

サブアプリケーションの開発とロード

サブアプリケーションのロードについて

Flex はメインアプリケーション内でサブアプリケーションをロードまたはアンロードします。サブアプリケーションを総合的なアプリケーションアーキテクチャの一部として使用する理由は、次のとおりです。

- メインアプリケーションのサイズを縮小する
- 関連する機能をサブアプリケーションに含める
- 異なるアプリケーションにロードできる再利用可能なサブアプリケーションを作成する
- サードパーティ製アプリケーションをメインアプリケーションに統合する

サブアプリケーションのロード方法によって、メインアプリケーションとサブアプリケーション間の相互運用性レベルが決定されます。メインアプリケーションにサブアプリケーションをロードする際は、次のような点を考慮します。

信頼できるアプリケーション そのアプリケーションとはどの程度の信頼関係がありますか。信頼できるアプリケーションでは、メインアプリケーションとの相互運用性も高くなります。信頼されていないアプリケーションでも、限られた範囲でメインアプリケーションとの何らかの相互運用が可能です。ただし一般的に、ロードしたアプリケーションの開発と展開を完全に制御できない場合、そのアプリケーションは信頼できないものと見なしてください。

バージョン管理 アプリケーションは Flex フレームワークと同じバージョンでコンパイルされていますか。サブアプリケーションのデフォルトのロード方法では、すべてのアプリケーションが同じバージョンのフレームワークでコンパイルされているものと見なされます。ただし、Flex はバージョンの異なるフレームワークでコンパイルされたアプリケーションもロードできます。このようなアプリケーションはマルチバージョン化されたアプリケーションと呼ばれます。マルチバージョン化されたアプリケーションでは、ロードするメインアプリケーションとの相互運用性レベルに多少の制限があります。しかし、大規模なアプリケーションではより柔軟に使用できます。

信頼関係のレベルとバージョン管理の方法は、サブアプリケーションがロードされるアプリケーションドメインとセキュリティドメインによって決定されます。

Flex にロードされるサブアプリケーションには 3 つの主な種類があります。

サンドボックス化されたアプリケーション は、独自のセキュリティドメインにロードされ、マルチバージョン化も可能です。サードパーティ製アプリケーションをロードする場合には、サンドボックス化されたアプリケーションの使用が推奨されます。また、RPC または DataServices 関連機能を使用する場合にも、サブアプリケーションはサンドボックスとしてロードする必要があります。

マルチバージョン化されたアプリケーション は、ロードしたメインアプリケーションとは異なるバージョンの Flex フレームワークでコンパイルが可能です。その場合、メインアプリケーションやその他のサブアプリケーションとの相互運用性は、シングルバージョンアプリケーションの場合に比べ制限されます。

シングルバージョンアプリケーション は、メインアプリケーションと同じバージョンのコンパイラでコンパイルされたことが保証されているアプリケーションです。メインアプリケーションとの相互運用性は最も高くなっていますが、サブアプリケーションのソースを完全に制御する必要があります。

サブアプリケーションの使用はモジュールの使用といくつかの類似点があります。2 つのアプローチの比較については、「13 ページの「[ロードされたアプリケーションとモジュールの比較](#)」を参照してください。

関連項目

- 31 ページの「[サンドボックス化されたアプリケーションの開発](#)」
- 39 ページの「[マルチバージョン化されたアプリケーションの開発](#)」
- 17 ページの「[サブアプリケーションの作成とロード](#)」

アプリケーションドメインについて

アプリケーションドメインにはクラス定義が含まれています。アプリケーションには、システムドメインと呼ばれる 1 つの最上位アプリケーションドメインがあります。アプリケーションドメインはシステムドメインの子ノードとして定義されます。サブアプリケーションを別のメインアプリケーションにロードする場合、兄弟、子、現在の 3 つのいずれかのアプリケーションドメインにロードできます。サブアプリケーションを兄弟アプリケーションドメインにロードすると、サブアプリケーションのアプリケーションドメインはメインアプリケーションのアプリケーションドメインと同じ親を持ちます。さらに、すべてのその他の兄弟アプリケーションのピアになります。サブアプリケーションを子アプリケーションドメインにロードすると、サブアプリケーションのアプリケーションドメインは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子になります。サブアプリケーションを現在のアプリケーションドメインにロードすると、サブアプリケーションはメインアプリケーションと同じアプリケーションドメインにロードされます。サブアプリケーションがどこからクラス定義を取得するかは、これらの場所で決まります。

SWFLoader および各 Loader コントロールは、デフォルトでサブアプリケーションが子アプリケーションドメインにロードするよう設定されています。サブアプリケーションとメインアプリケーションが、異なる Flex フレームワークでコンパイルされていると、ランタイムエラーの原因となります。これは、アプリケーションが同じクラスの異なる定義に対してコンパイルされた場合に起こります。

このような場合、マルチバージョン化されたサブアプリケーションをメインアプリケーションがロードするよう指定できます。これは、サブアプリケーションを兄弟アプリケーションドメインにロードすることで行います。そうすることで、サブアプリケーションはクラス定義を親から取得せず、独自に行います。2 つのアプリケーションが、バージョンの異なる Flex フレームワークでコンパイルされた場合でも併用できます。

SWFLoader の `loadForCompatibility` プロパティの値を設定して、サブアプリケーションのアプリケーションドメインを指定します。このプロパティの値を `true` に設定すると、サブアプリケーションは兄弟アプリケーションドメインにロードされます。このプロパティの値を `false` に設定すると、サブアプリケーションは子アプリケーションドメインにロードされます。このプロパティのデフォルト値は `false` で、サブアプリケーションはマルチバージョン化されません。

また、LoaderContext オブジェクト上でもアプリケーションドメインの指定ができます。これは、SWFLoader コントロールを使用する場合に、loaderContext プロパティの値を指定して行います。詳細については、「21 ページの「[LoaderContext の指定](#)」を参照してください。

システムドメイン

Flash Player で定義されているクラスはシステムドメインにあります。システムドメインは他のすべてのアプリケーションドメインの親です。メインアプリケーションのアプリケーションドメインはシステムドメインの子です。兄弟アプリケーションドメインにサブアプリケーションをロードした場合、そのサブアプリケーションもシステムドメインの子となります。システムドメイン内で定義されたクラスは、サブアプリケーションまたはメインアプリケーションで定義し直すことはできません。これらのアプリケーションはすべて Flash Player と共通定義を共有します。これらの定義には DisplayObject、Event、Sprite などのクラスが含まれます。これらの共有されたクラス定義は playerglobal.swc ファイルに含まれています。

兄弟アプリケーションドメイン

サブアプリケーションを含むアプリケーションドメインは、サブアプリケーションがそのクラス定義をどこから取得するかを決定します。メインアプリケーションがサブアプリケーションを兄弟アプリケーションドメインにロードした場合、サブアプリケーションは独自の非プレーヤクラス定義を定義します。これはマルチバージョン化されたアプリケーションの設定です。兄弟アプリケーションドメインにロードされたアプリケーションは、他のアプリケーションを兄弟アプリケーションドメインにロードできます。

メインアプリケーションの兄弟アプリケーションドメインにあるサブアプリケーションは、メインアプリケーションと通信が可能です。サブアプリケーションは、次の条件を満たす限り、メソッドを呼び出し、アプリケーションのプロパティにアクセスすることができます。

- Flash Player で定義されている強い型のみが使用されている。
- サブアプリケーションが別のセキュリティドメインに含まれていない。

兄弟アプリケーションドメインのサブアプリケーションがメインアプリケーションと通信する機能は、子アプリケーションドメインのサブアプリケーションほどスムーズではありません。例えば、サブアプリケーションがポップアップコントロールを起動すると、メインアプリケーションにイベントが渡され、コントロールが起動されます。このようなイベントの受け渡しにより通信は多少制限されますが、多少の相互運用性もあります。この種類のアプリケーション開発の詳細については、「39 ページの「[マルチバージョン化されたアプリケーションの開発](#)」を参照してください。

個別のセキュリティドメインに含まれているアプリケーションは、自動的に個別の兄弟アプリケーションドメインに含まれます。そのため、信頼されていないサブアプリケーションをメインアプリケーションにロードすると、そのサブアプリケーションは独自のクラス定義を持ちます。

関連項目

39 ページの「[マルチバージョン化されたアプリケーションの開発](#)」

子アプリケーションドメイン

メインアプリケーションがサブアプリケーションをそのアプリケーションドメインの子アプリケーションドメインにロードした場合、サブアプリケーションはそのクラス定義をメインアプリケーションから取得します。これはアプリケーションのロードのデフォルト設定です。バージョンの異なる Flex フレームワークでアプリケーションがコンパイルされていると、ランタイムエラーを起こす原因となります。SWF ファイルがクラス設定を行う際、既に定義されているクラスはアプリケーションドメインに追加されません。ここでは一番最初に定義されたクラスが、そのクラスの唯一の定義となります。後からアプリケーションドメインにロードした定義は無視されます。

メインアプリケーションの子アプリケーションドメインにあるサブアプリケーションは、メインアプリケーションと通信が可能です。この場合、メインアプリケーションと最高レベルの相互運用性があります。これはマルチバージョン化されていない大規模アプリケーションに典型的に見られる状態で、SWFLoader のデフォルト動作となっています。

関連項目

17 ページの「[サブアプリケーションの作成とロード](#)」

現在のアプリケーションドメイン

サブアプリケーションを（別の兄弟アプリケーションドメインまたは子アプリケーションドメインではなく）現在のアプリケーションドメインにロードした場合、サブアプリケーションのクラス定義は無視されることがよくあります。これはドメイン中の最初の定義が使用されるために起こります。後からドメインにロードした定義は無視されます。新しく追加したクラス定義は、メインアプリケーションで使用できます。

現在のアプリケーションドメインは RSL やその他の特殊にコンパイルされたリソースに使用することが多く、サブアプリケーションのロードには通常は使用しません。

セキュリティドメインについて

セキュリティドメインはアプリケーション間の信頼関係のレベルを定義します。アプリケーション間の信頼関係が強くなれば、相互運用性も高くなります。一般的に、サブアプリケーションがメインアプリケーションと同じセキュリティドメインにロードされた場合は、最高レベルの相互運用性がアプリケーション間にあります。

多くのリモートアプリケーションやマルチバージョン化されたアプリケーションの場合と同様に、サブアプリケーションを異なるセキュリティドメインにロードすると、メインアプリケーションとの相互運用性が制限されます。また、別々のセキュリティドメインにあるサブアプリケーション間でも通信機能は制限されます。これらはサンドボックス化されたアプリケーションと呼ばれるものです。

ロードの際にサブアプリケーションをメインアプリケーションと同じセキュリティドメインにロードするかどうかを決定します。メインアプリケーションと同じセキュリティドメインにリモートのサブアプリケーションをロードするには、`trustContent` プロパティの値を `true` に設定してください。この動作はアプリケーションがメインアプリケーションとは異なるウェブドメインまたはサブドメインからロードされる場合のみ適用します。サブアプリケーションがメインアプリケーションと同じウェブドメインからロードされる場合、デフォルトでは同じセキュリティドメインにロードされるように設定されています。`loadForCompatibility` プロパティの値の設定は、`trustContent` プロパティには影響しません。

場合によっては、メインアプリケーションと同じドメインにあるサブアプリケーションを別のセキュリティドメインにロードしたい場合もあります。これには例えば、サブアプリケーションがサードパーティ製である場合やサンドボックス化されたアプリケーションと同レベルの相互運用性を保ちたい場合などがあります。これは、1つの方法として、同一サーバ上に名前の異なるサブドメインを設定し、そこからサブアプリケーションをロードすることにより実行できます。詳細については、「15 ページの「[同じドメインおよびクロスドメインアプリケーションのロード](#)」を参照してください。

AIR を使用の際は、`trustContent` プロパティの値を `true` に設定することができません。

`trustContent` プロパティの値を `true` に設定していない場合、リモートサーバのサブアプリケーションはデフォルト設定として別のセキュリティドメインにロードされます。

また、`LoaderContext` オブジェクトにもセキュリティドメインの指定ができます。これは、`SWFLoader` コントロールを使用する場合に、`loaderContext` プロパティの値を指定して行います。これは同じセキュリティドメインにサブアプリケーションをロードしたい場合にのみ行います。詳細については、「21 ページの「[LoaderContext の指定](#)」を参照してください。

サンドボックス化されたアプリケーションにはアプリケーションの相互運用性に関する制限が最も多くあります。これには次のような制限が含まれます。

ステージ サブアプリケーションからステージへのアクセスは、いくつかのステージプロパティとメソッドに限られています。

マウス 他のセキュリティドメインのオブジェクトからはマウスのイベントを取得できません。

ピクセル 他のセキュリティドメインにあるアプリケーションで描画されたピクセルにはアクセスできません。

プロパティ アプリケーションは他のセキュリティドメインにあるオブジェクトの参照を取得することが可能ですが、セキュリティ上の理由からこのような取得は避けてください。いくつかのプロパティ（`Stage` や、別のアプリケーションのインスタンスを作成する `DisplayObject` の親など）にも制限があります。これらの制限のほか、別のセキュリティドメインのアプリケーションは、定義上、別のアプリケーションドメインにあることになります。その結果、そのアーキテクチャのすべての制限に従います。しかし、マルチバージョン化ができることによる利点もあります。

