error:SecurityError)
{
  trace("A SecurityError has occurred.");
}
```

上記のコードの一部は、サーバーに送信される XML パケットを含む `dataXml` という名前の XML インスタンスを作成します。次に、`"text/xml"` に URLRequest `contentType` プロパティを設定し、XML パケットのコンテンツに URLRequest `data` プロパティを設定します。このコンテンツは、`XML.toXMLString()` メソッドを使用してストリングに変換されます。最後に、新しい URLLoader インスタンスを作成し、`URLLoader.load()` メソッドを使用してリモートスクリプトに要求を送信します。

URL リクエストで渡すパラメータを指定するには、次の 3 つの方法があります。

- URLVariables コンストラクタで指定する方法
- `URLVariables.decode()` メソッドで指定する方法
- URLVariables オブジェクト自体の特定のプロパティとして設定する方法

URLVariables のコンストラクタまたは `URLVariables.decode()` メソッドで変数を定義する場合、アンパサンド文字 (`&`) は必ず URL エンコードしておく必要があります。アンパサンドには特別な意味があり、区切り文字として機能するからです。たとえば、アンパサンドを渡す場合、アンパサンドはパラメータの区切り記号として機能するため、`&` から `%26` に変更することにより、アンパサンドを URL エンコードする必要があります。

外部ドキュメントからのデータのロード

動的なアプリケーションを ActionScript 3.0 で作成する場合、外部のファイルまたはサーバーサイドスクリプトからデータをロードするようにすると、ActionScript ファイルの編集と再コンパイルを必要としない動的なアプリケーションを実現できます。たとえば、" 今日の一言 " アプリケーションを作成する場合、サーバーサイドスクリプトで、データベースからランダムに選んだ一言を取得して 1 日 1 回テキストファイルに保存するようにします。そうすれば、ActionScript アプリケーションでは静的なテキストファイルをロードするだけでよく、毎回データベースへのクエリを発行する必要はなくなります。

次のコードでは、URLRequest オブジェクトと URLLoader オブジェクトを作成し、それによって "params.txt" という外部のテキストファイルから内容をロードします。

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

上記のコードの一部を、次のように簡素化できます。

```
var loader:URLLoader = new URLLoader(new URLRequest("params.txt"));
```

デフォルトでは、要求メソッドを定義しない場合、Flash Player は HTTP `GET` メソッドを使用してコンテンツをロードします。`POST` メソッドでデータを送信する必要がある場合は、次のように、静的定数 `URLRequestMethod.POST` を使用して `request.method` プロパティに `POST` を設定します。

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

実行時にロードされる外部ドキュメント params.txt には、次のデータが含まれています。

```
monthNames=January,February,March,April,May,June,July,August,September,October,
   November,December&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,
   Saturday
```

このファイルには、`monthNames` と `dayNames` の 2 つのパラメータが含まれます。各パラメータの内容は、ストリングとして解析されるカンマ区切りリストです。このリストは `String.split()` メソッドを使用して分割し、配列に変換できます。

> **ヒント**
> 外部データファイル内の変数名には、予約語やプログラミング言語の構成要素を含めないでください。そのような変数名を使用すると、コードの解読やデバッグ作業が困難になります。

データのロードが完了すると `Event.COMPLETE` イベントが送出され、次のように、URLLoader の `data` プロパティで外部ドキュメントの内容にアクセスできるようになります。

```
private function completeHandler(event:Event):void
{
   var loader2:URLLoader = URLLoader(event.target);
   trace(loader2.data);
}
```

リモートドキュメントに名前と値のペアが含まれる場合、次のように URLVariables クラスを使用してロードされたファイルのコンテンツを渡すことにより、データを解析することができます。

```
private function completeHandler(event:Event):void
{
  var loader2:URLLoader = URLLoader(event.target);
  var variables:URLVariables = new URLVariables(loader2.data);
  trace(variables.dayNames);
}
```

外部ファイルからの個々の名前と値のペアは、URLVariables オブジェクトのプロパティとして作成されます。このコード例にある variables オブジェクトの各プロパティは、ストリングとして扱われます。名前 / 値ペアの値がアイテムのリストである場合は、次のように `String.split()` メソッドを呼び出すことでストリングから配列に変換できます。

```
var dayNameArray:Array = variables.dayNames.split(",");
```

ヒント	外部テキストファイルから数値データをロードするには、トップレベル関数の `int()`、`uint()`、`Number()` などを使用して値を数値に変換する必要があります。

リモートファイルの内容をロードして新しい URLVariables オブジェクトを作成する方法とは別に、`URLLoader.dataFormat` プロパティに URLLoaderDataFormat クラスの静的プロパティのいずれかを設定する方法もあります。`URLLoader.dataFormat` プロパティに設定できる値は次の 3 つのうちいずれかです。

- `URLLoaderDataFormat.BINARY`-`URLLoader.data` プロパティには、ByteArray オブジェクトに格納されたバイナリデータが含まれます。
- `URLLoaderDataFormat.TEXT`-`URLLoader.data` プロパティには、String オブジェクトのテキストが含まれます。
- `URLLoaderDataFormat.VARIABLES`-`URLLoader.data` プロパティには、URLVariables オブジェクトに格納された URL エンコードされた変数が含まれます。

次のコードでは `URLLoader.dataFormat` プロパティを `URLLoaderDataFormat.VARIABLES` に設定しているため、ロードしたデータは自動的に解析されて URLVariables オブジェクトが作成されます。

```
package
{
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLLoader;
  import flash.net.URLLoaderDataFormat;
  import flash.net.URLRequest;

  public class URLLoaderDataFormatExample extends Sprite
  {
    public function URLLoaderDataFormatExample()
    {
```

```
      var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  params.txt");
      var variables:URLLoader = new URLLoader();
      variables.dataFormat = URLLoaderDataFormat.VARIABLES;
      variables.addEventListener(Event.COMPLETE, completeHandler);
      try
      {
        variables.load(request);
      }
      catch (error:Error)
      {
        trace("Unable to load URL: " + error);
      }
    }
    private function completeHandler(event:Event):void
    {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
    }
  }
}
```

メモ

URLLoader.dataFormat のデフォルト値は URLLoaderDataFormat.TEXT です。

次の例に示すように、外部ファイルから XML をロードする方法も、URLVariables をロードする場合と同じです。URLRequest インスタンスと URLLoader インスタンスを作成し、それらを使用してリモート XML ドキュメントをダウンロードします。ファイルのダウンロードが完了すると、Event.COMPLETE イベントが送出され、外部ファイルの内容は XML インスタンスに変換されます。このインスタンスは XML のメソッドとプロパティを使用して解析できます。

```
package
{
  import flash.display.Sprite;
  import flash.errors.*;
  import flash.events.*;
  import flash.net.URLLoader;
  import flash.net.URLRequest;

  public class ExternalDocs extends Sprite
  {
    public function ExternalDocs()
    {
      var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  data.xml");
      var loader:URLLoader = new URLLoader();
      loader.addEventListener(Event.COMPLETE, completeHandler);
      try
      {
        loader.load(request);
```

```
        }
        catch (error:ArgumentError)
        {
          trace("An ArgumentError has occurred.");
        }
        catch (error:SecurityError)
        {
          trace("A SecurityError has occurred.");
        }
      }
      private function completeHandler(event:Event):void
      {
        var dataXML:XML = XML(event.target.data);
        trace(dataXML.toXMLString());
      }
    }
}
```

外部スクリプトとの通信

URLVariables クラスを使用すると、外部データファイルからデータをロードできるだけでなく、サーバーサイドスクリプトに変数を送信し、サーバーの応答を処理することもできます。たとえば、ゲームを作成する場合、ユーザーの得点をサーバーに送信して高得点者リストに掲載できるかどうか確認することや、ユーザーのログイン情報をサーバーに送信して検証することなどが考えられます。サーバーサイドスクリプトでは、ユーザー名とパスワードを処理してデータベースに照会し、入力されたユーザー情報が有効かどうかの確認を応答として返します。

次のコードでは、variables という URLVariables オブジェクトを作成し、このオブジェクトを使用して name という新しい変数を作成します。次に、変数の送信先となるサーバーサイドスクリプトの URL を指定する URLRequest オブジェクトを作成します。その後、URLRequest オブジェクトの method プロパティに、HTTP POST リクエストで変数を送信することを設定します。URL リクエストに URLVariables オブジェクトを追加するために、先ほど作成した URLVariables オブジェクトを URLRequest オブジェクトの data プロパティに設定します。最後に、URLLoader インスタンスを作成し、URLLoader.load() メソッドを呼び出してリクエストを送信します。

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
  loader.load(request);
}
```

```
catch (error:Error)
{
  trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
  trace(event.target.data.welcomeMessage);
}
```

次のコードには、前の例で使用された ColdFusion® greeting.cfm ドキュメントのコンテンツが含まれています。

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
  <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

他の Flash Player インスタンスへの接続

LocalConnection クラスを使用すると、HTML コンテナや内蔵プレーヤーあるいはスタンドアローンプレーヤーの SWF など、異なる Flash Player インスタンス間で通信することができます。これにより、たとえば Web ブラウザ上の SWF ファイルやデスクトップアプリケーションに組み込まれた SWF など複数の Flash Player インスタンス間でデータの共有を実現し、非常に用途の広いアプリケーションを作成できます。

LocalConnection クラス

LocalConnection クラスを使用すると、`fscommand()` メソッドや JavaScript を使用することなく他の SWF ファイルに命令を送信できる SWF ファイルを開発することができます。LocalConnection オブジェクトでの通信相手は、同じクライアントコンピュータ上で動作する SWF ファイルに限られますが、通信相手が同じアプリケーションで動作している必要はありません。たとえば、ブラウザ上で動作している SWF ファイルとプロジェクタの SWF ファイルで情報を共有しつつ、プロジェクタがローカルの情報を管理し、ブラウザ上の SWF がリモート接続を実行するといった分担もできます。プロジェクタは、スタンドアローンアプリケーションとして実行可能な形式で保存される SWF ファイルです。つまり、実行可能ファイルの内部に埋め込まれているため、プロジェクに Flash Player がインストールされている必要はありません。

LocalConnection オブジェクトは、異なるバージョンの ActionScript を使用した SWF 間の通信に使用できます。

- ActionScript 1.0 または 2.0 の LocalConnection オブジェクトは、ActionScript 2.0 で作成された LocalConnection オブジェクトと通信できます。
- ActionScript 1.0 または 2.0 の LocalConnection オブジェクトは、ActionScript 3.0 で作成された LocalConnection オブジェクトと通信できます。

Flash Player は、このバージョンが異なる LocalConnection オブジェクト間の通信を自動的に処理します。

LocalConnection オブジェクトの最も簡単な使用方法は、同じドメイン内の LocalConnection オブジェクト間だけで通信することです。この場合、セキュリティの問題はありません。しかし、異なるドメイン間で通信を行う必要がある場合は、セキュリティ対策を実施する必要があります。いくつかの方法があります。詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』で、`send()` メソッドの `connectionName` パラメータ、`allowDomain()` メソッド、および `domain` エントリ (LocalConnection クラスリスト内) の説明を参照してください。

ヒント	LocalConnection オブジェクトを使用して 1 つの SWF ファイル内でデータを送受信することは可能ですが、お勧めしません。代わりに、共有オブジェクトを使用してください。

LocalConnection オブジェクトにコールバックメソッドを追加するには、次の 3 つの方法があります。

- LocalConnection クラスのサブクラスを作成してメソッドを追加する方法
- `LocalConnection.client` プロパティに、メソッドを実装したオブジェクトを設定する方法
- LocalConnection を拡張した動的なクラスを作成し、動的にメソッドを追加する方法

第 1 の方法では、LocalConnection クラスを拡張することでコールバックメソッドを追加します。LocalConnection インスタンスに動的にメソッドを追加するのではなく、カスタムクラス内にメソッドを定義します。この方法を使用する例を次のコードに示します。

```
package
{
  import flash.net.LocalConnection;
  public class CustomLocalConnection extends LocalConnection
  {
    public function CustomLocalConnection(connectionName:String)
    {
      try
      {
        connect(connectionName);
      }
      catch (error:ArgumentError)
      {
        // サーバーは既に作成され、接続済み
      }
    }
    public function onMethod(timeString:String):void
    {
      trace("onMethod called at: " + timeString);
    }
  }
}
```

DynamicLocalConnection クラスの新しいインスタンスを作成する場合、次のコードを使用できます。

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

コールバックメソッドを追加する 2 つ目の方法は、LocalConnection.client プロパティを使用する方法です。次のように、カスタムクラスを作成し、そのクラスの新しいインスタンスを client プロパティに設定します。

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

LocalConnection.client プロパティは、呼び出されるオブジェクトのコールバックメソッドを指定します。このコードでは、client プロパティに CustomClient というカスタムクラスの新しいインスタンスを設定しています。client プロパティのデフォルト値は、現在の LocalConnection インスタンスです。たとえば、あるウィンドウ内のボタンが別のウィンドウの表示を切り替えるアプリケーションのように、同じメソッドのセットが設定されていながら動作の異なる 2 つのデータハンドラがある場合、client プロパティを使用することができます。

CustomClient クラスを作成するには、次のようなコードを使用します。

```
package
{
  public class CustomClient extends Object
  {
    public function onMethod(timeString:String):void
    {
      trace("onMethod called at: " + timeString);
    }
  }
}
```

コールバックメソッドを追加する 3 つ目の方法は、ダイナミッククラスを作成してメソッドを動的に関連付ける方法です。この方法は次のコードが示すように、ActionScript の以前のバージョンで LocalConnection クラスを使用する方法と非常によく似ています。

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

コールバックメソッドは、次のコードを使用して、このクラスに動的に追加できます。

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// ここにコードを追加する
public function onMethod(timeString:String):void
{
  trace("onMethod called at: " + timeString);
}
```

以前のコールバックメソッドの追加方法はコードの移植があまり簡単ではないため、お勧めしません。また、この方法でローカル接続を作成すると、ダイナミックプロパティへのアクセスは sealed プロパティにアクセスするより大幅に時間を要するため、パフォーマンスの問題が起きるおそれがあります。

2 つの Flash Player インスタンス間のメッセージの送信

LocalConnection クラスを使用すると、Flash Player の異なるインスタンス間での通信ができます。たとえば、1 つの Web ページ上に複数の Flash Player インスタンスがある場合や、ポップアップウィンドウ内の Flash Player インスタンスから提供されるデータを別の Flash Player インスタンスで取得する場合などが考えられます。

次のコードでは、サーバーとして機能するローカル接続オブジェクトを定義し、他の Flash Player インスタンスからの接続を受け付けます。

```
package
{
  import flash.net.LocalConnection;
  import flash.display.Sprite;
  public class ServerLC extends Sprite
  {
    public function ServerLC()
    {
      var lc:LocalConnection = new LocalConnection();
      lc.client = new CustomClient1();
      try
      {
        lc.connect("conn1");
      }
      catch (error:Error)
      {
        trace("error:: already connected");
      }
    }
  }
}
```

このコードは、最初に lc という名前の LocalConnection オブジェクトを作成し、次にカスタムクラス CustomClient1 に対して client プロパティを設定します。このローカル接続インスタンスに対して、他の Flash Player インスタンスからメソッドが呼び出されると、Flash Player は CustomClient1 クラスの該当するメソッドを探します。

何らかの Flash Player インスタンスがこの SWF ファイルに接続し、当該ローカル接続に対して何らかのメソッドを呼び出そうとすると、そのたびに、client プロパティに設定したクラス (CustomClient1 クラス) に対してリクエストが送信されます。

```
package
{
  import flash.events.*;
  import flash.system.fscommand;
  import flash.utils.Timer;
  public class CustomClient1 extends Object
  {
    public function doMessage(value:String = ""):void
```

```
    {
      trace(value);
    }
    public function doQuit():void
    {
      trace("quitting in 5 seconds");
      this.close();
      var quitTimer:Timer = new Timer(5000, 1);
      quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
    }
    public function closeHandler(event:TimerEvent):void
    {
      fscommand("quit");
    }
  }
}
```

LocalConnection サーバーを作成するには、LocalConnection.connect() メソッドを呼び出して一意の名前を指定します。指定した名前の接続が既に存在する場合は ArgumentError エラーが発生します。このエラーは、オブジェクトが接続済みであるために接続の試行が失敗したことを示します。

次のコードでは、conn1 という名前で新しいソケット接続を作成します。

```
try
{
  connection.connect("conn1");
}
catch (error:ArgumentError)
{
  trace("Error! Server already exists\n");
}
```

メモ

ActionScript の以前のバージョンでは、指定した接続名が既に使用されている場合に LocalConnection.connect() メソッドが Boolean 値を返しました。ActionScript 3.0 では、指定した接続名が既に使用されている場合はエラーが発生します。

第 2 SWF ファイルから第 1 SWF ファイルに接続するには、送信側の LocalConnection オブジェクトに新しい LocalConnection オブジェクトを作成し、接続名と実行するメソッド名を使用して LocalConnection.send() メソッドを呼び出す必要があります。たとえば、前の例で作成した LocalConnection オブジェクトに接続するには次のコードを使用します。

```
sendingConnection.send("conn1", "doQuit");
```

このコードは、接続名 conn1 を使用して既存の LocalConnection オブジェクトに接続し、リモート SWF ファイルの doQuit() メソッドを呼び出します。リモート SWF ファイルにパラメータを送信する場合は、次のように、send() メソッドに対するメソッド名の指定の後に追加パラメータを指定します。

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

異なるドメインの SWF ドキュメントへの接続

特定のドメインからの通信だけを許可するには、LocalConnection クラスの allowDomain() メソッドまたは allowInsecureDomain() メソッドを呼び出し、この LocalConnection オブジェクトへのアクセスを許可された 1 つまたは複数のドメインのリストを渡します。

ActionScript の以前のバージョンでは、開発者が LocalConnection.allowDomain() および LocalConnection.allowInsecureDomain() をコールバックメソッドとして実装する必要があり、また、いずれも戻り値として Boolean 値を返すようにする必要がありました。ActionScript 3.0 では、LocalConnection.allowDomain() および LocalConnection.allowInsecureDomain() はいずれも内蔵メソッドとなりました。開発者はアクセスを許可するドメイン名のリストを指定して、これらのメソッドを Security.allowDomain() や Security.allowInsecureDomain() と同様に呼び出すだけで済みます。

LocalConnection.allowDomain() および LocalConnection.allowInsecureDomain() メソッドに対して指定できる特別な値として、* および localhost の 2 つがあります。アスタリスク値 (*) は、すべてのドメインにアクセスを許可することを示します。localhost というストリングは、ローカル環境にインストールされている SWF ファイルから当該 SWF ファイルへの呼び出しを許可することを示します。

Flash Player 8 から、ローカル SWF ファイルに関するセキュリティ制限が導入されました。これにより、インターネットへのアクセスを許可された SWF ファイルは、ローカルファイルシステムにはアクセスできません。localhost を指定した場合、すべてのローカル SWF ファイルは当該 SWF ファイルにアクセスできます。セキュリティ Sandbox 内から、呼び出し元コードがアクセスを認められていない SWF ファイルに対して LocalConnection.send() メソッドで接続しようとすると、securityError イベント (SecurityErrorEvent.SECURITY_ERROR) が送出されます。このエラーを回避するには、受信側の LocalConnection.allowDomain() メソッドで、呼び出し元のドメインを指定してください。

同じドメイン内の SWF ファイル間でのみ通信を行う場合は、connectionName パラメータに対して、先頭がアンダースコア (_) 以外で始まり、しかもドメイン名を含まない名前を指定します (例 : myDomain:connectionName)。また、それと同じストリングを LocalConnection.connect(connectionName) コマンドでも使用します。

異なるドメインにある SWF ファイル間での通信を実装する場合は、connectionName パラメータに対して、先頭がアンダースコアで始まる名前を指定します。このアンダースコアを付けると、受信側 LocalConnection オブジェクトを含む SWF ファイルのドメイン間におけるポータビリティが高まります。考えられる 2 つの状況を次に示します。

- connectionName のストリングがアンダースコアで始まっていない場合、Flash Player によってスーパードメイン名にプレフィクスとコロンが追加されます (例 : myDomain:connectionName)。この処理には他のドメインにある同じ名前の接続との競合を回避する意味がありますが、その代わり、このスーパードメインをすべての送信側 LocalConnection オブジェクトで指定する必要があります (例 : myDomain:connectionName)。受信側 LocalConnection オブジェクトを含んだ SWF ファイルを別のドメインに移動すると、Flash Player によって追加されるプレフィクスは、スーパードメインの変更を反映して変化します (例 : anotherDomain:connectionName)。したがって、新しいスーパードメインを参照するようにすべての送信側 LocalConnection オブジェクトを手動で編集する必要が生じます。
- connectionName のストリングがアンダースコアで始まる場合 (例 : _connectionName)、Flash Player によってストリングにプレフィクスが追加されることはありません。そのため、受信側と送信側の両方の LocalConnection オブジェクトで、connectionName に同じストリングを使用できます。すべてのドメインからの接続を受け付けることを受信側オブジェクトが LocalConnection.allowDomain() によって指定していれば、受信側 LocalConnection オブジェクトを含んだ SWF ファイルを別のドメインに移動した場合でも、送信側 LocalConnection オブジェクトに変更を加える必要はありません。

ソケットの接続

ActionScript 3.0 で可能なソケットの接続には、XML ソケット接続とバイナリソケット接続の 2 種類があります。XML ソケットでリモートサーバーに接続する場合は、明示的に接続を閉じるまで維持されるサーバー接続を確立できます。これにより、サーバー接続を確立し直す処理を繰り返すことなくサーバー / クライアント間で XML データの交換を継続できます。XML ソケットサーバーを使用するもう 1 つのメリットは、ユーザーが明示的にデータを要求する必要がないことです。要求がなくてもサーバーからのデータ送信ができ、また、XML ソケットサーバーに接続済みのクライアントすべてに対してデータを送信できます。

バイナリソケット接続は、XML ソケットに似ていますが、クライアント / サーバー間で交換するデータが XML パケットである必要はないという点が異なります。データはバイナリ情報として伝送されます。このため、電子メールサーバー (POP3、SMTP、IMAP) やニュースサーバー (NNTP) などさまざまなサービスとの接続に使用できます。

Socket クラス

ActionScript 3.0 で新設された Socket クラスを使用すると、ActionScript でソケット接続を確立して生のバイナリデータを読み書きできます。これは XMLSocket クラスに似ていますが、送受信するデータの形式に制約がありません。Socket クラスは、バイナリプロトコルを使用するサーバーとの通信に役立ちます。バイナリソケット接続を使用すれば、POP3、SMTP、IMAP、NNTP などさまざまなインターネットプロトコルによる通信のコードを記述できます。したがって、Flash Player から電子メールサーバーやニュースサーバーに接続することもできます。

Flash Player からサーバーに直接接続するには、サーバーが使用するバイナリプロトコルに従う必要があります。バイト順序にビッグエンディアンを使用するサーバーと、リトルエンディアンを使用するサーバーがあります。標準の " ネットワークバイト順序 " がビッグエンディアンとされているため、インターネット上の大部分のサーバーはビッグエンディアンのバイト順序を使用しています。また、Intel x86 アーキテクチャでリトルエンディアンが採用されているため、リトルエンディアンのバイト順序が使用されることもよくあります。データを送受信する対象のサーバーに適したエンディアンバイト順序を使用してください。IDataInput および IDataOutput インターフェイスで実行されるすべての操作と、これらのインターフェイスを実装するクラス (ByteArray、Socket、URLStream) では、デフォルトでビッグエンディアン形式 (先頭が最上位バイト) を使用します。これは、Java および公式なネットワークバイト順序に適合するためです。使用するエンディアンを変更する場合は、`endian` プロパティに `Endian.BIG_ENDIAN` または `Endian.LITTLE_ENDIAN` を設定します。

ヒント	Socket クラスでは、IDataInput および IDataOutput インターフェイス (flash.utils パッケージ) で実装するすべてのメソッドを継承しています。Socket の読み書きにはそれらのメソッドを使用してください。

XMLSocket クラス

ActionScript には、サーバーとの連続的な接続を確立できる XMLSocket ビルトインクラスがあります。接続が維持されるため待ち時間の問題が解消されるので、チャットやマルチプレーヤーゲームなどのリアルタイムアプリケーションによく使用されます。従来の HTTP ベースによるチャットソリューションでは、サーバーへのポーリング処理を頻繁に実行し、HTTP リクエストを使用して新しいメッセージをダウンロードします。それに対し、XMLSocket によるチャットソリューションではサーバーとの接続を開いたまま維持するため、クライアントから要求されなくても、新着メッセージをサーバーから直ちに送信できます。

ソケット接続を確立するには、ソケット接続の要求を受け付けて SWF ファイルに応答を送るサーバーサイドアプリケーションを作成する必要があります。このようなサーバーサイドアプリケーションは、Java、Python、Perl などのプログラミング言語で作成できます。XMLSocket クラスを使用するには、サーバーコンピュータで動作するデーモンが XMLSocket クラスで使用するプロトコルを処理できる必要があります。プロトコルの説明を次の一覧に示します。

- XML メッセージは、全二重 TCP/IP ストリームソケット接続を介して送られます。
- 個々の XML メッセージは完全な XML ドキュメントであり、ゼロ (0) バイトで終了します。
- 1 つの XMLSocket 接続を使用して送受信できる XML メッセージの数に制限はありません。

メモ

XMLSocket クラスは、ファイアウォールをトンネリングによって自動的に通過することはできません。RTMP プロトコルとは異なり、XMLSocket には HTTP トンネリング機能が備わっていないためです。HTTP トンネリングが必要な場合は、Flash Remoting または (RTMP をサポートする) Flash Media Server の使用をお勧めします。

XMLSocket オブジェクトがサーバーに接続する方法と接続先については、次の制限があります。

- `XMLSocket.connect()` メソッドが接続できる TCP ポートの番号は、1024 以上です。この制限により、`XMLSocket` オブジェクトと通信するサーバーデーモンにも、1024 以上のポート番号を割り当てる必要があります。1024 未満のポート番号は、FTP (21)、Telnet (23)、SMTP (25)、HTTP (80)、POP3 (110) などのシステムサービスによって使用されることが多いため、セキュリティ上の理由から、XMLSocket オブジェクトではこれらのポートにアクセスできません。こうしたリソースが不適切な方法でアクセスされたり悪用されたりする可能性を小さくするために、ポート番号が制限されています。
- `XMLSocket.connect` メソッドは、SWF ファイルが存在するのと同じドメイン内のコンピュータにしか接続できません。この制限は、ローカルディスクから再生される SWF ファイルには適用されません (この制限は、`URLLoader.load()` のセキュリティ規則と同じです)。特定のドメインからのアクセスを許可するセキュリティポリシーファイルを作成すると、SWF ファイルが存在するドメイン以外で実行されるサーバーデーモンに接続できます。

メモ

XMLSocket オブジェクトと通信できるようにサーバーを設定することは、場合によっては困難が伴います。リアルタイムのインタラクティブ機能を必要としないアプリケーションでは、XMLSocket クラスではなく URLLoader クラスを使用してください。

XMLSocket クラスの `XMLSocket.connect()` メソッドと `XMLSocket.send()` メソッドを使用し、ソケット接続を介してサーバーへ XML を送信 (またはサーバーから受信) することができます。`XMLSocket.connect()` メソッドは、Web サーバーのポートに対してソケット接続を確立します。`XMLSocket.send()` メソッドは、ソケット接続で指定されたサーバーに XML オブジェクトを送信します。

XMLSocket.connect() メソッドを呼び出すと、Flash Player によってサーバーへの TCP/IP 接続が開かれ、次のいずれかのイベントが発生するまでその接続が開いたまま維持されます。

- XMLSocket クラスの XMLSocket.close() メソッドが呼び出される。
- XMLSocket オブジェクトへの参照が 1 つも存在しなくなる。
- Flash Player が終了する。
- 接続が切れる (モデムの切断など)。

Java XML ソケットサーバーの作成と接続

次のコードは、Java で作成した単純な XMLSocket サーバーの例です。接続を受け付け、受信したメッセージをコマンドプロンプトウィンドウに表示します。デフォルトでは新しいサーバーをローカルマシンのポート番号 8080 に作成しますが、サーバーを起動する際のコマンドラインでポート番号を変更することもできます。

新しいテキストドキュメントを作成し、次のコードを入力します。

```
import java.io.*;
import java.net.*;

class SimpleServer
{
  private static SimpleServer server;
  ServerSocket socket;
  Socket incoming;
  BufferedReader readerIn;
  PrintStream printOut;

  public static void main(String[] args)
  {
    int port = 8080;

    try
    {
      port = Integer.parseInt(args[0]);
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      // 例外をキャッチして続行する
    }

    server = new SimpleServer(port);
  }

  private SimpleServer(int port)
  {
    System.out.println(">> Starting SimpleServer");
    try
```

```
    {
      socket = new ServerSocket(port);
      incoming = socket.accept();
      readerIn = new BufferedReader(new
  InputStreamReader(incoming.getInputStream()));
      printOut = new PrintStream(incoming.getOutputStream());
      printOut.println("Enter EXIT to exit.\r");
      out("Enter EXIT to exit.\r");
      boolean done = false;
      while (!done)
      {
        String str = readerIn.readLine();
        if (str == null)
        {
          done = true;
        }
        else
        {
          out("Echo: " + str + "\r");
          if(str.trim().equals("EXIT"))
          {
            done = true;
          }
        }
        incoming.close();
      }
    }
    catch (Exception e)
    {
      System.out.println(e);
    }
  }

  private void out(String str)
  {
    printOut.println(str);
    System.out.println(str);
  }
}
```

ドキュメントを SimpleServer.java としてハードディスクに保存し、Java コンパイラを使用してこれをコンパイルすると、SimpleServer.class という名前の Java クラスが作成されます。

コマンドプロンプトを開き、`java SimpleServer` と入力して XMLSocket サーバーを起動します。"SimpleServer.class" ファイルはローカルコンピュータ上またはネットワーク上のどこに配置しても使用できます。Web サーバーのルートディレクトリに置く必要はありません。

ヒント	ファイルを置いた場所が Java クラスパスに含まれていないためにサーバーを起動できない場合は、`java -classpath . SimpleServer` と入力すれば起動できることがあります。

ActionScript アプリケーションから XMLSocket に接続するには、次のように XMLSocket クラスの新しいインスタンスを作成し、ホスト名とポート番号を指定して `XMLSocket.connect()` メソッドを呼び出します。

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

`XMLSocket.connect()` の呼び出しによって、呼び出し元のセキュリティ Sandbox の外部サーバーや 1024 未満のポート番号に接続しようとした場合、`securityError` (flash.events.SecurityErrorEvent) イベントが発生します。

サーバーからデータを受信するたびに、`data` イベント (`flash.events.DataEvent.DATA`) が送出されます。

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
   trace("[" + event.type + "] " + event.data);
}
```

XMLSocket サーバーにデータを送信するには、`XMLSocket.send()` メソッドを使用して XML オブジェクトまたは XML ストリングを渡します。Flash Player によってパラメータが String オブジェクトに変換され、その内容に終端を示すゼロ (0) バイトを付加したデータが XMLSocket サーバーに送信されます。

```
xmlsock.send(xmlFormattedData);
```

`XMLSocket.send()` メソッドは、データが正常に転送されたかどうかを示す値を返しません。データの送信中にエラーが発生した場合は、IOError エラーがスローされます。

ヒント	XML ソケットサーバーに送信する個々のメッセージには、末尾に終端の改行文字 (`\n`) を付ける必要があります。

ローカルデータの保存

"Flash cookie" とも呼ばれる共有オブジェクトは、アクセスしたサイトによってコンピュータ上に作成されるデータファイルです。ほとんどの場合、共有オブジェクトは、ユーザーが頻繁にアクセスする Web サイトの外観をパーソナライズする場合など、Web ブラウジングの操作性を向上するために使用されます。共有オブジェクト自体がコンピュータ上にあるデータにアクセスすることはできません。さらに重要な点は、ユーザーが意図的に情報を提供しない限り、共有オブジェクトがユーザーの電子メールアドレスやその他の個人情報にアクセスしてデータを記憶することはできないということです。

新しい共有オブジェクトのインスタンスは、静的な `SharedObject.getLocal()` メソッドまたは `SharedObject.getRemote()` メソッドを使用して作成することができます。`getLocal()` メソッドでは、現在のクライアントだけがアクセスできるローカルの永続共有オブジェクトに対してロードを試みます。一方、`getRemote()` メソッドでは、何らかのサーバー (Flash Media Server など) によって複数クライアントでの共有ができるリモートの共有オブジェクトに対してロードを試みます。目的のローカル共有オブジェクトやリモート共有オブジェクトが存在しない場合、`getLocal()` メソッドと `getRemote()` メソッドは新しい SharedObject インスタンスを返します。

次のコードでは、`test` というローカル共有オブジェクトをロードしようとしています。この共有オブジェクトが存在しなければ、指定した名前の新しい共有オブジェクトが作成されます。

```
var so:SharedObject = SharedObject.getLocal("test");
trace("SharedObject is " + so.size + " bytes");
```

test という名前の共有オブジェクトが見つからない場合、サイズが 0 バイトの新しい共有オブジェクトが作成されます。該当する共有オブジェクトが既に存在する場合は、現在のサイズ (バイト単位) が返されます。

共有オブジェクトにデータを格納するには、次のようにデータオブジェクトに値を割り当てます。

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.now = new Date().time;
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

名前が `test` でパラメータが `now` に設定された共有オブジェクトが既に存在する場合、既存の値は上書きされます。共有オブジェクトが既に存在するかどうかを確認するには、次の例に示すように、`SharedObject.size` プロパティを使用します。

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
   // 共有オブジェクトは存在しない
   trace("created...");
   so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

前のコードでは、size パラメータを使用して、指定された名前を持つ共有オブジェクトインスタンスが既に存在するかどうかを確認しています。次のコードでは、実行するたびに共有オブジェクトを改めて作成します。共有オブジェクトをユーザーのハードディスクに保存するには、次の例に示すように、SharedObject.flush() メソッドを明示的に呼び出す必要があります。

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
   // 共有オブジェクトは存在しない
   trace("created...");
   so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

flush() メソッドを使用してユーザーのハードディスクに共有オブジェクトを書き込む場合は、次の例に示すように、Flash Player の設定マネージャ (www.macromedia.com/support/documentation/jp/flashplayer/help/settings_manager07.html) を使用してローカル記憶域を明示的に無効にしてあるかどうかを慎重に確認する必要があります。

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

共有オブジェクトの data プロパティ内でプロパティの名前を指定することにより、共有オブジェクトから値を取得することができます。たとえば、次のコードでは、SharedObject インスタンスが作成された時点からの経過時間 (分単位) を Flash Player に表示します。

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
   // 共有オブジェクトは存在しない
   trace("created...");
   so.data.now = new Date().time;
}
var ageMS:Number = new Date().time - so.data.now;
trace("SharedObject was created " + Number(ageMS / 1000 / 60).toPrecision(2) + "
   minutes ago");
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

前のコードを初めて実行すると、test という名前の新しい SharedObject が初期サイズ 0 バイトで作成されます。初期サイズは 0 バイトで if ステートメントの条件が true になるため、now という新しいプロパティが共有オブジェクトに追加されます。共有オブジェクト作成時からの経過時間は、現在時刻から now プロパティの値を減算して求めています。以後、このコードが再度実行される際には共有オブジェクトのサイズが 0 より大きくなっているため、共有オブジェクト作成時からの経過時間をトレース出力します。

共有オブジェクトのコンテンツの表示

共有オブジェクトに保存する値は、data プロパティ内に格納されます。ループで共有オブジェクトインスタンス内の個々の値にアクセスするには、次のように for..in ループを使用します。

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
  trace(i + ":\t" + so.data[i]);
}
```

セキュアな SharedObject の作成

getLocal() または getRemote() でローカルまたはリモートの SharedObject を作成する際にオプションの secure というパラメータを指定すると、その共有オブジェクトについて、HTTPS 経由で配信される SWF ファイルのみにアクセスを制限するかどうかを定義できます。HTTPS 経由で配信される SWF ファイルにおいて、このパラメータに true を指定すると、セキュアな共有オブジェクトを新しく作成、または既存のセキュアな共有オブジェクトに対する参照を取得できます。このセキュアな共有オブジェクトは、HTTPS 経由で配信される SWF ファイルで、secure パラメータに true を指定して SharedObject.getLocal() を呼び出した場合にのみ読み書きできます。HTTPS 経由で配信される SWF ファイルにおいて、このパラメータに false を指定すると、共有オブジェクトを新しく作成、または既存の共有オブジェクトに対する参照を取得でき、

この場合の共有オブジェクトは HTTPS 以外の接続を経由して配信される SWF ファイルからも読み書きできます。HTTPS 以外の接続を経由して配信される SWF ファイルにおいて、このパラメータに true を指定しようとすると、新しい共有オブジェクトの作成 (または既存のセキュアな共有オブジェクトへのアクセス) に失敗してエラーがスローされ、共有オブジェクトには null が設定されます。次のコードを、HTTPS 以外の接続を経由して配信される SWF ファイルで実行しようとすると、SharedObject.getLocal() メソッドでエラーがスローされます。

```
try
{
  var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
  trace("Unable to create SharedObject.");
}
```

このパラメータの値にかかわらず、作成された共有オブジェクトは、各ドメインに割り当てられた総ディスク容量の一部を使用します。

ファイルのアップロードおよびダウンロード操作

FileReference クラスを使用すると、クライアントとサーバー間におけるファイルのアップロード機能とダウンロード機能を追加することができます。ユーザーはダイアログボックス (Windows OS の [開く] ダイアログボックスなど) によって、アップロードするファイルやダウンロード先の場所を指定するよう要求されます。

ActionScript で作成した個々の FileReference オブジェクトは、ユーザーのハードディスク上にある同じファイルを参照します。このオブジェクトには、ファイルのサイズ、種類、名前、作成日、および変更日の情報を格納するプロパティがあります。

メモ

creator プロパティは Mac OS でのみサポートされており、その他すべてのプラットフォームでは null を返します。

FileReference クラスのインスタンスを作成する方法は 2 種類あります。1 つは、次のコードのように new 演算子を使用する方法です。