詳細については、「31 ページの「[サンドボックス化されたアプリケーションの開発](#)」を参照してください。

サブアプリケーションをロードするアプリケーションによく見られるタイプ

サブアプリケーションをロードするアプリケーションには通常、次のようなタイプがあります。

- マルチバージョン化された、またはマルチバージョン化されていない大規模アプリケーション
- マッシュアップ

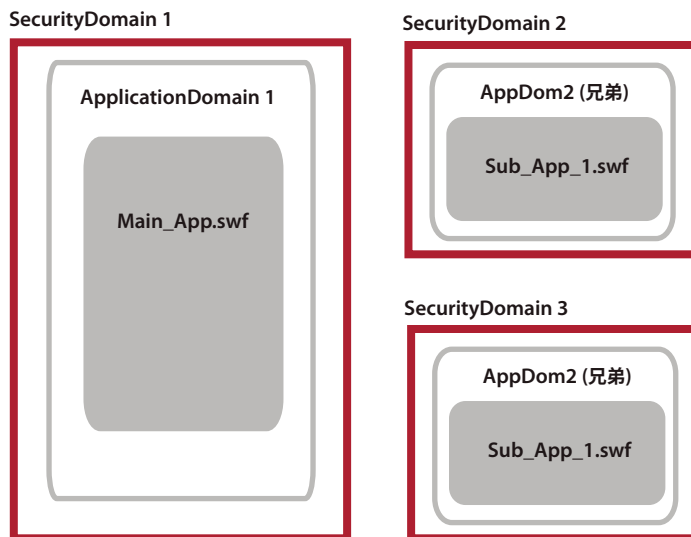
- ポータル
- ダッシュボード

サンドボックス化されたアプリケーションについて

サンドボックス化されたアプリケーションは、よく見られる、サブアプリケーションをロードするアプリケーションのタイプです。通常、サンドボックス化されたアプリケーションはサードパーティによってコンパイルおよびホストされたアプリケーションをロードします。メインアプリケーションは、このようなサブアプリケーションを開発したサードパーティと信頼関係があるとは限りません。さらに、メインアプリケーションの開発者にはサブアプリケーションのコンパイルに使用された Flex フレームワークのバージョンがわかりません。その結果、サンドボックス化されたアプリケーションは、メインアプリケーションとサブアプリケーション間の相互運用性が非常に低くなりますが、多くの場合、マルチバージョン化されています。サンドボックス化されたアプリケーションによく見られるタイプにポータルがあります。

サンドボックス化されたアプリケーションは、RPC クラスや DataService 関連機能の使用するのに追加コーディングが最小限で済みます。信頼関係を持つマルチバージョン化されたアプリケーションでは、ほとんどの場合、RPC クラスがアプリケーション全体で機能するのにブートストラップローダーを必要とします。サンドボックス化されたアプリケーションでは必要ありません。そのため、多くのマルチバージョン化されたアプリケーションで、サンドボックス化されたアプリケーションの使用を推奨しています。

次の図は、サンドボックス化されたアプリケーションの境界を示しています。サブアプリケーションは、個別の兄弟アプリケーションドメインにロードされます。これは、バージョンの異なるフレームワークでコンパイルされたアプリケーション同士が相互運用できるよう、それぞれのアプリケーションが独自のクラス定義を含んでいることを意味します。サブアプリケーションは個別のセキュリティドメインにも含まれます。そのため、サブアプリケーション同士は信頼関係がなく、その結果、相互運用性にも制限があります。



関連項目

31 ページの「[サンドボックス化されたアプリケーションの開発](#)」

マルチバージョン化された大規模アプリケーションについて

マルチバージョン化された大規模アプリケーションの多くは、組織内の異なるグループによって開発され、多くの異なるコンポーネントから構成されています。これには例えば、フォームマネージャなどがあります。

マルチバージョン化されたアプリケーションは、メインアプリケーションのアプリケーションドメインと並んでいる兄弟アプリケーションドメインにサブアプリケーションをロードします。サブアプリケーションはメインアプリケーションと同じセキュリティドメインにもロードされます。

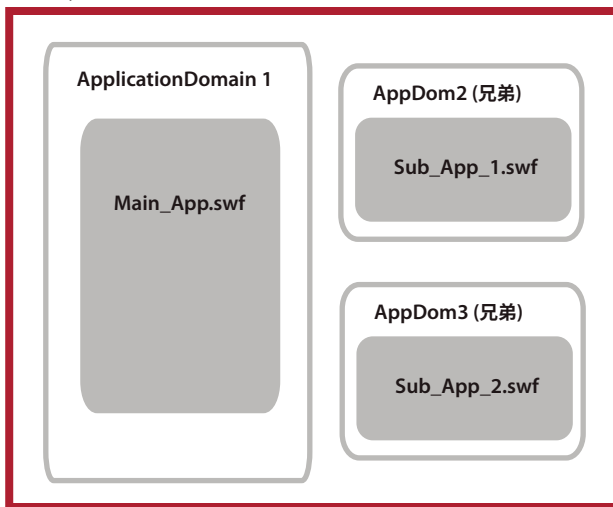
マルチバージョン化されたアプリケーションにはサンドボックス化されたアプリケーションの利点が数多く適用されますが、RPC クラスと DataService 関連機能と併用できるようにするにはコーディングを追加する必要があります。一般に、サブアプリケーションでこれらのクラスが使用される場合は、信頼関係がある場合でもロード時にサンドボックス化されたアプリケーションを使用するアプローチを採用してください。詳細については、「40 ページの「マルチバージョン化されたアプリケーションでの RPC および DataService クラスの使用」」を参照してください。

実際の展開では、通常、メインアプリケーションとサブアプリケーションはすべて同じウェブドメインにあるため、信頼関係を持っています。これらのアプリケーションは、たとえ異なるウェブドメインに展開されても、通常は、信頼関係を維持するために同じセキュリティドメインにロードされます。このようにドメインの異なるアプリケーションを同じセキュリティドメインにロードするプロセスはインポートロードと呼ばれます。このプロセスは、ロードしたサブアプリケーションのソースを信頼できることがわかっているまれな状況でのみ使用してください。

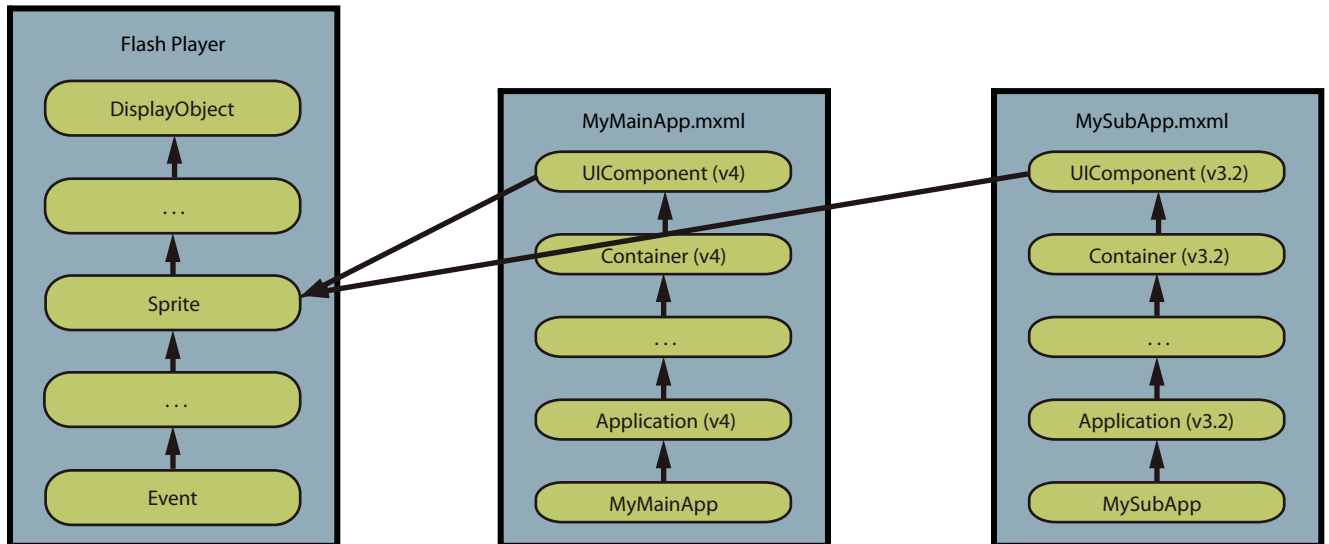
メインアプリケーションをロードするサブアプリケーションをサードパーティがコンパイルする場合もあります。この場合、開発者にはサブアプリケーションのコンパイルに使用された Flex フレームワークのバージョンがわからないことがあります。また、開発者はサブアプリケーションのソースコードにアクセスできないこともあります。その結果、メインアプリケーションと同じバージョンのフレームワークで再コンパイルすることができません。

下の図は、マルチバージョン化された大規模アプリケーションの境界を示しています。サブアプリケーションは、個別の兄弟アプリケーションドメインにロードされます。これは、各アプリケーションがバージョンの異なるフレームワークでコンパイルされるよう、独自のクラス定義を含んでいることを意味します。すべてのアプリケーションが同じセキュリティドメインにロードされるため、相互に信頼関係を持っています。

SecurityDomain 1



兄弟アプリケーションドメインのサブアプリケーションは、メインアプリケーションのクラス定義とは別に、独自のクラス定義を保存します。下の図は、マルチバージョン化された大規模アプリケーションのクラス定義を示しています。この図では、サブアプリケーションは Flex 3.2 でコンパイルされた独自の定義を使用しています。メインアプリケーションは Flex 4 でコンパイルされた独自の定義を使用しています。



関連項目

39 ページの「[マルチバージョン化されたアプリケーションの開発](#)」

シングルバージョンの大規模アプリケーションについて

バージョン管理をサポートしていない大規模なアプリケーションとは、同じバージョンの Flex フレームワークでコンパイルされたサブアプリケーションのみがロードできるアプリケーションです。組織内の単一グループでは通常、このようなアプリケーションが作成されます。このような状況では、開発プロセス中にグループ内で Flex フレームワークのバージョンやその他基準を統一することができます。

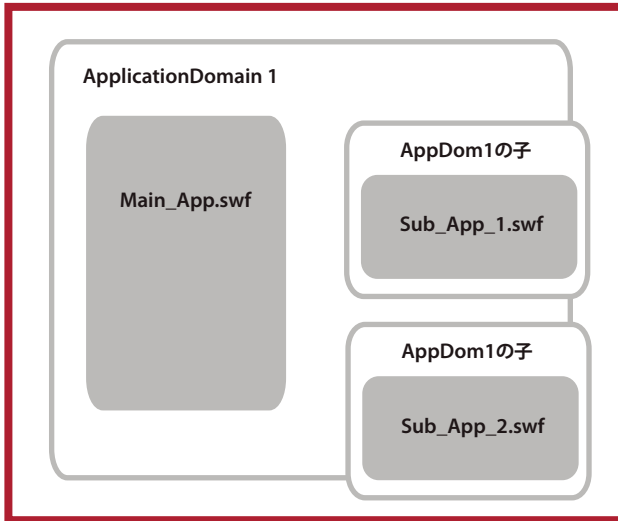
SWFLoader コントロールがロードするアプリケーションのデフォルトのタイプは、シングルバージョンの大規模アプリケーションです。サブアプリケーションをメインアプリケーションにロードする際に SWFLoader のデフォルト設定をそのまま使用した場合、サブアプリケーションはマルチバージョン化されません。この場合、サブアプリケーションはメインアプリケーションの子アプリケーションドメインにロードされます。

シングルバージョンの大規模アプリケーションは、その保持するが困難な場合があります。これは、アプリケーションを新しいバージョンのフレームワークで再コンパイルすると、常にすべてのサブアプリケーションを再コンパイルする必要があります。さらに、バージョンの異なるフレームワークでコンパイルされている可能性のある、サードパーティ製サブアプリケーションを追加することもより困難になります。また、多くの場合、サブアプリケーションを再コンパイルするソースコードにアクセスできません。

シングルバージョンの大規模アプリケーションとそのすべてのサブアプリケーションは通常、同じウェブドメインに展開されます。これは、組織内の特定のグループがアプリケーションの開発と管理の両方を行うことが多いためです。その結果、これらのアプリケーションは相互に信頼関係を持っています。たとえ異なるウェブドメインに展開されても、これらのアプリケーションは同じセキュリティドメインにインポート（ロード）されます。これにより、アプリケーションは信頼関係を持つことができます。