```
import flash.net.FileReference;
var myFileReference:FileReference = new FileReference();
```

あるいは、FileReferenceList.browse() メソッドを呼び出すこともできます。このメソッドは、アップロードするファイルの選択用ダイアログボックスをユーザーのシステム上で開き、ユーザーがファイルを正常に選択すると FileReference オブジェクトの配列を作成します。個々の FileReference オブジェクトは、ユーザーがダイアログボックスで指定した 1 つのファイルを表します。FileReference オブジェクトに含まれる各種の FileReference プロパティ (name、size、modificationDate など) には、次のいずれかが行われるまでは何もデータが格納されていません。

- FileReference.browse() メソッドまたは FileReferenceList.browse() メソッドが呼び出され、ユーザーがファイルピッカーからファイルを選択する。
- FileReference.download() メソッドが呼び出され、ユーザーがファイルピッカーからファイルを選択する。

メモ

ダウンロードを実行する場合、FileReference.name プロパティだけはダウンロードの完了より前に設定されます。ダウンロードが完了すると、すべてのプロパティの値が設定されます。

FileReference.browse()、FileReferenceList.browse()、FileReference.download() いずれかのメソッドに対する呼び出しの実行中、ほとんどの Player では SWF の再生が続けられます。

FileReference クラス

FileReference クラスを使用すると、ユーザーのコンピュータとサーバーとの間でファイルのアップロードおよびダウンロード機能を実現できます。ユーザーに対してアップロードするファイルやダウンロード先の場所の指定を求めるために、OS のダイアログボックスを表示します。個々の FileReference オブジェクトはユーザーのディスク上にある 1 つのファイルを表し、ファイルのサイズ、タイプ、名前、作成日、変更日、クリエータに関する情報をプロパティとして保持します。

FileReference インスタンスを作成するには、次の 2 つの方法があります。

- 次のように new 演算子で FileReference コンストラクタを使用する方法。
  `var myFileReference:FileReference = new FileReference();`
- `FileReferenceList.browse()` を呼び出す方法 (このメソッドによって FileReference オブジェクトの配列が作成される)。

アップロード処理およびダウンロード処理では、当該 SWF ファイル自体が属するドメイン内のファイルと、クロスドメインポリシーファイルで指定されたドメイン内のファイルに対してのみアクセスできます。アップロードまたはダウンロードを開始する SWF ファイルがファイルサーバーと同じドメインに属していない場合は、ファイルサーバー上にポリシーファイルを配置する必要があります。

メモ

browse() または download() のアクションを複数並行して実行することはできません。これは、一度に 1 つしかダイアログボックスを表示できないためです。

ファイルのアップロードを処理するサーバースクリプトは、次の要素を含んだ HTTP POST リクエストを受け付ける必要があります。

- Content-Type : 設定されている値は multipart/form-data。
- Content-Disposition : name 属性に "Filedata"、filename 属性に元のファイルの名前。FileReference.upload() メソッドの uploadDataFieldName パラメータを指定すると、name 属性に独自の値を指定できます。
- ファイルの内容を表すバイナリデータ。

HTTP POST リクエストの例を次に示します。

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
```

```
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

Test File
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

次に示すサンプルの HTTP POST 要求は、3 つの POST 変数を送信します。つまり、api_sig、api_key、および auth_token です。この例では、カスタムアップロードデータフィールド名の値は、"photo" です。

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

サーバーへのファイルのアップロード

サーバーにファイルをアップロードするには、まず browse() メソッドを呼び出して、ユーザーにファイルの選択を求めます。次に、FileReference.upload() メソッドが呼び出されると、選択したファイルがサーバーに転送されます。ユーザーが FileReferenceList.browse() メソッドを使用して複数のファイルを選択した場合、Flash Player は FileReferenceList.fileList と呼ばれる選択されたファイルの配列を作成します。その後、各ファイルを個別に FileReference.upload() メソッドでアップロードします。

メモ

FileReference.browse() メソッドを使用する場合は、ファイルを 1 つだけアップロードできます。複数のファイルをアップロードできるようにするには、FileReferenceList.browse() メソッドを使用する必要があります。

デフォルトでは、ユーザーはシステムのファイルピッカーダイアログボックスを使用して、ローカルコンピュータ上にある任意のタイプのファイルを選択できます。ただし、次のように FileFilter クラスを使用してファイルタイプのフィルタを作成し、ファイルフィルタインスタンスの配列を browse() メソッドで指定すれば、選択できるファイルを限定することもできます。

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif,
  *.png)", "*.jpg; *.jpeg; *.gif; *.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt;
  *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

ユーザーがシステムのファイルピッカーでファイルを選択し、[開く] ボタンをクリックすると、Event.SELECT イベントが送出されます。FileReference.browse() メソッドを使用してファイルをアップロードする場合は、次のコードで Web サーバーにファイルを送信する必要があります。

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
  var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
  trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  fileUploadScript.cfm")
  try
  {
    fileRef.upload(request);
```

```
    }
    catch (error:Error)
    {
      trace("Unable to upload file.");
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

ヒント

FileReference.upload() メソッドでは、URLRequest.method および URLRequest.data プロパティを使用して、POST または GET メソッドで変数のデータをサーバーに送信することもできます。

FileReference.upload() メソッドを使用してファイルをアップロードしようとする際には、次のイベントが送出されることがあります。

- Event.OPEN: アップロード処理が開始したときに送出されます。
- ProgressEvent.PROGRESS: ファイルのアップロード処理中に定期的に送出されます。
- Event.COMPLETE: ファイルのアップロード処理が正常に完了したときに送出されます。
- SecurityErrorEvent.SECURITY_ERROR: セキュリティ侵害が原因でアップロードが失敗した場合に送出されます。
- HTTPStatusEvent.HTTP_STATUS: HTTP エラーが原因でアップロードが失敗した場合に送出されます。
- IOErrorEvent.IO_ERROR: 次のいずれかの原因でアップロードが失敗した場合に送出されます。
  - Flash Player によるファイルの読み込み中、書き込み中、または転送中に入出力エラーが発生した場合。
  - SWF ファイルが、認証 (例 : ユーザー名とパスワード) を必要とするサーバーにファイルをアップロードしようとした場合。Flash Player には、アップロード時にユーザーがパスワードを入力する手段が用意されていません。
  - url パラメータに無効なプロトコルが指定されている場合。FileReference.upload() メソッドでは、HTTP または HTTPS を使用する必要があります。

ヒント

認証を必要とするサーバーに対する Flash Player のサポートは完全ではありません。認証用のユーザー名とパスワードを入力するダイアログボックスを表示できるのは、ブラウザプラグインまたは Microsoft ActiveX® コントロールを使用してブラウザ上で実行される SWF ファイルにおいて、ダウンロードを実行する場合のみです。プラグインまたは ActiveX コントロールを使用する場合のアップロードと、スタンドアローン Player または外部 Player を使用する場合のアップロードおよびダウンロードでは、ファイル転送に失敗します。

Flash Player によるファイルのアップロードを受け付けるスクリプトを ColdFusion で作成する場合は、次のようなコードを使用します。

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#"
  nameconflict="OVERWRITE" />
```

この ColdFusion コードでは、Flash Player から送信されたファイルをアップロードし、ColdFusion テンプレートと同じディレクトリに保存します。同じ名前のファイルが既に存在する場合は上書きされます。このコードではファイルのアップロードに必要な最低限の処理しか行っていないため、実際の運用環境ではこのスクリプトを使用しないでください。データ検証のコードを付け加えて、特定のファイルタイプ (イメージなど) についてのみアップロードを許可し、サーバーサイドスクリプトなど危険が生じる可能性のあるファイルを受け付けないようにすることをお勧めします。

次のコードは PHP によるファイルのアップロード処理の例で、データ検証も実行します。このスクリプトでは、アップロード先ディレクトリにアップロードするファイルの数を 10 個まで、ファイルサイズを 200 KB までに制限し、しかも、JPEG、GIF、PNG のいずれかのファイルだけをアップロードしてファイルシステムに保存できるようにしています。

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
  move_uploaded_file($_FILES['Filedata']['tmp_name'], "./temporary/
  ".$_FILES['Filedata']['name']);
  $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
  if ($type == 1 || $type == 2 || $type == 3)
  {
    rename("./temporary/".$_FILES['Filedata']['name'], "./images/
  ".$_FILES['Filedata']['name']);
  }
  else
  {
    unlink("./temporary/".$_FILES['Filedata']['name']);
  }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory))
{
  array_push($files, array('./images/'.$file, filectime('./images/'.$file)));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
  $files_to_delete = array_splice($files, 0, count($files) -
  $MAXIMUM_FILE_COUNT);
  for ($i = 0; $i < count($files_to_delete); $i++)
```

```
  {
    unlink($files_to_delete[$i][0]);
  }
}
print_r($files);
closedir($directory);

function sorter($a, $b)
{
  if ($a[1] == $b[1])
  {
    return 0;
  }
  else
  {
    return ($a[1] < $b[1]) ? -1 : 1;
  }
}
?>
```

このアップロードスクリプトには、POST または GET リクエストメソッドを使用して追加の変数を渡すこともできます。追加の POST 変数をアップロードスクリプトに送信するには、次のようなコードを使用します。

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
  var params:URLVariables = new URLVariables();
  params.date = new Date();
  params.ssid = "94103-1394-2345";
  var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  FileReferenceUpload/fileupload.cfm");
  request.method = URLRequestMethod.POST;
  request.data = params;
  fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
  trace("uploaded");
}
```

この例では、新しい URLVariables オブジェクトを作成してリモートのサーバーサイドスクリプトに送信しています。ActionScript の以前のバージョンでは、変数をサーバーアップロードスクリプトに送信するには値をクエリ文字列に含めて渡す必要がありました。ActionScript 3.0 では、URLRequest オブジェクトを使用して POST または GET メソッドで変数をリモートスクリプトに送信します。これにより、データのセット数が多い場合でも容易に、簡潔に送信処理を記述できるようになりました。変数の受け渡しに使用する GET または POST リクエストメソッドを指定するには、URLRequest.method プロパティに URLRequestMethod.GET (GET メソッドの場合) または URLRequestMethod.POST (POST メソッドの場合) を設定します。

また、ActionScript 3.0 では、upload() メソッドの第 2 パラメータを指定することで、デフォルトの Filedata アップロードファイルフィールド名を変更できます。前のコード例では、この操作の例としてデフォルト値の Filedata を Custom1 に置き換えています。

Flash Player のデフォルトではテストアップロードが試行されませんが、upload() メソッドの第 3 パラメータに true を指定するとこの動作を変更できます。テストアップロードの目的は、実際のファイルアップロードを正常に実行できるかどうか試すことと、(認証が必要な場合に) 認証を通過できるかどうか確認することです。

メモ　テストアップロードは、現在のところ Windows 版の各種 Flash Player でのみ実行されます。

サーバーからのファイルのダウンロード

FileReference.download() メソッドを使用すると、サーバー上のファイルをユーザーがダウンロードできます。このメソッドには、request および defaultFileName の 2 つのパラメータがあります。第 1 パラメータには、ダウンロードするファイルの URL を表す URLRequest オブジェクトを指定します。第 2 パラメータはオプションです。ダウンロードファイルのダイアログボックスに表示されるデフォルトのファイル名を指定できます。第 2 パラメータの defaultFileName を省略すると、指定した URL に含まれているファイル名が使用されます。

次のコードでは、SWF ドキュメントと同じディレクトリから "index.xml" というファイルをダウンロードします。

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

デフォルトの名前を "index.xml" ではなく "currentnews.xml" に設定するには、次のように defaultFileName パラメータを指定します。

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

サーバーにあるファイルに意味のわかりにくい名前が付いている場合や、ファイル名がサーバーで自動生成される場合、ファイル名の変更は非常に役立ちます。また、ファイルを直接にダウンロードするのではなくサーバーサイドスクリプトによってダウンロードする場合も、defaultFileName パラメータを明示的に指定することをお勧めします。たとえば、指定した URL 変数に基づいて特定のファイルをダウンロードするようなサーバーサイドスクリプトに対しては、defaultFileName パラメータを指定する必要があります。指定しない場合、ダウンロードするファイルのデフォルトの名前はサーバーサイドスクリプトの名前と同じになります。

データ解析用サーバースクリプトを表す URL にパラメータを追加すると、download() メソッドを使用してサーバーにデータを送信できます。次の ActionScript 3.0 コードでは、ColdFusion スクリプトに渡すパラメータに基づいてドキュメントをダウンロードします。

```
package
{
  import flash.display.Sprite;
  import flash.net.FileReference;
  import flash.net.URLRequest;
  import flash.net.URLRequestMethod;
  import flash.net.URLVariables;

  public class DownloadFileExample extends Sprite
  {
    private var fileToDownload:FileReference;
    public function DownloadFileExample()
    {
      var request:URLRequest = new URLRequest();
      request.url = "http://www.[yourdomain].com/downloadfile.cfm";
      request.method = URLRequestMethod.GET;
      request.data = new URLVariables("id=2");
      fileToDownload = new FileReference();
      try
      {
        fileToDownload.download(request, "file2.txt");
      }
      catch (error:Error)
      {
        trace("Unable to download file.");
      }
    }
  }
}
```

次のコードは "download.cfm" という ColdFusion スクリプトの例です。URL 変数の値に応じて、サーバー上にある 2 つのファイルのうちいずれかをダウンロードします。

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
  <cfcase value="2">
    <cfcontent type="text/plain" file="#ExpandPath('two.txt')#" deletefile="No"
  />
  </cfcase>
  <cfdefaultcase>
    <cfcontent type="text/plain" file="#ExpandPath('one.txt')#" deletefile="No"
  />
  </cfdefaultcase>
</cfswitch>
```

FileReferenceList クラス

FileReferenceList クラスを使用すると、サーバーサイドスクリプトにアップロードするファイルをユーザーが指定できます。ファイルのアップロードについては、ユーザーが指定した個々のファイルに対して FileReference.upload() メソッドを呼び出すことにより処理する必要があります。

次のコードでは、2 つの FileFilter オブジェクト (imageFilter および textFilter) を作成し、それらを配列として FileReferenceList.browse() メソッドに渡しています。これにより、OS のファイルダイアログボックスにはファイルタイプを指定するフィルタが 2 つ表示されます。

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg, *.gif,
  *.png)", "*.jpg; *.jpeg; *.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt;
  *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
  var success:Boolean = fileRefList.browse(new Array(imageFilter, textFilter));
}
catch (error:Error)
{
  trace("Unable to browse for files.");
}
```

FileReferenceList クラスを使用して複数のファイルを選択およびアップロードできるようにする場合も、ファイルの選択は FileReference.browse() と同様に行いますが、複数のファイルを選択できる点が異なります。複数のファイルをアップロードする際は、次のように、個々のファイルについて FileReference.upload() を呼び出す必要があります。

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  fileUploadScript.cfm");
  var file:FileReference;
  var files:FileReferenceList = FileReferenceList(event.target);
  var selectedFileArray:Array = files.fileList;
  for (var i:uint = 0; i < selectedFileArray.length; i++)
  {
    file = FileReference(selectedFileArray[i]);
    file.addEventListener(Event.COMPLETE, completeHandler);
    try
    {
      file.upload(request);
    }
    catch (error:Error)
    {
      trace("Unable to upload files.");
    }
  }
}
function completeHandler(event:Event):void
{
  trace("uploaded");
}
```

配列内の各 FileReference オブジェクトに対して Event.COMPLETE イベントを登録しているため、各ファイルのアップロードが完了するたびに、Flash Player によって completeHandler() メソッドが呼び出されます。

例 : Telnet クライアントの構築

この Telnet の例では、Socket クラスを使用してリモートサーバーに接続し、データを伝送する手法を示します。この例は、次の手法を示します。

- Socket クラスを使用したカスタム Telnet クライアントの作成
- ByteArray オブジェクトを使用したリモートサーバーへのテキストの送信
- リモートサーバーから受信したデータの処理

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。Telnet アプリケーションのファイルは、Samples/Telnet フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
TelnetSocket.mxml	MXML ユーザーインターフェイスから構成されるメインアプリケーションファイル。
com/example/programmingas3/Telnet/Telnet.as	リモートサーバーへの接続、データの送受信と表示など、アプリケーションに Telnet クライアント機能を提供します。

Telnet ソケットアプリケーションの概要

メイン TelnetSocket.mxml ファイルは、アプリケーション全体で使用するユーザーインターフェイス (UI) の作成に責任を負います。

このファイルは、UI の他に、ユーザーを指定されたサーバーに接続するために、`login()` と `sendCommand()` の 2 つのメソッドも定義しています。

次のコードは、メインアプリケーションファイル内の ActionScript を示しています。

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
  telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
  console.title = "Connecting to " + serverName.text + ":" + portNumber.text;
  console.enabled = true;
}
private function sendCommand():void
{
  var ba:ByteArray = new ByteArray();
  ba.writeMultiByte(command.text + "\n", "UTF-8");
  telnetClient.writeBytesToSocket(ba);
  command.text = "";
}
```

コードの 1 行目は、Telnet クラスをカスタム com.example.programmingas.socket パッケージから読み込みます。コードの 2 行目は、Telnet クラスのインスタンス、telnetClient を宣言します。このインスタンスは、後で connect() メソッドによって初期化されます。次に、connect() メソッドが宣言され、前に宣言された telnetClient 変数を初期化します。このメソッドは、ユーザーが指定した Telnet サーバーの名前とポートを引き渡すほか、ソケットサーバーからのテキスト応答の表示に使用される表示リスト上の TextArea コンポーネントへの参照を引き渡します。connect() メソッドの最後の 2 行は、Panel の title プロパティを設定し、Panel コンポーネントを有効にします。このコンポーネントを使用すると、リモートサーバーへデータを送信できます。メインアプリケーションファイル内の最後のメソッド、sendCommand() は、ユーザーのコマンドを ByteArray オブジェクトとしてリモートサーバーへ送信するために使用されます。

Telnet クラスの概要

Telnet クラスは、リモート Telnet サーバーへの接続と、データの送受信に責任を負います。

Telnet クラスは、次のプライベート変数を宣言します。

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

最初の変数、serverURL は、ユーザーが指定した接続先サーバーアドレスを格納します。

2 番目の変数、portNumber は、Telnet サーバーが現在稼働しているポートの番号です。デフォルトでは、Telnet サービスはポート 23 上で稼働します。

3 番目の変数、socket は、serverURL 変数と portNumber 変数によって定義されたサーバーへの接続を試みる Socket インスタンスです。

4 番目の変数、ta は、ステージ上の TextArea コンポーネントインスタンスへの参照です。このコンポーネントは、リモート Telnet サーバーからの応答やエラーメッセージを表示するために使用されます。

最後の変数、state は、その Telnet クライアントがどのオプションをサポートするかを決定するために使用される数値です。

前に述べたように、Telnet クラスのコンストラクタ関数は、メインアプリケーションファイル内の connect() メソッドによって呼び出されます。

Telnet コンストラクタは 3 つのパラメータを受け取ります。server、port、および output です。server パラメータと port パラメータは、Telnet サーバーが稼働しているサーバーの名前とポート番号を指定します。最後のパラメータ、output は、サーバーの出力がユーザーに対して表示されるステージ上の TextArea コンポーネントインスタンスへの参照です。

```
public function Telnet(server:String, port:int, output:TextArea)
{
  serverURL = server;
  portNumber = port;
  ta = output;
  socket = new Socket();
  socket.addEventListener(Event.CONNECT, connectHandler);
  socket.addEventListener(Event.CLOSE, closeHandler);
  socket.addEventListener(ErrorEvent.ERROR, errorHandler);
  socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
  socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
  Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
  try
  {
    msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
    socket.connect(serverURL, portNumber);
  }
  catch (error:Error)
  {
    msg(error.message + "\n");
    socket.close();
  }
}
```

ソケットへのデータの書き込み

Socket 接続にデータを書き込むには、Socket クラス内のいずれかの書き込みメソッド (writeBoolean()、writeByte()、writeBytes()、writeDouble() など) を呼び出した後、flush() メソッドで出力バッファ内のデータをフラッシュします。Telnet サーバー内では、データは writeBytes() メソッドを使用してソケット接続へ書き込まれます。このメソッドは、パラメータとしてバイト配列を受け取り、それを出力バッファへ送信します。writeBytesToSocket() メソッドは、次のとおりです。

```
public function writeBytesToSocket(ba:ByteArray):void
{
  socket.writeBytes(ba);
  socket.flush();
}
```

このメソッドは、メインアプリケーションファイルの sendCommand() メソッドによって呼び出されます。

ソケットサーバーからのメッセージの表示

ソケットサーバーからメッセージを受け取るか、イベントが発生すると、カスタム msg() メソッドが呼び出されます。このメソッドは、ステージ上の TextArea にストリングを付加し、カスタム setScroll() メソッドを呼び出します。これにより、TextArea コンポーネントは最下部までスクロールします。msg() メソッドは、次のとおりです。

```
private function msg(value:String):void
{
  ta.text += value;
  setScroll();
}
```

TextArea コンポーネントの内容を自動的にスクロールしなかった場合、サーバーからの最新の応答を参照するには、テキスト領域にあるスクロールバーを手動でドラッグする必要があります。

TextArea コンポーネントのスクロール

setScroll() メソッドには、TextArea コンポーネントの内容を垂直方向にスクロールする 1 行だけの ActionScript が入っており、これにより、ユーザーは返されたテキストの最後の行を見ることができます。次のコードは、setScroll() メソッドを示しています。

```
public function setScroll():void
{
  ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

このメソッドは、現在表示されている文字の一番上の行の行番号を示す verticalScrollPosition プロパティを設定し、そのプロパティを maxVerticalScrollPosition プロパティの値に設定します。

例 : ファイルのアップロードとダウンロード

FileIO は、Flash Player でファイルのダウンロードとアップロードを行う手法を示した例です。それらの手法は、次のとおりです。

- ユーザーのコンピュータへのファイルのダウンロード
- ユーザーのコンピュータからサーバーへのファイルのアップロード
- 進行中のダウンロードのキャンセル
- 進行中のアップロードのキャンセル

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。FileIO アプリケーションのファイルは、Samples/FileIO フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
FileIO.fla または FileIO.mxml	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/fileio/FileDownload.as	サーバーからファイルをダウンロードするためのメソッドを含んでいるクラス。
com/example/programmingas3/fileio/FileUpload.as	ファイルをサーバーへアップロードするためのメソッドを含んでいるクラス。

FileIO アプリケーションの概要

FileIO アプリケーションには、ユーザーが Flash Player を使用してファイルをアップロードまたはダウンロードできるユーザーインターフェイスが含まれています。このアプリケーションは最初に、FileUpload と FileDownload という 1 対のカスタムコンポーネントを定義します。これらのコンポーネントは、com.example.programmingas3.fileio パッケージに入っています。それぞれのカスタムコンポーネントが `contentComplete` イベントを送出した後、コンポーネントの `init()` メソッドが呼び出され、ProgressBar および Button コンポーネントインスタンスへの参照を引き渡します。これらを使用して、ユーザーはファイルのアップロードまたはダウンロードの進行状況を知ることができ、進行中のファイル転送をキャンセルすることもできます。

FileIO.mxml ファイルの関連コードを次に示します (Flash バージョンでは、ステージに配置されたコンポーネントが FLA ファイルに含まれており、その名前はこの手順で説明する Flex コンポーネントの名前と一致します)。

```
<example:FileUpload id="fileUpload"
  creationComplete="fileUpload.init(uploadProgress, cancelUpload);" />
<example:FileDownload id="fileDownload"
  creationComplete="fileDownload.init(downloadProgress, cancelDownload);" />
```

次のコードは、プログレスバーと 2 つのボタンが入っている [ファイルのアップロード] パネルを示しています。最初のボタン、`startUpload` は `FileUpload.startUpload()` メソッドを呼び出し、このメソッドは `FileReference.browse()` メソッドを呼び出します。次のコードは、[ファイルのアップロード] パネルのコードです。

```
<mx:Panel title="Upload File" paddingTop="10" paddingBottom="10"
  paddingLeft="10" paddingRight="10">
  <mx:ProgressBar id="uploadProgress" label="" mode="manual" />
  <mx:ControlBar horizontalAlign="right">
```

```
    <mx:Button id="startUpload" label="Upload..."
  click="fileUpload.startUpload();" />
    <mx:Button id="cancelUpload" label="Cancel"
  click="fileUpload.cancelUpload();" enabled="false" />
  </mx:ControlBar>
</mx:Panel>
```

このコードは、ProgressBar コンポーネントインスタンスと、2 つの Button コンポーネントボタンインスタンスをステージ上に配置します。ユーザーが [アップロード] ボタンをクリックすると (startUpload)、オペレーティングシステムのダイアログボックスが起動し、リモートサーバーへアップロードするファイルをユーザーが選択できるようになります。もう一方のボタン、cancelUpload は、デフォルトでは無効にされていますが、ユーザーがファイルのアップロードを開始すると有効になり、ユーザーが任意の時点でファイル転送を中止できるようにします。

[ファイルのダウンロード] パネルのコードは、次のとおりです。

```
<mx:Panel title="Download File" paddingTop="10" paddingBottom="10"
  paddingLeft="10" paddingRight="10">
  <mx:ProgressBar id="downloadProgress" label="" mode="manual" />
  <mx:ControlBar horizontalAlign="right">
    <mx:Button id="startDownload" label="Download..."
  click="fileDownload.startDownload();" />
    <mx:Button id="cancelDownload" label="Cancel"
  click="fileDownload.cancelDownload();" enabled="false" />
  </mx:ControlBar>
</mx:Panel>
```

このコードはファイルアップロードコードに非常によく似ています。ユーザーが [ダウンロード] ボタンをクリックすると (startDownload)、FileDownload.startDownload() メソッドが呼び出され、FileDownload.DOWNLOAD_URL 変数で指定されたファイルのダウンロードが開始されます。ファイルのダウンロードが進行するにつれて、プログレスバーが更新され、ファイルのどれくらいのパーセンテージがダウンロードされたかが示されます。ユーザーは、cancelDownload ボタンをクリックすることにより、任意の時点でダウンロードをキャンセルできます。このボタンは、進行中のファイルダウンロードを即時に停止します。

リモートサーバーからのファイルのダウンロード

リモートサーバーからのファイルのダウンロードは、flash.net.FileReference クラスとカスタム com.example.programmingas3.fileio.FileDownload クラスによって処理されます。ユーザーが [ダウンロード] ボタンをクリックすると、Flash Player は FileDownload クラスの DOWNLOAD_URL 変数で指定されたファイルのダウンロードを開始します。

FileDownload クラスは、次のコードが示すように、冒頭で com.example.programmingas3.fileio パッケージ内の 4 つの変数を定義します。

```
/**
 * ユーザーのコンピュータへダウンロードするファイルの URL をハードコード化する
 */
private const DOWNLOAD_URL:String = "http://www.yourdomain.com/
file_to_download.zip";

/**
 * ファイルのダウンロードを操作する FileReference インスタンスを作成する
 */
private var fr:FileReference;

/**
 * ダウンロード ProgressBar コンポーネントへの参照を定義する
 */
private var pb:ProgressBar;

/**
 * 即座に停止する [ キャンセル ] ボタンへの参照を定義する
 * 現在のダウンロードが進行中
 */
private var btn:Button;
```

最初の変数、DOWNLOAD_URL には、ユーザーがメインアプリケーションファイル内の [ダウンロード] ボタンをクリックしたときに、ユーザーのコンピュータへダウンロードされるファイルへのパスが入っています。

2 番目の変数、fr は FileReference オブジェクトで、これは FileDownload.init() メソッド内で初期化され、ユーザーのコンピュータへのリモートファイルのダウンロードを処理します。

最後の 2 つの変数、pb と btn には、ステージ上の ProgressBar および Button コンポーネントインスタンスへの参照が入っており、これらのインスタンスは、FileDownload.init() メソッドによって初期化されます。

FileDownload コンポーネントの初期化

FileDownload コンポーネントは、FileDownload クラス内の init() メソッドを呼び出すことによって初期化されます。このメソッドは pb と btn の 2 つのパラメータを受け取ります。これらはそれぞれ、ProgressBar コンポーネントと Button コンポーネントのインスタンスです。

init() メソッドのコードは次のとおりです。

```
/**
 * コンポーネントへの参照を設定し、OPEN、PROGRESS、および
 * COMPLETE イベントのリスナーを追加する
 */
public function init(pb:ProgressBar, btn:Button):void
{
  // 呼び出しスクリプトから渡されるプログレスバーと [ キャンセル ] ボタン
  // への参照を設定する
  this.pb = pb;
  this.btn = btn;

  fr = new FileReference();
  fr.addEventListener(Event.OPEN, openHandler);
  fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
  fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

ファイルダウンロードの開始

ユーザーがステージ上の [ダウンロード] ボタンコンポーネントをクリックすると、startDownload() メソッドがファイルダウンロードプロセスを開始します。次のコードは、startDownload() メソッドを示しています。

```
/**
 * DOWNLOAD_URL 定数で指定されたファイルのダウンロードを開始する。
 */
public function startDownload():void
{
  var request:URLRequest = new URLRequest();
  request.url = DOWNLOAD_URL;
  fr.download(request);
}
```

最初に、startDownload() メソッドは新しい URLRequest オブジェクトを作成し、ターゲットの URL を DOWNLOAD_URL 変数で指定された値に設定します。次に、FileReference.download() メソッドが呼び出され、新規に作成された URLRequest オブジェクトがパラメータとして引き渡されます。これにより、オペレーティングシステムはユーザーのコンピュータ上にダイアログボックスを表示し、ユーザーに、要求したドキュメントを保存する場所を選択するよう求めます。ユーザーが場所を選択すると、open イベント (Event.OPEN) が送出され、openHandler() メソッドが起動されます。

openHandler() メソッドは、ProgressBar コンポーネントの label プロパティのテキストフォーマットを設定し、[キャンセル] ボタンを有効にします。このボタンにより、ユーザーは進行中のダウンロードを即時に停止できます。openHandler() メソッドは、次のとおりです。

```
/**
 * OPEN イベントが送出されたらプログレスバーのラベルを変更し、
 * [ キャンセル ] ボタンを有効にして、ユーザーがダウンロード操作を
 * 中止できるようにする
 */
private function openHandler(event:Event):void
{
  pb.label = "DOWNLOADING %3%%";
  btn.enabled = true;
}
```

ファイルダウンロードの進行状況の監視

ファイルがリモートサーバーからユーザーのコンピュータへダウンロードされるとき、一定の間隔で progress イベント (ProgressEvent.PROGRESS) が送出されます。progress イベントが送出されたときは、常に progressHandler() メソッドが起動され、ステージ上の ProgressBar コンポーネントインスタンスが更新されます。progressHandler() メソッドのコードは、次のとおりです。

```
/**
 * ファイルのダウンロード中に、プログレスバーのステータスを更新する。
 */
private function progressHandler(event:ProgressEvent):void
{
  pb.setProgress(event.bytesLoaded, event.bytesTotal);
}
```

progress イベントには bytesLoaded と bytesTotal の 2 つのプロパティが含まれており、これらは、ステージ上の ProgressBar コンポーネントを更新するために使用されます。これにより、ユーザーはファイルのどれくらいの分量が既にダウンロードを完了し、どれくらいの分量がまだ残っているかを知ることができます。ユーザーは、プログレスバーの下にある [キャンセル] ボタンをクリックすることにより、任意の時点でファイル転送を中止できます。

ファイルのダウンロードが正常に完了した場合は、complete イベント (Event.COMPLETE) が completeHandler() メソッドを起動します。このメソッドはユーザーにファイルのダウンロードが完了したことを通知し、[キャンセル] ボタンを無効にします。completeHandler() メソッドのコードは次のとおりです。

```
/**
 * ダウンロードの完了後、プログレスバーのラベルをもう一度だけ変更し、
 * ダウンロードが既に完了しているため、[ キャンセル ] ボタンは
 * 無効にする。
 */
private function completeHandler(event:Event):void
{
```

```
    pb.label = "DOWNLOAD COMPLETE";
    btn.enabled = false;
  }
```

ファイルダウンロードの中止

ユーザーは、ステージ上の [キャンセル] ボタンをクリックすることにより、任意の時点でファイル転送を中止し、それ以上のバイトのダウンロードを停止することができます。次のコードは、ダウンロードをキャンセルするためのコードを示しています。

```
  /**
   * 現在のファイルダウンロードを中止する。
   */
  public function cancelDownload():void
  {
    fr.cancel();
    pb.label = "DOWNLOAD CANCELLED";
    btn.enabled = false;
  }
```

最初に、このコードはファイル転送を即時に停止し、それ以上データがダウンロードされないようにします。次に、プログレスバーの label プロパティが更新され、ダウンロードが正常にキャンセルされたことをユーザーに通知します。最後に、[キャンセル] ボタンが無効にされます。これにより、ユーザーは再びファイルのダウンロードを試みるまで、このボタンをクリックできなくなります。

リモートサーバーへのファイルのアップロード

ファイルのアップロードプロセスは、ファイルのダウンロードプロセスに非常によく似ています。FileUpload クラスは、次のコードに示すように、同じ 4 つの変数を宣言します。

```
private const UPLOAD_URL:String = "http://www.yourdomain.com/
  your_upload_script.cfm";
private var fr:FileReference;
private var pb:ProgressBar;
private var btn:Button;
```

FileDownload.DOWNLOAD_URL 変数と異なり、UPLOAD_URL 変数にはユーザーのコンピュータからファイルをアップロードするサーバーサイドスクリプトへの URL が入っています。残りの 3 つの変数は、それらに対応する FileDownload クラス内の変数と同じ働きをします。

FileUpload コンポーネントの初期化

FileUpload コンポーネントには init() メソッドが入っており、このメソッドはメインアプリケーションから呼び出されます。このメソッドは pb と btn の 2 つのパラメータを受け取ります。これらのパラメータは、ステージ上の ProgressBar および Button コンポーネントインスタンスへの参照です。次に、init() メソッドは前に FileUpload クラス内で定義された FileReference オブジェクトを初期化します。最後に、このメソッドは 4 つのイベントリスナーを FileReference オブジェクトに割り当てます。init() メソッドのコードは次のとおりです。

```
public function init(pb:ProgressBar, btn:Button):void
{
  this.pb = pb;
  this.btn = btn;

  fr = new FileReference();
  fr.addEventListener(Event.SELECT, selectHandler);
  fr.addEventListener(Event.OPEN, openHandler);
  fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
  fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

ファイルのアップロードの開始

ファイルのアップロードは、ユーザーがステージ上の [アップロード] ボタンをクリックしたときに開始されます。このボタンは、FileUpload.startUpload() メソッドを起動します。このメソッドは FileReference クラスの browse() メソッドを呼び出し、後者のメソッドにより、オペレーティングシステムは、ユーザーにリモートサーバーへアップロードするファイルを選択するよう求めるシステムダイアログボックスを表示します。次のコードは、startUpload() メソッドのコードです。

```
public function startUpload():void
{
  fr.browse();
}
```

ユーザーがアップロードするファイルを選択すると、select イベント (Event.SELECT) が送出され、selectHandler() メソッドが起動されます。selectHandler() メソッドは新しい URLRequest オブジェクトを作成し、URLRequest.url プロパティを前にコード内で定義された UPLOAD_URL 定数の値に設定します。最後に、FileReference オブジェクトは選択されたファイルを指定されたサーバーサイドスクリプトへアップロードします。selectHandler() メソッドのコードは次のとおりです。

```
private function selectHandler(event:Event):void
{
  var request:URLRequest = new URLRequest();
  request.url = UPLOAD_URL;
  fr.upload(request);
}
```

FileUpload クラス内の残りのコードは、FileDownload クラス内で定義されたコードと同じです。ユーザーは、任意の時点でアップロードを終了する場合、[キャンセル] ボタンをクリックします。これにより、プログレスバー上のラベルが設定され、ファイル転送が即時に停止します。プログレスバーは、`progress` イベント (`ProgressEvent.PROGRESS`) が送出されると必ず更新されます。同様に、アップロードが完了すると、プログレスバーが更新され、ファイルが正常にアップロードされたことをユーザーに通知します。その後、[キャンセル] ボタンは、ユーザーが新しいファイル転送を開始するまで無効にされます。

第 23 章

クライアントのシステム環境

23

この章では、ユーザーのシステムを操作する方法について説明します。ここでは、サポートされている機能を判別する方法と、ユーザーが入力メソッドエディタ (IME) をインストールしてある場合に、IME を使用して複数言語の SWF ファイルを構築する方法を示します。また、アプリケーションドメインの一般的な使用方法も示します。

目次

クライアントのシステム環境の基礎 662
System クラスの使用 664
Capabilities クラスの使用 665
ApplicationDomain クラスの使用 666
IME クラスの使用 670
例 : システム機能の検出 675

クライアントのシステム環境の基礎

クライアントのシステム環境の概要

高度な ActionScript アプリケーションを構築していると、ユーザーのオペレーティングシステムの詳細およびアクセス機能を知る必要がある場合があります。クライアントのシステム環境は、flash.system パッケージ内のクラスのコレクションです。これらのクラスを使用すると、次のようなシステムレベルの機能にアクセスできます。

- SWF を実行しているアプリケーションとセキュリティドメインの判別
- 画面サイズ (解像度) などのユーザーの Flash Player の機能、および MP3 オーディオなどの特定の機能が使用可能かどうかの判別
- IME を使用した複数言語サイトの構築
- Flash Player のコンテナ (HTML ページ、コンテナアプリケーションなど) の操作
- ユーザーのクリップボードへの情報の保存

また、flash.system パッケージには IMEConversionMode および SecurityPanel クラスも含まれています。これらでは、それぞれ IME クラスと Security クラスで使用する静的定数が定義されています。

クライアントのシステム環境の一般的なタスク

この章では、ActionScript を使用したクライアントシステムの操作に関する、次のような一般的なタスクについて説明します。

- アプリケーションが使用しているメモリの量の確認
- ユーザーのクリップボードへのテキストのコピー
- 次のようなユーザーのコンピュータ性能の確認
  - 画面の解像度、色、DPI、およびピクセル縦横比
  - オペレーティングシステム
  - ストリーミングオーディオ、ストリーミングビデオ、および MP3 再生のサポート
  - インストールされた Flash Player がデバッガ版かどうか
- アプリケーションドメインの操作
  - アプリケーションドメインの定義
  - SWF ファイルのコードをアプリケーションドメインに分離
- アプリケーションでの IME の操作
  - IME がインストールされているかどうかの確認
  - IME 変換モードの確認および設定
  - テキストフィールドに対する IME の無効化
  - IME 変換実行の検知

重要な概念と用語

次の参照リストに、この章で使用されている重要な用語を示します。

- オペレーティングシステム : Microsoft Windows、Mac OS X、Linux など、コンピュータで実行されるメインプログラム。他のすべてのアプリケーションはこのプログラムの内部で実行されます。
- クリップボード : コピーまたはカットしたテキストやアイテムを保持するためのオペレーティングシステムのコンテナ。コンテナ内のアイテムはアプリケーションにペーストできます。
- アプリケーションドメイン : 異なる SWF ファイルに使用されているクラスを分離するメカニズム。これにより、複数の SWF ファイルで同じ名前の別のクラスが使用されていても、そのクラスが他のクラスを上書きすることはありません。
- IME (入力メソッドエディタ) : 標準キーボードを使用して複雑な文字や記号を入力するために使用するプログラム (またはオペレーティングシステムツール)。
- クライアントシステム : プログラミング用語では、クライアントは個々のコンピュータで実行されるアプリケーションの一部 (またはアプリケーション全体) で、単一のユーザーによって使用されます。クライアントシステムはユーザーのコンピュータ上の基になるオペレーティングシステムです。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章のすべてのコードリストには、テスト対象の値を書き出すための適切な `trace()` 関数呼び出しが含まれています。この章のコードリストをテストするには :

1. 空の Flash ドキュメントを作成します。
2. タイムラインのキーフレームを選択します。
3. [アクション] パネルを開いて、コードリストを [スクリプト] ペインにコピーします。
4. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   コードリストの `trace()` 関数の結果が [出力] パネルに表示されます。

後半のコードリストの一部はより複雑で、クラスとして作成されています。これらの例をテストするには :

1. 空の Flash ドキュメントを作成して、コンピュータに保存します。
2. 新しい ActionScript ファイルを作成して、Flash ドキュメントと同じディレクトリに保存します。ファイル名は、コードリスト内のクラスの名前と同じ名前にしてください。たとえば、コードリストで SystemTest という名前のクラスが定義されている場合は、SystemTest.as という名前で ActionScript ファイルを保存します。
3. コードリストを ActionScript ファイルにコピーして、ファイルを保存します。

4. Flash ドキュメントで、ステージの何もない部分または作業スペースをクリックして、ドキュメントのプロパティインスペクタを起動します。
5. プロパティインスペクタの [ドキュメントクラス] フィールドに、テキストからコピーした ActionScript クラスの名前を入力します。
6. [制御]-[ムービープレビュー] を使用して、プログラムを実行します。

   例の結果が [出力] パネルに表示されます。

サンプルコードリストのテストの詳細については、60 ページの「章内のコード例のテスト」を参照してください。

System クラスの使用

System クラスには、ユーザーのオペレーティングシステムを操作したり、Adobe Flash Player の現在のメモリ使用状況を取得できるメソッドとプロパティが格納されています。System クラスのメソッドとプロパティを使用すると、`imeComposition` イベントの待ち受け、ユーザーの現行コードページを使用した外部テキストファイルのロードまたは Unicode としてのロードを行うための Flash Player に対する指示、ユーザーのクリップボードの内容の設定も行うことができます。

実行時におけるユーザーのシステムに関するデータの取得

`System.totalMemory` プロパティを調べると、Flash Player が現在使用しているメモリの量をバイト単位で判別できます。このプロパティを使用してメモリ使用量を監視すれば、空きメモリの状況の変化に応じてアプリケーションの動作を最適に調整できます。たとえば、特定のビジュアル効果を使用するとメモリ使用量が大幅に増加する場合、状況によってその効果を変化または完全に無効にすることが考えられます。

`System.ime` プロパティは、現在インストールされている IME (入力メソッドエディタ) への参照です。このプロパティを使用すると、`addEventListener()` メソッドで `imeComposition` イベント (`flash.events.IMEEvent.IME_COMPOSITION`) を待ち受けることができます。

System クラスの 3 番目のプロパティは `useCodePage` です。`useCodePage` を `true` に設定すると、Flash Player が実行されているオペレーティングシステムの通常のコードページを使用して外部テキストファイルがロードされます。このプロパティを `false` に設定すると、Flash Player は外部ファイルを Unicode として解釈します。

System.useCodePage を true に設定すると、Flash Player が実行されているオペレーティングシステムの通常のコードページに属する文字を使用して外部テキストファイルの内容が記述されていないと、テキストは表示されないことに注意してください。たとえば、中国語を含んだ外部テキストファイルをロードする場合、English Windows コードページを使用しているシステム上では文字が表示されません。これは、English Windows コードページには中国語の文字が含まれないためです。

SWF ファイル内で使用される外部テキストファイルをすべてのプラットフォームのユーザーが表示できるようにするには、すべての外部テキストファイルを Unicode で保存し、System.useCodePage をデフォルトの false の設定のままにします。これにより、Flash Player ではテキストが Unicode として解釈されます。

クリップボードへのテキストの保存

System クラスには setClipboard() というメソッドがあります。このメソッドを使用すると、Flash Player から指定したストリングをユーザーのクリップボードの内容として設定できます。セキュリティ上の理由から、Security.getClipboard() メソッドはありません。そのようなメソッドは、悪意あるサイトがユーザーのクリップボードへ最後にコピーされたデータにアクセスすることを許すおそれがあるからです。

次のコードは、セキュリティエラーが発生したときに、ユーザーのクリップボードへエラーメッセージをコピーする方法の例です。このエラーメッセージは、ユーザーがアプリケーションの潜在的なバグを報告する場合に役に立ちます。

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
  var errorString:String = "[" + event.type + "] " + event.text;
  trace(errorString);
  System.setClipboard(errorString);
}
```

Capabilities クラスの使用

Capabilities クラスを使用すると、開発者は SWF ファイルの実行環境に関する情報を取得できます。Capabilities クラスのさまざまなプロパティにより、ユーザーのシステムにおける画面解像度、アクセシビリティソフトウェアに対するサポートの有無、ユーザーのオペレーティングシステムの言語、および、現在インストールされている Flash Player のバージョンを知ることができます。

Capabilities クラスのプロパティを確認すれば、実際のユーザー環境に応じてアプリケーションの動作を最適に調整できます。たとえば、Capabilities.screenResolutionX および Capabilities.screenResolutionY プロパティを調べることにより、ユーザーのシステムで使用されているディスプレイ解像度を判別し、どのサイズのビデオが最適かを判断できます。また、Capabilities.hasMP3 プロパティを調べれば、外部 MP3 ファイルをロードする前にユーザーのシステムにおける MP3 再生のサポート状況を知ることができます。

次のコードでは、クライアント環境にインストールされている Flash Player のバージョン情報を正規表現で解析しています。

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
  trace("input: " + result.input);
  trace("platform: " + result[1]);
  trace("majorVersion: " + result[2]);
  trace("minorVersion: " + result[3]);
  trace("buildNumber: " + result[4]);
  trace("internalBuildNumber: " + result[5]);
}
else
{
  trace("Unable to match RegExp.");
}
```

ユーザーのシステムの機能に関する情報をサーバーサイドスクリプトに送信してデータベースに保存するには、次のような ActionScript コードを使用します。

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

ApplicationDomain クラスの使用

ApplicationDomain クラスの目的は、ActionScript 3.0 定義のテーブルを保存することです。SWF ファイル内のすべてのコードは、アプリケーションドメイン内に存在するように定義されます。アプリケーションドメインは、同じセキュリティドメイン内にある複数のクラスを分離するために使用します。それにより、1 つのクラスについて複数の定義を用意することや、子が親の定義を再利用することが可能になります。

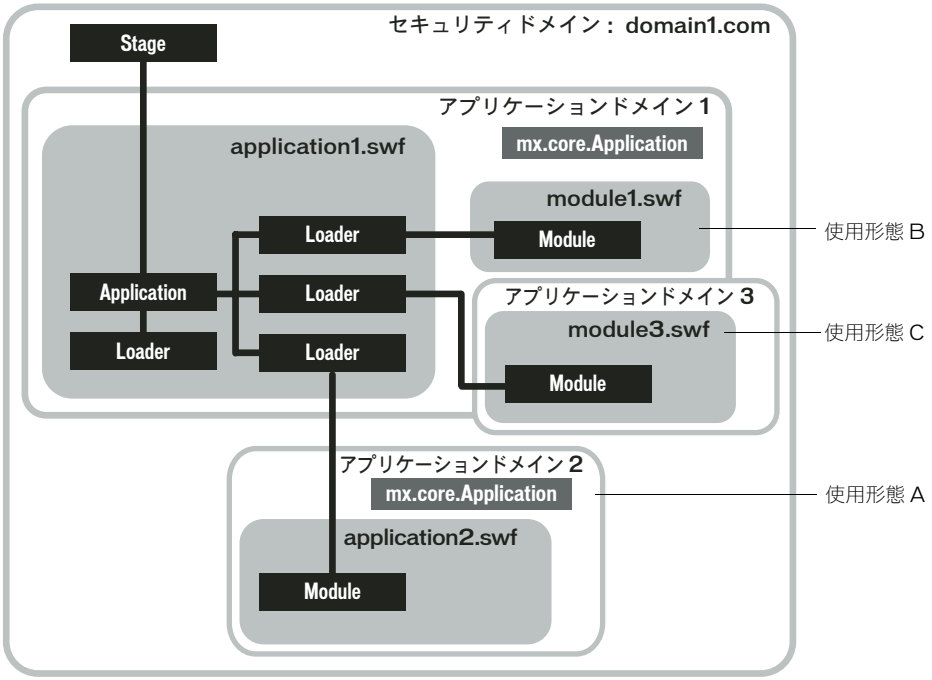
ActionScript 3.0 で作成された外部 SWF ファイルを Loader クラス API でロードするときは、アプリケーションドメインを使用できます。ActionScript 1.0 または ActionScript 2.0 で作成された SWF ファイルのイメージをロードするときは、アプリケーションドメインを使用できないことに注意してください。ロードされたクラスに含まれているすべての ActionScript 3.0 定義は、アプリケーションドメイン内に格納されます。SWF ファイルをロードするとき、LoaderContext オブジェクトの applicationDomain パラメータを ApplicationDomain.currentDomain に設定することにより、ファイルを Loader オブジェクトと同じアプリケーションドメインに含めるよう指定できます。ロードされた SWF ファイルを同じアプリケーションドメインに置くと、そのクラスに直接アクセスできます。これは、関連するクラス名によってアクセスできるメディアが埋め込まれた SWF ファイルをロードする場合や、ロードされた SWF ファイルのメソッドにアクセスする場合に便利です。次に例を示します。

```
package
{
  import flash.display.Loader;
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLRequest;
  import flash.system.ApplicationDomain;
  import flash.system.LoaderContext;

  public class ApplicationDomainExample extends Sprite
  {
    private var ldr:Loader;
    public function ApplicationDomainExample()
    {
      ldr = new Loader();
      var req:URLRequest = new URLRequest("Greeter.swf");
      var ldrContext:LoaderContext = new LoaderContext(false,
  ApplicationDomain.currentDomain);
      ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
      ldr.load(req, ldrContext);
    }
    private function completeHandler(event:Event):void
    {
      ApplicationDomain.currentDomain.getDefinition("Greeter");
      var myGreeter:Greeter = Greeter(event.target.content);
      var message:String = myGreeter.welcome("Tommy");
      trace(message); // Hello, Tommy
    }
  }
}
```

それ以外に、アプリケーションドメインを操作するときに注意すべき点は、次のとおりです。

- SWF ファイル内のすべてのコードは、アプリケーションドメイン内に存在するように定義されます。" 現在のドメイン " とは、メインアプリケーションが実行される場所のことです。現在のドメインおよび他のすべてのアプリケーションドメインは " システムドメイン " 内に属します。つまり、システムドメインにはすべての Flash Player クラスが含まれます。
- システムドメインを除くすべてのアプリケーションドメインには、親ドメインが関連付けられています。メインアプリケーションのアプリケーションドメインの親ドメインは、システムドメインです。ロードされるクラスは、そのクラスの親によってまだ定義されていない場合にのみ定義されます。ロードされるクラスの定義を新しい定義でオーバーライドすることはできません。

次の図に示すアプリケーションでは、さまざまな SWF ファイルから domain1.com という単一のドメイン内にコンテンツをロードしています。ロードするコンテンツによっては、異なるアプリケーションドメインを使用することもできます。このアプリケーションに含まれる個々の SWF ファイルでは、それぞれに適した異なる形態でアプリケーションドメインを使用しています。各形態の使い分け方について次に説明します。



メインアプリケーションファイルは application1.swf です。これには、他の SWF ファイルからコンテンツをロードする Loader オブジェクトが入っています。このシナリオでは、現在のドメインは Application domain 1 です。使用形態 A、使用形態 B、および使用形態 C は、アプリケーション内の各 SWF ファイルに適切なアプリケーションドメインを設定するさまざまな手法の例を示しています。

使用形態 A : システムドメインの子ドメインを作成することにより、子 SWF ファイルを分離します。図では、システムドメインの子ドメインとして Application domain 2 を作成しています。"application2.swf" ファイルを Application domain 2 内にロードしているため、このファイルに含まれるクラス定義は "application1.swf" のクラス定義と分離されます。

この形態の用途としては、古いアプリケーションにおいて同じアプリケーションの新バージョンを動的にロードする場合に競合の発生を防ぐことが考えられます。異なるバージョンを異なるアプリケーションドメインに分離すれば、それぞれにおいて同じクラス名を使用していても競合が発生しません。

次のコードでは、システムドメインの子ドメインとしてアプリケーションドメインを作成します。

```
request.url = "application2.swf";
request.applicationDomain = new ApplicationDomain();
```

使用形態 B : 現在のクラス定義に新しいクラス定義を追加します。図では、"module1.swf" に対するアプリケーションドメインとして現在のドメイン (Application domain 1) を設定しています。この場合、アプリケーションに現在含まれている各種のクラス定義に、新しいクラス定義を加えることになります。この形態は、メインアプリケーションのランタイム共有ライブラリに使用できます。ロードされた SWF は、リモート共有ライブラリ (RSL) として扱われます。アプリケーションが開始する前にプリローダーによって RSL をロードするには、このテクニックを使用します。

次のコードでは、現在のドメインをアプリケーションドメインとして設定します。

```
request.url = "module1.swf";
request.applicationDomain = ApplicationDomain.currentDomain;
```

使用形態 C : 現在のドメインの子ドメインを新しく作成することにより、親のクラス定義を使用します。図では、"module3.swf" に対するアプリケーションドメインとして現在のドメインの子ドメインを設定しており、子ドメインでは、親ドメインに含まれるすべてのクラス定義を使用します。この形態の用途としては、複数画面の RIA (高度なインターネットアプリケーション) を構成するモジュールをメインアプリケーションの子としてロードし、メインアプリケーションで定義されている型をモジュール内でも使用できるようにすることが考えられます。クラスを更新される際に必ず後方互換性を維持するようにし、また、ロードされるモジュールよりもメインアプリケーションを常に新しくしておけば、メインアプリケーションに含まれるクラス定義はモジュール内でも使用されます。また、このようにして新しいアプリケーションドメイン内にロードしたモジュールは、アンロードすれば、すべてのクラス定義をガベージコレクションの対象とすることができます (ただし、子 SWF への参照を確実に破棄する必要があります)。

このテクニックでは、ロードする側にあるシングルトンオブジェクトや静的クラスメンバーを、ロードされる側のモジュールでも共有できます。

次のコードでは、現在のドメインの子ドメインを新しく作成します。

```
request.url = "module3.swf";
request.applicationDomain = new
  ApplicationDomain(ApplicationDomain.currentDomain);
```

IME クラスの使用

IME クラスを使用すると、オペレーティングシステムの IME を Flash Player 内で操作できます。

ActionScript から、次の事項を調べることができます。

- ユーザーのコンピュータに IME がインストールされているかどうか (`Capabilities.hasIME`)
- ユーザーのコンピュータで IME が有効になっているかどうか (`IME.enabled`)
- 現在の IME で使用されている変換モード (`IME.conversionMode`)

テキスト入力フィールドを特定の IME コンテキストに関連付けることができます。それにより、入力フィールド間をフォーカスが移動するのに従って IME の変換モード (ひらがな、全角数字、半角数字、直接入力など) の切り替えができます。

IME を使用すると、中国語、日本語、韓国語といったマルチバイトの言語で ASCII 以外の文字を入力できるようになります。

IME の使用方法の詳細については、アプリケーションの開発対象であるオペレーティングシステムのマニュアルを参照してください。追加情報については、次の Web サイトを参照してください。

- http://www.microsoft.com/globaldev/default.mspx
- http://developer.apple.com/documentation/
- http://java.sun.com/

注	IME がユーザーのコンピュータ上でアクティブでない場合、IME のメソッドまたはプロパティの呼び出しは、`Capabilities.hasIME` を除き、失敗します。IME を手動でアクティブにすると、それ以降の ActionScript による IME のメソッドまたはプロパティの呼び出しは、意図したとおりに動作します。たとえば、日本語 IME を使用する場合は、IME のメソッドまたはプロパティを呼び出す前に IME をアクティブにする必要があります。

IME がインストールされ有効になっているかどうかの確認

IME のメソッドまたはプロパティを呼び出す前に、必ずユーザーのコンピュータを調べ、IME が現在インストールされ有効にされているかどうかを確認する必要があります。次のコードは、いずれかのメソッドを呼び出す前に、ユーザーが IME をインストール済みで、それがアクティブであるかどうかを確認する方法を示しています。

```
if (Capabilities.hasIME)
{
  if (IME.enabled)
  {
    trace("IME is installed and enabled.");
  }
  else
  {
    trace("IME is installed but not enabled.Please enable your IME and try
  again.");
  }
}
else
{
  trace("IME is not installed.Please install an IME and try again.");
}
```

上記のコードは、最初に、Capabilities.hasIME プロパティを使用して、ユーザーが IME をインストールしているかどうかを確認します。このプロパティが true に設定されている場合は、次に IME.enabled プロパティを使用し、ユーザーの IME が現在有効にされているかどうかを確認します。

現在有効になっている IME 変換モードの判別

複数言語アプリケーションを構築するときは、ユーザーが現在どの変換モードをアクティブにしているかを判別することが必要な場合もあります。次のコードは、ユーザーが IME をインストールしているかどうか、また、インストールしている場合は、どの IME 変換モードが現在アクティブであるかを確認する方法を示しています。

```
if (Capabilities.hasIME)
{
  switch (IME.conversionMode)
  {
    case IMEConversionMode.ALPHANUMERIC_FULL:
      tf.text = "Current conversion mode is alphanumeric (full-width).";
      break;
    case IMEConversionMode.ALPHANUMERIC_HALF:
      tf.text = "Current conversion mode is alphanumeric (half-width).";
      break;
    case IMEConversionMode.CHINESE:
```

```
        tf.text = "Current conversion mode is Chinese.";
        break;
      case IMEConversionMode.JAPANESE_HIRAGANA:
        tf.text = "Current conversion mode is Japananese Hiragana.";
        break;
      case IMEConversionMode.JAPANESE_KATAKANA_FULL:
        tf.text = "Current conversion mode is Japanese Katakana (full-
  width).";
        break;
      case IMEConversionMode.JAPANESE_KATAKANA_HALF:
        tf.text = "Current conversion mode is Japanese Katakana (half-
  width).";
        break;
      case IMEConversionMode.KOREAN:
        tf.text = "Current conversion mode is Korean.";
        break;
      default:
        tf.text = "Current conversion mode is " + IME.conversionMode + ".";
        break;
  }
}
else
{
  tf.text = "Please install an IME and try again.";
}
```

上記のコードでは、最初に、ユーザーが IME をインストールしているかどうかを確認します。次に、現在の IME がどの変換モードを使用しているかを確認するために、IME.conversionMode プロパティを IMEConversionMode クラス内のそれぞれの定数と照合します。

IME 変換モードの設定

ユーザーの IME の変換モードを変更するときは、コードが try..catch ブロック内にラップされていることを確認する必要があります。なぜなら、conversionMode プロパティを使用して変換モードを設定すると、IME が変換モードを設定できない場合にエラーがスローされる可能性があるからです。次のコードは、IME.conversionMode プロパティを設定するときの try..catch ブロックの使用方法を示しています。

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
  try
  {
    IME.enabled = true;
    IME.conversionMode = IMEConversionMode.KOREAN;
    statusText.text = "Conversion mode is " + IME.conversionMode + ".";
```

```
  }
  catch (error:Error)
  {
    statusText.text = "Unable to set conversion mode.\n" + error.message;
  }
}
```

上記のコードは、最初に、ユーザーに対してステータスメッセージを表示するためのテキストフィールドを作成します。次に、このコードは、IME がインストールされている場合、IME を有効にして変換モードを韓国語に設定します。ユーザーのコンピュータに韓国語 IME がインストールされていない場合は Flash Player によってエラーがスローされ、`try..catch` ブロックによってキャッチされます。`try..catch` ブロックは、前に作成されたテキストフィールドにエラーメッセージを表示します。

特定のテキストフィールドに対する IME の無効化

場合によっては、ユーザーが文字を入力する間、ユーザーの IME を無効にした方が望ましいときもあります。たとえば、数値入力だけを受け入れるテキストフィールドがある場合、IME が起動されてデータ入力操作が遅くなることは望ましくありません。

次の例は、`FocusEvent.FOCUS_IN` イベントと `FocusEvent.FOCUS_OUT` イベントを待ち受け、それらのイベントに応じてユーザーの IME を無効にする方法を示しています。

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
  if (Capabilities.hasIME)
  {
    IME.enabled = false;
  }
}
```

```
function focusOutHandler(event:FocusEvent):void
{
  if (Capabilities.hasIME)
  {
    IME.enabled = true;
  }
}
```

この例は、phoneTxt と nameTxt という 2 つの入力テキストフィールドを作成した後、2 つのイベントリスナーを phoneTxt テキストフィールドに追加します。ユーザーが phoneTxt テキストフィールドにフォーカスを設定すると、FocusEvent.FOCUS_IN イベントが送出され、IME は無効になります。phoneTxt テキストフィールドがフォーカスを失うと、FocusEvent.FOCUS_OUT イベントが送出され、IME が再び有効になります。

IME 入力イベントの待ち受け

IME 入力イベントは、入力ストリングが設定されようとしているときに送出されます。たとえば、ユーザーが IME を有効にしてあり、それをアクティブにして日本語でストリングを入力する場合、ユーザーが入力ストリングを選択すると同時に IMEEvent.IME_COMPOSITION イベントが送出されます。IMEEvent.IME_COMPOSITION イベントを待ち受けるためには、次の例に示すように、System クラス (flash.system.System.ime.addEventListener(...)) の静的プロパティ ime にイベントリスナーを追加する必要があります。

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
  IME.enabled = true;
  try
  {
    IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
  }
  catch (error:Error)
```

```
    {
      outputTxt.text = "Unable to change IME.";
    }
    System.ime.addEventListener(IMEEvent.IME_COMPOSITION, imeCompositionHandler);
}
else
{
    outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
    outputTxt.text = "you typed: " + event.text;
}
```

上記のコードは 2 つのテキストフィールドを作成し、表示リストに追加します。最初のテキストフィールド、inputTxt は、ユーザーが日本語テキストを入力できる入力テキストフィールドです。2 番目のテキストフィールド、outputTxt は、ユーザーに対するエラーメッセージを表示するか、ユーザーが inputTxt テキストフィールドに入力した日本語ストリングをエコーするための動的テキストフィールドです。

例 : システム機能の検出

この CapabilitiesExplorer の例は、flash.system.Capabilities クラスを使用してユーザーの Flash Player がどのような機能をサポートしているかを判別する方法を示しています。この例から、次の手法を学習できます。

- Capabilities クラスを使用してユーザーの Flash Player がサポートする機能を検出する方法
- ExternalInterface クラスを使用してユーザーのブラウザがサポートしているブラウザ設定を検出する方法

このサンプルのアプリケーションファイルを入手するには、
www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。
CapabilitiesExplorer アプリケーションのファイルは、Samples/CapabilitiesExplorer フォルダにあります。このアプリケーションは、次のファイルで構成されています。

ファイル	説明
CapabilitiesExplorer.fla または CapabilitiesExplorer.mxml	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル。
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	配列へのシステム Capabilities の追加、項目のソート、ExternalInterface クラスを使用したブラウザ機能の取得など、アプリケーションの主要な機能を提供するクラス。
capabilities.html	External API との通信に必要な JavaScript を格納する HTML コンテナ。

CapabilitiesExplorer の概要

CapabilitiesExplorer.mxml ファイルは、CapabilitiesExplorer アプリケーションのユーザーインターフェイスを設定することに責任を負います。ユーザーの Flash Player 機能は、ステージ上の DataGrid コンポーネント内に表示されます。また、ユーザーが HTML コンテナからアプリケーションを実行している場合、しかも External API が使用可能な場合は、ユーザーのブラウザ機能も表示されます。

メインアプリケーションファイルの `creationComplete` イベントが送出されると、`initApp()` メソッドが起動されます。`initApp()` メソッドは com.example.programmingas3.capabilities.CapabilitiesGrabber クラスの中から `getCapabilities()` メソッドを呼び出します。`initApp()` メソッドのコードは次のとおりです。

```
private function initApp():void
{
  var dp:Array = CapabilitiesGrabber.getCapabilities();
  capabilitiesGrid.dataProvider = dp;
}
```

`CapabilitiesGrabber.getCapabilities()` メソッドは、Flash Player 機能およびブラウザ機能のソートされた配列を返します。その後、その配列はステージ上の `capabilitiesGrid` DataGrid コンポーネントインスタンスの `dataProvider` プロパティに設定されます。

CapabilitiesGrabber クラスの概要

CapabilitiesGrabber クラスの静的 getCapabilities() メソッドは、flash.system.Capabilities クラスからの各プロパティを配列 (capDP) に追加します。その後、CapabilitiesGrabber クラス内の静的 getBrowserObjects() メソッドを呼び出します。getBrowserObjects() メソッドは、External API を使用して、ブラウザのナビゲータオブジェクトをループ処理します。このオブジェクトには、ブラウザの機能が入っています。getCapabilities() メソッドは、次のとおりです。

```
public static function getCapabilities():Array
{
  var capDP:Array = new Array();
  capDP.push({name:"Capabilities.avHardwareDisable",
  value:Capabilities.avHardwareDisable});
  capDP.push({name:"Capabilities.hasAccessibility",
  value:Capabilities.hasAccessibility});
  capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
  ...
  capDP.push({name:"Capabilities.version", value:Capabilities.version});
  var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
  if (navArr.length > 0)
  {
    capDP = capDP.concat(navArr);
  }
  capDP.sortOn("name", Array.CASEINSENSITIVE);
  return capDP;
}
```

getBrowserObjects() メソッドは、ブラウザのナビゲータオブジェクトに含まれる各プロパティの配列を返します。その配列の長さが 1 項目以上である場合、ブラウザ機能の配列 (navArr) が Flash Player 機能の配列 (capDP) に付加され、配列全体がアルファベット順でソートされます。最後に、ソートされた配列がメインアプリケーションファイルへ返され、メインアプリケーションファイルがデータグリッドにデータを入れます。getBrowserObjects() メソッドのコードは次のとおりです。

```
private static function getBrowserObjects():Array
{
  var itemArr:Array = new Array();
  var itemVars:URLVariables;
  if (ExternalInterface.available)
  {
    try
    {
      var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
      itemVars = new URLVariables(tempStr);
      for (var i:String in itemVars)
      {
        itemArr.push({name:i, value:itemVars[i]});
      }
    }
    catch (error:SecurityError)
```

```
    {
      // ignore
    }
  }
  return itemArr;
}
```

現在のユーザー環境で External API が使用可能な場合、Flash Player は JavaScript の `JS_getBrowserObjects()` メソッドを呼び出します。このメソッドはブラウザのナビゲータオブジェクトをループ処理し、URL エンコードされた値のストリングを ActionScript に返します。このストリングは、その後、URLVariables オブジェクト (`itemVars`) に変換され、`itemArr` 配列に追加されます。その配列が、呼び出し元のスクリプトへ返されます。

JavaScript との通信

CapabilitiesExplorer アプリケーションを構築する最後の部分は、ブラウザのナビゲータオブジェクト内の各項目をループ処理して名前と値のペアを一時配列に付加するために必要な JavaScript を作成することです。container.html ファイルに入っている JavaScript `JS_getBrowserObjects()` メソッドのコードは、次のとおりです。

```
<script language="JavaScript">
  function JS_getBrowserObjects()
  {
    // ブラウザの各項目を保持する配列を作成する。
    var tempArr = new Array();

    // ブラウザのナビゲータオブジェクトに含まれている各項目をループ処理する。
    for (var name in navigator)
    {
      var value = navigator[name];

      // 現在の値がストリングまたは Boolean オブジェクトなら
      // 配列に追加し、そうでなければ項目を無視する。
      switch (typeof(value))
      {
        case "string":
        case "boolean":

          // 配列に追加される一時ストリングを作成する。
          // 必ず JavaScript の
          //  escape() 関数を使用して値を URL エンコードする。
          var tempStr = "navigator." + name + "=" + escape(value);
          // URL エンコードした名前と値のペアを配列にプッシュする。
          tempArr.push(tempStr);
          break;
      }
    }
    // ブラウザの画面オブジェクトに入っている各項目をループ処理する。
```

```
    for (var name in screen)
    {
      var value = screen[name];

      // 現在の値が数値ならそれを配列に追加し、数値でなければ
      // 項目を無視する。
      switch (typeof(value))
      {
        case "number":
          var tempStr = "screen." + name + "=" + escape(value);
          tempArr.push(tempStr);
          break;
      }
    }
    // 配列を、URL エンコードされた名前と値のペアからなるストリングとして返す。
    return tempArr.join("&");
  }
</script>
```

このコードは最初に、ナビゲータオブジェクト内にあるすべての名前と値のペアを保持する一時配列を作成します。次に、ナビゲータオブジェクトが `for..in` ループでループ処理され、現在の値のデータ型が評価されます。これにより、不要な値が除外されます。このアプリケーションでは、String 値または Boolean 値だけが注目され、それ以外のデータ型 (関数または配列など) は無視されます。ナビゲータオブジェクト内のそれぞれの String 値または Boolean 値は、`tempArr` 配列に付加されます。次に、ブラウザの画面オブジェクトが、`for..in` ループでループ処理され、それぞれの数値が `tempArr` 配列に追加されます。最後に、一時配列オブジェクトが、`Array.join()` メソッドでストリングに変換されます。この配列は、アンパサンド (&) を区切り記号として使用します。これにより、ActionScript は URLVariables クラスを使用してデータを簡単に解析できます。

第 24 章

プリント

24

Adobe® Flash® Player 9 では、OS のプリント機能インターフェイスと通信し、ページをプリントスプーラに送ることができます。Flash Player からスプーラに送信するページには、表示されているコンテンツ、動的なコンテンツ、または画面外にあってユーザーに表示されないコンテンツ (データベースの値や動的なテキストなど) を含めることができます。また、flash.printing.PrintJob クラスのプロパティがユーザーのプリンタ設定に基づいて Flash Player により設定されるため、適切なページ書式を指定できます。

この章では、flash.printing.PrintJob クラスのメソッドとプロパティを使用して、プリントジョブの作成、ユーザーのプリント設定の読み取り、および、Flash Player とユーザー OS のフィードバックに基づいたプリントジョブの調整を行う際の指針について詳しく説明します。

目次

プリントの基本 682
ページのプリント 683
Flash Player タスクとシステムプリント 684
サイズ、拡大率、用紙の向きの設定 687
例 : 複数ページのプリント 690
例 : 拡大 / 縮小、トリミング、および応答 692

プリントの基本

プリントの概要

ActionScript 3.0 では、PrintJob クラスを使用して、表示コンテンツのスナップショットを作成し、インクと紙を使った印刷結果として出力できます。コンテンツのプリント設定は画面表示の設定と似たところがあります。いずれの場合も、エレメントの位置やサイズを指定して、目的のレイアウトを作成します。ただし、プリントには画面レイアウトとは異なる点もあります。たとえば、プリンタではコンピュータのモニタとは異なる解像度が使用されます。コンピュータ画面のコンテンツは動的に変化しますが、印刷されたコンテンツは基本的に静止しています。また、プリントの場合は、固定ページサイズによる制約や複数ページプリントの可能性を考慮する必要があります。

これらの違いは当然のように思われるかもしれませんが、ActionScript でプリント設定する際には十分に注意してください。的確にプリントできるかどうかは、指定する値と使用するプリンタの特性との組み合わせによって決まります。そのため、PrintJob クラスには、考慮するべき使用プリンタの重要な特性を決定できるプロパティがあります。

一般的なプリントのタスク

この章では、次のような一般的なプリントタスクについて説明します。

- プリントジョブの開始
- プリントジョブへのページの追加
- ユーザーがプリントジョブをキャンセルするかどうかの判断
- ビットマップまたはベクターのいずれのレンダリングを使用するかの指定
- ページのサイズ、拡大率、用紙の向きの設定
- コンテンツの印刷領域の指定
- 画面サイズからページサイズへの変換
- 複数ページのプリントジョブの印刷

重要な概念と用語

次の参照リストに、この章で使用されている重要な用語を示します。

- スプーラ：オペレーティングシステムまたはプリンタドライバの一部で、印刷待機中のページを追跡し、プリンタが使用可能になるとページをプリンタに送信します。
- 用紙の向き : プリントされるコンテンツの、用紙に対する向き。水平 (横方向) または垂直 (縦方向)。
- プリントジョブ : 1 回の印刷結果を構成するページまたはページのセット。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章のコードリストの多くは、コードの一部を示したもので、プリントの完全な実用例や、コードまたは値をチェックするコードではありません。例をテストするには、プリントする要素を作成して、それらの要素に対してコードリストを使用する必要があります。この章の最後の 2 つの例は、プリントの完全な例です。これらの例には、プリントする内容を定義し、プリントタスクを実行するコードが含まれています。

サンプルコードリストをテストするには :

1. 新しい Flash ドキュメントを作成します。
2. タイムラインのフレーム 1 のキーフレームを選択して、[アクション] パネルを開きます。
3. コードリストを [スクリプト] ペインにコピーします。
4. メインメニューから [制御]-[ムービープレビュー] を選択して SWF ファイルを作成し、例をテストします。

ページのプリント

PrintJob クラスのインスタンスを使用して、プリントを処理できます。Flash Player からページをプリントする際の基本的な手順としては、次に示す 4 つのステートメントを順に使用します。

- `new PrintJob()`: 指定した名前で新しいプリントジョブのインスタンスを作成します。
- `PrintJob.start()`: OS のプリント処理を開始します。これにより、ユーザーに対してプリントダイアログボックスが表示され、また、プリントジョブの読み取り専用プロパティが設定されます。
- `PrintJob.addPage()`: プリントジョブのコンテンツに関する詳細を設定します。これには、Sprite オブジェクト (およびそれに含まれる子) の指定、プリント範囲のサイズ指定、および、イメージをベクターとビットマップのいずれの形式でプリントするかの指定が含まれます。`addPage()` を複数回呼び出せば、複数のページに複数のスプライトをプリントできます。
- `PrintJob.send()`: ページを OS のプリンタに送信します。

したがって、非常に単純なプリントジョブスクリプトは次のようになります (`package`、`import`、および `class` ステートメントはコンパイルするために必要です)。

```
package
{
  import flash.printing.PrintJob;
  import flash.display.Sprite;

  public class BasicPrintExample extends Sprite
  {
    var myPrintJob:PrintJob = new PrintJob();
```

```
    var mySprite:Sprite = new Sprite();

    public function BasicPrintExample()
    {
      myPrintJob.start();
      myPrintJob.addPage(mySprite);
      myPrintJob.send();
    }
  }
}
```

メモ

この例は、プリントジョブスクリプトの基本的なエレメントを示すためのもので、エラー処理は含まれていません。ユーザーによるプリントジョブのキャンセルに適切に対応するスクリプトを作成する場合は、684 ページの「例外および戻り値に対する処理」を参照してください。

何らかの理由で PrintJob オブジェクトのプロパティを " クリア " する必要がある場合は、PrintJob 変数に `null` を設定してください (例 : `myPrintJob = null`)。

Flash Player タスクとシステムプリント

OS のプリントインターフェイスに対するページの送出処理は Flash Player によって実行されるため、Flash Player が担当するタスクの範囲と、OS 自体のプリントインターフェイスが担当するタスクについて理解しておく必要があります。Flash Player では、プリントジョブの開始、プリンタのページ設定のうち一部の読み込み、OS に対するプリントジョブのコンテンツの送信、および、ユーザーまたはシステムによってプリントジョブがキャンセルされたかどうかの確認を実行します。その他の処理 (プリンタに特有のダイアログボックスの表示、スプールされたプリントジョブのキャンセル、プリンタのステータスに関するレポートなど) は、すべて OS で実行されます。Flash Player はプリントジョブの開始や書式指定に関する問題に応答しますが、その際に実行できる処理は、OS のプリント機能インターフェイスから取得した特定のプロパティや条件についてレポートを返すことだけに限られます。アプリケーション開発者は、レポートされたプロパティや条件に応じて何らかの処理を実行するコードを用意する必要があります。

例外および戻り値に対する処理

ユーザーによってプリントジョブがキャンセルされた場合に対応するため、`addPage()` および `send()` を呼び出す前には、`PrintJob.start()` メソッドが `true` を返したかどうか確認してください。これらのメソッドに対する呼び出しを次のように `if` ステートメントで囲むと、続行する前にキャンセルされたかどうかを簡単にチェックできます。

```
if (myPrintJob.start())
{
  // ここに addPage() および send() ステートメントを記述
}
```

PrintJob.start() が true の場合は、ユーザーが [印刷] を選択した (または Flash Player が [印刷] コマンドを開始した) ことを意味するため、addPage() および send() メソッドの呼び出しができます。

また、プリント処理の管理をしやすくするために、PrintJob.addPage() メソッドに対して Flash Player から例外が送出されるようになりました。これらのエラーをキャッチすることにより、ユーザーに情報と対処方法についての選択肢を示すことができます。PrintJob.addPage() メソッドが失敗した場合は、別の関数を呼び出すか、現在のプリントジョブを停止する方法もあります。例外をキャッチするには、addPage() の呼び出しを次のように try..catch ステートメントで囲みます。例に含まれている [params] の部分には、実際にプリントするコンテンツを指定するパラメータを指定します。

```
if (myPrintJob.start())
{
   try
   {
      myPrintJob.addPage([params]);
   }
   catch (error:Error)
   {
      // エラーを処理する
   }
   myPrintJob.send();
}
```

プリントジョブを開始した後は、PrintJob.addPage() でコンテンツを追加し、それによって例外 (ユーザーによるプリントジョブのキャンセルなど) が発生するかどうかを確認します。例外が発生した場合は、catch ステートメントに記述したロジックで、情報と対処方法の選択肢をユーザー (または Flash Player) に示すか、現在のプリントジョブを停止します。ページを正常に追加できた場合は、次に PrintJob.send() でページをプリンタに送信します。

Flash Player がプリントジョブをプリンタに送信する際に問題が発生した場合 (プリンタがオフラインになっていた場合など) も、同様にその例外をキャッチし、情報と対処方法の選択肢をユーザー (または Flash Player) に示すことができます (メッセージテキストと警告を Flash アニメーションで表示することなどが考えられます)。たとえば、次のように if..else ステートメント内でテキストフィールドに新しいテキストを設定するとします。

```
if (myPrintJob.start())
{
   try
   {
      myPrintJob.addPage([params]);
   }
   catch (error:Error)
   {
      // エラーを処理する
   }
```

```
  myPrintJob.send();
}
else
{
  myAlert.text = "Print job canceled";
}
```

実際に動作するサンプルについては、692 ページの「例 : 拡大 / 縮小、トリミング、および応答」を参照してください。

ページプロパティの操作

ユーザーが [印刷] ダイアログボックスで [OK] をクリックし、PrintJob.start() が true を返すと、プリンタの設定によって定義されるプロパティにアクセスできるようになります。このプロパティには、用紙の幅と高さ (pageHeight および pageWidth)、用紙の向きが含まれます。これらはプリンタの設定であり、Flash Player の制御下にあるプロパティではありません。したがって設定を変更することはできませんが、これらの情報を使用することで、コンテンツを設定に適した形式に整形してプリンタに送信できます。詳細については、687 ページの「サイズ、拡大率、用紙の向きの設定」を参照してください。

ベクター形式またはビットマップ形式のレンダリング

プリントジョブで各ページをベクターグラフィックとしてスプールするか、ビットマップイメージとしてスプールするかは、手動で設定できます。場合によっては、ベクター形式でプリントするとスプールファイルが小さくなり、画質もビットマップ形式でプリントするより向上することがあります。ただし、コンテンツにビットマップイメージが含まれている場合、アルファ透明度やカラー効果を再現するには、ページをビットマップイメージとしてプリントする必要があります。また、PostScript 非対応のプリンタでは、ベクターグラフィックは自動的にビットマップイメージに変換されます。ビットマップ形式でのプリントを指定するには、次のように、printAsBitmap パラメータに true を設定した PrintJobOptions オブジェクトを PrintJob.addPage() の第 3 パラメータとして渡します。