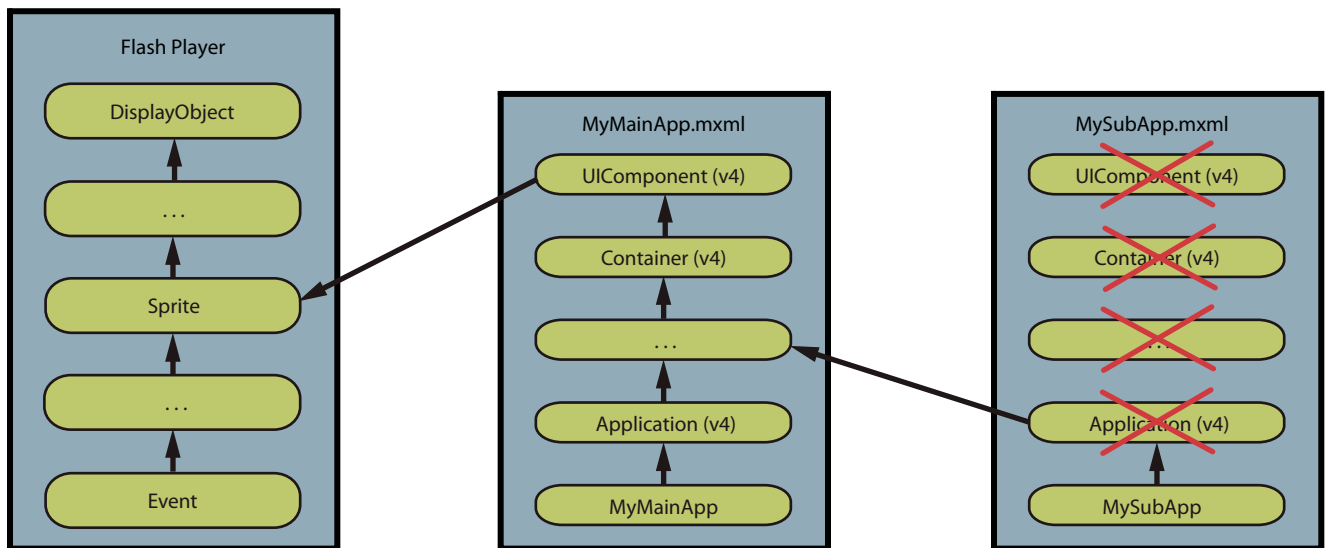
下の図は、マルチバージョン化をサポートしない大規模アプリケーションの境界を示しています。各サブアプリケーションは、メインアプリケーションのアプリケーションドメインの子アプリケーションドメインにロードされます。その結果、すべてのアプリケーションは同じクラス定義を使用します。これらのアプリケーションのいずれかが別バージョンのフレームワークでコンパイルされた場合、異なる API への呼び出しによってランタイムエラーが発生する可能性があります。すべてのアプリケーションは同じセキュリティドメインにあるので、相互に信頼関係を持っています。

SecurityDomain 1



子アプリケーションドメインのサブアプリケーションは、親アプリケーションドメインのアプリケーションからクラス定義を継承します。サブアプリケーションがメインアプリケーションで定義済みのクラスのいずれかを定義しようとする、子の定義は無視されます。メインアプリケーションで定義されていない特定のクラスを、複数のサブアプリケーションが定義しようとする、各サブアプリケーションで独自の定義が使用されます。

下の図は、シングルバージョンの大規模アプリケーションのクラス定義を示しています。この図において、サブアプリケーションは Flex 4 でコンパイルされたメインアプリケーションから定義を取得しています。Flex 4 で同様にコンパイルされたサブアプリケーションのクラス定義は無視されます。



関連項目

17 ページの「サブアプリケーションの作成とロード」

Flex マネージャクラスについて

Flex マネージャクラスは、Flex アプリケーションのさまざまなタスクを処理します。例えば、`DragManager` はドラッグアンドドロップに関連するすべての機能の処理を行います。このマネージャクラスはデータ整列化および `DragProxy` の生成、ドラッグ関連のイベント発生を管理します。

サブアプリケーションをメインアプリケーションと同じアプリケーションドメインにロードした場合、アプリケーションドメインには各マネージャの 1 つのインスタンスのみが含まれます。ただし、アプリケーションが異なるアプリケーションドメインにある場合は、システムドメイン内でマネージャのインスタンスが複数存在する可能性があります。

タスクの種類によっては、マネージャクラスはメインアプリケーションがサブアプリケーションをロードした際に渡されるイベントを介して通信できる場合もあります。例えば、フォーカスがサブアプリケーションに移動すると、サブアプリケーションの `FocusManager` はメインアプリケーションの `FocusManager` からコントロールを受け取ります。フォーカスがサブアプリケーションから離れると、サブアプリケーションの `FocusManager` はメインアプリケーションの `FocusManager` にコントロールを返します。

その他の場合は、システムドメインに存在するマネージャのアクティブなインスタンスは 1 つだけです。Flex はサブアプリケーションのマネージャを無効にします。例えば、`BrowserManager` は複数のインスタンスを持つことはできません。サブアプリケーションがメインアプリケーションの子アプリケーションドメインにある場合を除き、メインアプリケーションだけが `BrowserManager` にアクセスして、それを使用して通信できます。別のアプリケーションドメインにあるサブアプリケーションが `BrowserManager` を使用して通信する場合は、メインアプリケーションを介して通信する必要があります。

この場合、マネージャクラスとのインタラクションを処理するカスタムロジックを作成する必要があります。例えば、サブアプリケーションにメインアプリケーションのデフォープリックを使用させたい場合、そのためのカスタムクラスを作成できます。このカスタムクラスは、`BrowserManager` と通信できる場合には、サブアプリケーションからメインアプリケーションにメッセージを渡します。

一般的に、最上位のマネージャがユーザインタラクションを処理します。このマネージャは、サブアプリケーションから必要なことを指示するメッセージを受け取ります。

次のマネージャクラスは、すべてのアプリケーションによってリンクされています。

- `SystemManager`
- `LayoutManager`
- `StyleManager`
- `EffectsManager`
- `ResourceManager` (`ModuleManager` にリンク)
- `FocusManager`
- `CursorManager`
- `ToolTipManager`

次のマネージャクラスは、アプリケーションで使用されている場合にのみリンクされています。

- `DragManager`
- `BrowserManager`
- `HistoryManager`
- `PopUpManager` (サブアプリケーションがポップアップコントロールを使用する場合は、メインアプリケーションにこのマネージャの定義が含まれている必要があります)

メインアプリケーションが同じバージョンのフレームワークでコンパイルされたサブアプリケーションをロードする場合、SWF ファイルに保存される定義は各マネージャに対し 1 つだけです。ただし、サンドボックス化あるいはマルチバージョン化されたサブアプリケーションの場合は、メインアプリケーションとサブアプリケーションの両方が独自の定義を保存します。多くの場合、サブアプリケーションのマネージャクラスは、メインアプリケーションのマネージャのインスタンスにメッセージを渡します。こうして、メインアプリケーションのマネージャはタスクを処理できるようになります。

サンドボックス化またはマルチバージョン化されたアプリケーションでは、メインアプリケーションとサブアプリケーションの両方が独自のマネージャクラスバージョンを持ちます。シングルバージョンのアプリケーションでは、ほとんどのマネージャクラスが必要とするのは 1 つのバージョンのみです。マルチバージョンでないアプリケーションでは、ほとんどの場合、マネージャの定義を外部化できます。

SystemManager について クラス

すべてのアプリケーションは SystemManager クラスのインスタンスを持っています。アプリケーションが別のアプリケーションにロードされた場合は、サブアプリケーション用の SystemManager が作成されます。SystemManager は引き続きサブアプリケーションのアプリケーションウィンドウを管理しますが、サブアプリケーションがロードされた方法によっては、ポップアップコントロール、ツールヒントおよびカーソルは管理しない場合があります。サブアプリケーションをロードした SWFLoader の内容には、そのサブアプリケーションへの参照方法が含まれています。

メインアプリケーションの SystemManager は、それによってアプリケーション全体の各要素にアクセスできるため、重要です。例えば、最上位の SystemManager は次の役割を持っています。

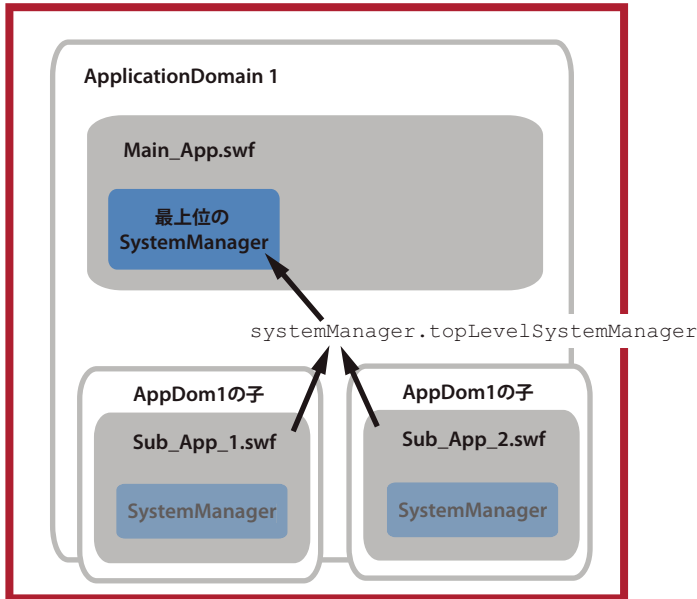
- メインアプリケーションと信頼できるすべてのサブアプリケーションのポップアップコントロール、ToolTip オブジェクトおよびカーソルすべての親となります。
- 信頼できるすべてのアプリケーションへのフォーカスを処理します。

最上位の SystemManager への参照を使用して、サブアプリケーションで発生したイベントをリスンするように登録できます。例えば、最上位の SystemManager にリスナーを追加すると、サブアプリケーションはそのアプリケーションドメイン外で発生したマウスイベントをリスンできるようになります。

サブアプリケーションから最上位の SystemManager にアクセスする方法は、使用しているサブアプリケーションの種類によって異なります。

サブアプリケーションが子アプリケーションドメインにロードされたアーキテクチャで、最上位の SystemManager にアクセスするには、SystemManager の topLevelSystemManager プロパティを使用します。次の図では、サブアプリケーションが systemManager.topLevelSystemManager を使用して、メインアプリケーションの SystemManager にアクセスしています。

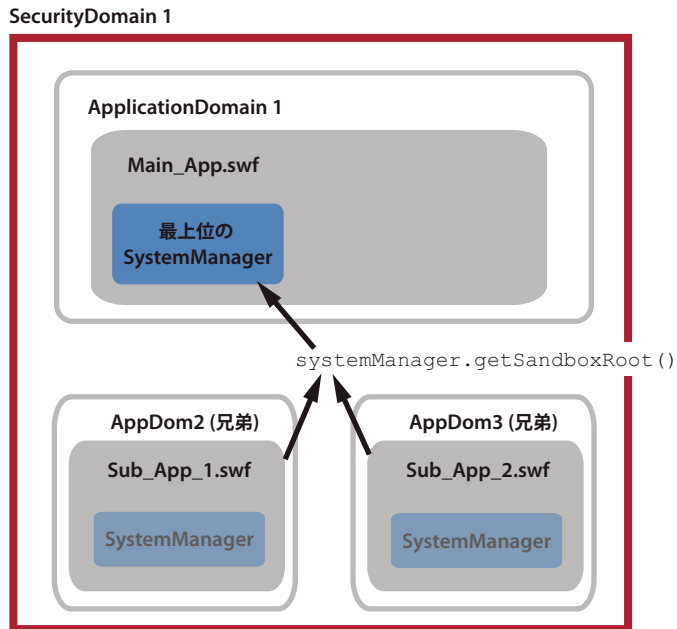
SecurityDomain 1



topLevelSystemManager プロパティを使用してメインアプリケーションの SystemManager にアクセスするコードの例については、「27 ページの「ロードされたアプリケーションによるマウスイベントのリッスン」」を参照してください。

シングルバージョンのアプリケーションでは、topLevelSystemManager プロパティは常にアプリケーションドメインにおける最上位の SystemManager を参照します。アプリケーションが（サンドボックス化またはマルチバージョン化されたアプリケーションとして）別のアプリケーションドメインにある場合、getSandboxRoot() メソッドを使用してそのセキュリティドメインにおける最上位の SystemManager を取得できます。