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

第 3 パラメータの値を指定しない場合、そのプリントジョブではベクター形式でのプリントがデフォルトで実行されます。

メモ

ビットマップ形式でのプリントを指定する必要がある場合に、printArea (第 2 パラメータ) を指定しないようにするには、printArea として null を渡します。

プリントジョブステートメントのタイミング

ActionScript 3.0 では、ActionScript の以前のバージョンと同様に、PrintJob オブジェクトが単一フレームに制限されません。ただし、[印刷] ダイアログボックスで [OK] ボタンをクリックすると印刷ステータス情報が表示されるため、ページをスプーラに送信したらただちに `PrintJob.addPage()` および `PrintJob.send()` を呼び出す必要があります。`PrintJob.send()` 呼び出しの格納されたフレームに遅延が到達すると、印刷処理が遅延します。

ActionScript 3.0 には、15 秒間のスクリプトタイムアウトの制限があります。したがって、プリントジョブのシーケンスにおける主要なステートメント間の時間が 15 秒を超えてはなりません。つまり、15 秒間のスクリプトタイムアウトの制限は、次の間隔に適用されます。

- `PrintJob.start()` と最初の `PrintJob.addPage()` との間
- `PrintJob.addPage()` とその次の `PrintJob.addPage()` との間
- 最後の `PrintJob.addPage()` と `PrintJob.send()` との間

これらの間隔のいずれかが 15 秒を超えていると、その次の PrintJob インスタンスの `PrintJob.start()` の呼び出しで `false` が返され、その次の PrintJob インスタンスの `PrintJob.addPage()` によって、Flash Player がランタイム例外をスローします。

サイズ、拡大率、用紙の向きの設定

683 ページの「ページのプリント」セクションで詳しく説明している基本的なプリントジョブ実行手順の場合は、指定したスプライトの画面サイズおよび位置を直接に反映した同等のプリントが出力されます。しかし、プリンタの解像度は環境によって異なり、また、プリンタの設定によってスプライトのプリント結果に悪影響が出る場合もあります。

Flash Player はオペレーティングシステムの印刷設定を読み取ることができますが、そのようなプロパティは読み取り専用です。値に応答することはできても、設定することはできません。したがって、プリンタの用紙サイズ設定を調べた後は、コンテンツのサイズなどを調整してプリンタに合わせることになります。また、プリンタの余白や用紙の向きに関する設定を調べることもできます。プリンタの設定に合わせるには、プリント範囲の指定、画面解像度とプリンタのポイント尺度との違いに関する調整、用紙のサイズと向きに応じた幾何学変換などが必要となります。

矩形によるプリント範囲の指定

PrintJob.addPage() メソッドでは、スプライトのうちプリントの対象とする範囲を指定できます。それには、第 2 パラメータの printArea として Rectangle オブジェクトを指定します。このパラメータの値については、次の 3 つの指定方法があります。

- 特定のプロパティを持つ Rectangle オブジェクトを前もって作成しておく場合は、それを次のように addPage() の呼び出しで使用します。

```
private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
myPrintJob.addPage(sheet, rect1);
```

- Rectangle オブジェクトを前もって用意しない場合は、次のように、呼び出しの時点で直接指定することもできます。

```
myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
```

- addPage() 呼び出しの第 3 パラメータを指定する必要がある場合に、プリント範囲の矩形を指定しないようにするには、次のように、第 2 パラメータとして null を使用します。

```
myPrintJob.addPage(sheet, null, options);
```

メモ

プリント寸法を矩形で指定する場合は、flash.display.Rectangle クラスを import で読み込んでください。

ポイントとピクセルの比較

矩形の幅と高さはピクセル値です。一方、プリンタで使用する測定単位はポイントです。ポイントの物理サイズは固定 (1/72 インチ) ですが、画面上のピクセルのサイズは解像度によって異なります。したがって、ピクセル値とポイント値の変換比率は、プリンタの設定と、スプライトが拡大または縮小されているかどうかによって変化します。72 ピクセル幅の伸縮されていない Sprite は、用紙上では 1 インチ幅で印刷されます。1 ポイントは 1 ピクセルに相当し、画面の解像度とは無関係です。

インチおよびセンチメートルと、ポイントおよび twip (1/20 ポイント) との間には、次の等式が成り立ちます。

- 1 ポイント = 1/72 インチ = 20 twip
- 1 インチ = 72 ポイント = 1440 twip
- 1 センチメートル = 567 twip

printArea パラメータを省略するか、このパラメータの指定が正しくない場合は、スプライト全体がプリントされます。

拡大 / 縮小

印刷前に Sprite オブジェクトを拡大 / 縮小するには、`PrintJob.addPage()` メソッドを呼び出す前に拡大 / 縮小のプロパティ (383 ページの「サイズの変更とオブジェクトの拡大 / 縮小」を参照) を設定します。印刷後、これらのプロパティを元の値に戻します。Sprite オブジェクトの拡大 / 縮小は、`printArea` プロパティとは無関係です。つまり、50 x 50 ピクセルのプリント範囲を指定した場合は常に 2500 個のピクセルがプリントされます。Sprite オブジェクトを拡大または縮小している場合、その設定を反映してプリント結果は伸縮しますが、プリントの対象となる 2500 個のピクセルは変わりません。

例については、692 ページの「例 : 拡大 / 縮小、トリミング、および応答」を参照してください。

用紙の向きが Landscape (横方向) または Portrait (縦方向) である場合のプリント

Flash Player では用紙の向きに関する設定を検出できるため、次の例に示すように、プリンタ設定に応じてコンテンツのサイズや向きを調整するロジックを ActionScript に組み込むことができます。

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
  mySprite.rotation = 90;
}
```

メモ

用紙の向きを調べるためにシステムの設定を読み取る場合は、次のように PrintJobOrientation クラスを import で読み込んでください。

```
import flash.printing.PrintJobOrientation;
```

PrintJobOrientation クラスには、用紙の向きを表す定数値が定義されています。

ページの幅と高さへの対応

用紙の向きに関する設定情報を扱う場合と同様、次のように `if` ステートメントで囲んだロジックを記述することで、ページの幅と高さの設定を読み取り、それに応じた処理ができます。その例を次のコードに示します。

```
if (mySprite.height > myPrintJob.pageHeight)
{
  mySprite.scaleY = .75;
}
```

また、余白の設定を調べるには、次のように用紙の寸法とページの寸法を比較します。

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

例 : 複数ページのプリント

複数ページのコンテンツをプリントする場合は、各ページのコンテンツを別々のスプライト (下の例では sheet1 および sheet2) に割り当て、それぞれについて PrintJob.addPage( ) を呼び出します。このテクニックを説明するコードを次に示します。

```
package
{
  import flash.display.MovieClip;
  import flash.printing.PrintJob;
  import flash.printing.PrintJobOrientation;
  import flash.display.Stage;
  import flash.display.Sprite;
  import flash.text.TextField;
  import flash.geom.Rectangle;

  public class PrintMultiplePages extends MovieClip
  {
    private var sheet1:Sprite;
    private var sheet2:Sprite;

    public function PrintMultiplePages():void
    {
      init();
      printPages();
    }

    private function init():void
    {
      sheet1 = new Sprite();
      createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80,
  height:130});
      sheet2 = new Sprite();
      createSheet(sheet2, "There was a great story to tell, and it ended
  quickly.\n\nThe end.", null);
    }

    private function createSheet(sheet:Sprite, str:String,
  imgValue:Object):void
    {
      sheet.graphics.beginFill(0xEEEEEE);
      sheet.graphics.lineStyle(1, 0x000000);
      sheet.graphics.drawRect(0, 0, 100, 200);
      sheet.graphics.endFill();

      var txt:TextField = new TextField();
      txt.height = 200;
      txt.width = 100;
      txt.wordWrap = true;
```

```
      txt.text = str;

      if (imgValue != null)
      {
        var img:Sprite = new Sprite();
        img.graphics.beginFill(0xFFFFFF);
        img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width,
imgValue.height);
        img.graphics.endFill();
        sheet.addChild(img);
      }
      sheet.addChild(txt);
    }

    private function printPages():void
    {
      var pj:PrintJob = new PrintJob();
      var pagesToPrint:uint = 0;
      if (pj.start())
      {
        if (pj.orientation == PrintJobOrientation.LANDSCAPE)
        {
          throw new Error("Page is not set to an orientation of portrait.");
        }

        sheet1.height = pj.pageHeight;
        sheet1.width = pj.pageWidth;
        sheet2.height = pj.pageHeight;
        sheet2.width = pj.pageWidth;

        try
        {
          pj.addPage(sheet1);
          pagesToPrint++;
        }
        catch (error:Error)
        {
        // エラーに応答する
        }

        try
        {
          pj.addPage(sheet2);
          pagesToPrint++;
        }
        catch (error:Error)
        {
          // エラーに応答する
        }
```

```
            if (pagesToPrint > 0)
            {
               pj.send();
            }
         }
      }
   }
}
```

例 : 拡大 / 縮小、トリミング、および応答

場合によっては、画面の表示とプリントとの外観上の違いに対応するために、表示オブジェクトのサイズ (またはその他のプロパティ) をプリント時に調整する必要が生じることがあります。プリントの前に表示オブジェクトのプロパティ (例 : scaleX および scaleY プロパティ) を調整する際には、プリント範囲を定義する矩形よりも大きくオブジェクトを拡大するとトリミングが発生することに注意してください。また、多くの場合には、ページをプリントした後でプロパティを元の値に戻す必要があります。

次のコードは、txt 表示オブジェクトの寸法を拡大 / 縮小 (ただし、背景にある緑色のボックスはそのまま) していることで、指定した矩形のサイズによってテキストフィールドがトリミングされる結果になる例です。プリントした後は、テキストフィールドを画面表示用の元のサイズに戻します。OS の [印刷] ダイアログボックスでプリントジョブがキャンセルされた場合は、Flash Player の表示内容を変更し、ジョブがキャンセルされたことを警告します。

```
package
{
   import flash.printing.PrintJob;
   import flash.display.Sprite;
   import flash.text.TextField;
   import flash.display.Stage;
   import flash.geom.Rectangle;

   public class PrintScaleExample extends Sprite
   {
      private var bg:Sprite;
      private var txt:TextField;

      public function PrintScaleExample():void
      {
         init();
         draw();
         printPage();
      }

      private function printPage():void
      {
         var pj:PrintJob = new PrintJob();
```

```
        txt.scaleX = 3;
        txt.scaleY = 2;
        if (pj.start())
        {
          trace(">> pj.orientation: " + pj.orientation);
          trace(">> pj.pageWidth: " + pj.pageWidth);
          trace(">> pj.pageHeight: " + pj.pageHeight);
          trace(">> pj.paperWidth: " + pj.paperWidth);
          trace(">> pj.paperHeight: " + pj.paperHeight);

          try
          {
            pj.addPage(this, new Rectangle(0, 0, 100, 100));
          }
          catch (error:Error)
          {
            // 何もしない
          }
          pj.send();
        }
        else
        {
          txt.text = "Print job canceled";
        }
        // txt の拡大 / 縮小プロパティを元に戻す
        txt.scaleX = 1;
        txt.scaleY = 1;
      }

      private function init():void
      {
        bg = new Sprite();
        bg.graphics.beginFill(0x00FF00);
        bg.graphics.drawRect(0, 0, 100, 200);
        bg.graphics.endFill();

        txt = new TextField();
        txt.border = true;
        txt.text = "Hello World";
      }

      private function draw():void
      {
        addChild(bg);
        addChild(txt);
        txt.x = 50;
        txt.y = 50;
      }
    }
}
```

第 25 章

External API の使用

25

ActionScript 3.0 External API を使用すると、Adobe Flash Player 9 が動作しているコンテナアプリケーションと ActionScript との間で簡単な手順による通信ができます。External API を使用する状況としては、HTML ページ内で SWF ドキュメントと JavaScript 間の操作を行う場合や、Flash Player を使用して SWF ファイルを表示するデスクトップアプリケーションを作成する場合などがあります。

この章では、External API を使用してコンテナアプリケーションを操作する方法、HTML ページ内で ActionScript と JavaScript との間でデータを受け渡す方法、および ActionScript とデスクトップアプリケーションとの間で通信を確立し、データを交換する方法について説明します。

メモ

この章では、SWF 内の ActionScript と、SWF をロードする Flash Player インスタンスへの参照が含まれるコンテナアプリケーションの間の通信についてのみ説明します。アプリケーション内でのその他の Flash Player の使用は、本マニュアルの範囲外です。Flash Player は、ブラウザのプラグインまたはプロジェクタ (スタンドアローンアプリケーション) として使用することを目的としています。それ以外の使用法については、サポートが制限される場合があります。

目次

External API 使用の基本 696
External API の要件と利点 699
ExternalInterface クラスの使用 700
例 : Web ページコンテナに対する External API の使用 705
例 : ActiveX コンテナに対する External API の使用 712

External API 使用の基本

External API の使用についての概要

SWF プロジェクタ作成時など、SWF ファイルが単独で実行されることもありますが、ほとんどの場合、SWF アプリケーションは他のアプリケーションの埋め込みエレメントとして実行されます。通常、SWF が含まれているコンテナは HTML ファイルですが、稀にデスクトップアプリケーションのユーザーインターフェイスの全部または一部として SWF ファイルが使用されることがあります。

より高度なアプリケーションで作業するようになると、SWF ファイルとコンテナアプリケーションの間の通信設定が必要になるでしょう。たとえば、一般的には、Web ページで HTML 形式でテキストなどの情報を表示させ、SWF ファイルでチャートやビデオなどの動的かつ視覚的なコンテンツを表示させます。この場合、Web ページ上のボタンを押すと SWF ファイルも変更されるようにしたいと思うかもしれません。ActionScript に含まれる External API というメカニズムにより、SWF ファイル内の ActionScript とコンテナアプリケーションに含まれるその他のコード間におけるこのような通信が容易になります。

一般的な External API のタスク

この章では、次のような一般的な External API のタスクについて説明します。

- コンテナアプリケーションに関する情報の取得
- コンテナアプリケーション (Web ページやデスクトップアプリケーションなど) における ActionScript を使用したコードの呼び出し
- コンテナアプリケーション内のコードからの ActionScript コードの呼び出し
- コンテナアプリケーションからの ActionScript コードの呼び出しを簡易化するプロキシの作成

重要な概念と用語

次の参照リストに、この章で使用されている重要な用語を示します。

- ActiveX コンテナ : アプリケーション内で SWF コンテンツを表示するための、Flash Player ActiveX コントロールのインスタンスが含まれるコンテナアプリケーション (Web ブラウザではない)。
- コンテナアプリケーション : Flash Player コンテンツを含む Web ブラウザや HTML ページなど、Flash Player が内部で SWF ファイルを実行するアプリケーション。

- プロジェクタ : スタンドアローン実行可能ファイルに変換された SWF ファイル。Flash Player および SWF ファイルのコンテンツを含みます。プロジェクタは、Adobe Flash CS3 Professional またはスタンドアローンの Flash Player を使用して作成します。プロジェクタは通常、SWF ファイルを CD-ROM で配布する場合や、ダウンロードサイズが問題にならず、ユーザーのコンピュータに Flash Player がインストールされているかどうかに関わらず確実に SWF ファイルが実行されるようにしたいと SWF 作成者が考えているような状況で使用されます。
- プロキシ : あるアプリケーション (“外部アプリケーション”) のコードを他のアプリケーション (“呼び出しアプリケーション ”) に代わって呼び出す仲介者となるアプリケーションまたはコード。呼び出しアプリケーションに値を返します。プロキシは次のようなさまざまな目的で使用されます。
  - 呼び出しアプリケーションにおけるネイティブ関数の呼び出しを、外部アプリケーションで理解可能なフォーマットに変換することにより、外部関数の呼び出しプロセスを簡単にする
  - 呼び出し元と外部アプリケーションとの直接通信を防止する、セキュリティおよびその他の制約に対処する
- 直列化 : オブジェクトやデータ値を、2 つのプログラミングシステム間 (インターネット上、1 台のコンピュータで実行中の 2 つの異なるアプリケーション間など) のメッセージにおいて値を受け渡しできるフォーマットに変換すること。

章内の例の使用

この章の進行に合わせて、サンプルコードリストを実際にテストすることができます。この章のコードリストの多くは、デモンストレーション目的の小さなコードリストであり、完全な実用例または値をチェックするコードではありません。External API を使用するには ActionScript コードおよびコンテナアプリケーション内のコードを作成する必要があることが定義されているので、サンプルをテストするには、コンテナ (SWF を含んだ Web ページなど) を作成して、コードリストを使用してコンテナとやり取りする必要があります。

ActionScript と JavaScript 間の通信の例をテストするには :

1. 新しい Flash ドキュメントを作成して、コンピュータに保存します。
2. メインメニューから [ファイル]-[パブリッシュ設定] を選択します。
3. [パブリッシュ設定] ダイアログボックスの [フォーマット] タブで、[Flash] および [HTML] チェックボックスが選択されていることを確認します。
4. [パブリッシュ] ボタンをクリックします。同じフォルダに、Flash ドキュメントを保存したときと同じ名前で SWF ファイルと HTML ファイルが生成されます。[OK] をクリックして、[パブリッシュ設定] ダイアログボックスを閉じます。
5. [HTML] チェックボックスを選択解除します。これにより HTML ページが生成されるので、その HTML ページを変更して、適切な JavaScript コードを追加します。[HTML] チェックボックスを選択解除すると、HTML ページを変更してから SWF ファイルをパブリッシュするときに、Flash により変更が上書きされることはありません。

6. [OK] をクリックして、[パブリッシュ設定] ダイアログボックスを閉じます。
7. HTML またはテキストエディタアプリケーションで、SWF をパブリッシュしたときに Flash で作成された HTML ファイルを開きます。HTML ソースコードでスクリプト要素を追加し、サンプルコードリストからそのスクリプト要素を JavaScript コードをコピーします。
8. HTML ファイルを保存して、Flash に戻ります。
9. タイムラインのフレーム 1 のキーフレームを選択して、[アクション] パネルを開きます。
10. ActionScript コードリストを [スクリプト] ペインにコピーします。
11. メインメニューから [ファイル]-[パブリッシュ] を選択し、変更内容を使用して SWF ファイルを更新します。
12. Web ブラウザで、編集した HTML ページを開いてページを表示し、ActionScript と HTML ページ間の通信をテストします。

ActionScript と ActiveX コンテナ間の通信の例をテストするには :

1. 新しい Flash ドキュメントを作成して、コンピュータに保存します。コンテナアプリケーションが SWF ファイルを検索する宛先のフォルダに保存することをお勧めします。
2. メインメニューから [ファイル]-[パブリッシュ設定] を選択します。
3. [パブリッシュ設定] ダイアログボックスの [フォーマット] タブで、[Flash] チェックボックスだけが選択されていることを確認します。
4. [Flash] チェックボックスの横の [ファイル] フィールドで、フォルダアイコンをクリックして、SWF ファイルがパブリッシュされるフォルダを選択します。SWF ファイルの場所を設定することにより、Flash ドキュメントを 1 つのフォルダで管理しながら、パブリッシュした SWF ファイルを別のフォルダ (コンテナアプリケーションのソースコードが含まれるフォルダなど) で管理できます。
5. タイムラインのフレーム 1 のキーフレームを選択して、[アクション] パネルを開きます。
6. サンプルの ActionScript コードを [スクリプト] ペインにコピーします。
7. メインメニューから [ファイル]-[パブリッシュ] を選択して、SWF ファイルを再びパブリッシュします。
8. ActionScript とコンテナアプリケーション間の通信をテストするためのコンテナアプリケーションを作成して実行します。

この章の最後の 2 つの例は、External API を使用して、それぞれ HTML ページと C# デスクトップアプリケーションと通信する完全な例です。これらの例には完全なコードが含まれており、その中には、External API を使用してコードを作成するときに使用する、ActionScript とコンテナエラーチェックコードも含まれています。External API を使用する別の完全な例については、『ActionScript 3.0 コンポーネントリファレンスガイド』を参照してください。

External API の要件と利点

External API は ActionScript の一部で、Flash Player のコンテナとして機能する "外部アプリケーション " (一般的には Web ブラウザまたはスタンドアロンのプロジェクターアプリケーション) で実行されるコードと ActionScript 間の通信用メカニズムを提供します。ActionScript 3.0 では、External API の機能が ExternalInterface クラスによって提供されます。Flash Player 8 よりも前のバージョンの Flash Player では、`fscommand()` アクションを使用してコンテナアプリケーション内での通信が実行されていました。ExternalInterface クラスは `fscommand()` に代わるもので、JavaScript と ActionScript との間のすべての通信用に推奨されます。

メモ	以前の `fscommand()` 関数を使用する必要がある場合 (たとえば、以前のバージョンのアプリケーションとの互換性を維持する場合や、サードパーティの SWF コンテナアプリケーションやスタンドアロン Flash Player で操作する場合など) は、パッケージレベルの関数として flash.system パッケージに用意されています。

ExternalInterface クラスは、ActionScript と Flash Player から、HTML ページ内の JavaScript や Flash Player のインスタンスを含むデスクトップアプリケーションへの通信を可能にするサブシステムです。

ExternalInterface クラスは次の条件でのみ使用できます。

- Windows 版 Internet Explorer の全サポートバージョン (5.0 以降)
- Flash Player ActiveX コントロールのインスタンスを使用するデスクトップアプリケーションなどのコンテナアプリケーション
- NPRuntime インターフェイスをサポートするすべてのブラウザ。現時点では次のブラウザが該当します。
  - Firefox 1.0 以降
  - Mozilla 1.7.5 以降
  - Netscape 8.0 以降
  - Safari 1.3 以降

その他の環境 (スタンドアローン Player で実行する場合など) では、`ExternalInterface.available` プロパティは `false` を返します。

ActionScript から、HTML ページの JavaScript 関数を呼び出すことができます。External API の機能は、次の点において `fscommand()` よりも強力です。

- `fscommand()` 関数から使用できる機能だけでなく、すべての JavaScript 関数を使用できます。
- 1 つのコマンドと 1 つのストリングパラメータという制限はなく、任意の個数のパラメータを任意の名前で渡すことができます。これにより、External API は `fscommand()` よりも非常に柔軟になっています。

- String パラメータだけでなく、さまざまなデータ型 (Boolean、Number、String など) を渡すことができます。
- 呼び出し結果の値を受け取ることができます。結果は呼び出しに対する戻り値として、直ちに ActionScript に戻されます。

警告	HTML ページの Flash Player インスタンス (`object` タグの `id` 属性) に付けられた名前にハイフン (`-`) またはその他の JavaScript で演算子として定義された文字 (`+`、`*`、`/`、`\`、`.` など) が含まれていると、コンテナ Web ページを Internet Explorer で表示した場合に、ActionScript からの ExternalInterface 呼び出しが機能しません。 また、Flash Player インスタンスを定義する HTML タグ (`object` および `embed` タグ) が HTML `form` タグ内でネストされていると、ActionScript からの ExternalInterface 呼び出しが機能しません。

ExternalInterface クラスの使用

ActionScript とコンテナアプリケーション間の通信には、コンテナで定義されたコード (JavaScript 関数など) を ActionScript から呼び出す方法と、呼び出し可能なように設計された ActionScript 関数をコンテナのコードから呼び出す方法の 2 種類の形式があります。いずれの場合も、呼び出される側のコードに情報が送信され、その結果が呼び出し側コードに返されます。

この通信を容易にするため、ExternalInterface クラスには 2 つの静的プロパティと 2 つの静的メソッドが用意されています。これらのプロパティとメソッドは、外部インターフェイス接続に関する情報を取得し、コンテナ内のコードを ActionScript から実行し、コンテナから ActionScript 関数を呼び出せるようにするために使用されます。

外部コンテナに関する情報の取得

`ExternalInterface.available` プロパティは、外部インターフェイス機能を提供するコンテナ内に現在の Flash Player が格納されているかどうかを示します。外部インターフェイスが提供されている場合、このプロパティは `true` になります。利用できない場合には `false` になります。

ExternalInterface クラスの他の機能を使用する前に、現在のコンテナにおいて外部インターフェイスによる通信がサポートされているかどうかを、次のようにして必ず確認してください。

```
if (ExternalInterface.available)
{
  // ExternalInterface メソッドをここで実行する
}
```

メモ	`ExternalInterface.available` プロパティで確認できるのは、現在のコンテナが ExternalInterface 接続をサポートするタイプなのかどうかです。現在使用しているブラウザの設定で JavaScript が有効化されているかどうかは確認できません。

ExternalInterface.objectID プロパティを使用すると、Flash Player インスタンスの一意の識別子 (特に、Internet Explorer における object タグの id 属性や、NPRuntime インターフェイスを使用するブラウザにおける embed タグの name 属性) を確認できます。この一意の ID は、ブラウザ内の現在の SWF ドキュメントを表し、SWF ドキュメントを参照する場合に使用できます (コンテナ HTML ページ内の JavaScript 関数の呼び出しなど)。Flash Player コンテナが Web ブラウザではない場合、このプロパティは null になります。

ActionScript からの外部コードの呼び出し

ExternalInterface.call() メソッドでは、コンテナアプリケーション内のコードが実行されます。パラメータは最低 1 つ必要で、コンテナアプリケーション内で呼び出される関数の名前を含んだストリングを指定します。ExternalInterface.call() メソッドに渡された追加パラメータは、関数呼び出しのパラメータとしてコンテナに渡されます。

```
// 外部関数 "addNumbers" を呼び出し、
// 2 つのパラメータを渡して、その関数の結果を
// 変数 "result" に割り当てる
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

コンテナが HTML ページの場合は、指定された名前の JavaScript 関数がこのメソッドによって呼び出されます。この名前は、格納された HTML ページの script エレメントで定義されている必要があります。JavaScript 関数の戻り値は ActionScript に返されます。

```
<script language="JavaScript">
  // 2 つの数値を加算し、その結果を ActionScript に返す
  function addNumbers(num1, num2)
  {
    return (num1 + num2);
  }
</script>
```

コンテナがその他の ActiveX コンテナの場合は、このメソッドによって Flash Player ActiveX コントロールから FlashCall イベントが送出されます。指定された関数名とパラメータは、Flash Player によって直列化されて XML ストリングになります。コンテナは、イベントオブジェクトの request プロパティでその情報にアクセスし、それを使用して自身のコードの実行方法を判定できます。値を ActionScript に返すには、コンテナコードで ActiveX オブジェクトの SetReturnValue() メソッドを呼び出し、その結果 (シリアル化された XML ストリング) をメソッドのパラメータとして渡します。この通信で使用される XML フォーマットの詳細については、703 ページの「External API の XML フォーマット」を参照してください。

コンテナが Web ブラウザでも、その他の ActiveX コンテナでも、呼び出しが失敗した場合やコンテナ側のメソッドが戻り値を返さなかった場合は、null が返されます。呼び出し側コードによるアクセスが、コンテナ環境の属するセキュリティ Sandbox によって許可されていない場合、ExternalInterface.call() メソッドは SecurityError 例外をスローします。この問題を回避するには、コンテナ環境の allowScriptAccess に適切な値を設定します。たとえば、HTML ページの allowScriptAccess の値を変更するには、object および embed タグの該当する属性を編集します。

コンテナからの ActionScript コードの呼び出し

コンテナが呼び出すことができるのは、関数内の ActionScript コードのみです。それ以外の ActionScript コードは、コンテナから呼び出すことはできません。コンテナアプリケーションから ActionScript 関数を呼び出すには、ExternalInterface クラスへの関数の登録と、コンテナコードからの関数の呼び出しという、2 つの処理が必要になります。

最初に、ActionScript 関数を登録し、コンテナから使用可能にすることを示す必要があります。ExternalInterface.addCallback() メソッドを次のように使用します。

```
function callMe(name:String):String
{
  return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

addCallback() メソッドには 2 つのパラメータがあります。1 つはストリングの関数名で、コンテナはその名前で関数を認識します。もう 1 つのパラメータは実際の ActionScript 関数で、定義された関数名をコンテナが呼び出すと実行されます。これらの名前は区別されるため、実際の ActionScript 関数が異なる名前であっても、コンテナで使用される関数名を指定できます。これは、匿名関数が指定された場合や、呼び出される関数が実行時に決定される場合など、関数名が不明な場合に非常に便利です。

ActionScript 関数が ExternalInterface クラスに登録されると、コンテナが実際に関数を呼び出せるようになります。その方法は、コンテナのタイプによって異なります。たとえば、Web ブラウザの JavaScript コードの場合、登録された関数名をまるで Flash Player ブラウザオブジェクトのメソッド (object または embed タグを表す JavaScript オブジェクトのメソッド) のように使用して ActionScript 関数が呼び出されます。つまり、パラメータが渡され、呼び出されたローカル関数から結果が返されます。

```
<script language="JavaScript">
  // callResult gets the value "busy signal"
  var callResult = flashObject.myFunction("my name");
</script>
...
```

```
<object id="flashObject"...>
  ...
  <embed name="flashObject".../>
</object>
```

あるいは、デスクトップアプリケーションで実行されている SWF ファイルの ActionScript 関数を呼び出す場合、登録された関数名とすべてのパラメータを直列化して XML フォーマットストリングにする必要があります。その後、ActiveX コントロールの `CallFunction()` メソッドとパラメータとして XML ストリングを呼び出すことで、呼び出しが実際に実行されます。この通信で使用される XML フォーマットの詳細については、703 ページの「External API の XML フォーマット」を参照してください。

いずれの場合も、ActionScript 関数の戻り値はコンテナコードに返されますが、呼び出し側がブラウザの JavaScript コードの場合は値として直接返され、呼び出し側が ActiveX コンテナの場合は XML フォーマットストリングとして直列化して返されます。

External API の XML フォーマット

Shockwave Flash ActiveX コントロールをホストするアプリケーションと ActionScript 間の通信では、関数呼び出しと値をエンコードする際に専用の XML フォーマットが使用されます。External API で使用される XML フォーマットには 2 つの部分があります。1 つのフォーマットは、関数呼び出しを表すために使用されます。もう 1 つのフォーマットは個々の値を表しています。このフォーマットは、関数のパラメータおよび関数の戻り値に使用されます。関数呼び出し用の XML フォーマットは、ActionScript との間の呼び出しに使用されます。ActionScript からの関数呼び出しに対して、Flash Player は XML をコンテナに渡します。コンテナからの呼び出しに対しては、Flash Player はこのフォーマットで XML ストリングを渡すコンテナアプリケーションが必要になります。次の XML は、XML フォーマットされた関数呼び出しの例を示しています。

```
<invoke name="functionName" returntype="xml">
  <arguments>
    ... (individual argument values)
  </arguments>
</invoke>
```

ルートノードは `invoke` ノードです。`name` は関数呼び出しの名前を表し、`returntype` は常に `xml` になります。関数呼び出しにパラメータが含まれている場合、`invoke` ノードには `arguments` 子ノードができます。さらにその子ノードは、次に説明する個別の値フォーマットを使用してフォーマットされたパラメータになります。

関数のパラメータや関数の戻り値などの個々の値では、実際の値に加えてデータ型情報を含んだフォーマッティングスキームが使用されます。次の表に、ActionScript クラス、およびそのデータ型の値のエンコードに使用する XML フォーマットの一覧を示します。

ActionScript のクラスまたは値	C# のクラスまたは値	フォーマット	コメント
null	null	<null/>	
Boolean `true`	bool true	<true/>	
Boolean `false`	bool false	<false/>	
String	string	<string> ストリング値 </string>	
Number、int、uint	single、double、int、uint	`<number>27.5</number>` `<number>-12</number>`	

Array (エレメントは型の混在可)	ArrayList や object[] など、エレメントの型の混在が可能なコレクション	`<array>`
		`  <property id="0">`
		`  <number>27.5</number>`
		`  </property>`
		`  <property id="1">`
		`  <string>Hello there!</string>`
		`  </property>`
		`  ...`
		`</array>`
			`property` ノードでは個々のエレメントが定義され、`id` 属性は 0 から始まるインデックスの数値です。

Object	ストリングキー付きの HashTable など、ストリングキーとオブジェクト値のある辞書	`<object>`
		`  <property id="name">`
		`  <string>John Doe</string>`
		`  </property>`
		`  <property id="age">`
		`  <string>33</string>`
		`  </property>`
		`  ...`
		`</object>`
			`property` ノードでは個々のプロパティが定義され、`id` 属性はプロパティ名 (ストリング) です。

その他のビルトインクラスまたはカスタムクラス		`<null/>`
		または
		`<object></object>`
			ActionScript では他のオブジェクトが null または空のオブジェクトとしてエンコードされます。いずれの場合も、すべてのプロパティ値が失われます。

メモ

この表では、ActionScript クラスに加えて同等の C# クラスを例示していますが、External API は、C# アプリケーションに限定されることなく、ActiveX コントロールをサポートする任意のプログラミング言語やランタイムとの通信に使用できます。

External API と ActiveX コンテナアプリケーションを使用して独自のアプリケーションを構築している場合は、ネイティブ関数の呼び出しを直列化 XML 形式に変換できるプロキシを作成すると便利です。C# で記述されたプロキシクラスの例については、717 ページの「ExternalInterfaceProxy クラスの内部」を参照してください。

例 : Web ページコンテナに対する External API の使用

このサンプルアプリケーションでは、ユーザーが自分自身とチャットできる Instant Messaging アプリケーションの場合に、Web ブラウザ内で ActionScript と JavaScript が通信するための方法を示しています (アプリケーション名 : Introvert IM)。メッセージは External API を使用して、Web ページ内の HTML 形式と SWF インターフェイスとの間で送信されます。この例で示す方法では、次のことを行います。

- ブラウザが通信可能なことを確認してから通信を設定することによる、通信の正しい初期化
- コンテナが External API をサポートしているかどうかの確認
- ActionScript からの JavaScript 関数の呼び出し、パラメータの受け渡し、および応答による値の受け取り
- JavaScript から呼び出される ActionScript メソッドの有効化、および呼び出しの実行

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。Introvert IM アプリケーションのファイルは、Samples/IntrovertIM_HTML フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
IntrovertIMApp.fla または IntrovertIMApp.mxml	Flash (FLA) または Flex (MXML) のメインアプリケーションファイル
com/example/programmingas3/introvertIM/IMManager.as	ActionScript とコンテナとの間の通信を確立および管理するクラス。
com/example/programmingas3/introvertIM/IMMessageEvent.as	コンテナからメッセージを受信したときに IMManager クラスから送出されるカスタムイベントタイプ。

ファイル	説明
com/example/programmingas3/introvertIM/IMStatus.as	アプリケーションで選択可能な、さまざまな "availability" ステータス値を値が表している列挙。
html-flash/IntrovertIMApp.html または html-template/index.template.html	Flash アプリケーションの HTML ページ (html-flash/IntrovertIMApp.html) または Adobe Flex アプリケーションのコンテナ HTML ページ作成に使用するテンプレート (html-template/index.template.html). このファイルには、アプリケーションのコンテナ部分を構成するすべての JavaScript 関数が格納されています。

ActionScript とブラウザ間の通信の準備

External API の最も一般的な用途の 1 つは、ActionScript アプリケーションと Web ブラウザとの通信です。External API を使用すると、ActionScript メソッドから JavaScript コードへの呼び出し、および JavaScript コードから ActionScript メソッドへの呼び出しを実現できます。ブラウザの仕組みは複雑であり、また、ブラウザ内部で実行されるページレンダリング処理が不明であることから、HTML ページ上の JavaScript コードが最初に実行される前に SWF ドキュメントのコールバックを登録できる保証はありません。したがって、JavaScript から SWF ドキュメントの関数を呼び出す前に、SWF ドキュメント側の接続受け付け準備ができたことを通知するために、必ず SWF ドキュメントから HTML ページへの呼び出しを実行してください。

たとえば、IMManager クラスで実行される一連の手順によって、ブラウザの通信準備ができ、SWF ファイルを通信で使用できることが、Introvert IM で判定されます。ブラウザの通信準備ができたことを判定する最初の手順は、次のように IMManager コンストラクタで実行されます。

```
public function IMManager(initialStatus:IMStatus)
{
  _status = initialStatus;

  // コンテナで External API を使用できることを確認する
  if (ExternalInterface.available)
  {
    try
    {
      // ここで isContainerReady() メソッドを呼び出し、そのメソッドが
      // コンテナを呼び出すことにより、Flash Player がロード済みでコンテナが
      // SWF からの呼び出しを受信可能なことを確認する
      var containerReady:Boolean = isContainerReady();
      if (containerReady)
      {
        // コンテナが準備できている場合は、SWF の関数を登録する
        setupCallbacks();
      }
      else
```

```
        {
          // コンテナの準備ができていない場合は、Timer を設定して
          // 100ms 間隔でコンテナを呼び出す。コンテナの準備ができたとの応答があれば、
          // タイマーを停止する
          var readyTimer:Timer = new Timer(100);
          readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
          readyTimer.start();
        }
      }
      ...
    }
    else
    {
      trace("このコンテナでは External インターフェイスを使用できません。");
    }
}
```

最初に、ExternalInterface.available プロパティを使用して External API が現在のコンテナで利用可能かどうかが確認されます。利用可能であれば、通信の設定手順が開始されます。外部アプリケーションと通信しようとするとセキュリティ例外やその他のエラーが発生することがあるため、コードは try ブロックでラップされています (簡単にするため、対応する catch ブロックは省略されています)。

次に、ここに示すように isContainerReady() メソッドが呼び出されます。

```
private function isContainerReady():Boolean
{
  var result:Boolean = ExternalInterface.call("isReady");
  return result;
}
```

isContainerReady() メソッドは、ExternalInterface.call() メソッドを使用して次のように JavaScript 関数 isReady() を呼び出します。

```
ActionScript で呼び出される <script language="JavaScript">
<!--
// ------- プライベート変数 -------
var jsReady = false;
...
// ------- の関数 -------
// ページが初期化されて JavaScript が利用可能になっているかどうかを確認するために呼び出す
function isReady()
{
  return jsReady;
}
...
// <body> タグの onload イベントから呼び出される
function pageInit()
{
  // JavaScript の準備ができたことを記録する
  jsReady = true;
```

```
}
...
//-->
</script>
```

isReady() 関数は、単純に jsReady 変数の値を返します。この変数の初期値は false です。Web ページの onload イベントがトリガされると、変数の値が true に変化します。つまり、ページのロード前に ActionScript から isReady() 関数が呼び出されると、JavaScript から ExternalInterface.call("isReady") に false が返され、その結果 ActionScript isContainerReady() メソッドから false が返されます。ページがロードされると、JavaScript isReady() 関数から true が返されるため、ActionScript isContainerReady() メソッドからも true が返されます。

IMManager コンストラクタでは、コンテナの準備状態に応じて 2 つのことが行われます。isContainerReady() から true が返された場合は、コードから単純に setupCallbacks() メソッドが呼び出されて、JavaScript での通信の設定手順が完了します。一方、isContainerReady() から false が返された場合、手順は実質的に保留状態になります。Timer オブジェクトが作成され、次のように 100 ミリ秒間隔で timerHandler() メソッドを呼び出すよう指示されます。

```
private function timerHandler(event:TimerEvent):void
{
   // コンテナの準備ができているかどうかを確認する
   var isReady:Boolean = isContainerReady();
   if (isReady)
   {
      // コンテナの準備ができている場合は何も確認する必要がないため、
      // タイマーを停止する
      Timer(event.target).stop();
      // コンテナからの呼び出しが可能な ActionScript メソッドを
      // 設定する
      setupCallbacks();
   }
}
```

timerHandler() メソッドが呼び出されるたびに、isContainerReady() メソッドの結果が再確認されます。コンテナが初期化されると、このメソッドから true が返されます。次に、コードでは Timer を停止して setupCallbacks() メソッドを呼び出し、ブラウザの通信の設定手順を終了します。

JavaScript への ActionScript メソッドの公開

上記の例で示したように、ブラウザの準備ができたことがコードで判定されると、setupCallbacks() メソッドが呼び出されます。このメソッドでは、次のように JavaScript からの呼び出しを受信するよう ActionScript が準備されます。

```
private function setupCallbacks():void
{
  // SWF クライアント関数をコンテナに登録する
  ExternalInterface.addCallback("newMessage", newMessage);
  ExternalInterface.addCallback("getStatus", getStatus);
  // SWF の呼び出し準備ができたことをコンテナに通知する
  ExternalInterface.call("setSWFIsReady");
}
```

setCallBacks() メソッドは、ExternalInterface.addCallback() を呼び出して JavaScript から呼び出し可能な 2 つのメソッドを登録することにより、コンテナとの通信準備のタスクを終了します。このコードでは、ActionScript のメソッド名と同じ名前が第 1 パラメータになっています。JavaScript ("newMessage" および "getStatus") は、この名前によってメソッドを認識します (この場合、異なる名前を使用することに意味がないため、同じ名前を再利用しました)。最後に、ExternalInterface.call() メソッドを使用して JavaScript 関数 setSWFIsReady() が呼び出され、ActionScript 関数の登録されたコンテナが通知されます。

ActionScript からブラウザへの通信

Introvert IM アプリケーションは、コンテナページでの JavaScript 関数の呼び出しに関するさまざまな例を示しています。最も単純な場合 (setupCallbacks() メソッドの例) には、パラメータを渡したり、返される値を受け取ることなく、JavaScript 関数 setSWFIsReady() が呼び出されます。

```
ExternalInterface.call("setSWFIsReady");
```

isContainerReady() メソッドの別の例では、ActionScript から isReady() 関数が呼び出され、それに対する応答としてブール値を受け取ります。

```
var result:Boolean = ExternalInterface.call("isReady");
```

External API を使用して JavaScript 関数にパラメータを渡すこともできます。たとえば、IMManager クラスの sendMessage() メソッドについて考えてみます。このメソッドは、ユーザーが自分の " 交信パートナー " に新しいメッセージを送信するときに呼び出されます。

```
public function sendMessage(message:String):void
{
  ExternalInterface.call("newMessage", message);
}
```

ここでも、指定された JavaScript 関数が ExternalInterface.call() を使用して呼び出され、ブラウザに新しいメッセージが通知されます。また、メッセージ自身は ExternalInterface.call() の追加パラメータとして渡され、結果的に JavaScript 関数 newMessage() のパラメータとして渡されます。

JavaScript からの ActionScript コードの呼び出し

通信は双方向形式でサポートされ、Introvert IM アプリケーションも例外ではありません。Flash Player IM クライアントがメッセージ送信のために JavaScript を呼び出すだけでなく、HTML フォームも JavaScript コードを呼び出して、メッセージを SWF ファイルに送信したり、SWF ファイルに情報を問い合わせます。たとえば、接続が完了し、通信準備ができたことを SWF ファイルがコンテナに通知する場合、ブラウザが最初に行うことは、IMManager クラスの `getStatus()` メソッドを呼び出して、初期のユーザー availability ステータスを SWF IM クライアントから受信することです。これは、次のように Web ページの `updateStatus()` 関数で行われます。

```
<script language="JavaScript">
...
function updateStatus()
{
  if (swfReady)
  {
    var currentStatus = getSWF("IntrovertIMApp").getStatus();
    document.forms["imForm"].status.value = currentStatus;
  }
}
...
</script>
```

このコードでは、`swfReady` 変数の値が確認されます。この変数により、SWF ファイルのメソッドが ExternalInterface クラスに登録されたことが SWF ファイルからブラウザに通知されたかどうかが追跡されます。SWF ファイルで通信の受信準備ができている場合、その次の行 (`var currentStatus =` ...) では実際に IMManager クラスの `getStatus()` メソッドが呼び出されます。コードのこの行では、次の 3 つのことが行われます。

- `getSWF()` JavaScript 関数が呼び出され、SWF ファイルを表す JavaScript オブジェクトへの参照が返されます。`getSWF()` に渡されたパラメータによって、HTML ページ内に複数の SWF ファイルが存在する場合にどのブラウザオブジェクトが返されるかが決まります。このパラメータに渡される値は、`object` タグの `id` 属性、および SWF ファイルの埋め込みに使用される `embed` タグの `name` 属性と一致している必要があります。
- SWF ファイルへの参照を使用することで、SWF オブジェクトのメソッドであるかのように `getStatus()` メソッドが呼び出されます。この場合、関数名 "`getStatus`" が使用されるのは、`ExternalInterface.addCallback()` を使用して ActionScript 関数がこの名前で登録されるからです。
- `getStatus()` ActionScript メソッドでは値が返され、その値が `currentStatus` 変数に割り当てられて、それがさらに `status` テキストフィールドのコンテンツ (`value` プロパティ) として割り当てられます。

sendMessage() JavaScript 関数は、ActionScript 関数へのパラメータの受け渡しを示しています (sendMessage() は、ユーザーが HTML ページの [送信] ボタンを押したときに呼び出される関数です)。

```
<script language="JavaScript">
...
function sendMessage(message)
{
  if (swfReady)
  {
    ...
    getSWF("IntrovertIMApp").newMessage(message);
  }
}
...
</script>
```

newMessage() ActionScript メソッドにはパラメータが 1 つ必要なため、JavaScript コード内の newMessage() メソッドの呼び出しで JavaScript message 変数をパラメータとして使用することで、この変数が ActionScript に渡されます。

ブラウザタイプの検出

ブラウザがコンテンツにアクセスする方法はさまざまであるため、この例の getSWF() JavaScript 関数に示すように、必ず JavaScript を使用してユーザーが実行しているブラウザを検出すること、およびウィンドウまたはドキュメントオブジェクトを使用して、ブラウザ固有のシンタックスに従ってムービーにアクセスすることが重要です。

```
<script language="JavaScript">
...
function getSWF(movieName)
{
  if (navigator.appName.indexOf("Microsoft") != -1)
  {
    return window[movieName];
  }
  else
  {
    return document[movieName];
  }
}
...
</script>
```

スクリプトによってユーザーのブラウザタイプが検出されない場合、HTML コンテナの SWF ファイルを再生したときに予期しない動作が発生することがあります。

例 : ActiveX コンテナに対する External API の使用

この例は、External API を使用した、ActiveX コントロールを使用するデスクトップアプリケーションと ActionScript との間の通信を示しています。この例では、ActionScript コードおよび同じ SWF ファイルを含めて Introvert IM を再び使用しているため、ActionScript の External API の使用については説明しません。前述の例をよく理解しておくと、この例を理解しやすくなります。

この例のデスクトップアプリケーションは、Microsoft Visual Studio .NET を使用して C# で記述されています。説明の中心は、ActiveX コントロールを使用した External API の操作方法です。この例は、次のことを示しています。

- Flash Player ActiveX コントロールをホスティングしているデスクトップアプリケーションからの ActionScript 関数の呼び出し
- ActionScript からの関数呼び出しの受信、および ActiveX コンテナでのその処理
- ActiveX コンテナへのメッセージ送信に Flash Player が使用するプロキシクラスを使用した、直列化 XML フォーマットの詳細の非表示

このサンプルのアプリケーションファイルを入手するには、www.adobe.com/go/learn_programmingAS3samples_flash_jp を参照してください。Introvert IM C# ファイルは Samples/IntrovertIM_CSharp フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
AppForm.cs	C# Windows Forms インターフェイスで構成されるメインアプリケーションファイル。
bin/Debug/IntrovertIMApp.swf	アプリケーションでロードされる SWF ファイル。
ExternalInterfaceProxy/ ExternalInterfaceProxy.cs	External Interface との通信のための ActiveX コントロールのラッパーとして機能するクラス。ActionScript を呼び出し、ActionScript からの呼び出しを受信するためのメカニズムを提供します。
ExternalInterfaceProxy/ ExternalInterfaceSerializer.cs	Flash Player の XML フォーマットメッセージを .NET オブジェクトに変換するタスクを実行するクラス。
ExternalInterfaceProxy/ ExternalInterfaceEventArgs.cs	このファイルは、カスタム Delegate とイベント引数という 2 つの C# タイプ (クラス) を定義します。これは、ActionScript からの関数呼び出しをリスナーに通知するため、ExternalInterfaceProxy クラスで使用されます。

ファイル	説明
ExternalInterfaceProxy/ ExternalInterfaceCall.cs	このクラスは、関数名とパラメータのプロパティを使用して、ActionScript から ActiveX コンテナへの関数呼び出しを表す値オブジェクトです。
bin/Debug/IntrovertIMApp.swf	アプリケーションでロードされる SWF ファイル。
obj/AxInterop.ShockwaveFlashObjects.dll、 obj/Interop.ShockwaveFlashObjects.dll	管理コードから Flash Player (Adobe Shockwave® Flash) ActiveX コントロールにアクセスする際に必要な Visual Studio .NET によって作成されたラッパーアセンブリ。

Introvert IM C# アプリケーションの概要

このサンプルアプリケーションは、互いに通信する 2 つのインスタントメッセージクライアントプログラム (1 つは SWF ファイル内、もう 1 つは Windows Forms に埋め込み) を示しています。ユーザーインターフェイスには Shockwave Flash ActiveX コントロールのインスタンスが含まれ、ActionScript IM クライアントが格納された SWF ファイルがこのインスタンスにロードされます。このインターフェイスには、Windows Forms IM クライアントを構成するいくつかのテキストフィールドも含まれています。1 つはメッセージ入力用フィールド (`MessageText`) で、もう 1 つはクライアント間で送信されるメッセージの複製 (`Transcript`) を表示し、第 3 のフィールド (`Status`) は SWF IM クライアントで設定された availability ステータスを表示します。

Shockwave Flash ActiveX コントロールの追加

Shockwave Flash ActiveX コントロールを独自の Windows Forms アプリケーションに含めるには、最初にそれを Microsoft Visual Studio Toolbox に追加する必要があります。

コントロールをツールボックスに追加するには :

1. Visual Studio Toolbox を開きます。
2. Visual Studio 2003 の [Windows フォーム] セクション、または Visual Studio 2005 の任意のセクションを右クリックします。Visual Studio 2003 ではコンテキストメニューで [項目の追加 / 削除] を選択します。Visual Studio 2005 では [項目の選択] を選択します。

   これにより、[ツールボックスのカスタマイズ] (2003) または [ツールボックス項目の選択] (2005) ダイアログボックスが表示されます。
3. [COM コンポーネント] タブを選択します。このタブには、Flash Player ActiveX コントロールなど、そのコンピュータにあるすべての COM コンポーネントが表示されます。
4. スクロールして Shockwave Flash Object を探し、選択します。

   この項目が表示されない場合は、システムに Flash Player ActiveX がインストールされているか確認してください。

ActionScript から ActiveX コンテナへの通信について

External API を使用した ActiveX コンテナアプリケーションとの通信は、Web ブラウザとの通信と同様になりますが、大きな違いが 1 つあります。前述したように、ActionScript が Web ブラウザと通信する場合、開発者が考慮している限り、関数は直接呼び出されます。関数の呼び出し方法と応答方法の詳細は、プレイヤー間で受け渡しできるようフォーマットされ、ブラウザは認識されません。ただし、ActiveX コンテナアプリケーションとの通信に External API が使用される場合、Flash Player はメッセージ (関数呼び出しおよび戻り値) をある XML フォーマットでアプリケーションに送信し、コンテナアプリケーションからの関数呼び出しと戻り値に同じ XML フォーマットが使用される必要があります。ActiveX コンテナアプリケーションの開発者は、適切なフォーマットの関数呼び出しと応答を認識し、作成可能なコードを作成する必要があります。

サンプルの Introvert IM C# には、メッセージのフォーマッティング防止の可能なクラスが含まれていますが、ActionScript 関数の呼び出しおよび ActionScript からの呼び出しの受信時は、標準のデータ型を使用して操作することができます。ExternalInterfaceProxy クラスを他のヘルパークラスと組み合わせるとこの機能が提供され、任意の .NET プロジェクトで再利用することで External API との通信が容易になります。

次のコードは、メインアプリケーションフォーム (AppForm.cs) から引用したもので、ExternalInterfaceProxy クラスを使用して実現された単純な操作を示しています。

```
public class AppForm : System.Windows.Forms.Form
{
   ...
   private ExternalInterfaceProxy proxy;
   ...
   public AppForm()
   {
      ...
      // このアプリケーションを登録することにより、プロキシが ActionScript からの
      // 呼び出しを受信したときに通知を受信する
      proxy = new ExternalInterfaceProxy(IntrovertIMApp);
      proxy.ExternalInterfaceCall += new
   ExternalInterfaceCallEventHandler(proxy_ExternalInterfaceCall);
      ...
   }
   ...
```

このアプリケーションは、`proxy` という名前の ExternalInterfaceProxy インスタンスを宣言して作成し、参照をユーザーインターフェイスの Shockwave Flash ActiveX コントロールに渡します (`IntrovertIMApp`)。次に、プロキシの `ExternalInterfaceCall` イベントを受信するための `proxy_ExternalInterfaceCall()` メソッドが、コードで登録されます。このイベントは、Flash Player から関数呼び出しがあったときに ExternalInterfaceProxy クラスによって送出されます。このイベントへのサブスクライブは、C# コードが ActionScript からの関数呼び出しを受信し、応答する方法になっています。

ActionScript から関数呼び出しがあると、ExternalInterfaceProxy インスタンス (proxy) はその呼び出しを受信し、それを XML フォーマットに変換します。そして、プロキシの ExternalInterfaceCall イベントのリスナーとなっているオブジェクトに通知します。AppForm クラスの場合、proxy_ExternalInterfaceCall() メソッドでは次のようにイベントが処理されます。

```
/// <summary>
/// SWF で ActionScript ExternalInterface 呼び出しが行われると、
/// プロキシから呼び出される
/// </summary>
private object proxy_ExternalInterfaceCall(object sender,
ExternalInterfaceCallEventArgs e)
{
  switch (e.FunctionCall.FunctionName)
  {
    case "isReady":
      return isReady();
    case "setSWFIsReady":
      setSWFIsReady();
      return null;
    case "newMessage":
      newMessage((string)e.FunctionCall.Arguments[0]);
      return null;
    case "statusChange":
      statusChange();
      return null;
    default:
      return null;
  }
}
...
```

このメソッドは、ExternalInterfaceCallEventArgs インスタンスに渡されます。この例のインスタンス名は e です。これにより、このオブジェクトが ExternalInterfaceCall クラスのインスタンスである FunctionCall プロパティを取得します。

ExternalInterfaceCall インスタンスが、2 つのプロパティを持つ単純な値オブジェクトです。FunctionName プロパティには、ActionScript ExternalInterface.Call() ステートメントで指定された関数名が格納されています。ActionScript に何らかのパラメータが追加されると、そのパラメータは ExternalInterfaceCall オブジェクトの Arguments プロパティに格納されます。この場合、イベントを処理するメソッドは、トラフィックマネージャのように機能する switch ステートメントです。FunctionName プロパティの値 (e.FunctionCall.FunctionName) によって、AppForm クラスのどのメソッドが呼び出されるかが決まります。

前述のコードの switch ステートメントのブランチは、一般的なメソッド呼び出しシナリオを示しています。たとえば、すべてのメソッドは値を ActionScript に返すか (isReady() メソッド呼び出しなど)、null を返す (他のメソッド呼び出しの場合) 必要があります。ActionScript から渡されたパラメータへのアクセスは、newMessage() メソッド呼び出しに示されています (このメソッド呼び出しにより、Arguments 配列の第 1 エレメントであるパラメータ e.FunctionCall.Arguments[0] が渡されます)。

ExternalInterfaceProxy クラスを使用した C# からの ActionScript 関数の呼び出しは、ActionScript からの関数呼び出しの受信よりもはるかに単純です。ActionScript 関数を呼び出すには、次のように ExternalInterfaceProxy インスタンスの Call() メソッドを使用します。

```
   /// <summary>
   /// [ 送信 ] ボタンが押されると呼び出され、
   /// MessageText テキストフィールドの値がパラメータとして渡される
   /// </summary>
   /// <param name="message">送信するメッセージ </param>
   private void sendMessage(string message)
   {
      if (swfReady)
      {
         ...
         // ActionScript の newMessage 関数を呼び出す
         proxy.Call("newMessage", message);
      }
   }
   ...
   /// <summary>
   /// ActionScript 関数を呼び出して現在の "availability" ステータスを
   /// 取得し、それをテキストフィールドに書き込む
   /// </summary>
   private void updateStatus()
   {
      Status.Text = (string)proxy.Call("getStatus");
   }
   ...
}
```

この例で示されているように、ExternalInterfaceProxy クラスの Call() メソッドは、ActionScript の ExternalInterface.Call() メソッドと非常によく似ています。第 1 パラメータはストリングで、呼び出す関数の名前になります。その他のパラメータ (ここには示されていません) も ActionScript 関数に渡されます。ActionScript 関数から値が返された場合、その値は Call() メソッドで返されます (前述の例)。

ExternalInterfaceProxy クラスの内部

ActiveX コントロールのプロキシラッパーを使用することは必ずしも重要ではなく、独自のプロキシクラス (たとえば、別のプログラミング言語や別のプラットフォーム用) を作成することもできます。プロキシの作成についてはここで詳しく説明しませんが、この例におけるプロキシクラスの内部処理を理解することは有益です。

Shockwave Flash ActiveX コントロールの `CallFunction()` メソッドを使用すると、External API を使用して ActiveX コンテナから ActionScript 関数を呼び出すことができます。次の ExternalInterfaceProxy クラスの `Call()` メソッドの一部は、これを示しています。

```
// Shockwave Flash ActiveX コントロールの "_flashControl" で、
// SWF の ActionScript 関数を呼び出す
string response = _flashControl.CallFunction(request);
```

このコード部分で、`_flashControl` は Shockwave Flash ActiveX コントロールです。ActionScript 関数呼び出しは、`CallFunction()` メソッドを使用して行われます。このメソッドにはパラメータが 1 つあり (この例では `request`)、このパラメータは XML フォーマットされた命令の含まれたストリングです。呼び出される ActionScript 関数の名前および任意のパラメータが含まれています。ActionScript から返されたすべての値は XML フォーマットストリングにエンコードされ、`CallFunction()` 呼び出しの戻り値として送り返されます。この例では、この XML ストリングが `response` 変数に格納されます。

ActionScript からの関数呼び出しの受信は、段階的な手順で行われます。ActionScript から関数呼び出しがあると、Shockwave Flash ActiveX コントロールは FlashCall イベントを送出するため、SWF ファイルからの呼び出しを受信するためのクラス (ExternalInterfaceProxy クラスなど) ではそのイベント用のハンドラを定義しておく必要があります。ExternalInterfaceProxy クラスでは、このイベントハンドラ関数の名前は `_flashControl_FlashCall()` で、クラスコンストラクタ内のイベントを待ち受けるよう登録されています。

```
private AxShockwaveFlash _flashControl;

public ExternalInterfaceProxy(AxShockwaveFlash flashControl)
{
  _flashControl = flashControl;
  _flashControl.FlashCall += new
  _IShockwaveFlashEvents_FlashCallEventHandler(_flashControl_FlashCall);
}
...
private void _flashControl_FlashCall(object sender,
  _IShockwaveFlashEvents_FlashCallEvent e)
{
  // イベントオブジェクトの request プロパティ ("e.request") を使用して
  // アクションを実行する
  ...
  // 値を ActionScript に返す
  // 返された値は、最初に XML フォーマットストリングとしてエンコードする
  _flashControl.SetReturnValue(encodedResponse);
}
```

イベントオブジェクト (e) には `request` プロパティ (`e.request`) があります。このプロパティは、関数名やパラメータなど、関数呼び出しに関する情報が XML フォーマットで格納された文字列になっています。この情報をコンテナで使用すると、実行するコードを判別できます。ExternalInterfaceProxy クラスでは、要求が XML フォーマットから ExternalInterfaceCall オブジェクトに変換され、同じ情報がアクセスしやすい形式で提供されます。ActiveX コントロールの `SetReturnValue()` メソッドを使用すると、関数の結果を ActionScript の呼び出し側に返すことができます。この場合も、得られたパラメータを External API で使用される XML フォーマットにエンコードする必要があります。

Shockwave Flash ActiveX コントロールをホストするアプリケーションと ActionScript 間の通信では、関数呼び出しと値をエンコードする際に専用の XML フォーマットが使用されます。サンプルの Introvert IM C# では、ExternalInterfaceProxy クラスによって、アプリケーションフォーム内のコードで、Flash Player で使用する XML フォーマットの詳細を無視して、ActionScript との間で受け渡す値を直接操作できるようになっています。これを実現するため、ExternalInterfaceProxy クラスでは ExternalInterfaceSerializer クラスのメソッドを使用して、実際に XML メッセージを .NET オブジェクトに変換しています。ExternalInterfaceSerializer クラスには、次の 4 つのパブリックメソッドがあります。

- `EncodeInvoke()`: 引数の関数名と C# ArrayList を適切な XML フォーマットにエンコードします。
- `EncodeResult()`: 結果値を適切な XML フォーマットにエンコードします。
- `DecodeInvoke()`: ActionScript からの関数呼び出しをデコードします。FlashCall イベントオブジェクトの `request` プロパティは `DecodeInvoke()` メソッドに渡され、このメソッドで呼び出しが ExternalInterfaceCall オブジェクトに変換されます。
- `DecodeResult()`: ActionScript 関数の呼び出しの結果として受信した XML をデコードします。

これらのメソッドは、C# の値を External API の XML フォーマットにエンコードし、XML を C# オブジェクトにデコードします。Flash Player で使用する XML フォーマットの詳細については、703 ページの「External API の XML フォーマット」を参照してください。

第 26 章

Flash Player セキュリティ

26

セキュリティは、Adobe、ユーザー、Web サイト所有者、およびコンテンツ開発者にとって大きな問題です。このため、Adobe Flash Player 9 には、ユーザー、Web サイト所有者、およびコンテンツ開発者を保護するための一連のセキュリティルールとコントロールが用意されています。この章では、Flash アプリケーションを開発する場合の Flash Player セキュリティモデルの操作方法について解説します。この章で取り上げる SWF ファイルは、特に説明がない限り、ActionScript 3.0 でパブリッシュされ、したがって Flash Player 9 以降で実行されるものとします。

この章の目的はセキュリティの概要を説明することで、あらゆる実装の詳細、使用方法、特定の API を使用する際の問題などについて包括的な説明をするものではありません。Flash Player セキュリティの概念の詳細については、Flash Player 9 セキュリティに関するホワイトペーパー (www.adobe.com/go/fp9_0_security_jp) を参照してください。

目次

Flash Player セキュリティの概要 720
アクセス許可管理の概要 722
セキュリティサンドボックス 732
ネットワーク API の制限 735
フルスクリーンモードのセキュリティ 737
コンテンツのロード 738
クロススクリプト 742
ロードされたメディアへのデータとしてのアクセス 746
データのロード 749
セキュリティドメインに読み込まれた SWF ファイルからの埋め込みコンテンツのロード 752
古いコンテンツの操作 752
LocalConnection 許可の設定 753
ホスト Web ページのスクリプトへのアクセスの制御 754
共有オブジェクト 755
カメラ、マイク、クリップボード、マウス、キーボードのアクセス 757

Flash Player セキュリティの概要

Flash Player セキュリティの大部分は、ロードされた SWF ファイル、メディア、およびその他のアセットの元のドメインに基づいています。www.example.com などの特定のインターネットドメインの SWF ファイルは、常にそのドメインのすべてのデータにアクセスできます。これらのアセットは " セキュリティサンドボックス " と呼ばれる同じセキュリティグループに配置されます (詳細については、732 ページの「セキュリティサンドボックス」を参照してください)。

たとえば、SWF ファイルは、ドメイン内の SWF ファイル、ビットマップ、オーディオ、テキストファイル、およびその他のアセットをダウンロードできます。また、同じドメインの 2 つの SWF 間のクロススクリプトも、両方のファイルが ActionScript 3.0 を使用して記述されていれば可能です。" クロススクリプト " は、SWF ファイルが ActionScript を使用して別の SWF ファイルのプロパティ、メソッド、およびオブジェクトにアクセスする機能です。しかし、ActionScript 3.0 で記述された SWF ファイルと旧バージョンの ActionScript で記述された SWF ファイルではクロススクリプトはサポートされません。この場合は、LocalConnection クラスを使用して通信できます。詳細については、742 ページの「クロススクリプト」を参照してください。

デフォルトでは、次の基本的なセキュリティルールが常に適用されます。

- 同じセキュリティサンドボックス内のリソースは、常に互いにアクセスできます。
- リモートサンドボックス内の SWF ファイルは、ローカルファイルおよびデータにはアクセスできません。

Flash Player は、次のサイトを個々のドメインと見なし、それぞれについてセキュリティサンドボックスを個別に設定します。

- `http://example.com`
- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`
- `http://192.0.34.166`

http://example.com などの名前付きドメインが http://192.0.34.166 などの特定の IP アドレスに対応している場合も、Flash Player は両方に対して別々のセキュリティサンドボックスを設定します。

開発者が SWF ファイルにそれ自体のサンドボックス以外のサンドボックスのアセットへのアクセスを許可するために使用できる基本的な方法には、次の 2 つがあります。

- `Security.allowDomain()` メソッド (731 ページの「作成者 (開発者) コントロール」を参照)
- クロスドメインポリシーファイル (726 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」を参照)

デフォルトでは、SWF ファイルが他のドメインの ActionScript 3.0 SWF ファイルをクロススクリプトし、他のドメインのデータをロードする機能は使用できません。この機能は、ロードされた SWF ファイルの `Security.allowDomain()` メソッドへの呼び出しを使用して許可できます。詳細については、742 ページの「クロススクリプト」を参照してください。

Flash Player セキュリティモデルでは、コンテンツのロードとデータへのアクセスまたはロードは区別されています。

- コンテンツのロード - コンテンツは Flash Player が再生できるビジュアルメディアを含むメディア、オーディオ、ビデオ、または表示メディアを含む SWF ファイルと定義されます。" データ " は、ActionScript コードからのみアクセスできるものとして定義されます。コンテンツは、Loader クラス、Sound クラス、NetStream クラスなどのクラスを使用してロードできます。
- データとしてのコンテンツへのアクセスまたはデータのロード - データには 2 とおりの方法でアクセスできます。ロードしたメディアコンテンツからデータを抽出する方法と、XML ファイルなどの外部ファイルからデータを直接ロードする方法です。ロードしたメディアからデータを抽出するには、Bitmap オブジェクト、`BitmapData.draw()` メソッド、`Sound.id3` プロパティ、または `SoundMixer.computeSpectrum()` メソッドを使用します。データをロードするには、たとえば URLStream、URLLoader、Socket、XMLSocket などのクラスを使用します。

Flash Player セキュリティモデルは、コンテンツのロードとデータへのアクセスのために、さまざまな規則を定義しています。一般に、コンテンツのロードには、データへのアクセスより制限が少なくなっています。

一般に、コンテンツ (SWF ファイル、ビットマップ、MP3 ファイル、およびビデオ) はどこからでもロードできますが、コンテンツがロードする側の SWF ファイルのドメイン以外のドメインにある場合は、別々のセキュリティサンドボックスに分割されます。

コンテンツのロードには、次のような制限があります。

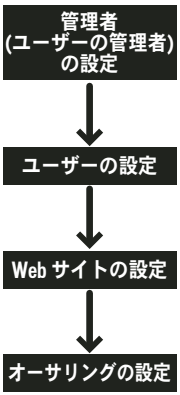
- デフォルトでは、ローカルの SWF ファイル (ユーザーのハードディスクなどの非ネットワークアドレスからロードされた SWF ファイル) は、local-with-filesystem システムサンドボックスに分類されます。これらのファイルでは、ネットワークからコンテンツをロードできません。詳細については、733 ページの「ローカルサンドボックス」を参照してください。
- RTMP (Real-Time Messaging Protocol) サーバーでは、コンテンツへのアクセスを制限することができます。詳細については、742 ページの「RTMP サーバーを使用して送信されたコンテンツ」を参照してください。

ロードされたメディアがイメージ、オーディオ、またはビデオの場合、セキュリティサンドボックス外部にある SWF ファイルのドメインがメディアの元のドメインにあるクロスドメインポリシーファイルに含まれていなければ、ピクセルデータやサウンドデータなどのメディアのデータにその SWF ファイルからアクセスすることができません。詳細については、746 ページの「ロードされたメディアへのデータとしてのアクセス」を参照してください。

その他のロードされたデータの形態としては、テキストファイルまたは XML ファイルがあり、これらは URLLoader オブジェクトを使用してロードされます。この場合も、別のセキュリティサンドボックスからのデータにアクセスするには、元のドメインでのクロスドメインポリシーファイルによって許可が与えられている必要があります。詳細については、749 ページの「URLLoader と URLStream の使用」を参照してください。

アクセス許可管理の概要

Flash Player クライアントランタイムのセキュリティモデルは、SWF ファイル、ローカルデータ、インターネット URL などのオブジェクトであるリソースに基づいて設計されています。ステークホルダーは、これらのリソースの所有者または使用者です。ステークホルダーはそのリソースを管理 (セキュリティ設定) できます。また、各リソースにはステークホルダーが 4 つあります。次の図に示すように、Flash Player ではこれらの管理の権限階層が厳密に適用されます。



セキュリティ管理の階層

これは、たとえば、管理者がリソースへのアクセスを制限した場合、他のステークホルダーはこの制限をオーバーライドすることはできないことを示します。

管理者、ユーザー、および Web サイトの管理については、以降のセクションで詳しく説明します。この章では、作成者 (開発者) 設定についても説明します。

管理ユーザー制御

コンピュータの管理ユーザー (管理者権限でログインするユーザー) は、そのコンピュータの全ユーザーに有効な Flash Player セキュリティ設定を適用できます。家庭用コンピュータなどのエンタープライズ環境以外では、通常は管理者権限も持つユーザーは 1 人です。エンタープライズ環境でも、個々のユーザーが管理者権限を持つ場合があります。

管理ユーザーの管理には次の 2 種類があります。

- mms.cfg ファイル
- Global Flash Player Trust ディレクトリ

mms.cfg ファイル

Mac OS X システムでは、mms.cfg ファイルの場所は /Library/Application Support/Macromedia/mms.cfg です。Microsoft Windows システムでは、ファイルはシステムディレクトリの Macromedia Flash Player フォルダにあります (たとえば、Windows XP のデフォルトのインストールでは C:¥windows¥system32¥macromed¥flash¥mms.cfg です)。

Flash Player が起動すると、このファイルからセキュリティ設定を読み取り、機能を制限するために使用します。

mms.cfg ファイルには、管理者が次のタスクを実行するために使用する設定が含まれます。

- **データのロード** - ローカル SWF ファイルの読み取りを制限し、ファイルのダウンロードとアップロードを無効にし、永続共有オブジェクトに記憶域制限を設定します。
- **プライバシーコントロール** - マイクとカメラアクセスを無効にし、SWF ファイルによる表示コンテンツの再生を禁止し、ブラウザウィンドウに表示される URL に一致しないドメインの SWF ファイルによる永続共有オブジェクトへのアクセスを禁止します。
- **Flash Player の更新** - 最新バージョンの Flash Player を確認する間隔を設定し、Flash Player の更新情報をチェックする URL を指定し、Flash Player の最新バージョンをダウンロードする URL を指定し、Flash Player の自動更新を完全に無効にします。
- **旧ファイルのサポート** - 古いバージョンの SWF ファイルを local-trusted サンドボックスに配置するかどうかを指定します。
- **ローカルファイルのセキュリティ** - ローカルファイルを local-trusted サンドボックスに配置できるかどうかを指定します。
- **フルスクリーンモード** - フルスクリーンモードを無効にします。

SWF ファイルは、`Capabilities.avHardwareDisable` および `Capabilities.localFileReadDisable` プロパティを呼び出すことにより、無効にされている機能に関する一部の情報にアクセスできます。ただし、mms.cfg ファイルのほとんどの設定は、AcrionScript から照会できません。

コンピュータのセキュリティとプライバシーに関するセキュリティを、アプリケーションから独立して適用するには、システム管理者のみが mms.cfg ファイルを変更する必要があります。mms.cfg ファイルは、アプリケーションインストーラが使用するためのものではありません。管理権限を持って実行しているインストーラは mms.cfg ファイルの内容を変更できますが、アドビ システムズ社では、そのような使用状況はユーザーの信頼を裏切るものであると考えており、インストーラの作成者に "mms.cfg" ファイルを決して変更しないように強く要請しています。

Global Flash Player Trust ディレクトリ

管理ユーザーおよびインストーラアプリケーションは、指定されたローカル SWF ファイルを信頼できるファイルとして登録できます。これらの SWF ファイルは、local-trusted サンドボックスに割り当てられます。割り当てられたファイルは、他の SWF ファイルと自由に通信し、リモートまたはローカルを問わずどこからでもデータをロードできます。ファイルは、Global Flash Player Trust ディレクトリで信頼できるファイルとして指定されます。このディレクトリは mms.cfg ファイルと同じディレクトリ内にあり、現在のユーザーに固有の次の場所にあります。

- Windows: system\Macromed\Flash\FlashPlayerTrust

  (例 : C:¥windows¥system32¥Macromed¥Flash¥FlashPlayerTrust)
- Mac: app support/Macromedia/FlashPlayerTrust

  (例 : /Library/Application Support/Macromedia/FlashPlayerTrust)

Flash Player Trust ディレクトリには、任意の数のテキストファイルを格納でき、それぞれのファイルは、信頼できるパスを 1 行に 1 パスずつ示したリストです。1 つのパスを 1 つの SWF ファイル、HTML ファイル、またはディレクトリとすることもできます。コメント行は、# 記号で始まります。たとえば、次のテキストが入っている Flash Player トラスト構成ファイルは、指定されたディレクトリとそのすべてのサブディレクトリにあるすべてのファイルに、信頼されている状態を付与します。

```
# Trust files in the following directories:
C:¥Documents and Settings¥All Users¥Documents¥SampleApp
```

トラスト構成ファイル内に示されるパスは、常にローカルパスであるか SMB ネットワークパスであることが必要です。トラスト構成ファイル内の HTTP パスは、すべて無視されます。信頼できるのは、ローカルファイルだけです。

競合を避けるため、それぞれのトラスト構成ファイルにはインストールするアプリケーションに対応したファイル名を付け、.cfg のファイル拡張子を使用してください。

ローカルで実行する SWF ファイルをインストーラアプリケーションから配布する開発者は、インストーラアプリケーションに Global Flash Player Trust ディレクトリへ構成ファイルを追加させて、配布するファイルに対するすべての権限を付与することができます。インストーラアプリケーションは、管理者権限のあるユーザーが実行する必要があります。mms.cfg ファイルと異なり、Global Flash Player Trust ディレクトリはトラスト許可を付与するインストーラアプリケーションのために組み込まれています。管理ユーザーとインストーラアプリケーションは、どちらも Global Flash Player Trust ディレクトリを使用して、信頼できるローカルアプリケーションを指定できます。

また個々のユーザーの Flash Player Trust ディレクトリも使用できます (次のセクション「ユーザーの管理」を参照してください)。

ユーザーの管理

Flash Player には、権限を設定するためのユーザーレベルのメカニズムが 3 つあります。つまり、設定 UI、設定マネージャ、および User Flash Player Trust ディレクトリです。

設定 UI および設定マネージャ

設定 UI は、特定のドメインの設定をすばやくインタラクティブに行うメカニズムです。設定マネージャは、詳細なインターフェイスを提供し、多数またはすべてのドメインのアクセス許可を対象とするグローバルな変更を行う機能を備えています。また、SWF ファイルによって新しいアクセス許可が要求され、セキュリティやプライバシーに関する実行時決定が必要なとき、ユーザーが Flash Player 設定を調整できるダイアログボックスが表示されます。

設定マネージャおよび設定 UI には、次のセキュリティ関連のオプションがあります。

- カメラとマイクの設定 - ユーザーは Flash Player からコンピュータ上のカメラとマイクへのアクセスを制御できます。ユーザーは、すべてのサイトまたは特定のサイトのアクセスを許可または拒否できます。すべてのサイトまたは特定のサイトの設定を指定しなかった場合、SWF ファイルがカメラまたはマイクにアクセスしようとするとダイアログボックスが表示され、ユーザーは SWF ファイルのデバイスへのアクセスを許可するかどうかを選択できます。ユーザーは、使用するカメラまたはマイクを指定することも可能で、マイクの感度を設定できます。
- 共有オブジェクトの記憶域の設定 - ユーザーは永続共有オブジェクトの格納にドメインで使用できるディスク容量を選択できます。任意の数のドメインについてこれらの設定を行うことができ、新しいドメインのデフォルト設定を指定できます。デフォルトのディスク容量は 100 KB です。永続共有オブジェクトの詳細は、ActionScript 3.0 コンポーネントリファレンスガイドの SharedObject クラスを参照してください。

メモ

mms.cfg ファイルで行われる設定は (723 ページの「管理ユーザー制御」を参照)、設定マネージャに反映されません。

設定マネージャの詳細については、www.adobe.com/go/settingsmanager_jp を参照してください。

User Flash Player Trust ディレクトリ

ユーザーおよびインストーラアプリケーションは、指定されたローカル SWF ファイルを信頼できるファイルとして登録できます。これらの SWF ファイルは、local-trusted サンドボックスに割り当てられます。割り当てられたファイルは、他の SWF ファイルと自由に通信し、リモートまたはローカルを問わずどこからでもデータをロードできます。ユーザーは、User Flash Player Trust ディレクトリでファイルを信頼できるファイルに指定します。このディレクトリは、Flash 共有オブジェクトの記憶領域と同じディレクトリ内にあり、現在のユーザーに固有の次の場所にあります。

- Windows: app data¥Macromedia¥Flash Player¥#Security¥FlashPlayerTrust

  (例 : C:¥Documents and Settings¥JohnD¥Application Data¥Macromedia¥Flash Player¥#Security¥FlashPlayerTrust)

- Mac: app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (例 : /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

この設定は、現在のユーザーだけに影響を及ぼし、コンピュータにログインする他のユーザーには影響を及ぼしません。管理権限のないユーザーがシステムの個人領域にアプリケーションをインストールする場合、インストーラは User Flash Player Trust ディレクトリに、アプリケーションをそのユーザーの信頼できるアプリケーションとして登録できます。

ローカルで実行する SWF ファイルをインストーラアプリケーションによって配布する開発者は、インストーラアプリケーションに User Flash Player Trust ディレクトリへ構成ファイルを追加させて、配布するファイルに対するすべての権限を付与することができます。この場合でも、User Flash Player Trust ディレクトリファイルは、管理ユーザーのアクション (インストール) によって開始されるため、管理ユーザーの管理と見なされます。

また、管理ユーザーまたはインストーラが、コンピュータの全ユーザーのアプリケーションの登録先に使用する Global Flash Player Trust ディレクトリもあります (723 ページの「管理ユーザー制御」を参照)。

Web サイトの管理 (クロスドメインポリシーファイル)

Web サーバーにあるデータを他のドメインの SWF ファイルから利用できるようにするために、サーバー上にクロスドメインポリシーファイルを作成できます。クロスドメインポリシーファイルは XML ファイルであり、サーバーのデータとドキュメントをどのドメインの SWF ファイルが利用できるかを示すものです。特定のドメインのみを指定することも、すべてのドメインを指定することもできます。サーバーのポリシーファイルに指定されたドメインが提供する SWF ファイルであれば、サーバーのデータやアセットにアクセスすることができます。

クロスドメインポリシーファイルは、次のようなアセットへのアクセスに影響を及ぼします。

- ビットマップ、サウンド、およびビデオのデータ
- XML ファイルとテキストファイルのロード
- ソケットと XML ソケット接続へのアクセス
- 他のセキュリティドメインから、ロードする SWF ファイルのセキュリティドメインへの SWF ファイルの読み込み

詳細は、この章の残りの部分で説明します。

ポリシーファイルのシンタックス

次の例は、`*.example.com`, `www.friendOfExample.com` および `192.0.34.166` に置かれている SWF ファイルへのアクセスを許可するポリシーファイルを示しています。