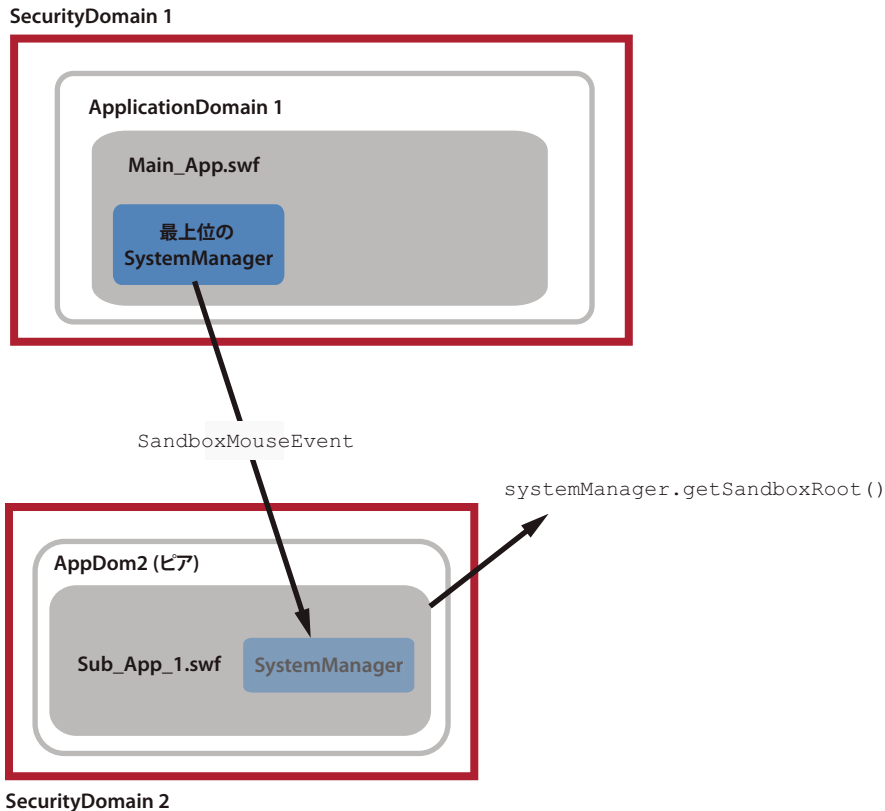
次の図は、サブアプリケーションが `systemManager.getSandboxRoot()` を使用して、別のアプリケーションドメインにあるメインアプリケーションの `SystemManager` を参照する場合を示しています。



`getSandboxRoot()` メソッドを使用してメインアプリケーションの `SystemManager` にアクセスするコードの例については、「43 ページの「マルチバージョン化されたアプリケーションでのマウスイベントのリッスン」」を参照してください。

サブアプリケーションがメインアプリケーションとは別のセキュリティドメインにある場合（またはサンドボックス化されたアプリケーションのように信頼関係がない場合）、サブアプリケーションからメインアプリケーションの `SystemManager` の参照を得ることはできません。この場合は、`SandboxMouseEvent` および `InterDragManagerEvent` をリッスンして、アプリケーション間の通信を行うことができます。どのアプリケーションからもこれらのイベントをトリガできます。サブアプリケーションの `SystemManager` がこれらのイベントを検知した時点で、それらのイベントを処理することができます。

次の図は、別々のセキュリティドメインにある2つのアプリケーションが、イベントの受け渡しによって通信するアーキテクチャを示しています。サブアプリケーションは `getSandboxRoot()` メソッドを呼び出して、それ自体の `SystemManager` への参照を実行します。これにより、`SystemManager` はメインアプリケーションの `SystemManager` が送出したイベントをリスンできます。



`SandboxMouseEvent` オブジェクトのプロパティは、典型的なイベントオブジェクトのプロパティとは異なります。例えば、`target` プロパティおよび `currentTarget` プロパティは、イベントを送出した `SandboxBridge`、またはそのイベントを再送出した `SystemManager` になります。これらのプロパティは、イベントを送出した特定のオブジェクトとは違います。これらのプロパティによって通常特定されるオブジェクトに対して、信頼関係のない子アプリケーションが参照できるようにすべきではありません。同様に、`stageX` および `stageY`、`localX`、`localY` プロパティも参照できません。

`SandboxMouseEvent` オブジェクトを使用して、サブアプリケーションのセキュリティドメインの外部で発生するマウスイベントを処理する例については、「35 ページの「[サンドボックス化されたアプリケーションでのマウスイベントのリスン](#)」を参照してください。

ロードされたアプリケーションとモジュールの比較

大規模なアプリケーションを設計するときは、そのアーキテクチャを検討してください。アプリケーションに下位のアプリケーションをロードおよびアンロードする1つのメインアプリケーションがある場合、採用するアプローチを決定する際に、モジュールを使用した場合とサブアプリケーションを使用した場合の利点を比べます。

様々なフォームで構成されるアプリケーションの例を検討してください。モジュール化アプリケーションでは、メインアプリケーションは個々のフォームを `ModuleManager` を持つモジュールとしてロードします。サブアプリケーションを使用するアプリケーションでは、個々のフォームは `SWFLoader` によってメインアプリケーションにロードされる個別のアプリケーションとなります。これらの2つのアプローチ手法は多くの点で類似していますが、異なる点も多くあります。

モジュールを使用する理由の多くは、サブアプリケーションにも当てはまります。メインアプリケーションは通常、多くの機能をそれ自体に埋め込まずに、サブアプリケーションをロードします。これにより、一般的に最初にダウンロードするメインアプリケーションのサイズが小さくなり、ロード時間も短縮されます。さらに、これにより関連機能のカプセル化が促進され、開発と管理が容易になります。

モジュールとサブアプリケーションには、次のような類似点があります。

- コンパイルされた SWF ファイルである
- 実行時にロードおよびアンロードできる
- 関連機能のカプセル化を促す
- 非同期型の開発環境が可能である
- メインアプリケーションを再コンパイルせずに再コンパイルできる
- デバッグが可能
- ローカルまたはリモートでロードできる（適切な権限が必要）
- ブラウザでキャッシュされる
- 事前のロードが可能
- ロード状況を知るための進行状況を示すイベントを発生できる
- メインアプリケーションのメソッドとプロパティに動的にアクセスできる

また、モジュールとサブアプリケーションは異なります。最大の違いは、モジュールは一般的にそのホストアプリケーションとクラスを共有していることです。クラスの共有により、モジュールとメインアプリケーションの間に依存関係が築かれます。一方、サブアプリケーションは一般的に独自のクラスのバージョンを保持しており、これによりサブアプリケーションとメインアプリケーションの間の依存関係は弱くなります。

その他のモジュールとサブアプリケーションの違いは、次のとおりです。

ファイルサイズ 共有クラスは外部化できるため、多くの場合、モジュールとそのホストアプリケーションのファイルサイズは小さめです。サブアプリケーションでは必ずしも共有クラスを外部化できるとは限りません。アプリケーションのバージョンが異なる場合は、それぞれのアプリケーションが独自のクラスのセットを持つ必要があります。

再利用 モジュールは、より緊密にホストアプリケーションにバインドされています。モジュールは、ロードするアプリケーションと通信するためにインターフェースを使用します。このため、アプリケーションがモジュールを使用している際に、そのアプリケーションに変更が加えられた場合、メインアプリケーションを Flex の最新バージョンに移行すると、すべてのモジュールを再コンパイルしなければなりません。

バージョン管理 モジュールはマルチバージョンには対応していません。メインアプリケーションとすべてのモジュールは、同じバージョンの Flex フレームワークでコンパイルされる必要があります。

マネージャークラス 一般的に、モジュールとそのホストアプリケーションはマネージャークラスを共有します。モジュールは子アプリケーションドメインの中にあるのに対して、サブアプリケーションは多くの場合、マネージャークラスの独自のインスタンスを持っており、これによりマルチバージョン化が可能です。

ActionScript のみ モジュールは MXML ベースにも ActionScript にもなり得るのに対し、アプリケーションおよびサブアプリケーションは一般的に純粋な ActionScript にはなり得ません。

アプリケーションドメイン のサブアプリケーションは兄弟アプリケーションドメインまたは子アプリケーションドメインにロードできますが、モジュールはホストアプリケーションの子アプリケーションドメインにロードする必要があります。

セキュリティドメイン モジュールはホストアプリケーションと同じセキュリティドメインにロードされる必要があります。サブアプリケーションは同じセキュリティドメインまたは異なるセキュリティドメインにロードできます。

モジュールについて詳しくは、「モジュール化アプリケーションの作成」を参照してください。

SWFLoader と Loader コントロールの比較

SWFLoader クラスと Loader クラスを使用して、サブアプリケーションをメインアプリケーションにロードできます。ほとんどの場合、SWFLoader クラスを使用します。このクラスは Loader クラスをラップして、サブアプリケーションのメインアプリケーションへのロードを容易にするための追加機能を提供します。

SWFLoader コントロールには次の機能があります。

- Flex のスタイルおよびエフェクトをサポートします。Loader クラスは固有のスタイルおよびエフェクトをサポートしません。
- ロードの進行状況を個別に監視できるようにします (Loader クラスを使用する場合は、最初に LoaderInfo オブジェクトへの参照を得る必要があります)。
- UIComponent なので、SWFLoader コントロールは表示リストに関与でき、子を表示リストに追加するために追加のコードを書く必要がありません。
- コンテンツのサイズとスケールを自動的に変更します。
- SWF ファイルは Application クラスのインスタンスである必要はなく、Application の存在のみをチェックし、サイズ変更は別に処理します。
- マルチバージョン化が可能です。Loader クラスには、マルチバージョン化のサポートは組み込まれていません。

ドメイン間の通信

異なるアプリケーションドメインに存在するメインアプリケーションとサブアプリケーションがある場合、アプリケーション間の通信方法として推奨されるのは、イベントの受け渡しです。メインアプリケーションに影響を及ぼす問題がサブアプリケーションで生じた場合、サブアプリケーションはイベントをトリガします。このイベントはメインアプリケーションにおいて捕捉可能です。イベントは、そのイベントに反応するために必要なプロパティを定義します。例えば、ToolTip のあるサブアプリケーションでユーザがマウスポインタを移動させたとき、サブアプリケーションの ToolTipManager は ToolTip を作成し、メインアプリケーションの ToolTipManager に対して ToolTip の表示を要求するイベントを送信します。このインタラクションは、同じセキュリティドメインにあるアプリケーションドメイン間で機能します。

セキュリティドメインを越える通信は、LoaderInfo.sharedEvent ディスパッチャおよびカスタムイベントを使用して行われます。このプロセスは極めて透過的であり、どのイベントをイベントリスナーに登録すべきかだけを知る必要があります。LoaderInfo.sharedEvent クラスは、サブアプリケーション上で、メインアプリケーションのマネージャが処理する必要があるイベントを送出します。このイベントにはそれ自体を定義するデータが含まれており、そのデータはセキュリティドメイン全体に伝わります。カスタムイベントクラスは、受信側によってセキュリティドメイン全体における強い型付けを行うことはできないので、注意が必要です。Flash Player によって定義されたクラスにのみ、強い型付けが可能です。

多くの場合、セキュリティドメインを越える通信は開発者に対して透過的です。Flex フレームワークは、イベント受け渡しの詳細とデータマーシャリングを処理します。例えば、FocusManager または PopupManager が管理する機能を使用する場合、アプリケーションの相互運用性を維持するために追加のコードを書くことは通常ありません。基底の Flex クラスはイベントをトリガしてドメインを越えてデータをマーシャリングし、これによりシームレスなユーザー体験を実現します。オブジェクトを定義するためのカスタムクラスを使用する際、場合によっては、セキュリティドメイン全体にわたりデータをマーシャリングするためのカスタムコードを書く必要があります。

同じドメインおよびクロスドメインアプリケーションのロード

メインアプリケーションとサブアプリケーションの間で行われるインタラクションのレベルは、それらの間の信頼関係レベルに依存します。こうした信頼関係は、ある場合とない場合があります。デフォルトでは、別のウェブドメインからロードされたアプリケーションには信頼関係がなく、同じウェブドメインからロードされたアプリケーションには信頼関係があります。ただし、クロスドメインのアプリケーションを信頼できるようにすることは可能です。

同じドメインのアプリケーションとは、メインアプリケーションと同じウェブドメインからメインアプリケーションにロードされたアプリケーションのことです。デフォルトでは、これらは同じセキュリティドメインにロードされ、信頼関係を持ちます。信頼できるアプリケーションは、メインアプリケーションと同じアプリケーションドメインにロードできます。これはそれらのアプリケーションがメインアプリケーションと同じクラス定義を共有することを意味します。信頼できるアプリケーションは、別の兄弟アプリケーションドメインにもロード可能で、これはそれらのアプリケーションがすべてのクラスに対して独自の定義を持っており、マルチバージョン化できることを意味します。

クロスドメインのアプリケーションとは、メインアプリケーションとは別のウェブドメインからメインアプリケーションにロードされたアプリケーションのことです。例えば、メインアプリケーションが `domainA.com` にあり、サブアプリケーションが `domainB.com` にある場合、サブアプリケーションはクロスドメインのアプリケーションとなります。クロスドメインのアプリケーションは、多くの場合信頼できないアプリケーションとして知られていますが、リモートアプリケーションを信頼できるように指定することは可能です (AIR を使用する場合を除く)。

クロスドメインの SWF ファイルをロードするときは、次の点を考慮してください。

- メインアプリケーションが `Security.allowDomain()` メソッドを呼び出す必要が生じる場合もあります。
- ターゲットアプリケーションからデータをロードしようとする場合、サブアプリケーションのサーバは `crossdomain.xml` を要求する可能性があります。
- メインアプリケーションは `trustContent` プロパティの値を設定して、メインアプリケーションとサブアプリケーションの間の信頼関係レベルを定義することができます (`crossdomain.xml` ファイルがそれを許可しており、AIR を使用していない場合)。