```
<?xml version="1.0"?>
<cross-domain-policy>
  <allow-access-from domain="*.example.com" />
  <allow-access-from domain="www.friendOfExample.com" />
  <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

SWF ファイルが別のドメインからデータにアクセスしようとすると、Flash Player は自動的にそのドメインからポリシーファイルのロードを試みます。データをアクセスしようとしている SWF ファイルのドメインがポリシーファイルに含まれている場合、データへのアクセスが自動的に許可されます。

デフォルトでは、ポリシーファイルの名前は `crossdomain.xml` にし、ファイルはサーバーのルートディレクトリに置く必要があります。しかし、SWF ファイルは、`Security.loadPolicyFile()` メソッドを呼び出して、別の名前または別のディレクトリ内をチェックできます。クロスドメインポリシーファイルは、ロード元のディレクトリとその子ディレクトリにのみ、適用されます。このため、ルートディレクトリのポリシーファイルは、サーバー全体に適用されますが、任意のサブディレクトリからロードされたポリシーファイルは、そのディレクトリおよびサブディレクトリにのみ適用されます。

ポリシーファイルは、そのファイルが存在する特定のサーバーへのアクセスにのみ影響を及ぼします。たとえば、https://www.adobe.com:8080/crossdomain.xml にあるポリシーファイルは、HTTPS 経由でポート 8080 から www.adobe.com に対して行われるデータロード呼び出しにのみ適用されます。

クロスドメインポリシーファイルには、<cross-domain-policy> タグが 1 つ含まれています。このタグには、<allow-access-from> タグが含まれている場合もあれば含まれていない場合もあります。各 <allow-access-from> タグには、domain という属性が 1 つ含まれています。この属性は、正確な IP アドレス、正確なドメイン、ワイルドカードドメイン (任意のドメイン) のいずれかを指定します。ワイルドカードドメインを指定するには、アスタリスク (*) 1 つを使用するか、アスタリスクの後に接尾辞を続けます。前者はすべてのドメインとすべての IP アドレスを意味し、後者は指定の接尾辞で終わるドメインのみを意味します。接尾辞はドットで始める必要があります。ただし、この先頭のドットは検索には使用されません。ドットを除いた接尾辞に一致するドメインだけが検索されます。たとえば、foo.com は *.foo.com の一部と見なされます。IP ドメインの指定にはワイルドカードを使用できません。

IP アドレスを指定した場合は、IP シンタックス (http://65.57.83.12/flashmovie.swf など) を使用してその IP アドレスからロードされた SWF ファイルにのみアクセスが付与されます。ドメイン名シンタックスを使用してロードされた SWF ファイルには付与されません。Flash Player は、DNS 解決を行いません。

次のように、すべてのドメインのドキュメントに対してアクセスを許可できます。

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

各 <allow-access-from> タグには、オプションの secure 属性もあり、デフォルトは true です。ポリシーファイルが HTTPS サーバー上に置かれているとき、非 HTTPS サーバー上の SWF ファイルによって、HTTPS サーバーからデータをロードするには、この属性を false に設定します。

secure 属性を false に設定すると、HTTPS のセキュリティが影響を受けます。特に、この属性を false に設定すると、セキュアなコンテンツがスヌープ攻撃やスプーフィング攻撃にさらされる可能性があります。secure 属性を false に設定しないことを強くお勧めします。

ロードするデータが HTTPS サーバー上にあり、それをロードする SWF ファイルが HTTP サーバー上にある場合は、セキュアなデータのすべてのコピーを HTTPS の保護下に置いておくことができるよう、ロードする SWF ファイルを HTTPS サーバーへ移動してください。しかし、ロードする SWF ファイルを HTTP サーバー上に置いておく必要があると判断した場合は、次のコードに示すように、secure="false" 属性を <allow-access-from> タグに追加します。

```
<allow-access-from domain="www.example.com" secure="false" />
```

<allow-access-from> タグを含まないポリシーファイルは、サーバーにポリシーが設定されていない場合と同じ効果があります。

ソケットポリシーファイル

ActionScript オブジェクトは、2 種類のサーバー接続をインスタンス化します。ドキュメントベースのサーバー接続とソケット接続です。Loader、Sound、URLLoader、URLStream のような ActionScript オブジェクトは、ドキュメントベースのサーバー接続をインスタンス化し、URL からファイルをロードします。ActionScript の Socket オブジェクトと XMLSocket オブジェクトはソケット接続を作成し、ロードされたドキュメントでなくストリーミングデータを処理します。Flash Player は、2 種類のポリシーファイルをサポートします。ドキュメントベースのポリシーファイルとソケットポリシーファイルです。ドキュメントベースの接続にはドキュメントベースのポリシーファイルが必要であり、ソケット接続にはソケットポリシーファイルが必要です。

Flash Player では、試みられている接続で使用するプロトコルと同じ種類のプロトコルを使用してポリシーファイルが伝送される必要があります。たとえば、ポリシーファイルを HTTP サーバーに置いた場合、他のドメインの SWF ファイルは、その HTTP サーバーからデータをロードできます。しかし、同じサーバーでソケットポリシーファイルを提供しなければ、他のドメインの SWF ファイルはそのサーバーにソケットレベルで接続できなくなります。ソケットポリシーファイルを取得する手段は、接続の手段に一致する必要があります。

ソケットサーバーで提供するポリシーファイルのシンタックスは、他のポリシーファイルとほぼ同じですが、アクセスを許可するポートも指定する必要がある点が異なります。ポリシーファイルが 1024 未満のポート番号から提供される場合、任意のポートへのアクセスが許可されます。ポリシーファイルがポート 1024 以上のポートから提供される場合、1024 以上のポートへのアクセスのみが許可されます。許可するポートは、`<allow-access-from>` タグの `to-ports` 属性で指定します。指定できる値は、単一のポート番号、ポート範囲、およびワイルドカードです。

次に、XMLSocket ポリシーファイルの例を示します。

```
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="507" />
   <allow-access-from domain="*.example.com" to-ports="507,516" />
   <allow-access-from domain="*.example2.com" to-ports="516-523" />
   <allow-access-from domain="www.example2.com" to-ports="507,516-523" />
   <allow-access-from domain="www.example3.com" to-ports="*" />
</cross-domain-policy>
```

ポリシーファイルが最初に Flash Player 6 に導入されたとき、ソケットポリシーファイルのサポートはありませんでした。ソケットサーバーへの接続は、ソケットサーバーと同じホストのポート 80 上にある HTTP サーバーのクロスドメインポリシーファイルのデフォルトの場所に置かれているポリシーファイルによって許可されていました。既存のサーバー配置を保存できるよう、Flash Player 9 では、この機能が引き続きサポートされています。しかし、現在の Flash Player のデフォルトは、ソケット接続と同じポート上のソケットポリシーファイルを取得することです。HTTP ベースのポリシーファイルを使用してソケット接続を許可する場合は、次のようなコードを使用して HTTP ポリシーファイルを明示的に要求する必要があります。

```
Security.loadPolicyFile("http://socketServerHost.com/crossdomain.xml")
```

さらに、ソケット接続を許可するためには、HTTP ポリシーファイルがクロスドメインポリシーファイルのデフォルトの場所からのみ提供され、それ以外の HTTP の場所から提供されないことが必要です。HTTP サーバーから取得されたポリシーファイルは、ポート 1024 以上のすべてのポートへのアクセスを暗黙に許可します。HTTP ポリシーファイル内に `to-ports` 属性があっても、それは無視されます。

ソケットポリシーファイルの詳細については、749 ページの「ソケットへの接続」を参照してください。

ポリシーファイルのプリロード

サーバーからのデータのロードまたはソケットへの接続は非同期操作であり、Flash Player は、単にクロスドメインポリシーファイルがダウンロードを終了するのを待ってから、主たる操作を開始します。ただし、イメージからのピクセルデータの抽出またはサウンドからのサンプルデータの抽出は、同期操作です。クロスドメインポリシーファイルをロードしなければ、データを抽出できません。メディアをロードするときは、クロスドメインポリシーファイルの有無をチェックするよう指定する必要があります。

- `Loader.load()` メソッドを使用する場合、LoaderContext オブジェクトである `context` パラメータの `checkPolicyFile` プロパティを設定します。
- `<img>` タグを使用してテキストフィールドにイメージを埋め込むときは、`<img checkPolicyFile = "true" src = "example.jpg">` のように、`<img>` タグの `checkPolicyFile` 属性を `"true"` に設定します。
- `Sound.load()` メソッドを使用する場合、SoundLoaderContext オブジェクトである `context` パラメータの `checkPolicyFile` プロパティを設定します。
- NetStream クラスを使用する場合、NetStream オブジェクトの `checkPolicyFile` プロパティを設定します。

これらのいずれかのパラメータを設定すると、Flash Player は最初に、既にそのドメイン用にダウンロードされたポリシーファイルがあるかどうかをチェックします。次に、`Security.loadPolicyFile()` メソッドへの保留中の呼び出しがスコープ内にあるかどうかを調べ、もしそうであれば、それらの呼び出しを待ちます。その後、サーバー上のデフォルトの場所にあるクロスドメインポリシーファイルを探します。

作成者 (開発者) コントロール

セキュリティ権限を付与するために使用される主な ActionScript API は、`Security.allowDomain()` メソッドで、これは、指定されたドメイン内の SWF ファイルに権限を付与します。次の例では、SWF ファイルは www.example.com ドメインに置かれた SWF ファイルへのアクセスを許可します。

```
Security.allowDomain("www.example.com")
```

次のメソッドは次の権限を許可します。

- SWF ファイル間のクロススクリプト (742 ページの「クロススクリプト」を参照)
- 表示リストへのアクセス (745 ページの「表示リスト内の移動」を参照)
- イベント検出 (745 ページの「イベントセキュリティ」を参照)
- Stage オブジェクトのプロパティとメソッドへのフルアクセス (744 ページの「ステージのセキュリティ」を参照)

`Security.allowDomain()` メソッドを呼び出す主な目的は、外部ドメインにある SWF ファイルに、`Security.allowDomain()` メソッドを呼び出す SWF ファイルをスクリプトする許可を与えることです。詳細については、742 ページの「クロススクリプト」を参照してください。

`Security.allowDomain()` メソッドにパラメータとして IP アドレスを指定しても、指定された IP アドレスに存在するすべてのアクセス元からのアクセスが許可されるわけではありません。許可されるのは、その IP アドレスに対応するドメイン名ではなく、URL として指定された IP アドレスを含むアクセス元からのアクセスだけです。たとえば、ドメイン名 www.example.com が IP アドレス 192.0.34.166 に対応している場合、`Security.allowDomain("192.0.34.166")` の呼び出しでは www.example.com へのアクセスは許可されません。

ワイルドカード `"*"` を `Security.allowDomain()` メソッドに渡して、すべてのドメインからのアクセスを許可できます。`"*"` ワイルドカードは " すべての " ドメインにある SWF ファイルに、呼び出しを行う SWF ファイルをスクリプトする許可を付与するので、使用に注意してください。

ActionScript には、`Security.allowInsecureDomain()` という第 2 の許可 API が組み込まれています。このメソッドは `Security.allowDomain()` メソッドとほとんど同じことをしますが、異なる点は、セキュアな HTTPS 接続によってサービスされた SWF ファイルから呼び出された場合に、HTTP などのセキュアでないプロトコルからサービスされる他の SWF ファイルに、呼び出し側の SWF ファイルへのアクセスを追加的に許可することです。しかし、セキュアなプロトコル (HTTPS) からのファイルと、セキュアでないプロトコル (HTTP など) からのファイルの間でスクリプティングを許可することはセキュリティ上望ましくありません。そのようなことをすると、セキュアなコンテンツがスヌープ攻撃やスプーフィング攻撃にさらされる可能性があります。そのような攻撃が可能になる理由は、次のとおりです。`Security.allowInsecureDomain()` メソッドは、HTTP 接続によってサービスされる SWF ファイルがセキュアな HTTPS データにアクセスすることを許可するので、HTTP サーバーとユーザーの間に割り込んだ攻撃者は、HTTP の SWF ファイルを独自のものに置き換えることにより、HTTPS データにアクセスできます。

セキュリティに関連したもう1つの重要なメソッドは、`Security.loadPolicyFile()` です。このメソッドは、Flash Player に標準以外の場所にクロスドメインポリシーファイルがあるかどうかをチェックさせます。詳細については、726 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」を参照してください。

セキュリティサンドボックス

クライアントコンピュータは、外部 Web サイトやローカルファイルシステムなどの多数のソースから個々の SWF ファイルを取得できます。Flash Player では、SWF ファイル、および共有オブジェクト、ビットマップ、サウンド、ビデオ、データファイルなどの他のリソースが Flash Player にロードされると、元の場所に基づいてセキュリティサンドボックスに個別に割り当てられます。次のセクションでは、Flash Player で適用される、任意のサンドボックス内の SWF ファイルがアクセスできるものを管理する規則について説明します。

セキュリティサンドボックスの詳細については、Flash Player 9 セキュリティに関するホワイトペーパーを参照してください。

リモートサンドボックス

Flash Player では、インターネットのアセット (SWF ファイルを含む) が Web サイトの元のドメインに対応する個別のサンドボックスに分類されます。デフォルトでは、それらのファイルは、それら自体のサーバーにあるリソースへのアクセスを許可されています。リモート SWF ファイルは、クロスドメインポリシーファイルと `Security.allowDomain()` メソッドなどの明示的な Web サイトおよび作成者の許可によって、他のドメインのその他のデータにアクセスできます。詳細については、726 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」と731 ページの「作成者 (開発者) コントロール」を参照してください。

リモート SWF ファイルは、ローカルファイルまたはローカルリソースをロードできません。

詳細については、Flash Player 9 セキュリティに関するホワイトペーパーを参照してください。

ローカルサンドボックス

" ローカルファイル " とは、`file:` プロトコルまたは UNC (Universal Naming Convention) パスを移用して参照されるすべてのファイルを表しています。ローカル SWF ファイルは、次の 3 つのローカルサンドボックスのいずれかに配置されます。

- local-with-filesystem サンドボックス - セキュリティを目的として、Flash Player はデフォルトですべてのローカル SWF ファイルおよびアセットを、local-with-file-system サンドボックスに配置します。このサンドボックスから、SWF ファイルは (たとえば、URLLoader クラスを使用して) ローカルファイルを読み取ることができますが、ネットワークとの通信はできません。その結果、ローカルデータがネットワークに漏れ出したり、不正に共有されたりすることがなくなります。
- local-with-networking サンドボックス - –SWF ファイルをコンパイルする際に、ローカルファイルとして実行される場合、ネットワークアクセスの許可を指定できます (734 ページの「ローカル SWF ファイルのサンドボックスタイプの設定」を参照)。これらのファイルは local-with-networking サンドボックスに配置されます。SWF ファイルが local-with-networking サンドボックスに割り当てられると、ローカルファイルへのアクセス権が失われます。代わりに、SWF ファイルはネットワークのデータにアクセスできるようになります。しかし、local-with-networking SWF ファイルは、そのための許可を得ない限り、クロスドメインポリシーファイルを使用するか、`Security.allowDomain()` メソッドの呼び出しを使用してネットワークからのデータを読み取ることはできません。そのような許可を付与するためには、クロスドメインポリシーファイルは、すべてのドメインへ許可を与える必要があり、それには、`<allow-access-from domain="*"/>` を使用するか、`Security.allowDomain("*")` を使用します。詳細については、726 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」および 731 ページの「作成者 (開発者) コントロール」を参照してください。
- local-trusted サンドボックス – (ユーザーまたはインストーラプログラムによって) 信頼できるファイルとして登録されたローカル SWF ファイルは、local-trusted サンドボックスに配置されます。システム管理者とユーザーは、セキュリティ上の配慮に基づき、local-trusted サンドボックス間でローカル SWF ファイルの再割り当て (移動) も実行できます (723 ページの「管理ユーザー制御」および 725 ページの「ユーザーの管理」を参照)。local-trusted サンドボックスに割り当てられた SWF ファイルは、他の SWF ファイルと自由に通信したり、どこからでも (リモートでもローカルでも) データをロードしたりすることができます。

local-with-networking サンドボックスと local-with-filesystem サンドボックスの間の通信は、local-with-filesystem サンドボックスとリモートサンドボックスの間の通信と同様に、厳しく禁じられています。そのような通信を可能にする許可を、Flash アプリケーションまたはユーザーや管理者が付与することはできません。

ローカル HTML ファイルとローカル SWF ファイル間の一方向のみのスクリプト – たとえば、ExternalInterface クラスを使用するには、HTML と SWF の両ファイルを local-trusted サンドボックスに配置する必要があります。その理由は、ブラウザのローカルセキュリティモデルが Flash Player のローカルセキュリティモデルと異なるからです。

local-with-networking サンドボックス内の SWF ファイルは、local-with-filesystem サンドボックス内の SWF ファイルをロードできません。local-with-filesystem サンドボックス内の SWF ファイルは、local-with-networking サンドボックス内の SWF ファイルをロードできません。

ローカル SWF ファイルのサンドボックスタイプの設定

local-with-filesystem サンドボックスまたは local-with-networking サンドボックス用の SWF ファイルを構成できます。そのためには、Adobe Flash CS3 Professional オーサリングツールで、ドキュメントのパブリッシュ設定を設定します。詳細については、「Using Flash の使用」の「Flash SWF ファイルフォーマットのパブリッシュオプションの設定」を参照してください。

コンピュータのエンドユーザーまたは管理者は、ローカル SWF ファイルが信頼できることを指定し、ローカルおよびネットワークの他のドメインからデータをロードできるようにすることができます。これは、Global Flash Player Trust ディレクトリおよび User Flash Player Trust ディレクトリで指定します。詳細については、723 ページの「管理ユーザー制御」および725 ページの「ユーザーの管理」を参照してください。

ローカルサンドボックスの詳細については、733 ページの「ローカルサンドボックス」を参照してください。

Security.sandboxType プロパティ

SWF ファイルの作成者は、読み取り専用の静的プロパティ `Security.sandboxType` を使用して、Flash Player で SWF ファイルを割り当てたサンドボックスのタイプを指定できます。Security クラスは、`Security.sandboxType` プロパティに指定できる値を表す次のような定数を含んでいます。

- `Security.REMOTE` - SWF ファイルはインターネット URL からダウンロードされ、ドメインベースのサンドボックス規則に従って機能します。
- `Security.LOCAL_WITH_FILE` - SWF ファイルはローカルファイルですが、ユーザーの信頼性はなく、ネットワークを指定してパブリッシュされているわけではありません。この SWF ファイルはローカルデータソースからの読み取りはできますが、インターネットとの通信はできません。
- `Security.LOCAL_WITH_NETWORK` - SWF ファイルはローカルファイルであり、ユーザーの信頼性はありませんが、ネットワークを指定してパブリッシュされています。この SWF ファイルはインターネットとの通信はできますが、ローカルデータソースからの読み取りはできません。
- `Security.LOCAL_TRUSTED` - SWF ファイルはローカルファイルであり、ユーザーの信頼性があり、設定マネージャまたは Flash Player トラスト構成ファイルを使用します。ローカルのデータソースから読み取ることも、インターネットでやり取りすることもできます。

ネットワーク API の制限

SWF ファイルのネットワーク機能へのアクセスは、SWF コンテンツを収めた HTML ページの <object> タグと <embed> タグに allowNetworking パラメータを設定して制御します。

allowNetworking に指定できる値は次のとおりです。

- "all" (デフォルト) - すべてのネットワーク API が SWF で許可されます。
- "internal" - SWF ファイルは、このセクションの後半に示すブラウザナビゲーションまたはブラウザインタラクション API を呼び出すことはできませんが、その他のネットワーク API を呼び出すことができます。
- "none" - SWF ファイルは、このセクションの後半に示すブラウザナビゲーションまたはブラウザインタラクション API を呼び出すことはできません。また、後半に示す SWF 間の通信 API を使用できません。

禁止された API を呼び出すと、SecurityError 例外がスローされます。

allowNetworking パラメータを設定するには、SWF ファイルの参照を含む HTML ページの <object> タグと <embed> タグで、次の例に示すように allowNetworking パラメータを追加し、その値を設定します。

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
   swflash.cab#version=9,0,18,0"
 width="600" height="400" id="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
   width="600" height="400"
   name="test" align="middle" type="application/x-shockwave-flash"
   pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML ページは、スクリプトを使用して SWF 埋め込みタグを生成することもできます。allowNetworking 設定が正しく挿入されるように、スクリプトを変更する必要があります。Flash および Adobe Flex Builder で生成された HTML ページは、AC_FL_RunContent( ) 関数を使用して SWF ファイルの参照を埋め込むため、次のようにスクリプトに allowNetworking パラメータ設定を追加する必要があります。

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

次の API は allowNetworking が "internal" に設定されている場合は禁止されます。

- navigateToURL()
- fscommand()
- ExternalInterface.call()

前記のリストの API 以外に、次の API も allowNetworking が "none" に設定されている場合は禁止されます。

- sendToURL()
- FileReference.download()
- FileReference.upload()
- Loader.load()
- LocalConnection.connect()
- LocalConnection.send()
- NetConnection.connect()
- NetStream.play()
- Security.loadPolicyFile()
- SharedObject.getLocal()
- SharedObject.getRemote()
- Socket.connect()
- Sound.load()
- URLLoader.load()
- URLStream.load()
- XMLSocket.connect()

選択された allowNetworking の設定により SWF ファイルがネットワーク API を使用できる場合でも、この章で示すセキュリティサンドボックスの制限により他の制限が課せられる場合があります。

allowNetworking が "none" に設定されている場合、TextField オブジェクトの htmlText プロパティの <img> タグで外部メディアを参照できません (SecurityError 例外がスローされます)。

allowNetworking を "none" に設定すると、ActionScript ではなく Flash オーサリングツールで追加された、インポート済み共有ライブラリのシンボルは、実行時にブロックされます。

フルスクリーンモードのセキュリティ

Flash Player 9.0.27.0 以降は、Flash コンテンツに画面全体を表示できるフルスクリーンモードをサポートしています。フルスクリーンモードに移行するためには、ステージの displayState プロパティは StageDisplayState.FULL_SCREEN 定数に設定します。詳細については、372 ページの「フルスクリーンモードの操作」を参照してください。

ブラウザで実行中の SWF ファイルについては、セキュリティ上の考慮事項がいくつかあります。

フルスクリーンモードを有効にするには、SWF ファイルの参照を含む HTML ページの <object> タグと <embed> タグで、次の例に示すように、allowFullScreen パラメータを追加し、その値を "true" に設定します (デフォルト値は "false")。

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
   swflash.cab#version=9,0,18,0"
 width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
   width="600" height="400"
   name="test" align="middle" type="application/x-shockwave-flash"
   pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML ページは、スクリプトを使用して SWF 埋め込みタグを生成することもできます。allowFullScreen 設定が正しく挿入されるように、スクリプトを変更する必要があります。Flash および Flex Builder で生成された HTML ページは、AC_FL_RunContent( ) 関数を使用して SWF ファイルの参照を埋め込むため、次のように allowFullScreen パラメータ設定を追加する必要があります。

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

フルスクリーンモードを開始する ActionScript は、マウスイベントまたはキーボードイベントへの応答としてのみ呼び出すことができます。別の状況で呼び出された場合、Flash Player は例外をスローします。

ユーザーはフルスクリーンモードの間は、テキスト入力フィールドにテキストを入力できません。すべてのキーボード入力およびキーボード関連の ActionScript は、フルスクリーンモードの間は無効ですが、例外としてキーボードショートカット ([Esc] キーを押すなど) により、アプリケーションは通常モードに戻ります。

コンテンツがフルスクリーンモードに移行すると、このモードの終了および通常モードへの復帰の方法を指示するメッセージが表示されます。メッセージは数秒間表示された後、消えます。

Stage オブジェクトの `displayState` プロパティを呼び出すと、ステージ所有者 (メイン SWF ファイル) と同じセキュリティサンドボックス内に存在しない呼び出し元について例外がスローされます。詳細については、744 ページの「ステージのセキュリティ」を参照してください。

管理者は、mms.cfg ファイルで `FullScreenDisable = 1` を設定することで、ブラウザで実行中の SWF ファイルのフルスクリーンモードを無効にできます。詳細については、723 ページの「管理ユーザー制御」を参照してください。

ブラウザでは、SWF ファイルは HTML ページ内に配置されなければフルスクリーンモードになりません。

フルスクリーンモードは、スタンドアロンプレイヤーまたはプロジェクタファイルは常に使用できます。

コンテンツのロード

SWF ファイルでは、次のタイプのコンテンツをロードできます。

- SWF ファイル
- イメージ
- サウンド
- ビデオ

SWF ファイルとイメージのロード

SWF ファイルおよびイメージ (JPG、GIF、または PNG ファイル) をロードするには、Loader クラスを使用します。local-with-filesystem サンドボックス以外にある SWF ファイルは、任意のネットワークドメインから SWF ファイルおよびイメージをロードできます。ローカルサンドボックス内にある SWF ファイルだけが、SWF ファイルとイメージをローカルファイルシステムからロードできます。しかし、local-with-networking サンドボックス内のファイルは、local-trusted サンドボックスまたは local-with-networking サンドボックス内のローカル SWF ファイルしかロードできません。local-with-networking サンドボックス内の SWF ファイルは、SWF ファイル以外のローカルコンテンツ (イメージなど) をロードしますが、ロードしたコンテンツの中のデータにはアクセスできません。

信頼できないソース (Loader オブジェクトのルート SWF ファイルのドメイン以外のドメインなど) から SWF ファイルをロードするとき、次のコードに示すように、Loader オブジェクトのマスクを定義して、ロードされたコンテンツ (Loader オブジェクトの子) がそのマスクの外部にあるステージの一部分に描画されないようにすることができます。

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

Loader オブジェクトの `load()` メソッドを呼び出す場合、LoaderContext オブジェクトである `context` を指定できます。LoaderContext クラスには、ロードされたコンテンツの使用方法のコンテキストを定義できる次の 3 つのプロパティが含まれています。

- `checkPolicyFile` - このプロパティは、SWF ファイルではなくイメージファイルをロードする場合にのみ使用します。Loader オブジェクトを含んでいるファイルのドメイン以外のドメインのイメージファイルに対して、このプロパティを指定します。このプロパティを `true` に設定した場合、Loader は元のサーバーでクロスドメインポリシーファイルをチェックします (726 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」を参照)。サーバーが Loader ドメインに許可を与えた場合、Loader ドメイン内にある SWF ファイルの ActionScript は、ロードされたイメージ内のデータにアクセスできます。つまり、`Loader.content` プロパティを使用して、ロードされたイメージを表す Bitmap オブジェクトへの参照を取得するか、`BitmapData.draw()` メソッドを使用して、ロードされたイメージのピクセルにアクセスできます。

- `securityDomain` - このプロパティは、イメージではなく SWF ファイルをロードする場合にのみ使用します。Loader オブジェクトを含むファイルのドメイン以外のドメインの SWF ファイルに対してこのプロパティを指定します。現時点で `securityDomain` プロパティに指定できる値は 2 つだけです。つまり、`null`（デフォルト）と `SecurityDomain.currentDomain` です。`SecurityDomain.currentDomain` を指定した場合、これはロードされた SWF ファイルを、ロードする側の SWF ファイルのサンドボックスへ読み込むよう要求します。つまり、ロードされた SWF ファイルが、ロードする側の SWF ファイル自体のサーバーからロードされたかのように動作することを意味します。これが許可されるのは、ロードされた SWF ファイルのサーバー上でクロスドメインポリシーファイルがあり、ロードする側の SWF ファイルのドメインによるアクセスを許可している場合だけです。要求されたポリシーファイルが検出された場合、ロードする側とロードされる側は、同じサンドボックス内にあるので、ロードが開始された後、自由にお互いをスクリプトすることができます。サンドボックスの読み込みは、ほとんどの場合、通常のロードを実行した後に、ロードされた SWF ファイルに `Security.allowDomain()` メソッドを呼び出させることに置き換えることができます。後者の方法の方が使いやすい場合もあります。なぜなら、ロードされた SWF ファイルはそれ自体の自然なサンドボックス内に入り、それ自体の実際のサーバー上にあるリソースにアクセスできるからです。
- `applicationDomain` - ActionScript 1.0 や ActionScript 2.0 で記述されたイメージまたは SWF ファイルではなく、ActionScript 3.0 で記述された SWF ファイルをロードする場合にのみ、このプロパティを使用します。ファイルをロードするときに、ファイルを特定のアプリケーションドメインに配置するよう指定できます。デフォルトのままでは、ファイルは新しいアプリケーションドメイン（ロードする側の SWF ファイルのアプリケーションドメインの子）に配置されます。アプリケーションドメインはセキュリティドメインのサブユニットです。したがって、ターゲットアプリケーションドメインを指定できるのは、ロードしようとしている SWF ファイルがユーザー自身のセキュリティドメインのものである場合だけです。なぜなら、それはユーザー自身のサーバーに入っているか、`securityDomain` プロパティを使用してセキュリティドメインへ正しく読み込まれた SWF ファイルだからです。アプリケーションドメインを指定し、ロードされた SWF ファイルが別のセキュリティドメインに属している場合、`applicationDomain` で指定したドメインは無視されます。詳細については、666 ページの「ApplicationDomain クラスの使用」を参照してください。

詳細については、402 ページの「ロードするコンテンツの指定」を参照してください。

Loader オブジェクトの重要なプロパティの 1 つに、`contentLoaderInfo` プロパティがあります。これは LoaderInfo オブジェクトです。他のほとんどのオブジェクトと異なり、LoaderInfo オブジェクトはロードする側の SWF ファイルとロードされたコンテンツの間で共有され、常に双方からアクセスできます。ロードされたコンテンツが SWF ファイルである場合、その SWF ファイルは `DisplayObject.loaderInfo` プロパティを通じて LoaderInfo オブジェクトにアクセスできます。LoaderInfo オブジェクトには、ロードの進行状況、ロードする側とロードされる側の URL、ロードする側とロードされる側の信頼関係、およびその他の情報が含まれています。詳細については、401 ページの「ロードの進行状況の監視」を参照してください。

サウンドとビデオのロード

local-with-filesystem サンドボックス内に入っている以外のすべての SWF ファイルで、`Sound.load()`、`NetConnection.connect()`、および `NetStream.play()` メソッドを使用して、ネットワークからサウンドおよびビデオをロードできます。

ローカル SWF ファイルだけが、ローカルファイルシステムからメディアをロードできます。それらのロードされたファイルのデータにアクセスできるのは、local-with-filesystem サンドボックスまたは local-trusted サンドボックスに入っている SWF ファイルだけです。

それ以外にも、ロードされたメディアのデータへのアクセスには制限があります。詳細については、746 ページの「ロードされたメディアへのデータとしてのアクセス」を参照してください。

テキストフィールドに <img> タグを使用した SWF ファイルとイメージのロード

SWF ファイルおよびビットマップをテキストフィールドにロードするには、次のコードに示すように、`<img>` タグを使用します。

```
<img src = 'filename.jpg' id = 'instanceName' >
```

この方法でロードされたコンテンツには、次のコードのように TextField インスタンスの `getImageReference()` メソッドを使用してアクセスします。

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

ただし、この方法でロードされた SWF ファイルとイメージは、オリジンに対応するサンドボックスに置かれます。

テキストフィールド内の `<img>` タグを使用してイメージファイルをロードする場合、イメージ内のデータへのアクセスは、クロスドメインポリシーファイルによって許可できます。次のコードのように、`checkPolicyFile` 属性を `<img>` タグに追加してポリシーファイルをチェックします。

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

テキストフィールドに `<img>` タグを使用した SWF をロードする場合、`Security.allowDomain()` メソッドを呼び出して、その SWF ファイルのデータにアクセスすることを許可できます。

SWF 内に埋め込んだ Bitmap クラスを使用せず、テキストフィールド内の `<img>` タグを使用して外部ファイルをロードする場合、Loader オブジェクトは TextField オブジェクトの子として自動的に作成され、外部ファイルは、ActionScript 内の Loader オブジェクトを使用してファイルをロードする場合とまったく同じように、Loader の中へロードされます。その場合、`getImageReference()` メソッドは、自動的に作成された Loader を返します。この Loader オブジェクトは、呼び出しコードと同じセキュリティサンドボックス内に置かれているため、このオブジェクトにアクセスする場合にはセキュリティチェックは不要です。

しかし、ロードされたメディアにアクセスするために Loader オブジェクトの `content` プロパティを参照する場合は、セキュリティルールが適用されます。コンテンツがイメージの場合、クロスドメインポリシーファイルの実装が必要になり、コンテンツが SWF ファイルの場合、SWF ファイル内のコードを使って `allowDomain()` メソッドを呼び出す必要があります。

RTMP サーバーを使用して送信されたコンテンツ

Flash Media Server は、RTMP (Real-Time Media Protocol) を使用してデータ、オーディオ、およびビデオを提供します。SWF ファイルは、NetConnection クラスの `connect()` メソッドを使用してこのメディアをロードし、パラメータとして RTMP URL を渡します。Flash Media Server は、要求元ファイルのドメインに基づいて、接続を制限し、コンテンツがダウンロードされないようにすることができます。詳細については、Flash Media Server のマニュアルを参照してください。

RTMP ソースからロードしたメディアの場合、`BitmapData.draw()` および `SoundMixer.computeSpectrum()` メソッドを使用してランタイムグラフィックおよびサウンドデータを抽出することはできません。

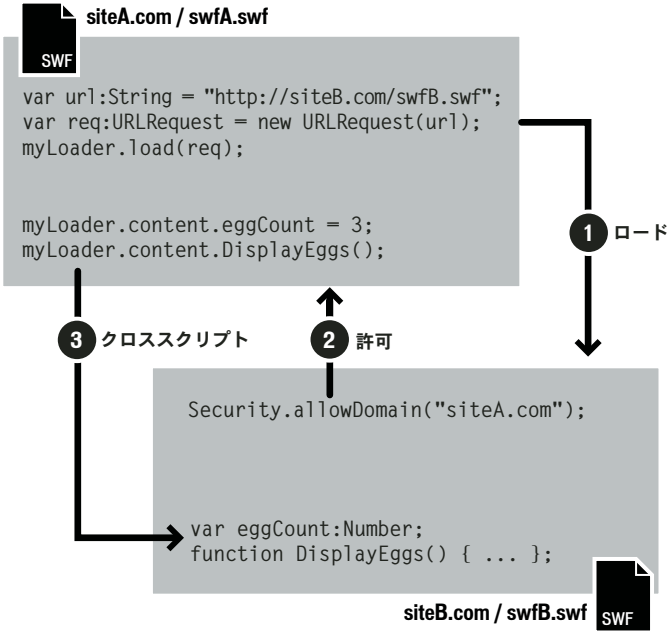
クロススクリプト

ActionScript 3.0 で記述された 2 つの SWF ファイルが同じドメインに所属している場合、たとえば、1 つの SWF ファイルの URL が http://www.example.com/swfA.swf、もう 1 つの SWF ファイルの URL が http://www.example.com/swfB.swf である場合、片方の SWF ファイルはもう一方の変数、オブジェクト、プロパティ、メソッドなどを検査および変更でき、その逆も可能です。これを " クロスドメインスクリプト " と呼びます。

AVM1 SWF ファイルと AVM2 SWF ファイル間でのクロススクリプトはサポートされません。AVM1 SWF ファイルは、ActionScript 1.0 または ActionScript 2.0 を使用して作成されたファイルです。AVM1 および AVM2 は、ActionScript 仮想マシンを意味します。しかし、LocalConnection クラスを使用すると、AVM1 と AVM2 間でデータを送信できます。

ActionScript 3.0 で記述された 2 つの SWF ファイルが異なるドメインに所属している場合 (たとえば、http://siteA.com/swfA.swf および http://siteB.com/swfB.swf)、デフォルトでは、swfA.swf で swfB.swf をスクリプトすることも、swfB.swf で swfA.swf をスクリプトすることも許可されません。`Security.allowDomain()` を呼び出すことで、SWF ファイルで他のドメインの SWF ファイルをスクリプトすることが許可されます。`Security.allowDomain("siteA.com")` を呼び出すことで、swfB.swf での siteA.com の SWF ファイルのスクリプト許可が付与されます。

クロスドメインの場合は、関与する 2 つのドメインを明確にすることが重要です。説明のため、ここでは、クロススクリプトを実行する側を " アクセス元 " (通常、アクセスする SWF) と呼び、他を "アクセス先" (通常、アクセスされる SWF) と呼びます。次の図に示すように、siteA.swf が siteB.swf をスクリプトする場合、siteA.swf がアクセス元で、siteB.swf がアクセス先です。



`Security.allowDomain()` メソッドを使用して確立されるクロスドメイン許可は非対称です。前の例では、siteA.swf は siteB.swf をスクリプトできますが、siteB.swf は siteA.swf をスクリプトできません。siteA.swf で、siteB.com の SWF ファイルに siteA.swf をスクリプトする許可を与えるための `Security.allowDomain()` メソッドを呼び出していないためです。対称的な許可を設定するには、両方の SWF ファイルで `Security.allowDomain()` メソッドを呼び出します。

Flash Player では、SWF ファイルを他の SWF ファイルによるクロスドメインスクリプトから保護するだけでなく、HTML ファイルによるクロスドメインスクリプトからも保護します。HTML から SWF へのスクリプトは、`ExternalInterface.addCallback()` メソッドを通じて確立されたコールバックで発生させることができます。HTML から SWF へのスクリプトがドメインを越える場合、アクセス先 SWF ファイルは、アクセス元が SWF ファイルの場合と同様に、`Security.allowDomain()` メソッドを呼び出す必要があります。さもなければ、操作は失敗します。詳細については、731 ページの「作成者 (開発者) コントロール」を参照してください。

また、Flash Player は SWF から HTML へのスクリプトのセキュリティ管理も行います。詳細については、754 ページの「ホスト Web ページのスクリプトへのアクセスの制御」を参照してください。