信頼されていないアプリケーションは、メインアプリケーションとは別のセキュリティドメインにロードされます。これにより、セキュリティの観点から、Flash Player は互いに信頼関係のないアプリケーション間には制限された相互運用性をのみを提供します。

目的のウェブサーバがポリシーファイルを持つかどうかにかかわらず、すべての SWF ファイルのコンテンツは、どのウェブドメインからでもロードできます。ただし、SWF ファイルのデータ読み取り権限が必要です。クロスドメインのアプリケーションをロードしてそのデータにアクセスするには、リモートサブアプリケーションのホストであるターゲットサーバが、`crossdomain.xml` と呼ばれるポリシーファイルを保持している必要があります。このファイルは、ソースサーバからのアクセスを個別に許可する必要があります。前述の例において、ロードされたアプリケーションデータにアクセスしようとする場合、`domainB` がポリシーファイルを持ち、`domainA` がそれをロードできるようにする必要があります。ポリシーファイルがない場合、アプリケーションデータにアクセスしようすると、Flash Player でセキュリティエラーが発生します。ポリシーファイルはドメインのルートに存在するか、あるいはロードするアプリケーションの中で、その場所を明示的に特定する必要があります。`crossdomain.xml` ファイルについて詳しくは、「クロスドメインポリシーファイルの使用」を参照してください。

`www1.domainA.com` と `www2.domainA.com` のように異なるサブドメインも、サブアプリケーションがクロスドメインであるかどうかを決定する際に適用されます。この例において、メインアプリケーションが `www1.domainA.com` にあり、サブアプリケーションが `www2.domainA.com` にある場合、デフォルトで、これは信頼できないサブアプリケーションであると見なされます。サブアプリケーションからデータをロードするために、サブアプリケーションのドメインがポリシーファイルを要求します。

サブアプリケーションは別ドメインからロードされなければならないため、クロスドメインのアプリケーションの試験的ロードを、Flex Builder の中で直接実行することはできません。このため、メインアプリケーションおよびサブアプリケーションを Flex Builder でビルドするには、アプリケーションを個別のテスト用サーバに配置するようコンパイルプロセスを作成してください。

クロスドメインのアプリケーションは「インポートしてロードする」ことができます。これはメインアプリケーションと同じセキュリティドメインにロードされることを意味します。これを実行するには、SWFLoader で `trustContent` フラグを `true` に設定するか、または `LoaderContext` で同じセキュリティドメインにアプリケーションをロードするよう指定します。クロスドメインのアプリケーションをインポートしてロードするのは、アプリケーションが確実に信頼できる場合に限りです。LoaderContext を使用してサブアプリケーションをロードする方法については、「21 ページの「LoaderContext の指定」」を参照してください。

クロスドメインのアプリケーションを同じセキュリティドメインにロードする別の方法として、`Security.allowDomain()` メソッドを使用する方法があります。最初にアプリケーションをサンドボックス化したアプリケーションとして（別のウェブドメインから、`trustContent` を `true` に設定せずに）ロードします。メインアプリケーションでは、サブアプリケーションのドメインの `allowDomain()` メソッドを、`SWFLoader` コントロールの `complete` イベントハンドラの中から呼び出します。サブアプリケーションでは、メインアプリケーションのドメインで `allowDomain()` メソッドを呼び出します。この呼び出しは、サブアプリケーションのライフサイクルの早い段階で行う必要があります。例えば、アプリケーションの `preinitialize` イベントの中などで行います。`allowDomain()` メソッドを使用してアプリケーションをロードする場合、ターゲットドメインに `crossdomain.xml` ファイルは必要ありません。

ローカル（または同じドメイン）のアプリケーションを別のセキュリティドメインにロードできます。これを実行するには、`SWFLoader` の `source` プロパティに対して、メインアプリケーションのパス名とは異なる URL 文字列を使用します。例えば、IP アドレスを `source` プロパティに指定できます。`Flash Player` がドメイン名を比較する際に、それらは異なると認識され、サブアプリケーションは信頼できないアプリケーションとしてロードされます。これは、ロードされたアプリケーションをサンドボックス化しながら、そのアプリケーションを同じウェブドメインに配置したい場合に一般的に用いられるアプローチです。また、サブアプリケーションが確実に別のセキュリティドメインにロードされるようにするため、同一サーバに別のサブドメインを作成することもできます。

サブアプリケーションは、アプリケーションがロードされた方法に応じてリモートデータにアクセスします。アプリケーションをインポートしてロードする場合、サブアプリケーションはメインアプリケーションのセキュリティドメインの内部から効果的に実行されています。これにより、アプリケーションのオリジナルドメインにあるデータへのアクセス許可が必要となります。サブアプリケーションを別のセキュリティドメインにロードする場合、ロードはオリジナルドメイン内部で実行され、そのドメインにあるリソースには通常どおりにアクセスできます。

サブアプリケーションの作成とロード

`SWFLoader` コントロールを使用して、サブアプリケーションをメインの Flex アプリケーションにロードします。`SWFLoader` コントロールのデフォルトの動作では、開発中のアプリケーションが、同じバージョンの Flex フレームワークを使用してコンパイルされた信頼できるサブアプリケーションをロードするものと見なします。これらのアプリケーションは、通常メインアプリケーションと同じウェブドメインからロードされます。

ロードできる種類のアプリケーションは次のとおりです。

サンドボックス化されたアプリケーション サンドボックス化されたアプリケーションは、別のセキュリティドメインにロードされるサブアプリケーションを含んでいます。このため、マルチバージョン化は可能ですが、信頼関係は持ってません。サードパーティ製アプリケーション、あるいは `RPC` クラスまたは `DataService` に関連した機能を使用する多くのマルチバージョン化されたアプリケーションでは、このアプローチが推奨されます。詳細については、「31 ページの「[サンドボックス化されたアプリケーションの開発](#)」」を参照してください。

マルチバージョン化されたアプリケーション マルチバージョン化されたアプリケーションとは、信頼できるサブアプリケーションをロードする一般的に非常に大規模なアプリケーションです。サブアプリケーションはメインアプリケーションと同じバージョンの Flex フレームワークでコンパイルされている場合も、そうでない場合もあります。詳細については、「39 ページの「[マルチバージョン化されたアプリケーションの開発](#)」」を参照してください。

大きなシングルバージョンのアプリケーションを開発する場合は、サブアプリケーションではなくモジュールの使用を検討してください。詳細については、「13 ページの「[ロードされたアプリケーションとモジュールの比較](#)」」を参照してください。

サブアプリケーションはメインアプリケーションや他のサブアプリケーションとは独立して実行できます。実行するサブアプリケーションに対し、メインアプリケーションまたは他のサブアプリケーションは依存関係を持つことはできません。

Flex Builder でサブアプリケーションを開発する場合は、メインアプリケーションをコンパイルしてもサブアプリケーションも一緒に再コンパイルされるわけではありません。それらは個別にコンパイルする必要があり、Ant または他の自動化されたビルドプロセスが便利です。モジュールの開発では、モジュールをロードするアプリケーションをコンパイルすると Flex Builder がモジュールも一緒にコンパイルしますが、このプロセスはそれとは異なります。

SWFLoader コントロールのあるアプリケーションのロード

The SWFLoader コントロールによって、Flex アプリケーションを他の Flex アプリケーションにロードできます。SWFLoader コントロールには、コンテンツのスケールを変更できるプロパティがあります。また、コンテンツのサイズに合うようにそれ自体のサイズを変更することも可能です。デフォルトでは、SWFLoader コントロールのサイズに合わせてコンテンツのスケールが変わります。SWFLoader コントロールは、要求に応じてコンテンツをロードするようにプログラムして、ロード操作の進行状況を監視することもできます。

SWF ファイルを SWFLoader コントロールにロードするには、SWFLoader コントロールの `source` プロパティの値を設定します。このプロパティはサブアプリケーションの SWF ファイルの場所を定義します。`source` プロパティの値を設定すると、Flex は指定された SWF ファイルをインポートして、それを実行します。Flex アプリケーションで SWFLoader タグを使用する簡単な例については、「SWFLoader コントロールの作成」を参照してください。

SWF ファイルの他に、SWFLoader コントロールは数種類の画像 (JPG、PNG、GIF ファイルなど) を SWF ファイルとしてロードすることもできます。SWFLoader コントロールと一緒にモジュール、RSL、スタイルモジュールをロードすることはできません。

SWFLoader コントロールのデフォルト動作は、ロードされた SWF ファイルの場所によって異なります。ロードされたアプリケーションのロード先がローカルの場合、デフォルトでは SWF ファイルも同じセキュリティドメインおよび子アプリケーションドメインにロードされます。SWFLoader コントロールは、信頼できるシングルバージョンのアプリケーションをロードします。

アプリケーションが、別のサーバから、あるいはメインアプリケーションとは別のサブドメインでロードされた場合、デフォルトではアプリケーションは別のセキュリティドメインと別の兄弟アプリケーションドメインにロードされます。これによって、マルチバージョン化されたアプリケーションが生成されますが、デフォルトでは信頼関係を持たないため、それ自体が一定の制限を持ちます。サードパーティが開発するすべてのアプリケーション、または RPC クラスや `DataService` に関連した機能を使用するマルチバージョン化されたアプリケーションでは、このようなアプローチが推奨されます。

どのようなアプリケーションをロードする場合でも、SWFLoader コントロールでアプリケーションドメインおよびセキュリティドメインを設定するプロパティの値を明示的に設定できます。これらのプロパティは、メインアプリケーションとサブアプリケーションの間の信頼関係レベルを定義し、アプリケーションがマルチバージョンであるかどうかを判断します。次の表で、SWFLoader コントロールのこれらのプロパティを説明します。

プロパティ	説明
loadForCompatibility	<p>ロードされた SWF ファイルがマルチバージョン化されたアプリケーションかどうかを判断します。</p> <p>このプロパティを true に設定すると、ロードされたアプリケーションを Flex フレームワークとは異なるバージョンでコンパイルして、アプリケーションがそのアプリケーションと対話するよう指示します。また、loadForCompatibility プロパティの値を true に設定しても、Loader の LoaderContext はサブアプリケーションをメインアプリケーションの子アプリケーションドメインではなく、兄弟アプリケーションドメインにロードするようになります。</p> <p>このプロパティを false に設定すると、バージョン間の互換性は許可されません。ロードされたアプリケーションがフレームワークとは異なるバージョンでコンパイルされると、そのアプリケーションはランタイムエラーを発生させる可能性があります (2つのアプリケーションが共有されたリソースを使用しており、その Flex フレームワークのバージョン間で API が変更されている場合)。アプリケーションが同じフレームワークのバージョンでコンパイルされた場合、そのアプリケーションは通常通りにメインアプリケーションをロードして対話します。</p> <p>loaderContext プロパティの値を明示的に設定すると、loadForCompatibility プロパティの値は無視されます。</p> <p>このプロパティを設定しても、サブアプリケーションのセキュリティドメインには影響はありません。</p> <p>loadForCompatibility プロパティのデフォルト値は、false です。</p> <p>Flex アプリケーションを開発する際、loadForCompatibility を true に設定する場合は、今後のロードを一貫させるこのメソッドを使用してください。アプリケーションをマルチバージョンからシングルバージョンのアプリケーションに、またはその逆に変更する場合、アプリケーションが同じように動作するとは限りません。</p> <p>詳細については、> を参照してください。SWFLoader.loadForCompatibility.</p>
loaderContext	<p>子 SWF ファイルがロードされるコンテキストを定義します。このコンテキストは、アプリケーションドメインおよびアプリケーションのセキュリティドメインを定義します。</p> <p>loaderContext プロパティを使用して、アプリケーションがロードされるセキュリティドメインを設定できます。SWF ファイルがロードされる前に、ポリシーファイルの配置を要求することも可能です。デフォルトでは、ローカルアプリケーションはサブアプリケーションをメインアプリケーションと同じセキュリティドメインにロードされます。リモート (またはクロスドメイン) アプリケーションでは、デフォルトでサブアプリケーションは異なるセキュリティドメインにロードされます。</p> <p>loaderContext プロパティを使用して、コンテキストに対するアプリケーションドメインを指定することもできます。デフォルトは、Loader のアプリケーションドメインの子です。マルチバージョン化されたアプリケーションをロードする場合は、デフォルトを兄弟アプリケーションドメインに変更してください。一般的に、アプリケーションドメインを指定するには、loaderContext プロパティではなく loadForCompatibility プロパティを使用します。</p> <p>loaderContext プロパティの値を明示的に設定する場合、SWFLoader コントロールはそれ自体の loadForCompatibility および trustContent プロパティの値を無視します。</p> <p>loaderContext プロパティのデフォルト値は null です。</p> <p>LoaderContext は ActionScript の中でのみ指定できます。loaderContext プロパティの使用には、いくつかの制限があります。詳細については、「21 ページの「LoaderContext の指定」」を参照してください。</p> <p>詳細については、> を参照してください。SWFLoader.loaderContext.</p>

プロパティ	説明
trustContent	<p>SWF ファイルがメインアプリケーションのセキュリティドメインにロードされるかどうかを判断します。</p> <p>メインアプリケーションのセキュリティドメインに SWF ファイルをロードするには、これを true に設定します。SWF ファイルが同じドメインあるいはクロスドメインかどうかにかかわらず、SWF ファイルはこれで信頼できるものと見なされます（ポリシーファイルがこれを許可していると想定します）。これは、SWF ファイルがメインアプリケーションのプロパティとメソッドにアクセス可能で、その逆も可能なことを意味します。サードパーティ製アプリケーションに関しては、そのアプリケーションの開発元を完全に信頼できる場合を除き、これを true に設定しないでください。</p> <p>クロスドメインの SWF ファイルを別のセキュリティドメインにロードするには、このプロパティを false に設定します。これにより、ファイルは信頼できないと見なされます。SWF ファイルはメインアプリケーションのプロパティとメソッドにはアクセスできず、またメインアプリケーションも SWF ファイルのプロパティとメソッドにアクセスできません。</p> <p>loaderContext プロパティの値を明示的に設定すると、trustContent プロパティの値は無視されます。</p> <p>trustContent プロパティのデフォルト値は false です。</p> <p>詳細については、> を参照してください。SWFLoader.trustContent.</p>

LoaderContext の指定

サブアプリケーションのローダコンテキストを指定すると、そのアプリケーションのロード先となるセキュリティドメインおよびアプリケーションドメインを制御できるようになります。ローダコンテキストを指定するには、LoaderContext オブジェクトで securityDomain および applicationDomain プロパティの値を設定します。

ローダコンテキストは ActionScript でのみ指定できます。<mx:SWFLoader> タグの属性としてこの値を設定することはできません。

次の例では、カスタム LoaderContext を SWFLoader に割り当てています。これは現在のセキュリティドメインとローダの兄弟アプリケーションドメインを定義します。これはシングルバージョンのサブアプリケーションのみに適用されます。マルチバージョンのアプリケーションまたはサンドボックス化されたアプリケーションには、allowDomain() メソッドを呼び出してメインアプリケーションとサブアプリケーション間の信頼を確立する必要があります。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/MainAppCustomLoaderContext.mx.xml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="initApp()">
  <mx:Script>
    <![CDATA[
      import flash.system.SecurityDomain;
      import flash.system.ApplicationDomain;

      private function initApp():void {
        var context:LoaderContext = new LoaderContext();

        /* Specify the current application's security domain. */
        context.securityDomain = SecurityDomain.currentDomain;

        /* Specify a new ApplicationDomain, which loads the sub-app into a
           peer ApplicationDomain. */
        context.applicationDomain = new ApplicationDomain();

        contentLoader.loaderContext = context;
        contentLoader.source = "http://yourdomain.com/SubApp3.swf";
      }
    ]]>
  </mx:Script>
  <mx:SWFLoader id="contentLoader"/>
</mx:Application>
```

この例では、アプリケーションの `creationComplete` イベントに反応して、`initApp()` メソッドを実行させています。代わりに MXML における SWFLoader コントロールの `source` プロパティを設定した場合は、ローダコンテキストを設定するために `preinitialize` イベントを使用します。アプリケーション起動時に、ローダコンテキストを設定するためにアプリケーションの `creationComplete` イベントを待機すると、時間がかかる場合があります。

`context.securityDomain` プロパティを `currentDomain` に設定すると、SWF ファイルがメインアプリケーションと同じセキュリティドメインにロードされ、信頼できるものと見なされます。ただし、これを実行するには、サブアプリケーションのサーバへのロードを許可するポリシーファイルが存在する必要があります。この結果、SWF ファイルは信頼され、ローカルでロードされた場合と同様に動作します。AIR では、`context.securityDomain` プロパティに対して、いかなる値も指定できません。

指定できる値は、`context.securityDomain` プロパティの `SecurityDomain.currentDomain` のみです。その他のセキュリティドメインを渡そうとすると、`SecurityError` 例外になります。`currentDomain` プロパティの値を一切指定しないと、リモート SWF ファイルがそれ自体のセキュリティドメインにロードされます。これは、SWFLoader の `trustContent` プロパティの値を `true` に設定するなどの他の方法でセキュリティドメインを設定しない限り発生します。

`context.applicationDomain` プロパティのアプリケーションドメインを指定するときは、サブアプリケーションを兄弟アプリケーションドメイン、子アプリケーションドメインまたはロードするアプリケーションと同じアプリケーションドメインに追加できます。次の表は、これらの各メソッドについて説明しています。

context.applicationDomain の値	結果
<code>new ApplicationDomain()</code>	サブアプリケーションを、ロードするアプリケーションと同じドメインにある兄弟アプリケーションドメインにロードします。ロードするアプリケーションが最上位のアプリケーションである場合は、ロードするアプリケーションおよびサブアプリケーションのアプリケーションドメインはどちらも、システムドメインの子になります。 兄弟アプリケーションドメインに置かれたアプリケーションは独自のクラス定義を持つことができるため、互換モードを使用する場合はこれを実行します。これにより、Flex フレームワークの異なるバージョンでアプリケーションのコンパイルが可能になります。
<code>new ApplicationDomain(ApplicationDomain.currentDomain)</code>	サブアプリケーションを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子であるアプリケーションドメインにロードします。クラスの衝突はアプリケーションがロードされる際に解決され、最初のクラス定義が使用されます。 マルチバージョンのアプリケーションをロードしたい場合は、このメソッドは使用しないでください。これは、同じアプリケーションドメインにロードされたアプリケーションは、Flex フレームワークの同一バージョンでコンパイルされる必要があるためです。
<code>application.currentDomain</code>	サブアプリケーションをメインアプリケーションと同じアプリケーションドメインにロードします。通常、サブアプリケーションをロードする場合は、この設定を使用しません。一般的に、RSL および特別にコンパイルされた他のリソースに対してのみ使用します。

サブアプリケーションをロードする際にローダコンテキストを指定する場合は、複数のアプリケーションを同一のアプリケーションドメインにロードしないでください。つまり、同じローダコンテキストは、複数のサブアプリケーションには使用できません。

SWFLoader コントロールによるアプリケーションのアンロード

SWFLoader コントロールを使用してロードしたサブアプリケーションをアンロードするには、次の例に示すように、SWFLoader コントロールの `unloadAndStop()` メソッドを呼び出します。

```
myLoader.unloadAndStop(true);
```

SWFLoader の `source` プロパティを `null` に設定することもできます。それによって `SWFLoaderObject.content.loaderInfo.loader.unload()` メソッドが呼び出されます。このメソッドは、信頼できるアプリケーションに対してのみ明示的に呼び出すことができます。

ロードしたサブアプリケーションが使用したメモリをすべて解放するために、サブアプリケーションの中のオブジェクトまたはクラスへの参照が残らないようにします。`unload()` メソッドはローダのサブアプリケーションのバイトへの参照を解放しますが、サブアプリケーションのコードがまだ使用されている場合、ガベージコレクションは実行されません。

同じ `SWFLoader` コントロールが新しいサブアプリケーションをロードすると、`Flash Player` もサブアプリケーションをアンロードします。元のコンテンツは、新しいアプリケーションがロードされる前に削除されます。

メインアプリケーションにないユーザインターフェイスクラスがサブアプリケーションに含まれている場合、それらのクラスのスタイルはメインアプリケーションの `StyleManager` にロードされます。サブアプリケーションによってサブアプリケーションのメモリが解放されることはありません。この場合、コンパイラのオプションを使用するか、ランタイムスタイルシートをロードして、サブアプリケーションのロード前にスタイルをロードしてください。

サブアプリケーションからのメインアプリケーションへのアクセス

サブアプリケーションからメインアプリケーションのメソッドとプロパティにアクセスするには、いくつかの方法があります。ただし、これらの方法は、メインアプリケーションのアプリケーションドメインに子としてロードされたサブアプリケーションに対してのみ適用できます。サンドボックス化されたアプリケーションやマルチバージョンのアプリケーションに対しては使用できません。

これらの方法には以下のものがあります。

- `Application` クラスの `application` プロパティの使用。このプロパティはアプリケーションのどこからでもルートアプリケーションにアクセスします。詳細については、「`mx.core.Application.application` プロパティの使用」を参照してください。
- `Application` クラスの `parentDocument` プロパティの使用。この方法は、複数のアプリケーションが埋め込まれている場合、直接の親アプリケーションにのみアクセスしたい場合に有用です。詳細については、「`parentDocument` プロパティの使用」を参照してください。

アクセスを許可されていないメンバーにアクセスを試みると、セキュリティエラーが発生する場合がありますので、注意してください。

メインアプリケーションからのサブアプリケーションへのアクセス

メインアプリケーションからサブアプリケーションにアクセスすることは可能ですが、メソッドとプロパティを直接参照することはできません。これは、サブアプリケーションがコンパイル時に埋め込まれた場合を除き、メインアプリケーションをコンパイルする際にコンパイラがサブアプリケーションのメンバーを判断できないためです。

ただし、サブアプリケーションの `SystemManager` への参照を得ることは可能です。その結果、サブアプリケーションのメンバーを、サブアプリケーションのオブジェクトの動的なプロパティおよびメソッドとして取り扱うことができます。

次の例では、リモートサブアプリケーションを `SWFLoader` にロードしています。ここでは、`trustContent` プロパティの値を `true` に設定して、サブアプリケーションをメインアプリケーションと同じセキュリティドメインにロードできるようにします。そして、`SWFLoader` の `content` プロパティを `SystemManager` に渡し、これによりアプリケーションのメンバーに動的にアクセスできるようにします。次にメソッドを呼び出して、サブアプリケーションのプロパティにアクセスします。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/MainAppUsingSubAppMembers.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      import mx.managers.SystemManager;

      private function getValueFromSubApp():void {
        /* Cast the SWFLoader's content as a SystemManager
           to access the sub-application. */
        var subApp:SubApp2 =
          (contentLoader.content as SystemManager).application as SubApp2;

        /* Call a method and access a property of the
           sub-application. */
        label1.text = subApp.doSomething() + subApp.answer;
      }
    ]]>
  </mx:Script>
  <mx:SWFLoader id="contentLoader" visible="false"
    height="0"
    width="0"
    trustContent="true"
    source="http://yourdomain.com/SubApp2.swf"
  />
  <mx:Panel id="myPanel2"
    paddingLeft="10"
    paddingBottom="10"
    paddingTop="10"
    paddingRight="10"
  >
    <mx:Label id="label1"/>
    <mx:Button id="b2" label="Call SubApp2" click="getValueFromSubApp()"/>
  </mx:Panel>
</mx:Application>
```

次の例は、前述の例のメインアプリケーションによってロードされるサブアプリケーションを示しています。ここでは、パブリックプロパティ `answer`、および `String` を返すパブリックメソッド `doSomething()` を定義しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/SubApp2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      // Define a public property.
      public var answer:int = 42;