ステージのセキュリティ

Stage オブジェクトの一部のプロパティとメソッドは、表示リストのスプライトやムービークリップで使用できます。

しかし、Stage オブジェクトには所有者があるといいます。つまり、ロードされた最初の SWF ファイルです。デフォルトでは、Stage オブジェクトの次のプロパティおよびメソッドは、ステージ所有者と同じセキュリティサンドボックス内の SWF ファイルでのみ使用できます。

プロパティ		メソッド
`align`	`showDefaultContextMenu`	`addChild()`
displayState	`stageFocusRect`	`addChildAt()`
frameRate	`stageHeight`	`addEventListener()`
`height`	`stageWidth`	`dispatchEvent()`
`mouseChildren`	`tabChildren`	`hasEventListener()`
`numChildren`	`textSnapshot`	`setChildIndex()`
`quality`	`width`	`willTrigger()`
`scaleMode`		

ステージ所有者のサンドボックス以外のサンドボックス内の SWF ファイルが上記のプロパティおよびメソッドにアクセスするには、ステージ所有者の SWF ファイルは、`Security.allowDomain()` メソッドを呼び出して、外部サンドボックスのドメインを許可する必要があります。詳細については、731 ページの「作成者 (開発者) コントロール」を参照してください。

`frameRate` プロパティは特殊なケースです。SWF ファイルはすべて `frameRate` プロパティを読み出すことができます。しかし、このプロパティを変更できるのは、ステージ所有者のサンドボックスに入っているか、`Security.allowDomain()` メソッドの呼び出しによって許可を付与された SWF ファイルだけです。

Stage オブジェクトの `removeChildAt()` および `swapChildrenAt()` メソッドについての制限もありますが、それらは他の制限と異なっています。これらのメソッドを呼び出すには、ステージ所有者と同じドメイン内にあることが必要なのではなく、コードが、影響を受ける子オブジェクトの所有者と同じドメイン内にあることが必要です。あるいは、子オブジェクトは `Security.allowDomain()` メソッドを呼び出すことができます。

表示リスト内の移動

SWF ファイルから他のサンドボックスからロードされた表示オブジェクトにアクセスする機能は制限されます。SWF ファイルが別のサンドボックス内にある別の SWF ファイルによって作成された表示オブジェクトにアクセスするには、アクセスされる側の SWF ファイルが `Security.allowDomain()` メソッドを呼び出し、アクセスする側の SWF ファイルのドメインによるアクセスを許可する必要があります。詳細については、731 ページの「作成者 (開発者) コントロール」を参照してください。

Loader オブジェクトでロードされた Bitmap オブジェクトにアクセスするには、クロスドメインポリシーファイルはイメージファイルのオリジンサーバー上に存在する必要があり、そのクロスドメインポリシーファイルは、SWF ファイルのドメインに、Bitmap オブジェクトへのアクセスを試みる許可を付与する必要があります (726 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」を参照)。

ロードされたファイル (および Loader オブジェクト) に対応する LoaderInfo オブジェクトは、次の 3 つのプロパティを含んでおり、これらのプロパティはロードされたオブジェクトと Loader オブジェクト (`childAllowsParent`、`parentAllowsChild`、および `sameDomain`) の間の関係を定義します。

イベントセキュリティ

表示オブジェクトに関連するイベントには、イベントを送出する表示オブジェクトのサンドボックスに基づくセキュリティアクセスの制限があります。表示リストのイベントは、バブリング段階とキャプチャ段階に分けられます (299 ページ、第 10 章の「イベントの処理」で説明)。バブリングおよびキャプチャ段階では、イベントは表示リスト内のソース表示オブジェクトから親表示オブジェクトに移行します。親オブジェクトがソースの表示オブジェクトと異なるセキュリティサンドボックス内にある場合、キャプチャおよびバブリング段階はその親オブジェクトの下で停止します。ただし、親オブジェクトの所有者とソースオブジェクトの所有者の間に相互の信頼関係がある場合は除きます。この相互の信頼関係は、次のようにして達成できます。

1. 親オブジェクトを所有する SWF ファイルは、ソースオブジェクトを所有する SWF ファイルのドメインを信頼するために、`Security.allowDomain()` メソッドを呼び出す必要があります。
2. ソースオブジェクトを所有する SWF ファイルは、`Security.allowDomain()` メソッドを呼び出して、親オブジェクトを所有する SWF ファイルのドメインを信頼する必要があります。

ロードされたファイル (および Loader オブジェクト) に対応する LoaderInfo オブジェクトは、次の 3 つのプロパティを含んでおり、これらのプロパティはロードされたオブジェクトと Loader オブジェクト (`childAllowsParent` および `parentAllowsChild`) の間の関係を定義します。

表示オブジェクト以外のオブジェクトから送出されたイベントの場合、セキュリティチェックやセキュリティ関連の影響はありません。

ロードされたメディアへのデータとしてのアクセス

ロードされたデータにアクセスするには、BitmapData.draw() や SoundMixer.computeSpectrum() などのメソッドを使用します。デフォルトでは、あるセキュリティサンドボックスの SWF ファイルでは、別のサンドボックス内にあるロードされたメディアによってレンダリングまたは再生されるグラフィックオブジェクトまたはオーディオオブジェクトからピクセルデータやオーディオデータを取得できません。ただし、次のメソッドを使用するとこの許可を付与できます。

- ロードされた SWF ファイルで、Security.allowDomain() メソッドを呼び出して、別のドメインにある SWF ファイルへのデータアクセスを許可します。
- ロードされたイメージ、サウンド、またはビデオの場合は、ロードされたファイルのサーバー上でクロスドメインポリシーファイルを追加します。このポリシーファイルでは、BitmapData.draw() または SoundMixer.computeSpectrum() メソッドを呼び出してファイルからデータを抽出しようとする SWF ファイルのドメインにアクセスを許可する必要があります。

次のセクションでは、ビットマップ、サウンド、およびビデオのデータにアクセスする方法を詳しく説明します。

ビットマップデータへのアクセス

BitmapData オブジェクトの draw() メソッドを使用すると、任意の表示オブジェクトの現在表示されているピクセルを BitmapData オブジェクトに描画できます。描画できるピクセルには、MovieClip オブジェクト、Bitmap オブジェクト、または任意の表示オブジェクトのピクセルがあります。draw() メソッドを使って BitmapData オブジェクトにピクセルを描画するには、次の条件を満たす必要があります。

- ロードされたビットマップ以外のソースオブジェクトの場合、そのソースオブジェクトおよび (Sprite または MovieClip オブジェクトの場合には) そのすべての子オブジェクトは、draw() メソッドを呼び出すオブジェクトと同じドメインに属しているか、Security.allowDomain() メソッドの呼び出しを済ませていて、呼び出し側からアクセスできる SWF ファイル内に存在する必要があります。
- ロードされたビットマップソースオブジェクトの場合、ソースオブジェクトは draw() メソッドを呼び出すオブジェクトと同じドメインに所属する必要があります。あるいはそのソースサーバーは、呼び出し側ドメインに権限を付与するクロスドメインポリシーファイルをインクルードしている必要があります。

これらの条件が満たされない場合は、SecurityError 例外が発生します。

イメージをロードするとき、Loader クラスの load() メソッドを使用すると、LoaderContext オブジェクトである context パラメータを指定できます。LoaderContext オブジェクトの checkPolicyFile プロパティを true に設定した場合、Flash Player は、ロードされるイメージが入っているサーバー上にクロスドメインポリシーファイルがあるかどうかをチェックします。クロスドメインポリシーファイルが存在し、そのファイルがロードを行おうとしている SWF ファイルのドメインを許可している場合、その SWF ファイルは Bitmap オブジェクト内のデータにアクセスできます。それ以外の場合は、アクセスできません。

テキストフィールド内の <img> タグによってロードされるイメージの中で、checkPolicyFile プロパティを指定することもできます。詳細については、741 ページの「テキストフィールドに <img> タグを使用した SWF ファイルとイメージのロード」を参照してください。

サウンドデータへのアクセス

次のサウンド関連の ActionScript 3.0 API にはセキュリティ制限があります。

- SoundMixer.computeSpectrum() メソッド - サウンドファイルと同じセキュリティサンドボックス内にある SWF ファイルで常に使用できます。他のサンドボックス内のファイルには、セキュリティチェックが行われます。
- SoundMixer.stopAll() メソッド - サウンドファイルと同じセキュリティサンドボックス内にある SWF ファイルで常に使用できます。他のサンドボックス内のファイルには、セキュリティチェックが行われます。
- Sound クラスの id3 プロパティ - サウンドファイルと同じセキュリティサンドボックス内にある SWF ファイルで常に使用できます。他のサンドボックス内のファイルには、セキュリティチェックが行われます。

それぞれのサウンドに、コンテンツサンドボックスとオーナーサンドボックスの 2 種類のサンドボックスが関連付けられています。

- サウンドのオリジンドメインはコンテンツサンドボックスを決定し、コンテンツサンドボックスは、サウンドの id3 プロパティと SoundMixer.computeSpectrum() メソッドによってサウンドからデータを抽出できるかどうかを決定します。
- サウンドの再生を開始したオブジェクトは、所有者サンドボックスを決定し、所有者サンドボックスは、SoundMixer.stopAll() メソッドを使用してサウンドを停止できるかどうかを決定します。

サウンドをロードするとき、Sound クラスの load() メソッドを使用すると、SoundLoaderContext オブジェクトである context パラメータを指定できます。SoundLoaderContext オブジェクトの checkPolicyFile プロパティを true に設定した場合、Flash Player は、サウンドがロードされるサーバー上にクロスドメインポリシーファイルがあるかどうかをチェックします。クロスドメインポリシーファイルがあり、そのファイルでロードする側の SWF ファイルのドメインが許可されている場合、SWF ファイルは Sound オブジェクトの id プロパティにアクセスできます。それ以外の場合はアクセスできません。また、checkPolicyFile プロパティの設定により、ロードされたサウンドに SoundMixer.computeSpectrum() メソッドを有効にできます。

SoundMixer.areSoundsInaccessible() メソッドを使用すると、SoundMixer.stopAll() メソッドの呼び出しが、呼び出し側からサウンド所有者のサンドボックスにアクセスできないためにすべてのサウンドを停止できないかどうかを知ることができます。

SoundMixer.stopAll() メソッドを呼び出すと、所有者サンドボックスが stopAll() の呼び出し側のそれと同じものであるサウンドが停止します。また、Security.allowDomain() メソッド (これによって stopAll() メソッドを呼び出す SWF ファイルのドメインによるアクセスを許可する) を呼び出した SWF ファイルによって再生が開始されたサウンドも停止します。それ以外のサウンドは停止せず、そのようなサウンドの存在は、SoundMixer.areSoundsInaccessible() メソッドを呼び出すことによって知ることができます。

computeSpectrum() メソッドを呼び出すには、再生中のすべてのサウンドが、このメソッドを呼び出すオブジェクトと同じサンドボックスに属するか、呼び出し側のサンドボックスに許可を与えたソースに属する必要があります。それ以外の場合は、SecurityError 例外が発生します。SWF ファイル内でライブラリの埋め込みサウンドからロードされたサウンドの場合、ロードされた SWF ファイルの Security.allowDomain() メソッドへの呼び出しを使用して許可を付与します。SWF ファイル以外のソースからロードされたサウンド (ロードされた MP3 ファイルまたは Flash Video からのサウンド) の場合、ソースサーバー上のクロスドメインポリシーファイルが、ロードされたメディア内のデータへのアクセスを許可します。サウンドが RTMP ストリームからロードされた場合は、computeSpectrum() メソッドを使用できません。

詳細については、731 ページの「作成者 (開発者) コントロール」および 726 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」を参照してください。

ビデオデータへのアクセス

ビデオの現在のフレームのピクセルデータをキャプチャするには、BitmapData.draw() メソッドを使用します。

ビデオには、次の 2 種類があります。

- RTMP ビデオ
- プログレッシブビデオ (RTMP サーバーなしに FLV ファイルからロードされる)

BitmapData.draw() メソッドを使用して RTMP ビデオにアクセスすることはできません。

プログレッシブビデオを source パラメータとして BitmapData.draw() メソッドを呼び出す場合は、BitmapData.draw() の呼び出し側が FLV ファイルと同じサンドボックスに属しているか、FLV ファイルのサーバーが、呼び出しを行う SWF ファイルのドメインに許可を与えるポリシーファイルを持っている必要があります。NetStream オブジェクトの checkPolicyFile プロパティを true に設定する方法で、ポリシーファイルがダウンロードされるように要求できます。

データのロード

SWF ファイルはサーバーから ActionScript 内にデータをロードでき、ActionScript からサーバーにデータを送信できます。データのロードはメディアのロードとは異なる種類の操作です。なぜなら、ロードされた情報は、メディアとして表示されず、ActionScript 内で直接表示されるからです。一般に、SWF ファイルは、それ自体のドメインからデータをロードできます。しかし、他のドメインからデータをロードするためには、通常ではクロスドメインポリシーファイルが必要になります。

URLLoader と URLStream の使用

データは、XML ファイルまたはテキストファイルとしてロードできます。URLLoader クラスと URLStream クラスの load() メソッドは、クロスドメインポリシーファイルの許可により制御されます。

load() メソッドを使用して、このメソッドを呼び出している SWF ファイルのドメイン以外のドメインからコンテンツをロードする場合、Flash Player はロードされるアセットのサーバー上にクロスドメインポリシーファイルがあるかどうかをチェックします。クロスドメインポリシーファイルが存在し、そのファイルでロードする側の SWF ファイルのドメインにアクセスが許可されている場合は、データをロードできます。

ソケットへの接続

デフォルトでは、ソケットおよび XML ソケット接続へのクロスドメインアクセスは無効になっています。また 1024 未満のポートでは、SWF ファイルと同じドメインでのソケット接続へのアクセスは、デフォルトで無効に設定されています。これらのポートへのアクセスは、次の場所のいずれかに所属するクロスドメインポリシーファイルを使用することで許可できます。

- メインソケット接続と同じポート
- 別のポート
- ソケットサーバーと同じドメイン内のポート 80 上の HTTP サーバー

メインソケット接続と同じポート、または別のポートからクロスドメインポリシーファイルを提供する場合、次の例に示すように、クロスドメインポリシーファイル内で to-ports 属性を使用して許可されるポートを列挙します。

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy
   SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- xmlsocket://socks.mysite.com のポリシーファイル -->
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="507" />
   <allow-access-from domain="*.example.com" to-ports="507,516" />
   <allow-access-from domain="*.example.org" to-ports="516-523" />
   <allow-access-from domain="adobe.com" to-ports="507,516-523" />
   <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

メインソケット接続と同じポートからソケットポリシーファイルを取得する場合、Socket.connect() メソッドまたは XMLSocket.connect() メソッドを呼び出します。また指定されたドメインが呼び出した SWF ファイルのドメインと異なる場合、取得を試みているメイン接続と同じポートからのポリシーファイルの取得が自動的に試みられます。メイン接続と同じサーバー上で別のポートからソケットポリシーファイルを取得するには、次のように特殊な "xmlsocket" シンタックスを使用して、Security.loadPolicyFile() メソッドを呼び出します。

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Socket.connect() メソッドまたは XMLSocket.connect() メソッドを呼び出す前に、Security.loadPolicyFile() メソッドを呼び出してください。その場合、Flash Player は、ユーザーのメイン接続を許可するかどうかを決める前に、ユーザーのポリシーファイル要求が満たされるのを待ちます。

ソケットサーバーを実装し、ソケットポリシーファイルを提供する必要がある場合は、ポリシーファイルを提供するのに、メイン接続を受け入れるのと同じポートを使用するか、それとも別のポートを使用するかを決めてください。いずれの場合も、サーバーはポリシーファイルを送信するか、メイン接続を確立するかの決定の前に、クライアントからの最初の送信を待機する必要があります。Flash Player は、ポリシーファイルを要求する場合、接続が確立されると同時に、必ず次のストリングを送信します。

```
<policy-file-request/>
```

サーバーはこの文字列を受信すると、ポリシーファイルを送信できます。同じ接続をポリシーファイル要求とメイン接続の両方に再使用しないでください。ポリシーファイルの送信後は、接続を閉じてください。そのようにしなかった場合、Flash Player はポリシーファイル接続を閉じてから、再接続してメイン接続を設定します。

詳細については、729 ページの「ソケットポリシーファイル」を参照してください。

データの送信

データ送信が行われるのは、SWF ファイルの ActionScript コードによりサーバーまたはリソースにデータが送信されるときです。データの送信は、ネットワークドメイン SWF ファイルで常に実行できます。ローカル SWF ファイルは、その SWF ファイルが local-trusted サンドボックスまたは local-with-networking サンドボックスに入っている場合にのみ、ネットワークアドレスにデータを送信できます。詳細については、733 ページの「ローカルサンドボックス」を参照してください。

`flash.net.sendToURL()` 関数を使用すると、URL にデータを送信できます。それ以外のメソッドも、URL に要求を送信します。それらのメソッドとしては、`Loader.load()` や `Sound.load()` などのロードメソッドと、`URLLoader.load()` や `URLStream.load()` などのデータロードメソッドがあります。

ファイルのアップロードとダウンロード

`FileReference.upload()` メソッドは、ユーザーによって選択されたファイルのリモートサーバーへのアップロードを開始します。`FileReference.upload()` メソッドを呼び出す前に、`FileReference.browse()` メソッドまたは `FileReferenceList.browse()` メソッドを呼び出す必要があります。

`FileReference.download()` メソッドを呼び出すと、ユーザーがリモートサーバーからファイルをダウンロードできるダイアログボックスが開きます。

メモ	サーバーでユーザー認証が必要な場合、ブラウザ内で実行される、つまり、ブラウザプラグインまたは ActiveX コントロールを使用する SWF ファイルでのみ、認証用のユーザー名とパスワードをユーザーが入力できるダイアログボックスを表示できます。ただし、それはダウンロードの場合のみです。Flash Player では、ユーザー認証が必要なサーバーへのアップロードはできません。

呼び出しを行う SWF ファイルが local-with-filesystem サンドボックス内にある場合、アップロードとダウンロードは許可されません。

デフォルトでは、SWF ファイルは、それ自体のサーバー以外のサーバーとの間でアップロードまたはダウンロードを行うことはできません。SWF ファイルは、別のサーバーへのアップロードまたはダウンロードが可能ですが、そのサーバーに、呼び出し側 SWF ファイルのドメインに許可を与えるクロスドメインポリシーファイルが格納されている必要があります。

セキュリティドメインに読み込まれた SWF ファイルからの埋め込みコンテンツのロード

SWF ファイルをロードするとき、ファイルをロードするために使用する Loader オブジェクトの `load()` メソッドの `context` パラメータを設定できます。このパラメータは、LoaderContext オブジェクトを取得します。この LoaderContext オブジェクトの `securityDomain` プロパティを `Security.currentDomain` に設定すると、Flash Player はロードされた SWF ファイルのサーバー上にクロスドメインポリシーファイルがあるかどうかをチェックします。クロスドメインポリシーファイルが存在し、そのファイルでロードする側の SWF ファイルのドメインが許可されている場合、SWF ファイルを読み込んだメディアとしてロードできます。この方法により、ロードする側のファイルは、SWF ファイルのライブラリにあるオブジェクトにアクセスできるようになります。

SWF ファイルから別のセキュリティサンドボックスにあるロードされた SWF ファイルのクラスにアクセスする代替の方法は、ロードされた SWF ファイルに、`Security.allowDomain()` メソッドを呼び出させ、呼び出す側の SWF ファイルのドメインにアクセスを許可することです。`Security.allowDomain()` メソッドの呼び出しを、ロードされた SWF ファイルのメインクラスのコンストラクタメソッドに追加し、その後、ロードする側の SWF ファイルに、Loader オブジェクトの `contentLoaderInfo` プロパティによって送出された `init` イベントに応答するイベントリスナーを追加させることができます。このイベントが送出される時点で、ロードされた SWF ファイルはコンストラクタメソッド内の `Security.allowDomain()` メソッドの呼び出しを完了しており、ロードされた SWF ファイル内のクラスは、ロードする側の SWF ファイルから使用可能です。ロードする側の SWF ファイルは、`Loader.contentLoaderInfo.applicationDomain.getDefinition()` を呼び出すことにより、ロードされた SWF ファイルからクラスを取得できます。

古いコンテンツの操作

Flash Player 6 では、特定の Flash Player 設定に使用されるドメインは、SWF ファイルのドメインの最後の部分に基づいています。この設定には、カメラとマイクの許可、記憶領域の割り当て、永続共有オブジェクトの記憶領域などがあります。

www.example.com のように SWF ファイルのドメインに 3 つ以上のセグメントが含まれている場合、ドメインの最初のセグメント (www) は削除され、ドメインの残りの部分が使用されます。したがって、Flash Player 6 では、www.example.com と store.example.com はどちらも example.com をこれらの設定のドメインとして使用します。同様に、www.example.co.uk と store.example.co.uk はどちらも example.co.uk をこれらの設定のドメインとして使用します。このため、example1.co.uk と example2.co.uk など、互いに無関係のドメインの SWF ファイルが、同じ共有オブジェクトにアクセスできるという問題が起きる場合があります。

Flash Player 7 以降では、Flash Player 設定は、デフォルトで SWF ファイルの正確なドメインに従って選択されます。たとえば、www.example.com の SWF ファイルは www.example.com の Flash Player 設定を使用し、store.example.com の SWF ファイルはそれとは別の store.example.com の Flash Player 設定を使用します。

ActionScript 3.0 で記述された SWF ファイルでは、`Security.exactSettings` が `true` に設定された場合 (デフォルト)、Flash Player は Player 設定に正確なドメインを使用します。`false` に設定された場合、Flash Player は Flash Player 6 で使用されるドメイン設定を使用します。`exactSettings` をデフォルト値から変更する場合、カメラやマイクの使用、永続共有オブジェクトの取得など、Flash Player によるプレイヤー設定の選択を要求するイベントが起こる前に、この変更を行う必要があります。

バージョン 6 の SWF ファイルをパブリッシュし、そこから永続共有オブジェクトを作成した場合、ActionScript 3.0 を使用する SWF から永続共有オブジェクトを取得するには、`SharedObject.getLocal()` を呼び出す前に、`Security.exactSettings` を `false` に設定する必要があります。

LocalConnection 許可の設定

LocalConnection クラスを使用すると、相互に指示を送ることができる SWF ファイルを作成できます。LocalConnection オブジェクトを使って通信できるのは、同じクライアントコンピュータ上で実行中の SWF ファイルだけです。ただし、これらは異なるアプリケーションで実行されていてもかまいません。たとえば、ブラウザで実行されている SWF ファイルと、プロジェクタで実行されている SWF ファイルとの間で通信することもできます。

どの LocalConnection 通信の場合も、送信側 SWF ファイルと受信側 SWF ファイルがあります。Flash Player では、デフォルトで同じドメイン内の SWF ファイル間で LocalConnection 通信が可能です。別のサンドボックス内の SWF ファイルの場合、受信側は `LocalConnection.allowDomain()` メソッドを使用して送信側に許可を与える必要があります。`LocalConnection.allowDomain()` メソッドへ引数として渡すストリングには、次のものを含めることができます。正確なドメイン名、IP アドレス、およびワイルドカードの *。

メモ	`allowDomain()` メソッドは、ActionScript 1.0 および 2.0 の形式から変更されています。これらの以前のバージョンでは、`allowDomain()` は実装するコールバックメソッドでした。ActionScript 3.0 では、`allowDomain()` は、ユーザーが呼び出す LocalConnection クラスのビルトインメソッドです。この変更のため、`allowDomain()` は `Security.allowDomain()` とほとんど同じように動作します。

SWF ファイルは LocalConnection クラスの `domain` プロパティを使用して、そのドメインを決定します。

ホスト Web ページのスクリプトへのアクセスの制御

外部スクリプトは、次の ActionScript 3.0 API を使用して実行できます。

- `flash.system.fscommand()` 関数
- `flash.net.navigateToURL()` 関数 (`navigateToURL("javascript: alert('Hello from Flash Player.')"` などのスクリプトステートメントを指定するとき)
- `flash.net.navigateToURL()` 関数 (`window` パラメータを `"_top"`、`"_self"`、または `"_parent"` に設定したとき)
- `ExternalInterface.call()` メソッド

ローカルに実行される SWF ファイルについては、SWF ファイルとこれを含む Web ページ (存在する場合) が local-trusted セキュリティサンドボックス内に置かれている場合にのみ、上記のメソッドの呼び出しが成功します。上記のメソッドの呼び出しが失敗するのは、コンテンツが local-with-networking または local-with-filesystem のいずれかのサンドボックス内にある場合です。

SWF ファイルをロードする HTML コード内の `AllowScriptAccess` パラメータは、SWF ファイル内から外部スクリプトを実行する機能を制御します。

SWF ファイルをホストする Web ページの HTML コード内でこのパラメータを設定します。このパラメータは、`PARAM` タグまたは `EMBED` タグの中で設定します。

`AllowScriptAccess` パラメータの可能な値は、次の 3 つです。`"always"`、`"sameDomain"`、または `"never"`

- `AllowScriptAccess` が `"sameDomain"` の場合、外部スクリプトが許可されるのは SWF ファイルと Web ページが同じドメイン内にある場合のみです。これは、AVM2 のコンテンツのデフォルト設定です。
- `AllowScriptAccess` が `"never"` の場合、外部スクリプトは常に失敗します。
- `AllowScriptAccess` が `"always"` に設定されている場合、外部スクリプトは常に成功します。

HTML ページ内の SWF ファイルで `AllowScriptAccess` パラメータが指定されていない場合、AVM2 のコンテンツのデフォルトは `"sameDomain"` です。

次に、HTML ページで `AllowScriptAccess` タグを設定する例を示します。

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
  codebase='http://download.adobe.com/pub/shockwave/cabs/flash/
  swflash.cab#version=9,0,0,0' height='100%' width='100%'>
<param name='AllowScriptAccess' value='never'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/getflashplayer'
  src='MyMovie.swf' height='100%' width='100%' AllowScriptAccess='never'/>
</object>
```

AllowScriptAccess パラメータを使用すると、あるドメインでホストされている SWF ファイルから別のドメインの HTML ページのスクリプトにアクセスすることを禁止できます。別のドメインでホストされているすべての SWF ファイルで AllowScriptAccess="never" を使用すると、HTML ページ内のスクリプトのセキュリティが確保されます。

詳細については、『ActionScript 3.0 コンポーネントリファレンスガイド』で次の項目を参照してください。

- flash.system.fscommand() 関数
- flash.net.navigateToURL() 関数
- ExternalInterface クラスの call() メソッド

共有オブジェクト

Flash Player は、共有オブジェクト (SWF ファイルの外部で存続する ActionScript オブジェクト) をユーザーのローカルファイルシステム上で使用することも、リモート RTMP サーバー上で使用することもできます。共有オブジェクトは、Flash Player 内のその他のメディアと同様に、いくつかのセキュリティサンドボックスに区分されます。しかし、共有オブジェクトのサンドボックスモデルは、少し異なっています。なぜなら、共有オブジェクトはドメインの境界を越えてアクセスできるリソースではないからです。共有オブジェクトは常に、SharedObject クラスのメソッドを呼び出す各 SWF ファイルのドメインに固有の共有オブジェクトストアから取得されます。通常、共有オブジェクトストアは SWF ファイルのドメインよりもさらに細分化されています。デフォルトでは、それぞれの SWF ファイルごとに、そのオリジン URL 全体に固有の共有オブジェクトストアが使用されます。

SWF ファイルは SharedObject.getLocal() および SharedObject.getRemote() メソッドの localPath パラメータを使用して、その URL の一部分だけに関連付けられた共有オブジェクトストアを使用することができます。この方法により、SWF ファイルは、他の URL からの別の SWF ファイルとの共有を許可できます。localPath パラメータとして '/' を渡した場合でも、独自のドメインに固有の共有オブジェクトストアが指定されます。

[Flash Player の設定] ダイアログボックスまたは設定マネージャを使用して、共有オブジェクトへのアクセスを制限できます。デフォルトでは、ドメインごとに最大 100KB のデータサイズまでの共有オブジェクトを作成できます。管理ユーザーおよび一般ユーザーは、ファイルシステムへの書き込み機能に制限を適用することもできます。詳細については、723 ページの「管理ユーザー制御」および 725 ページの「ユーザーの管理」を参照してください。

共有オブジェクトがセキュアであることを指定するには、SharedObject.getLocal() メソッドまたは SharedObject.getRemote() メソッドの secure パラメータに、true を指定します。secure パラメータについて、次の点に注意してください。

- このパラメータを true に設定すると、Flash Player は新しいセキュアな共有オブジェクトを作成するか、既存のセキュアな共有オブジェクトの参照を取得します。このセキュアな共有オブジェクトは、HTTPS 経由で配信される SWF ファイルで、secure パラメータを true に設定して SharedObject.getLocal() を呼び出した場合にのみ読み書きできます。
- このパラメータを false に設定した場合、Flash Player は新しい共有オブジェクトを作成するか、非 HTTPS 接続で配信された SWF ファイルから読み書きできる既存の共有オブジェクトへの参照を取得します。

呼び出し側の SWF ファイルが HTTPS URL からのものでない場合、SharedObject.getLocal() メソッドまたは SharedObject.getRemote() メソッドの secure パラメータに true を指定すると、SecurityError 例外が発生します。

共有オブジェクトストアの選択は、SWF ファイルのオリジン URL が基礎となります。このことは、SWF ファイルが単純な URL から提供されない 2 つの状況でも当てはまります。それらの状況は、読み込みロードとダイナミックロードです。読み込みロードとは、LoaderContext.securityDomain プロパティを SecurityDomain.currentDomain に設定して SWF ファイルをロードする状況のことです。この状況では、ロードされた SWF ファイルは、ロードする側の SWF ファイルのドメインで始まり、その後に実際のオリジン URL を指定した疑似 URL を持ちます。ダイナミックロードとは、Loader.loadBytes() メソッドを使用して SWF ファイルをロードすることです。この状況では、ロードされた SWF ファイルは、ロードする側の SWF ファイルの完全 URL で始まり、その後に整数の ID が続く疑似 URL を持ちます。読み込みロードとダイナミックロードのどちらの場合も、LoaderInfo.url プロパティを使用して SWF ファイルの疑似 URL を検査できます。疑似 URL は、共有オブジェクトストアを選択する目的で実 URL とまったく同じように扱われます。疑似 URL の一部または全部を使用するには、共有オブジェクトの localPath パラメータを指定します。

ユーザーおよび管理者は、" サードパーティ共有オブジェクト " の使用を無効にすることを選択できます。これは、Web ブラウザ内で実行されているすべての SWF ファイルが共有オブジェクトを使用することで、その場合、その SWF ファイルのオリジン URL は、ブラウザのアドレスバーに表示される URL と異なるドメインに属しています。ユーザーと管理者は、プライバシーの理由から、クロスドメイントラッキングを避けるために、サードパーティ共有オブジェクトの使用を無効にすることを選択できます。この制限を回避するために、共有オブジェクトを使用するすべての SWF ファイルが、ブラウザのアドレスバーに表示されているのと同じドメインから SWF ファイルが提供される HTML ページ構造内でのみロードされるようにすることもできます。サードパーティ SWF ファイルから共有オブジェクトを使用しようとして、サードパーティ共有オブジェクトの使用が無効にされている場合、SharedObject.getLocal() および SharedObject.getRemote() メソッドは null を返します。詳細については、www.adobe.com/products/flashplayer/articles/thirdpartylso を参照してください。

カメラ、マイク、クリップボード、マウス、キーボードのアクセス

SWF ファイルが `Camera.get()` または `Microphone.get()` メソッドを使用してユーザーのカメラまたはマイクにアクセスしようとした場合、Flash Player は [プライバシー] ダイアログボックスを表示します。このダイアログボックスで、ユーザーはカメラおよびマイクへのアクセスを許可または拒否できます。ユーザーと管理ユーザーは、mms.cfg ファイル、設定 UI、設定マネージャのコントロールを通じて、カメラアクセスをサイト別またはグローバルに無効にできます (723 ページの「管理ユーザー制御」と 725 ページの「ユーザーの管理」を参照)。ユーザー制限があると、`Camera.get()` および `Microphone.get()` メソッドは `null` 値を返します。`Capabilities.avHardwareDisable` プロパティを使用すると、カメラおよびマイクが管理上禁止されているか (`true`) または許可されているか (`false`) を決定できます。

`System.setClipboard()` メソッドでは、SWF ファイルでクリップボードの内容をプレーンテキストストリング文字に置き換えることができます。これにより、セキュリティリスクが発生することはありません。クリップボードにカットまたはコピーされるパスワードおよびその他の機密データがもたらすリスクを防ぐために、対応する "getClipboard" (読み取り) メソッドはありません。

Flash アプリケーションが監視できるのは、そのフォーカス内で発生するキーボードイベントとマウスイベントのみです。Flash アプリケーションは、別のアプリケーションでのキーボードイベントまたはマウスイベントを検出できません。

索引

記号

\ (円記号)
　ストリング内 195
　正規表現 279
!= 演算子 197
!== (厳密な不等価) 演算子 197
+ (連結) 演算子、XMLList 340
$ 置換コード 202
$ メタ文字 279
& (アンパサンド) 618
( ) (XML のフィルタ処理) 演算子 343
() (括弧) 演算子 99
() (括弧) メタ文字 279
.. (子孫アクセサ) 演算子、XML 340
. (ドット) 演算子 97、117
. (ドット) 演算子、XML 333、340
. (ドット) メタ文字 279
* (アスタリスク) 型注釈 81、84、91
* (アスタリスク) メタ文字 279
* (ワイルドカード) 演算子、XML 342
+ (加算) 演算子 198
+ (プラス) メタ文字 279
+= (加算後代入) 演算子 198、340
, (カンマ) 演算子 78
-as3 コンパイラオプション 231
-es コンパイラオプション 231
... (rest) パラメータ 124
/ (スラッシュ) 277、279
: (コロン) 演算子 83
>= 演算子 197
== 演算子 197
=== 演算子 197
\? (疑問符) 279
?: (条件) 演算子 108
@ (属性識別用) 演算子、XML 333、342
[(左角括弧) 279
] (右角括弧) 279
^ (キャレット) 279
__proto__ 64
__resolve 64
| (パイプ) 284

A

ActionScript
　ActionScript ファイルへの格納 45
　OOP サポートの歴史 161
　アプリケーションにインクルードする方法 44
　アプリケーションの構築 44
　開発プロセス 49
　旧バージョンとの互換性 22
　作成用ツール 47
　新機能 18
　説明 17、64
　テキストエディタを使用した作成 48
　マニュアル 14
　利点 18
ActionScript 1.0 161
ActionScript 2.0、プロトタイプチェーン 164
ActionScript 仮想マシン (AVM1) 161
ActionScript 仮想マシン 2 (AVM2) 161、166
ActionScript のコア Error クラス 262
addCallback() メソッド 743
addEventListener() メソッド 145、305、318
addListener() メソッド 305
allowDomain() メソッド
　img タグ 741
　LocalConnection クラス 628
　クロススクリプトの説明 742
　コンストラクタ 752
　コンテキストのロード 402
　サウンド 748
allowFullScreen 属性 737
allowInsecureDomain() メソッド 628

allowNetworking タグ 735
AllowScriptAccess パラメータ 754
application/x-www-form-urlencoded 617
ApplicationDomain クラス 403、666、740
apply() メソッド 231
arguments オブジェクト 120、122、124
arguments.callee プロパティ 122
arguments.caller プロパティ 124
arguments.length プロパティ 122
Array クラス
　concat() メソッド 221
　join() メソッド 221
　length プロパティ 217、223
　pop() メソッド 216
　push() メソッド 216、232
　reverse() メソッド 218
　shift() メソッド 216
　slice() メソッド 221
　sort() メソッド 218
　sortOn() メソッド 218、220
　splice() メソッド 216
　toString() メソッド 221
　unshift() メソッド 216
　拡張 230
　コンストラクタのアルゴリズム 231
as 演算子 87、149
AS3 名前空間 168、231
ASCII 文字 191
avHardwareDisable プロパティ 723
AVM1 (ActionScript 仮想マシン) 161
AVM1Movie クラス 360
AVM2 (ActionScript 仮想マシン 2) 161、166

B

beginGradientFill() メソッド 434
bitmap caching
　caching movie clips 389
Bitmap クラス 359、519
BitmapData オブジェクト、フィルタの適用 448
BitmapData クラス 519
Boolean クラス
　strict モードでの暗黙的な変換 93
　キャスト 95
Boolean データ型 89
browse() メソッド 751
bubbles プロパティ 309
ByteArray クラス 229

C

call() メソッド (ExternalInterface クラス) 735、754
callback methods
　ignoring 540
callee プロパティ 122
caller プロパティ 124
Camera クラス 549
cancelable プロパティ 308
Capabilities クラス 665
Capabilities.avHardwareDisable プロパティ 723
Capabilities.localFileReadDisable プロパティ 723
catch ブロック 250
charAt() メソッド 196
charCodeAt() メソッド 196
checkPolicyFile プロパティ 730
childAllowsParent プロパティ 745
class キーワード 132
clearInterval() 関数 186
clearTimeout() 関数 186
clone() メソッド (BitmapData クラス) 524
clone() メソッド (Event クラス) 311
ColdFusion 623
ColorTransform クラス 421
colorTransform プロパティ 421
computeSpectrum() メソッド (SoundMixer クラス)
　742、746、747
concat() メソッド
　Array クラス 221
　String クラス 198
connect() メソッド
　LocalConnection クラス 736
　NetConnection クラス 736、741
　Socket クラス 736
　XMLSocket クラス 631、736
content プロパティ (Loader クラス) 742
contentLoaderInfo プロパティ 401、752
contentType プロパティ 617
cookie 635
createBox() メソッド 420
createGradientBox() メソッド 434
CSS
　スタイル 497
　定義 487
　ロード 499
currentDomain プロパティ 752
currentTarget プロパティ 311

D

data プロパティ (URLRequest クラス) 618
dataFormat プロパティ 622
Date オブジェクト
　作成例 181
　時間値の取得 182
Date クラス
　date プロパティ 182
　day プロパティ 182
　fullYear プロパティ 182
　getMonth() メソッド 141、182
　getMonthUTC() メソッド 182
　getTime() メソッド 182
　getTimezoneOffset() メソッド 183
　hours プロパティ 182
　milliseconds プロパティ 182
　minutes プロパティ 182
　month プロパティ 182
　monthUTC プロパティ 182
　parse() メソッド 141
　seconds プロパティ 182
　setTime() メソッド 182
　コンストラクタ 181
　説明 179
date プロパティ 182
Date() コンストラクタ 181
day プロパティ 182
decode() メソッド 618
default xml namespace ディレクティブ 346
Delegate クラス 315
delete 演算子 118、217
Dictionary クラス
　useWeakReference パラメータ 226
　説明 224
dispatchEvent メソッド 319
DisplayObject クラス
　stage プロパティ 306
　サブクラス 358
　説明 355、363
DisplayObjectContainer クラス 355、360、364
displayState プロパティ 372、737
distance() メソッド 414
do..while ループ 114
DOM Events 仕様 299、304
domain プロパティ (LocalConnection クラス) 753
download() メソッド 736、751
draw() メソッド 402、739、742、746、748
dynamic 属性 133

E

E4X、「XML」を参照
ECMAScript Edition 4 草案 64
ECMAScript for XML、「XML」を参照
ECMAScript のコア Error クラス 260
Endian.BIG_ENDIAN 630
Endian.LITTLE_ENDIAN 630
enterFrame イベント 307
Error クラス
　ActionScript 262
　ECMAScript 260
　説明 259
ErrorEvent クラス 256、320
Event クラス
　bubbles プロパティ 309
　cancelable プロパティ 308
　clone() メソッド 311
　currentTarget プロパティ 311
　eventPhase プロパティ 310
　isDefaultPrevented() メソッド 312
　preventDefault() メソッド 304、312
　stopImmediatePropogation() メソッド 311
　stopPropogation() メソッド 311
　target プロパティ 310
　toString() メソッド 311
　type プロパティ 308
　サブクラス 312
　説明 308
　定数 308
　メソッドの種類 311
Event.COMPLETE 617
EventDispatcher クラス
　addEventListener() メソッド 145、305
　dispatchEvent メソッド 319
　IEventDispatch インターフェイス 149
　willTrigger() メソッド 319
　参照 97
eventPhase プロパティ 310
exactSettings プロパティ (Security クラス) 753
exec() メソッド 292
extends キーワード 152
External API
　XML フォーマット 703
　一般的なタスク 696
　概念と用語 696
　説明 696
　利点 699
　例 705
ExternalInterface クラス 699、735、754
ExternalInterface.addCallback() メソッド 743

F

FileReference クラス 639、736、751
FileReferenceList クラス 647、751
final 属性 85、143、146、156
Flash cookie 635
Flash Media Server 742
Flash Player
　IME 670
　インスタンス間の通信 623
　エンコードされた FLV との互換性 557
　デバッガバージョン 320
　バージョン 6 161
Flash Video、「FLV」を参照
Flash オーサリング、ActionScript に使用する場合 47
Flash タイムライン、ActionScript の追加 45
flash パッケージ 67
Flash マニュアル 15
flash.display パッケージ
　サウンド 565
　説明、表示のプログラミング 353
　テキスト 485
　ビットマップ 515
　描画 API 427
　フィルタ処理 443
　ムービークリップ 469
　ユーザー入力 599
flash.geom パッケージ 411
flash_proxy 名前空間 74
Flex、ActionScript に使用する場合 47
FLV
　Flash Player 557
　Macintosh 558
　サーバーでホスティングするための設定 557
　ファイル形式 533
for each..in ステートメント 113、225、345
for ループ 112
for ループ、XML 333、345
for..in ステートメント 112、225、345
frameRate プロパティ 370
fromCharCode() メソッド 196
fscommand() 関数 623、735、754
fullScreen イベント 372
fullYear プロパティ 182
function キーワード 116、139
function ステートメント 116
Function.apply() メソッド 231

G

g フラグ (正規表現) 288
GeometricShapes のサンプル 170
getDefinition() メソッド 752
getImageReference() メソッド 741
getLocal() メソッド 635、736、753、755
getMonth() メソッド 141、182
getMonthUTC() メソッド 182
getRect() メソッド 419
getRemote() メソッド 635、736、755
getter および setter
　オーバーライド 158
　説明 143
getter および setter のオーバーライド 158
getTime() メソッド 182
getTimer() 関数 186
getTimezoneOffset() メソッド 183
GIF グラフィック 400

H

hours プロパティ 182
HTML テキスト
　CSS 497
　表示 490
htmlText プロパティ 490
HTTP トンネリング 631

I

i フラグ (正規表現) 288
id3 プロパティ 747
IDataInput および IDataOutput インターフェイス 630
IEventDispatcher インターフェイス 148、317
if ステートメント 109
if..else ステートメント 109
IME
　Flash Player 内での操作 670
　入力イベント 674
　有効かどうかの確認 671
IME 変換モード
　設定 672
　判別 671
import ステートメント 69
indexOf() メソッド 199
init イベント 307
instanceof 演算子 87
int クラス、キャスト 93
int データ型 89

InteractiveObject クラス 360
internal 属性 69、71、137
intersection() メソッド 418
intersects() メソッド 419
is 演算子 86、149
isDefaultPrevented() メソッド 312
isNaN() グローバル関数 81

J

Java ソケットサーバー 632
join() メソッド 221
JPG グラフィック 400

L

Landscape (横方向) でのプリント 689
lastIndexOf() メソッド 199
length プロパティ
　arguments オブジェクト 122
　Array クラス 217
　ストリング 195
level プロパティ 320
lineGradientStyle() メソッド 434
load() メソッド (Loader クラス) 402、730、736
load() メソッド (Sound クラス) 730、736、741、751
load() メソッド (URLLoader クラス) 617、736
load() メソッド (URLStream クラス) 736、751
loadBytes() メソッド 402、730
Loader クラス 400、736、747、752
LoaderContext オブジェクト 730
LoaderContext クラス 402、739、747
LoaderInfo クラス
　表示オブジェクトへのアクセス 745
　ロードの進行状況の監視 401
loaderInfo プロパティ 401
loadPolicyFile() メソッド 736
LocalConnection クラス
　connectionName パラメータ 628
　許可 753
　制限 736
　説明 623
LocalConnection.allowDomain() メソッド 628、753
LocalConnection.allowInsecureDomain() メソッド 628
LocalConnection.client プロパティ 624、625
LocalConnection.connect() メソッド 736
localFileReadDisable プロパティ 723
localToGlobal() メソッド 414

M

m フラグ (正規表現) 288
Macintosh、FLV ファイル 558
match() メソッド 201
Matrix クラス
　オブジェクト、定義 420
　回転 420
　拡大 / 縮小 420
　グラデーションの定義 434
　傾斜 421
　定義 420
　変換 420
　例 421
MAX_VALUE (Number クラス) 90
method プロパティ (URLRequest クラス) 618
Microphone クラス 310
milliseconds プロパティ 182
MIN_VALUE (Number クラス) 90
minutes プロパティ 182
month プロパティ 182
monthUTC プロパティ 182
MorphShape クラス 360
MouseEvent クラス 305、312
movie clips
　caching 389
MovieClip オブジェクト、作成 475
MovieClip クラス 359
　フレームレート 370
mx.util.Delegate クラス 315

N

NaN 値 90
navigateToURL() 関数 735、754
NetConnection クラス 736
NetConnection.connect() メソッド 736、741
NetStream クラス 730、736、741
new 演算子 65
null 値 81、89、91、225
Number クラス
　isNaN() グローバル関数 81
　キャスト 93
　整数範囲 90
　精度 90
　デフォルト値 81
Number データ型 89

O

Object クラス
　valueOf() メソッド 167
　結合配列 223
　データ型 91
　プロトタイププロパティ 163、167
on() イベントハンドラ 303
onClipEvent() 関数 303
onCuePoint イベントハンドラ 539
override キーワード 143、144

P

package ステートメント 133
parentAllowsChild プロパティ 745
parse() メソッド 141
play() メソッド (NetStream クラス) 736
PNG グラフィック 400
Point オブジェクト
　座標空間の変換 414
　説明 414
　その他の使用 415
　ポイント間の距離 414
polar() メソッド 415
pop() メソッド 216
Portrait (縦方向) でのプリント 689
preventDefault() メソッド 304、312
printArea パラメータ 686
PrintJob ステートメント、タイミング 687
PrintJob() コンストラクタ 683
priority パラメータ、addEventListener() メソッド 318
private コンストラクタ、サポートされていない 140
private 属性 135
ProgressEvent.PROGRESS 617
protected 属性 137
Proxy クラス 74
public クラス 70
public 属性 135
push() メソッド 216、232

R

Real-Tiime Messaging Protocol コンテンツのセキュリティ 742
Rectangle オブジェクト
　結合 418
　交差 418
　サイズ変更 416
　再配置 416
　その他の使用 419
　定義 416
　プリント 688
RegExp クラス
　説明 273
　プロパティ 288
　メソッド 292
replace() メソッド 186、202
rest パラメータ 124
return ステートメント 119、141
reverse() メソッド 218
rotate() メソッド 420
RSS データ
　ポッドキャストチャンネル用のデータの読み取り 592
　ロード、例 349
RTMP コンテンツセキュリティ 742

S

s フラグ (正規表現) 288
sameDomain プロパティ 745
scale() メソッド 420
sealed クラス 87
search() メソッド 201
seconds プロパティ 182
Security クラス 736
Security.allowDomain() メソッド
　img タグ 741
　クロススクリプトの説明 742
　コンストラクタ 752
　コンテキストのロード 402
　サウンド 748
Security.currentDomain プロパティ 752
Security.exactSettings プロパティ 753
SecurityDomain クラス 403、740
send() メソッド (LocalConnection クラス) 624、736
sendToURL() 関数 736、751
Server、Flash Media 742
setClipboard() メソッド 757
setInterval() 関数 186
setter、「getter および setter」を参照
setTime() メソッド 182
setTimeout() メソッド 186
Shape クラス 359
SharedObject クラス 635、736
SharedObject.getLocal() メソッド 753、755
SharedObject.getRemote() メソッド 755
shift() メソッド 216
significand 90

SimpleButton クラス 359
SimpleClock の例 186
slice() メソッド
　Array クラス 221
　String クラス 199
Socket クラス 630、736、749
Sound クラス 730、736、741
SoundFacade クラス 592
SoundLoaderContext クラス 730
SoundMixer.computeSpectrum() メソッド 742、746、747
SoundMixer.stopAll() メソッド 747
splice() メソッド 216
split() メソッド 200
Sprite クラス 359
SpriteArranger クラスの例 405
Sprite、最初にロードされる 355、401
Stage
　説明 354
　表示オブジェクトコンテナ 356
Stage クラス 306
StageDisplayState クラス 737
standard モード 85、117
static 属性 137
StaticText クラス 360
stopAll() メソッド (SoundMixer クラス) 747
stopImmediatePropogation() メソッド 311
stopPropogation() メソッド 311
strict モード
　キャスト 93
　説明 83
　ドットシンタックス 117
　明示的な変換 93
　戻り値 119
　ランタイムエラー 85
String クラス
　charAt() メソッド 196
　charCodeAt() メソッド 196
　concat() メソッド 198
　fromCharCode() メソッド 196
　indexOf() メソッド 199
　lastIndexOf() メソッド 199
　match() メソッド 201
　replace() メソッド 202
　search() メソッド 201
　slice() メソッド 199
　split() メソッド 200
　substr() および substring() メソッド 199
　toLowerCase() および toUpperCase() メソッド 203
String データ型 90
strings
　about 192
StyleSheet クラス 497
substr() および substring() メソッド 199
super ステートメント 140、142、157
SWF ファイル
　インスタンス間の通信 626
　外部ロード 479
　旧バージョンのロード 480
　実行時環境の判別 665
　ドメイン間の通信 628
　ロード 400
　ロードされたファイルの読み込み 752
switch ステートメント 111
System.setClipboard() メソッド 757

T

target プロパティ 310
Telnet クライアントの例 649
test() メソッド 292
TextEvent クラス 304
TextField クラス 304、359
TextFormat クラス 497
TextLineMetrics クラス 512
TextSnapshot クラス 505
this キーワード 142、143、145、315
throw ステートメント 252
Timer クラス
　再生の監視 597
　説明 184
toLowerCase() メソッド 203
toString() メソッド
　Array クラス 221
　Event クラス 311
　説明 197
toUppercase() メソッド 203
Transform クラス
translate() メソッド 420
try..catch..finally ステートメント 250
twip 688
type プロパティ (Event クラス) 308

U

UIEventDispatcher クラス 304
uint クラス、キャスト 93
uint データ型 90、91
undefined 65、91、215

Unicode 文字 191
union() メソッド 418
unshift() メソッド 216
upload() メソッド 736、751
URI 72
URI (Uniform Resource Identifier) 72
URL エンコード 618
URLLoader クラス
　XML データのロード 338、349
　制限 736
　セキュリティ 749
　説明 617
URLLoader コンストラクタ 617
URLLoader.dataFormat プロパティ 622
URLLoader.load() メソッド 617、618
URLLoaderDataFormat.VARIABLES 622
URLRequest インスタンス 617、618
URLRequest.contentType プロパティ 617
URLRequest.data プロパティ 618
URLRequest.method プロパティ 618
URLRequestMethod.GET 619
URLRequestMethod.POST 619
URLStream クラス 736、749
URLVariables クラス 617
URLVariables.decode() メソッド 618
use namespace ディレクティブ 73、76、169
useCapture パラメータ、addEventListener() メソッド 318
useWeakReference パラメータ 226
UTC (世界標準時) 180

V

valueOf() メソッド (Object クラス) 167
var キーワード 78、138
Video クラス 534
void 91

W

while ループ 114
Wiki パーサーの例 293
willTrigger() メソッド 319
WordSearch の例 608

X

x フラグ (正規表現) 288
XML
　ActionScript 330
　E4X (ECMAScript for XML) 67、327、332
　External API のフォーマット 703
　for each..in ループ 113
　for ループ 333、345
　一般的なタスク 330
　親ノード 341
　概念と用語 331
　型変換 347
　基本概念 328
　空白 334
　構造の移動 340
　子ノード 341
　コメント 334
　処理命令 334
　属性へのアクセス 342
　ソケットサーバー 632
　中括弧演算子 ({ および }) 339
　データのロード 338、349
　ドキュメント 329
　名前空間 346
　フィルタ 343
　プロパティ 334
　変形 339
　変数の初期化 338
　メソッド 335
XML クラス 67
XML データのフィルタ処理 343
XML の中括弧演算子 ({ および }) 339
XMLDocument クラス 67、332
XMLList オブジェクト
　説明 336
　連結 340
XMLNode クラス 332
XMLParser クラス 332
XMLSocket クラス 339、349、630、736、749
XMLSocket.connect() メソッド 631、736
XMLTag クラス 332

あ

アクセサ関数、get および set 143
アクティベーションオブジェクト 127
アスタリスク (*) 型注釈 81、84、91
アスタリスク (*) メタ文字 279
アスタリスク (ワイルドカード) 演算子、XML 342

値
　引数の渡し 121
　変数への割り当て 78
アップキャスト 85
アニメーション 398
アプリケーション、開発の決定 44
アルファチャンネルマスク 396
アンチエイリアス、テキスト 503
アンパサンド (&) 618
暗黙的な型変換 92

い

位置
　ストリング内の文字 199
　表示オブジェクト 376
イベント
　「イベントリスナー」も参照
　enterFrame イベント 307
　error 256、319
　init イベント 307
　this キーワード 315
　イベントオブジェクト 308
　イベントフロー 300、306、309
　親ノード 307
　基本的な概念 31
　ステータス変化 258
　セキュリティ 745
　送出 300、319
　ターゲットノード 306
　デフォルト動作 304
　表示オブジェクト 374
イベントオブジェクト 300
イベントターゲット 300、306
イベントの送出 300
イベントハンドラ 303、539
イベントフロー 300、306、309
イベントリスナー
　ActionScript 3.0 の変更点 305
　管理 317
　クラス外 313
　クラスメソッドとして 314
　削除 318
　作成 313
　避けるべき手法 315
　説明 300
イメージ
　Bitmap クラスでの定義 359
　セキュリティ 746
　テキストフィールド 491
　フィルタ処理の例 468
　ロード 400
インクリメント、値 105
インスタンスプロパティ
　継承 153
　宣言 134
インスタンス変数 139
インスタンスメソッド 142
インスタンス、作成 37
インターフェイス
　拡張 150
　クラス内での実装 150
　説明 148
　定義 149
インデックス配列 214
引用符 194、195

う

埋め込みアセットクラス 148
埋め込みフォント
　使用 501
　定義 487

え

エラー
　ErrorEvent クラス 256、320
　print 684
　throw ステートメント 252
　再スロー 254
　処理について 242
　ステータスに基づくイベント 256
　タイプ 242、245
　デバッグツール 249
　独自のクラス 255
　非同期 245
　表示 253
エラーイベント 256、319
エラー処理
　一般的なタスク 243
　戦略 248
　ツール 247
　デフォルト動作 304
　用語 243
　例 320
円記号 (\)
　ストリング内 195
　正規表現 279

エンコード、アンパサンド (&) 618
› 演算子 103、197
‹ 演算子 103
演算子
　加算 107
　関係 107
　基本 105
　基本的な概念 39
　後置 105
　条件 108
　乗法 106
　説明 102
　前置 106
　代入 109
　単項 102、106
　等価 108、197
　ビット単位シフト 107
　ビット論理 108
　優先順位 103
　論理 108

お

オーディオ再生の監視 597
オーディオ再生の進行状況 597
オーディオのセキュリティ 747
オーバーロードされた演算子 102
大文字と小文字の置き換え、ストリング内 203
大文字と小文字の区別 97
オブジェクト
　インスタンス化 37
　基本的な概念 28
オブジェクト指向プログラミング
　一般的なタスク 130
　概念 131
オブジェクトのストリング表現 197
オブジェクトリテラル 223

か

カーソル、カスタマイズ 605
改行文字 195
開始角括弧 ([) 279
開始括弧 279
開発
　計画 44
　プロセス 49
外部 SWF ファイル、ロード 479
外部コード、ActionScript からの呼び出し 701
外部コンテナ、情報の取得 700
外部データ、ロード 617
外部ドキュメント、データのロード 619
改ページ文字 195
角括弧 ([および]) 文字 279
拡大 / 縮小
　ステージ 370
　歪みの制御 385
　表示オブジェクト 420
　プリント 689
　マトリックス 420
加算 (+) 演算子 198
加算演算子 107
加算後代入 (+=) 演算子 198
仮数 90
カスケーディングスタイルシート、「CSS」を参照
カスタム LocalConnection クライアント 625
カスタムデータ型、列挙 146
型指定されていない変数 65、81
型チェック
　コンパイル時 83
　実行時 85
型注釈 78、83
型付き配列 214
型の不一致 84
型変換 92、93、347
型、「データ型」を参照
括弧
　XML フィルタ処理演算子 343
　演算子 99
　空 115
　メタ文字 279
括弧 ({ および }) 演算子 117
ガベージコレクション 118、225
カメラ
　インストールの確認 551
　許可 552
　コンテンツを画面上に表示 550
　再生の状況 555
　セキュリティ 753、757
　入力のキャプチャ 549
カメラコンテンツを画面上に表示 550
カメラ入力のキャプチャ 549
カラー
　異なるイメージからの組み合わせ 390
　特定のカラーの変更 393
　背景 390
　表示オブジェクトでの調整 391
　表示オブジェクトの設定 392
関係演算子 107

関数
arguments オブジェクト 120
アクセサ 143
オブジェクト 125
括弧 115
再帰 123
スコープ 119、127
説明 115
タイミング 186
匿名 116、123
ネストされた 120、127
パラメータ 120
プロパティの追加 126
戻り値 119
呼び出し 115
関数オブジェクト 132
関数クロージャ 115、120、127
関数式 116
関数パラメータ 120
カンマ演算子 78

き

キーコード 602
キーボードセキュリティ 757
キーボード入力、キャプチャ 601
キーワード 100
キー、ストリング 223
基本演算子 105
基本クラス 152
基本的な概念
イベント 31
演算子 39
オブジェクト 28
オブジェクトインスタンスの作成 37
コメント 39
フロー制御 40
プロパティ 29
変数 26
メソッド 30
例 41
疑問符 (\?) メタ文字 279
キャスト 92、93、95
キャプチャ段階 306
キャレット (^) 文字 279
キューポイント
アクションのトリガー 539
ビデオ 538
共有オブジェクト
Flash Player の設定 753
コンテンツの表示 637
セキュリティ 637、755
説明 635
許可
LocalConnection クラス 753
カメラ 552

く

空白 334
区切り記号、ストリングを分割して配列を作成 200
クライアントのシステム環境
一般的なタスク 662
説明 662
クラス
base 152
dynamic 属性 133
internal 属性 137
private 属性 135
protected 属性 137
public クラス 70
public 属性 135
sealed 87
インスタンスプロパティの継承 153
カスタムクラスの作成 50
構成 53
コードの作成について 51
最上位のステートメント 134
サブクラス 152
サポートされていない抽象 133
静的プロパティ 158
静的プロパティとインスタンスプロパティを宣言 134
説明 132
属性 133
ダイナミック 87、118、136
定義 132
デフォルトのアクセス制御 137
特性 29
内での名前空間の定義 134
ビルトイン 65
プライベートクラス 66
プロパティ属性 135
本体 134
クラスオブジェクト 64、165
クラス継承 166
クラス定義、複数 666
クラスパス 69
グラデーション 434

グラフィック、ロード 400
繰り返し制御文字 (正規表現) 283
クリップボード
　セキュリティ 757
　テキストの保存 665
グループ、正規表現 285
グローバルオブジェクト 127
グローバルスコープ 127
グローバル変数 79
クロススクリプト 742
クロスドメインポリシーファイル 749
　checkPolicyFile プロパティ 402、747
　img タグ 741
　securityDomain プロパティ 740
　URLLoader および URLStream クラス 749
　データの抽出 746

け

継承
　インスタンスプロパティ 153
　固定プロパティ 166
　静的プロパティ 158
　定義 152
結合性、規則 103
検索、ストリング 201
検索、正規表現 291
厳密な不等価 (!==) 演算子 197

こ

後置演算子 105
コード、アプリケーションにインクルードする方法 44
コールバックメソッド
　処理 541
互換性、Flash Player と FLV ファイル 557
固定プロパティの継承 166
コメント
　XML 334
　説明 39、100
コロン (:) 演算子 83
コンストラクタ
　ActionScript 1.0 162
　説明 140
コンテキストメニュー、カスタマイズ 606
コンテンツ、動的ロード 400
コンパイラオプション 169、231
コンパイル時の型チェック 83
語、予約 100

さ

サーバーサイドスクリプト 622
再帰関数 123
最初にロードされる Sprite 355、401
再生
　オーディオの一時停止と再開 597
　オーディオの監視 597
　カメラ 555
　ビデオ 535
　フレームレートの制御 370
　ムービークリップ 472
サウンド
　サーバーとの送受信 590
　サンプルアプリケーション 591
　セキュリティ 741、747
座標空間
　定義 412
　変換 414
サブクラス 152
サブストリング
　区切り記号に基づいて作成 200
　検索と置換 199、200
　正規表現での一致 286
　説明 198
3 項演算子 102
参照、渡し 120

し

シーン、タイムラインを区切る 475
ジオメトリ
　一般的なタスク 412
　概念と用語 413、428
　説明 411
時間間隔 184
時間形式 180
時間単位の値 182
識別子 71
システム、ユーザーのシステムの判別 664
子孫アクセサ (..) 演算子、XML 340
実行時、ユーザーのシステムの判別 664
弱参照 226
シャドウ 160
終了角括弧 279
終了括弧 279
条件 (?:) 演算子 108
条件演算 109
乗法演算子 106
ショートカットメニュー (コンテキストメニュー) 606

シンタックス 97
シンタックスのキーワード 101
深度管理、向上 361

す

スーパークラス 152
スクリプトタイムアウトの制限 687
スコープ
　関数 119、127
　グローバル 127
　ブロックレベル 80
　変数 79
スコープチェーン 127、159
スタイルシート、「CSS」を参照
ステージ
　拡大 / 縮小 370
　セキュリティ 744
　説明 306
　プロパティ、設定 370
ステージ所有者 744
ステータスに基づくエラーイベント 256
ステータス変化イベント 258
ステートメントの終了 99
ストリーミングビデオ 538
ストリング
　length 195
　XML オブジェクトの変換 347
　XML 属性のデータ型の変換 348
　一般的なタスク 192
　インデックス位置 196
　大文字と小文字の変換 203
　サブストリング 198、200
　サブストリングの一致 286
　サブストリングの検索 199
　正規表現内の一致のチェック 292
　宣言 194
　テキストの置換 200
　パターン、検索 198、200
　比較 197
　文字で区切られたストリングへの配列の結合 238
　文字の位置 199
　用語 193
　例 203、204
　連結 198
ストリングキー 223
ストリング内のインデックス位置 196
ストリング内の二重引用符 194、195
ストリング内の一重引用符 194、195
スピーカーとマイク 588
スラッシュ 277、279
　円記号 (\) 195、279
　スラッシュ (/) 277、279
スラッシュシンタックス 98

せ

正規表現
　String メソッドのパラメータ 293
　繰り返し制御文字 283
　グループ 285
　検索 291
　作成 277
　サブストリングの一致のキャプチャ 286
　使用するメソッド 292
　スラッシュ区切り 277
　説明 274
　選択制御文字と文字グループ 286
　名前付きグループ 287
　パイプ (|) メタ文字を使用した選択制御 284
　フラグ 288
　プロパティ 288
　メタシーケンス 278、280
　メタ文字 278、279
　文字 278
　文字クラス 281
　例 293
正規表現での選択制御 284
正規表現内の dotall フラグ 290
正規表現内の extended フラグ 290
正規表現内の multiline フラグ 289
正規表現内の記号 278
正規表現内のグローバルフラグ 289
正規表現内の非キャプチャグループ 286
正規表現内の無視フラグ 289
正規表現の dotall プロパティ 288
正規表現の extended プロパティ 288
正規表現の global プロパティ 288
正規表現の ignoreCase プロパティ 288
正規表現の multiline プロパティ 288
正規表現のフラグ 288
静止テキスト
　アクセス 504
　作成 360
静止テキストフィールド 487
静的プロパティ
　XML 334
　継承 158
　スコープチェーン内 159
　宣言 134

静的変数 138
静的メソッド 141
正の無限大 90
世界標準時 (UTC) 180
セキュリティ
「クロスドメインポリシーファイル」も参照
allowNetworking タグ 735
img タグ 741
LocalConnection クラス 753
RTMP 742
URLLoader 749
URLStream 749
イベント関連 745
イメージ 746
カメラ 753、757
キーボード 757
共有オブジェクト 753、755
クリップボード 757
サウンド 741、747
ステージ 744
ソケット 749
データの送信 751
ビットマップ 746
ビデオ 741、748
表示リスト 745
ファイル、アップロードとダウンロード 751
フルスクリーンモード 737
ポート 749
マイク 753、757
マウス 757
読み込まれた SWF ファイル 752
ロードされたメディアへのデータとしてのアクセス 746
セミコロン 99
0 による除算 90
前置演算子 106

そ

ソースパス 69
ソートされていない配列 222
属性識別用 (@) 演算子、XML 333、342
速度、レンダリングの向上 388
ソケットサーバー 632
ソケットの接続 629

た

ターゲットノードまたは段階 306
ダイナミッククラス 87、118、136
ダイナミックテキストフィールド 487
代入演算子 109
タイマー 184
タイマーイベント 184
タイミング関数 186
タイムアウト制限 687
タイムゾーン 181、183
タイムライン、Flash 45
タブ文字 195
単項演算子 102、106

ち

置換コード 202
置換、ストリング内のテキスト 200
抽象クラス 133

つ

通信
Flash Player インスタンス間 623
SWF ファイル間 626
異なるドメインの SWF ファイル間 628

て

定数 101、138、308
データ
外部ロード 617
サーバーへの送信 622
セキュリティ 746、751
データ型
Boolean 89
int 89
void 91
カスタム 146
数値 89
ストリング 90
説明 27
単純および複雑 27
定義 82
デフォルト (型指定なし) 65
ユニット 90
データ構造 212
データの格納 635

テキスト
アンチエイリアス 503
一般的なタスク 487
概念と用語 487
クリップボードへの保存 665
シャープネス 503
使用可能なタイプ 489
スクロール 491、493
静的 360、504
説明 486
選択 493
操作 493
置換 200
入力のキャプチャ 494
入力の制限 496
範囲のフォーマット 500
表示 489
フォーマット 496、505
フォーマットの割り当て 497
太さ 503
テキストエディタ 48
テキスト行のメトリック 487、512
テキストのスクロール 491、493
テキストのフォーマット 496、500
テキストフィールド
HTML 497
IME の無効化 673
img タグとセキュリティ 741
イメージ 491
静止 487
ダイナミック 487
テキストのスクロール 491
入力 487
変更 489
テキストフィールドの img タグ 741
デクリメント、値 105
デバイスフォント 487
デバッガバージョン、Flash Player 320
デバッグ 249
デフォルト動作
キャンセル 309
定義 304
デフォルトのデータ型 65
デフォルトのパラメータ値 122

と

等価演算子 108、197
同期エラー 245
ドキュメントオブジェクトモデル (DOM) Level 3 Events 仕様 299、304
独自のエラークラス 255
独自のクラス 50
特性オブジェクト 166
匿名関数 116、123
時計の例 186
ドット (.) 演算子 97、117
ドット (.) 演算子、XML 333、340
ドット (.) メタ文字 279
ドットシンタックス 97
ドメイン、通信 628
ドラッグ & ドロップ
操作のキャプチャ 604
操作の作成 378
ドル記号 ($) 置換コード 202
ドル記号 ($) メタ文字 279

な

名前空間
AS3 168、231
flash_proxy 74
namespace キーワード 71
use namespace ディレクティブ 73、76、169
XML 346
アクセス制御指定子 72
参照 73
説明 71
定義 72、134
適用 73
デフォルトの名前空間 71
開く 73
ユーザー定義の属性 138
読み込み 76
名前付きグループ (正規表現) 287
名前の競合、回避 67、70

に

二項演算子 102
入力テキストフィールド 487
任意パラメータ 122

ね

ネストされた関数 120、127
ネストされたパッケージ 67
ネットワーク
　概念と用語 615
　制限 735
　説明 614
ネットワークバイト順序 630

の

ノード、XML でのアクセス 341

は

バー (|) 文字 284
背景色、不透明にする 390
バイト順序 630
パイプ (|) 文字 284
配列
　delete 演算子 217
　浅いコピー 229
　一般的なタスク 213
　インデックス付き 214
　オブジェクトキー 224
　キーと値のペア 223
　クエリ 221
　繰り返し処理 225
　結合 222
　結合配列およびインデックス付き配列の使用 228
　コピー 229
　コンストラクタ 215
　最大サイズ 214
　作成 200、215
　サポートされない型付き配列 214
　スーパーコンストラクタ 231
　説明 212
　ソート 218
　多次元 226、227
　長さ 217
　ネストされた配列と join() メソッド 222
　配列リテラル 98、215
　深いコピー 229
　用語 213
　要素の削除 216
　要素の挿入 216
　例 235
配列内のオブジェクトキー 224
配列の繰り返し処理 225
配列のスーパーコンストラクタ 231
配列のソート 218、219
バインドメソッド 128、144
8 進数 94
パッケージ
　最上位レベル 66、68
　作成 68
　説明 66
　ドット演算子 67、97
　ドットシンタックス 97
　ネストされたパッケージ 67
　読み込み 68
ハッシュ 222、224
パフォーマンス、表示オブジェクトの向上 387
バブリング段階 306
パラメータ
　値渡しと参照渡し 120
　任意または必須 122
汎用オブジェト 98、223

ひ

引数、参照渡しまたは値渡し 120
ピクセルの吸着 520
ピクセルレベルの衝突検出 523
ピクセル、個々の操作 521
ビジュアルオブジェクト、「表示オブジェクト」を参照
左角括弧 279
左括弧 279
左結合演算子 103
ビッグエンディアンバイト順序 630
日付と時刻
　説明 179
　例 180
日付の演算 183
必須パラメータ 122
ビット単位シフト演算子 107
ビットマップ
　Bitmap クラスの定義 359
　スムージング 520
　セキュリティ 746
　説明 516
　透明と不透明 517
　ファイル形式 516
ビットマップキャッシュ
　避けるべきシナリオ 389
　使用するシナリオ 388
　フィルタ 448
　利点と欠点 388
ビットマップ形式でのプリント 686

ビットマップデータ、コピー 524
ビットマップのスムージング 520
ビット論理演算子 108
ビデオ
Macintosh 558
quality 554
一般的なタスク 530
サーバーへの送信 556
再生 535
ストリーミング 538
ストリームの終端 537
セキュリティ 741、748
説明 530
メタデータ 546、548
ロード 534
ビデオジュークボックスの例 559
非同期エラー 245
非同期処理 319
表示アーキテクチャ 354、428
表示オブジェクト
size 383
アニメーション化 398
一般的なタスク 356
イベント 374
回転 394、420
拡大 / 縮小 383、385、420
型 358
カラーの設定 392
カラーの調整 391
キャッシュ 387
クリックとドラッグの例 409
グループ化 364
傾斜 420
コアクラスの継承 358
再配置の例 410
作成 363
サブクラス化 362
サブクラスの選択 375
深度管理 361
セキュリティ 745
説明 355
配置 376、377
ビットマップ 515
描画 API 427
表示リストへの追加 364
フィルタ 443、446、450
フィルタの削除 447
フェード 394
複雑なオブジェクトの構成 362
変換 420
マスク 395
マトリックスの変換 421
ムービークリップ 469
ユーザー入力 599
用語 357
リスト外 362
例 404、440
表示オブジェクトコンテナ 355、364
表示オブジェクトの回転 394、420
表示オブジェクトのグループ化 364
表示オブジェクトの傾斜 420
表示オブジェクトのマスク 395
表示コンテンツ、動的ロード 400
表示のプログラミング、説明 354
表示リスト
移動 368
イベントフロー 306
セキュリティ 745
説明 354
利点 360
表示リストオブジェクト 304
ピリオド (.)、「ドット」を参照
ビルトインクラス 65
ビルドパス 69

ふ

ファイル
アップロード 647、751
ダウンロード 751
ファイルサイズ、シェイプの場合の縮小 361
ファイルのアップロード 641、647、751
ファイルのダウンロード 645、751
ファサードクラス 592
フィルタ
BitmapData オブジェクトへの適用 448
一般的なタスク 444
イメージ、例 468
作成 445
実行時の変更 449
説明 448
ビットマップキャッシュ 448
表示オブジェクトでの削除 447
表示オブジェクトとビットマップオブジェクト用 450
表示オブジェクトへの適用 446
フィルタとビットマップのキャッシュ 448
フェード、表示オブジェクト 394
フォーカス、操作での管理 607

フォント
埋め込み 487、501
デバイス 487
複合値 82
複合リテラル 98
複数のクラス定義 666
不等価 (!=) 演算子 197
不透明な背景 390
負の無限大 90
プライベートクラス 66
プラス記号 (+) 279
プリミティブ型、明示的な変換 92
プリミティブ値 65、82
プリント
Rectangle オブジェト 688
一般的なタスク 682
概念と用語 682
拡大 / 縮小 689
説明 682
タイムアウト 687
範囲の指定 688
複数ページ、例 690
ページ 683
ページの高さと幅 689
ページプロパティ 686
ベクターまたはビットマップ 686
ポイント 688
向き 689
例外と戻り値 684
フルスクリーンモード 372、373、737
フレーム、ジャンプ 474
プレーヤー、「Flash Player」を参照
フロー制御、基本的な概念 40
プログラムフロー 109
プログラム、基本的な定義 26
ブロックレベルのスコープ 80
プロトタイプオブジェクト 118、163、166
プロトタイプチェーン 64、163
プロトタイププロパティ 163、167
プロパティ
ActionScript と他の言語の比較 64
XML 334
関数への追加 126
基本的な概念 29
正規表現 288
静的およびインスタンス 134、158
定義、ActionScript 3.0 135
プロパティアクセス演算子 224

へ

ページプロパティ 686
ベクター形式でのプリント 686
変換プロパティ 420、421
変換マトリックス Matrix クラスを参照
変数
var ステートメント 78
インスタンス 139
型 138
型指定されていない 65、81
型注釈 78、83
基本的な概念 26
許可されないオーバーライド 139
初期化 81、338
初期化されていない 81
スコープ 79
静的 138
宣言 138
デフォルト値 81

ほ

ホイスト 80
ポインタ (カーソル)、カスタマイズ 605
ポイントとピクセル 688
ポート、セキュリティ 749
星 (*)、「アスタリスク」を参照
ポッドキャストアプリケーション
拡張 598
作成 591
ポリモーフィズム 152

ま

マイク
アクセス 586
アクティビティの検知 589
セキュリティ 753、757
ローカルスピーカーへの転送 588
マウスカーソル、カスタマイズ 605
マウスのセキュリティ 757
マスク、アルファチャンネル 396
マップ 222、224
マトリックスの回転 420
マトリックスの傾斜 420、421
マトリックスの変換 420

マニュアル
ActionScript 14
ActionScript 3.0 のプログラミングの目次 13
Flash 15
アドビデベロッパーセンターおよびアドビデザインセンター 16

み

右括弧 279
右クリックメニュー (コンテキストメニュー) 606
右結合演算子 103

む

ムービークリップ
一般的なタスク 470
概念と用語 471
再生と停止 473
説明 470
早送り 473
フレームレート 370
巻き戻し 473
ムービークリップの停止 473
ムービークリップの早送り 473
ムービークリップの巻き戻し 473
無限大 90

め

明示的な型変換 92
メソッド
getter および setter 143、158
インスタンス 142
オーバーライド 156
基本的な概念 30
コンストラクタ 140
静的 141
定義 139
バインド 128、144
メタシーケンス、正規表現の 278、280
メタデータ、ビデオ 546、548
メタ文字、正規表現の 278
メモリ管理 225

も

文字
ストリング内 196、199
正規表現 278
文字クラス (正規表現) 281
文字クラス内のエスケープシーケンス 281
文字クラスの補集合 (正規表現) 282
文字コード 602
文字で区切られたストリング、配列を組み合わせる例 238
文字範囲、指定 282
モニタ、フルスクリーンモード 372

ゆ

ユーザー選択テキストのキャプチャ 494
ユーザー選択テキスト、キャプチャ 494
ユーザー操作、フォーカスの管理 607
ユーザー入力
一般的なタスク 600
概念と用語 600
説明 599
ユーザーのシステム、実行時の判別 664

よ

読み込まれた SWF ファイル 752
予約語 101
より大きい演算子 103、197
より大きいか等しい演算子 197
より小さい演算子 103、197
より小さいか等しい演算子 197

ら

ライブラリシンボルの書き出し 476
ライブラリシンボル、書き出し 476
ラッパーオブジェクト 82

り

リスト外表示オブジェクト 362
リスナー、「イベントリスナー」を参照
リテラル値
object 223
説明 98
配列リテラル 98、215
リトルエンディアンバイト順序 630

る

ループ
do..while 114
for 112
for (XML) 333、345
for each..in 113、225、345
for..in 112、225、345
while 114

れ

例
GeometricShapes 170
Matrix クラス 421
RSS データのロード 349
RunTimeAssetsExplorer 480
SimpleClock 186
SpriteArranger クラス 405
Telnet クライアントの構築 649
Web ページコンテナに対する External API の使用 705
Wiki パーサー 293
WordSearch 608
イメージのフィルタ処理 468
エラー処理 320
画面外ビットマップを使用してスプライトをアニメーション化する 528
サウンドアプリケーション 591
システム機能の検出 675
ストリング 203
正規表現 293
テキストフォーマット 505
配列 235
ビデオジュークボックス 559
表示オブジェクトのレイヤーの再配置 410
複数ページのプリント 690
例外 245
レイヤー、再配置 410
レキシカル環境 127
列挙 146
連結
XML オブジェクト 340
ストリング 198
連結 (+) 演算子、XMLList 340

ろ

ローカル記憶領域 635
ローカル変数 79
ロードされたオブジェクトの URL 402
ロードされたバイト数 402
ロードされたメディア、データとしてのアクセス 746
ロードの進行状況 402
ロード、グラフィック 400
ロード、コンテキスト 402
論理演算子 108

わ

ワイルドカード (*) 演算子、XML 342