      // Define a public method that returns a String.
      public function doSomething():String {
        return "The answer is ";
      }
    ]]>
  </mx:Script>
</mx:Application>
```

メインアプリケーションからサブアプリケーションにアクセスする方法の詳細については、「SWFLoader コントロール」を参照してください。

サブアプリケーションがリモートでロードされた場合、そのメンバーにアクセスできるのはアプリケーションがメインアプリケーションと同じセキュリティドメインにあるか、あるいはサブアプリケーションが `Security.allowDomain()` メソッドを呼び出す場合に限られます。この場合、サブアプリケーションのドメイン上でメインアプリケーションから

`Security.allowDomain()` メソッドを呼び出し、サブアプリケーションがこのメソッドをメインアプリケーションのドメインで呼び出す必要があります。`allowDomain()` メソッドも、サブアプリケーションのライフサイクルの早い段階で呼び出す必要があります。例えば、`preinitialize` イベントハンドラの中で呼び出してください。

ロードされたアプリケーションからのクラスインスタンスの作成

メインアプリケーションの中で、ロードされたサブアプリケーション内で定義されるクラスのインスタンスを生成できます。それにより、それらのオブジェクトをメインアプリケーションに追加し、表示リストの他のオブジェクトと対話するようにそれらのオブジェクトと対話できます。サブアプリケーションでクラス定義にアクセスするには、サブアプリケーションのアプリケーションドメインから定義を取得します。

それぞれのアプリケーションドメインには、すべてのクラス定義が含まれています。ロードしたアプリケーションの `LoaderContext` への参照によってアクセスできる `applicationDomain` オブジェクトがあります。`ApplicationDomain` オブジェクトには、`hasDefinition()` と `getDefinition()` の 2 つのメソッドがあり、`hasDefinition()` メソッドは、クラス定義が存在するかどうかを判定します。クラス定義が存在する場合は、`getDefinition()` メソッドを使用してメインアプリケーションの中にそのクラスのインスタンスを生成できます。

他のアプリケーションドメインで定義された `UIComponent` をメインアプリケーションの表示リストに追加することはできません。他のアプリケーションドメインで定義されたクラスのインスタンスを生成するには、サブアプリケーションをメインアプリケーションのアプリケーションドメインにある子アプリケーションドメインにロードする必要があります (`SWFLoader` の `loadForCompatibility` プロパティの値を `true` に設定することはできません)。サブアプリケーションは、メインアプリケーションと同じセキュリティドメインにもある必要があります (`trustContent` は `true` である必要があります)。

コンパイル時において、コンパイラはロードされたアプリケーションのクラスにアクセスできず、リンクをチェックできないため、実行時にロードしたアプリケーション内でクラスが定義されている場合、クラスのインスタンスは動的でのみ作成可能です。サブアプリケーションがコンパイル時に組み込まれていた場合は、インスタンスを動的作成する必要がなく、クラス定義をコンパイラで使用できます。この場合、ジェネリックなクラスタイプではなく特定のクラスタイプで、代わりに `new` キーワードを使用することができます。

次の例では、`SWFLoader` コントロールを使用してサブアプリケーションをメインアプリケーションにロードしています。これは `trustContent` プロパティの値を `true` に設定し、メソッド `createClassInstance()` を定義します。このメソッドは、ロードされたアプリケーションのアプリケーションドメインからカスタムクラスの定義を取得します。その後、このクラスのインスタンスを作成し、表示リストに追加する前にプロパティを設定します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/MainAppUsingSubAppDefinitions.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      import mx.core.UIComponent;
      public function createClassInstance():void {
        // Check that the definition exists.
        if (contentLoader.loaderContext.applicationDomain.hasDefinition('MyButton')) {
          var objClass:Class =
contentLoader.loaderContext.applicationDomain.getDefinition('MyButton') as Class;
          if (objClass != null) {
            var newObj:UIComponent = UIComponent(new objClass());

            // Set properties on the custom class as an associative array.
            newObj["label"] = "Click Me";

            // Add the new instance to the second panel in this application.
            myPanel2.addChild(newObj);
          }
        }
      }
    ]]>
  </mx:Script>
  <!-- The SWFLoader in the first panel loads the sub-application that contains a definition of MyButton. -->
  <mx:Panel id="myPanel" title="SubApp1 Loaded by main application">
    <mx:SWFLoader id="contentLoader" trustContent="true" source="SubApp1.swf"/>
  </mx:Panel>

  <!-- This application adds an instance of the MyButton class to the second panel after the content is loaded. -->
  <mx:Panel id="myPanel2" title="Instance of a Class Defined By SubApp1"/>
</mx:Application>
```

この場合、hasDefinition() メソッドはブール値を返し、ロードされた SWF にクラスが存在するかどうかを示します。クラスが存在する場合、サブアプリケーションは getDefinition() メソッドを使用して Class を取得可能です。その後、返されたクラスで new キーワードを使用すればインスタンスを作成できます。

createClassInstance() メソッドはパブリックである点にも注意してください。プライベートである場合、サブアプリケーションを呼び出せなくなります。

サブアプリケーション SubApp1.swf は、MyButton と呼ばれるカスタムクラスを静的にリンクします。また、完了時に親アプリケーションのメソッドを呼び出します。このメソッドを親アプリケーションの applicationComplete イベントで呼び出すことはできません。その理由は、イベントが初期化プロセスの早い段階でトリガーされる可能性が高いからです。サブアプリケーションがロードおよび初期化を完了してから、定義されているクラスのインスタンスを作成してください。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/SubApp1.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:custom="*"
  layout="vertical"
  creationComplete="initApp()"
  applicationComplete="mx.core.Application.application.createClassInstance()"
>
  <mx:Script>
    <![CDATA[
      private function initApp():void {
        var child:DisplayObject = getChildAt(0);
        var childClassName:String = getQualifiedClassName(child);

        // Show that the qualified class name of the custom button is MyButton.
        trace(childClassName);
      }
    ]]>
  </mx:Script>
  <custom:MyButton id="myButtonId" label="Click Me"/>
</mx:Application>
```

MyButton クラスは Button を拡張し、色を赤に定義する単純な MXML コンポーネントです。次の例は、このカスタムクラスを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/MyButton.mxml -->
<mx:Button xmlns:mx="http://www.adobe.com/2006/mxml" color="red">
</mx:Button>
```

ロードされたアプリケーションによるマウスイベントのリッスン

メインアプリケーションにあるマウスイベントは、子アプリケーションドメイン内にあるロードされたサブアプリケーションからリッスン可能です。これらのイベントをリッスンするには、サブアプリケーションの `systemManager.topLevelSystemManager` 上で `MOUSE_UP` や `MOUSE_MOVE` などのイベントをリッスンします。サブアプリケーションが子アプリケーションドメインにある場合、`topLevelSystemManager` プロパティは、メインアプリケーションの `SystemManager` を参照します。

次のアプリケーションは、子アプリケーションドメインにロードされたサブアプリケーション内の `topLevelSystemManager` 上のマウスイベントにアクセスする方法を示しています。サブアプリケーションが子アプリケーションドメインにない場合は、別の方法で `SystemManagers` にアクセスする必要があります。詳しくは、「35 ページの「[サンドボックス化されたアプリケーションでのマウスイベントのリッスン](#)」および「43 ページの「[マルチバージョン化されたアプリケーションでのマウスイベントのリッスン](#)」」の例を参照してください。

```

<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/ZoomerPattern2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="setup()"
    height="250"
>
    <mx:Script>
    <![CDATA[
        import mx.core.UIComponent;
        import mx.managers.PopUpManager;

        [Bindable]
        public var data:Array = ["Ice Cream", "Fudge", "Whipped Cream", "Nuts"];

        public var zoomTool:UIComponent;

        public function setup():void {
            // Draw the zoom rectangle.
            zoomWidget.graphics.lineStyle(1);
            zoomWidget.graphics.beginFill(0, 0);
            zoomWidget.graphics.drawRect(0, 0, 17, 17);
            zoomWidget.graphics.endFill();

            // Listen for mouse down events.
            zoomWidget.addEventListener(MouseEvent.CLICK, zoom_mouseDownHandler);
        }

        private var lastX:int;
        private var lastY:int;

        private function zoom_mouseDownHandler(event:MouseEvent):void {
            // When the mouse is down, listen for the move and up events.
            systemManager.topLevelSystemManager.addEventListener(
                MouseEvent.CLICK, zoom_mouseMoveHandler, true);
            systemManager.topLevelSystemManager.addEventListener(
                MouseEvent.CLICK, zoom_mouseUpHandler, true);

            // Update the last position of the mouse.
            lastX = event.stageX;
            lastY = event.stageY;

            // Create and pop up the zoomTool. This is what is dragged around.
            // It must be a popup so that it can float over other content.
            zoomTool = new UIComponent();
            PopUpManager.addPopUp(zoomTool, this);
            var pt:Point = new Point(zoomWidget.transform.pixelBounds.x,
                zoomWidget.transform.pixelBounds.y);
            pt = zoomTool.parent.globalToLocal(pt);
            zoomTool.x = pt.x;
            zoomTool.y = pt.y;
            zoomTool.graphics.lineStyle(1);
            zoomTool.graphics.beginFill(0, 0);
            zoomTool.graphics.drawRect(0, 0, 17, 17);
            zoomTool.graphics.endFill();

            // Hide the rectangle that was the target.
            zoomWidget.visible = false;
        }

        private function zoom_mouseMoveHandler(event:MouseEvent):void {
            // Update the position of the dragged rectangle.
    
```

```

zoomTool.x += event.stageX - lastX;
zoomTool.y += event.stageY - lastY;
lastX = event.stageX;
lastY = event.stageY;

var bm:BitmapData = new BitmapData(16, 16);

// Capture the bits on the screen.
bm.draw(DisplayObject(systemManager.topLevelSystemManager), new
  Matrix(1, 0, 0, 1, -zoomTool.transform.pixelBounds.x - 2,
    -zoomTool.transform.pixelBounds.y - 2));

// Create a Bitmap to hold the bits.
if (zoomed.numChildren == 0) {
  var bmp:Bitmap = new Bitmap();
  zoomed.addChild(bmp);
} else
  bmp = zoomed.getChildAt(0) as Bitmap;

// Set the bits.
bmp.bitmapData = bm;

// Zoom in on the bits.
bmp.scaleX = bmp.scaleY = 8;
}

private function zoom_mouseUpHandler(event:Event):void {
  // Remove the listeners.
  systemManager.topLevelSystemManager.removeEventListener(
    MouseEvent.MOUSE_MOVE, zoom_mouseMoveHandler, true);
  systemManager.topLevelSystemManager.removeEventListener(
    MouseEvent.MOUSE_UP, zoom_mouseUpHandler, true);

  // Replace the target rectangle.
  zoomWidget.visible = true;

  // Remove the dragged rectangle.
  PopUpManager.removePopUp(zoomTool);
}

]]>
</mx:Script>
<mx:HBox>
  <mx:HBox backgroundColor="0x00eeee" height="140" paddingTop="4" paddingRight="4">
    <mx:Label text="Drag Rectangle"/>
    <mx:UIComponent id="zoomWidget" width="17" height="17"/>
    <mx:Canvas id="zoom"
      borderStyle="solid"
      borderThickness="2"
      width="132"
      height="132"
    >
      <mx:UIComponent id="zoomed" width="128" height="128"/>
    </mx:Canvas>
  </mx:HBox>
  <mx>List dataProvider="{data1}"/>
</mx:HBox>
</mx:Application>

```

次の例では、前述の例のアプリケーションをメインアプリケーションにロードしています。デフォルト設定でサブアプリケーションをロードします。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/MainZoomerPattern2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" backgroundColor="#FFFFFF">
    <mx:Text text="Default (trusted application in child ApplicationDomain):"/>
    <mx:SWFLoader id="swf1" source="ZoomerPattern2.swf"/>
</mx:Application>
```

この例では、サブアプリケーションの境界外で長方形をドラッグすることはできず、mouseUp イベントはトリガーされません。

この例のアプリケーションは、systemManager.topLevelSystemManager プロパティを使用してメインアプリケーションの SystemManager への参照を取得しています。アプリケーションがスタンドアロンであった場合、次の例のように systemManager プロパティを使用してイベントリスナーを登録できます。

```
systemManager.addEventListener(MouseEvent.MOUSE_MOVE, zoom_mouseMoveHandler, true);
systemManager.addEventListener(MouseEvent.MOUSE_UP, zoom_mouseUpHandler, true);
```

これはアプリケーションがスタンドアロンの場合には機能しますが、子としてロードされた場合には機能しません。サブアプリケーションの SystemManager ではない最上位の SystemManager は、zoomTool クラスの親になります。

ロードされたアプリケーション内の埋め込みフォント

同じアプリケーションドメインに置かれた各 Flex アプリケーションとモジュールは、フォント名を指定すれば同じ組み込みフォントを使用できます。例えば、メインアプリケーションとサブアプリケーションが同じアプリケーションドメインにある限り、メインアプリケーションに組み込まれているフォントをサブアプリケーションでも使用できます。

ロードされたアプリケーション内のモデルとシングルトン

シングルトンとして実装されたモデルおよびメインアプリケーションとそのサブアプリケーションが共有するその他のシングルトンは、サブアプリケーションとメインアプリケーションが別々のアプリケーションドメインにある場合は動作しません。

モデルと他のシングルトンを共有するには、マーシャリングコードを自分で記述するか、クラス定義を保存するブートストラップローダを作成します。

データモデルや他のシングルトンは信頼できないアプリケーションと共有しないことをお勧めします。

関連項目

45 ページの「ブートストラップローディング」

Optimizing loaded applications

メインアプリケーションとサブアプリケーションを作成する際は、できる限りクラス定義が重複しないようにしてください。つまり、各クラスの定義は 1 つのみにし、アプリケーションの合計サイズができる限り小さくなるようにしてください。

特に組み込みアセットが含まれていたり、リンクされたクラスライブラリが多数ある場合、SWF ファイルが大きくなる可能性があります。メインアプリケーションとロードされたアプリケーションが必ず同じバージョンの Flex フレームワークでコンパイルされる場合は、重複したアセットを RSL やその他コンパイル手法で外部化しても構いません。

一般的に、サブアプリケーションはサブアプリケーション自身の RSL を読み込む必要があります。サブアプリケーションをメインアプリケーションの RSL または他のサブアプリケーションの RSL に対してコンパイルするのは標準的な方法ではありません。そのような方法でコンパイルしたとしても、サブアプリケーションは起動時に自身の RSL をロードします。結果として RSL が 2 回ロードされるため、RSL を使用する利点がなくなってしまいます。

サンドボックス化またはマルチバージョン化されたアプリケーションを開発する場合、クラス定義が異なる可能性があるので重複するクラス定義を必ず外部化できるとは限りません。例えば、2つのアプリケーションがバージョンの異なる Flex フレームワークでコンパイルされている場合、マネージャークラス (LayoutManager や CursorManager など) およびフレームワーク内のその他クラスが各アプリケーションで定義されている必要があります。したがって両方のアプリケーションを同じ RSL でコンパイルする手法で、これらのクラスを外部化しないでください。

旧バージョンの Flex では、FocusManager などの特定のマネージャークラスを使用しないメインアプリケーションにモジュールやサブアプリケーションをロードした場合、これらのマネージャークラスを初期化する必要がありました。各アプリケーションがそれぞれのクラス定義を持っている、兄弟アプリケーションドメインにサブアプリケーションをロードすれば、初期化の必要がなくなります。ただし、信頼されていないサブアプリケーションがモーダルダイアログボックスを開くことのないよう、各セキュリティドメインには PopUpManager クラスの定義が含まれている必要があります。

重複するアセットの外部化については、「アプリケーションクラスの外部化」を参照してください。

サンドボックス化されたアプリケーションの開発

サンドボックス化されたアプリケーションは、別のセキュリティドメインにロードされるサブアプリケーションを含んでいます。定義により、これらアプリケーションは別々のアプリケーションドメインにもロードされることになります。このため、マルチバージョン化は可能ですが、信頼関係は持てません。信頼関係を持っていないため、メインアプリケーションとの相互運用性は制限されます。サンドボックス化されたアプリケーションのタイプはポータル、マッシュアップおよびダッシュボードです。

サードパーティ製アプリケーションを使用している場合はすべて、サンドボックス化されたアプリケーションとしてロードしてください。また、RPC クラスあるいは DataServices 関連機能を使用するマルチバージョン化されたアプリケーションを使用している場合も、サンドボックス化されたアプリケーションとしてロードするよう検討してください。サンドボックス化されたアプリケーションとしてロードしない場合は、ブートストラップローダなどの追加コードが必要です。

サンドボックス化されたコンフィギュレーションでは、ロードされたサブアプリケーションは、メインアプリケーションと同じドメインにないことがほとんどです。これは、ロードされたアプリケーションをデフォルトで信頼するとは限らないということです。さらに、すべてのサブアプリケーションが必ずしも同時に表示されるとは限らないため、サンドボックス化されたメインアプリケーションは、常にサブアプリケーションのロードとアンロードが可能である必要があります。

サンドボックス化されたアプリケーションでは、各サブアプリケーションは個別のアプリケーションドメインと個別のセキュリティドメインにロードされます。セキュリティドメイン間の相互運用性は非常に制限されます。サブアプリケーションは大半のステージのプロパティ、メソッドおよびイベントにアクセスできません。他のセキュリティドメインからマウスイベントおよびキーボードイベントを取得することはできません。またメインアプリケーションとのドラッグ&ドロップも実行できず、ポップアップコントロールはサブアプリケーションの境界にクリップされます。メインアプリケーションとサブアプリケーション間の共有データは、マーシャリングする必要があります。

アプリケーションオブジェクトの親チェーンを使用して、異なるセキュリティドメインにあるサブアプリケーションからメインアプリケーションのプロパティにアクセスしようとする、実行時にセキュリティエラーが生じます。さらに SWFLoader.content オブジェクトからアプリケーションにアクセスすることもできません。

サンドボックス化されたアプリケーションのアーキテクチャについては、「5 ページの「[サンドボックス化されたアプリケーションについて](#)」」を参照してください。

サンドボックス化されたアプリケーション内のポップアップコントロール

ポップアップコントロールは、アプリケーションのセキュリティドメインのサンドボックスルートの下に置かれます。これは、サンドボックスのルートが子からのモーダルウィンドウを表示するリクエストをするためです。

ポップアップコントロールはサンドボックスのルートの下に置かれるため、サンドボックス化されたアプリケーション内のポップアップをセンタリングすると、メインアプリケーションではなく、サブアプリケーションが占める画面領域の中央にポップアップが表示されます。このため、ポップアップコントロールがスクロールバーでクリップされたり、サブアプリケーションの前面に表示されることもあります。

メインアプリケーションからの個別のセキュリティドメイン内のサブアプリケーションは、次のような動作を行います。

- モーダルダイアログボックスが起動するとメインアプリケーションがグレー表示になりますが、ポップアップはサブアプリケーションの境界内であればドラッグ可能です。
- ポップアップをセンタリングすると、メインアプリケーションではなく、サブアプリケーションの前面中央にポップアップが表示されます。
- ポップアップコントロールはサブアプリケーションの上のみドラッグできます。ポップアップをサブアプリケーションの外にドラッグすると、ポップアップがクリップされます。
- ポップアップを初めて起動すると、フォーカスがポップアップコントロールに移動します。

サンドボックス化されたアプリケーションでは、ウィンドウまたはダイアログボックスをアプリケーションの境界外に表示できません。この規則はダイアログボックスをすべてのアプリケーションの前面に表示することで、信頼関係のないアプリケーションによるパスワードフィッシングを防ぎます。ポップアップウィンドウを表示する際、`PopUpManager` は親アプリケーションとポップアップに信頼関係があるかどうかを確認した後、ポップアップをホストするか親にたずねます。親がポップアップをホストする場合、ポップアップは親のコンテンツおよび子のコンテンツの前面に表示されます。メインアプリケーションとサブアプリケーションとの間に相互の信頼関係がない場合、`PopUpManager` はダイアログボックスをローカルにホストし、メインアプリケーション自身のコンテンツの前面にだけ表示されるようにします。ただし親と子の信頼関係がある場合、ダイアログボックスが子のアプリケーション境界でクリップされることはありません。

メインアプリケーションとサブアプリケーションに信頼関係がない場合、メインアプリケーションの `SWFLoader` は `scrollRect` によるマスキングとスクロールバーを使用し、サンドボックス化されたアプリケーションのコンテンツをアプリケーションスペース内に制限します。

`ColorPicker`、`ComboBox`、`DateField`、`PopUpButton`、`PopUpMenuButton`、`Menu` などのポップアップ関連コントロールは、通常の位置でクリップされてしまう場合、コンテンツを予期しない方法で表示することがあります。

サンドボックス化されたアプリケーション内の Alert コントロール

サンドボックス化されたアプリケーションの他のポップアップコントロールと同様に、`Alert` もロードされたアプリケーションの端にクリップされます。`Alert` が表示されると、メインアプリケーションとすべてのサブアプリケーションがモーダルダイアログボックスで覆われ、コントロールとのインタラクションができなくなります。このエフェクトは、`Alert` ボックスを開いたサブアプリケーションとその子アプリケーションにのみ適用され、親アプリケーションや兄弟アプリケーションには適用されません。

サンドボックス化されたアプリケーション内のスタイルとスタイルモジュール

子アプリケーションが親とは異なるアプリケーションドメインまたはセキュリティドメインにある場合、`StyleManager` は、親アプリケーションのスタイルを子アプリケーションに渡しません。このため、スタイルはサブアプリケーション内で定義し、メインアプリケーションからスタイルを引き継ぐサブアプリケーションにスタイルに依存しないようにしてください。同様に、メインアプリケーションはサブアプリケーションのスタイルを引き継ぎません。

メインアプリケーションとサブアプリケーションで同じランタイムスタイルシートを使用したい場合は、スタイルモジュールをメインアプリケーションとサブアプリケーションの両方にロードしてください。サブアプリケーションは、メインアプリケーションにロードされたスタイルモジュール内で定義されているスタイルを引き継ぎません。同様にメインアプリケーションは、サブアプリケーションにロードされたスタイルモジュール内で定義されているスタイルを引き継ぎません。

スタイルモジュールは、ロード先のアプリケーションと同じバージョンの Flex フレームワークでコンパイルする必要があります。同一バージョンのフレームワークでコンパイルしない場合、メインアプリケーションとサブアプリケーションで同じスタイルモジュールをロードできない可能性があります。

スタイルモジュールをサブアプリケーションにロードするときにアプリケーションドメインを指定しないと、モジュールはサブアプリケーションの兄弟アプリケーションドメインにロードされます。これにより、サブアプリケーションがスタイルモジュールで定義されているクラスを使用しようとすると、エラーが発生する可能性があります。スタイルモジュールをサブアプリケーションにロードしてそのスタイルを使用するには、スタイルモジュールをサブアプリケーションの子アプリケーションドメインにロードします。StyleManager の loadStyleDeclarations() メソッドには、applicationDomain と securityDomain の 2 つのパラメータがあります。これらのプロパティを使用して、スタイルモジュールのロード先のアプリケーションドメインとセキュリティドメインを制御することができます。

次の例では、スタイルモジュールをサブアプリケーションの子アプリケーションドメインにロードします。

```
private function loadStyle():void {  
    /* Load style module into a child ApplicationDomain by specifying  
       ApplicationDomain.currentDomain. */  
    var eventDispatcher:IEventDispatcher = StyleManager.loadStyleDeclarations(  
        currentTheme + ".swf", true, false, ApplicationDomain.currentDomain);  
    eventDispatcher.addEventListener(StyleEvent.COMPLETE, completeHandler);  
}
```

スタイルモジュールの使用の詳細については、「実行時にスタイルシートのロード」を参照してください。

サンドボックス化されたアプリケーション内のフォント

同じアプリケーションドメインに置かれた各アプリケーションは、フォント名を指定すれば同じ組み込みフォントを使用できます。ただし、サブアプリケーションが異なるアプリケーションドメインにロードされている場合（サンドボックス化されたアプリケーションと同様）は、サブアプリケーションにフォントを組み込む必要があります。

サンドボックス化されたアプリケーション内のフォーカス

メインアプリケーションおよびサブアプリケーションにある FocusManager クラスを統合すると、シームレスなフォーカススキームを作成できます。サブアプリケーションがメインアプリケーションと同じセキュリティドメインにあるかどうかに関係なく、ユーザはサブアプリケーション内を Tab キーで移動できます。また Shift + Tab キーでも移動できます。FocusManager クラスは、セキュリティドメイン間の相互運用性をサポートする数少ないマネージャクラスの 1 つです。

アプリケーションがロードされると、メインアプリケーションはフォーカス候補リストにある SWF ファイルを追跡します。フォーカスをサブアプリケーションに移動すると、ユーザがフォーカスをサブアプリケーションの外に動かすまでの間、フォーカスはサブアプリケーションの FocusManager に引き継がれます。フォーカスをサブアプリケーション外に移動すると、メインアプリケーションの FocusManager はコントロールを再開します。

ポップアップを閉じると、フォーカスは最後にフォーカスがあった場所に戻ります。この動作は他のポップアップやメインアプリケーション内でも同じです。

フォーカスが別のセキュリティサンドボックス内のコントロール上にある場合、そのアプリケーションの FocusManager 上で getFocus() メソッドを呼び出すと、null が返されます。UIComponent.getFocus() メソッドを呼び出した場合も null が返されます。

FocusManager の moveFocus() メソッドを使うと、プログラムによってフォーカスを別の FocusManager の管轄下のコントロールに移すことができます。また、フォーカスのコントロールを別の FocusManager に移すことも可能です。

アプリケーションドメイン間のフォーカスは、モダルダイアログボックスに対しても管理できます。別の最上位ウィンドウがアクティブになっている場合、SystemManager は以前アクティブだった最上位ウィンドウ内の FocusManager を非アクティブにします。SystemManager は他のウィンドウの FocusManager もアクティブにし、ウィンドウの深度 (z-order) を変更します。

サンドボックス化されたアプリケーション内のカーソル

各アプリケーションが異なるセキュリティドメインにある場合、サブアプリケーションのカスタムカーソルは、そのサブアプリケーションに割り当てられた画面領域上のみ表示されます。マウスをサブアプリケーションの境界外に動かすと、カーソルのコントロールをメインアプリケーションの `CursorManager` に渡します。

これらの規則はビジーカーソルにも適用されます。ビジーカーソルが表示された状態で、異なるセキュリティドメインにあるメインアプリケーションにビジーカーソルを移動すると、メインアプリケーションで最後に使用したカーソルに変わります。

サンドボックス化されたアプリケーションのローカライズ

スタイルモジュールと同様に、メインアプリケーションはサブアプリケーションで使用されているリソースモジュールにアクセスできません。逆に、サブアプリケーションはメインアプリケーションのリソースモジュールにアクセスできません。各サブアプリケーションはそれぞれのリソースモジュールをロードする必要があります。

マルチバージョン化されたアプリケーションには、それぞれの `ResourceManager` インスタンスがあります。その結果、各サブアプリケーションはそれぞれの `localeChain` を持つことになります。

複数のサブアプリケーションに同名かつ同言語のリソースバンドルが存在する場合は、最初のサブアプリケーションが使用され、それ以外のサブアプリケーションにある、同名のリソースバンドルのコンテンツは無視されます。これらのサブアプリケーションは、最初に定義されたバンドルを使用します。

スタイルモジュールと同様に、リソースモジュールはサブアプリケーションの子アプリケーションドメインにロードします。リソースバンドルのロード先のアプリケーションドメインとセキュリティドメインを制御します。`IResourceManager` の `loadResourceModule()` メソッドには、`applicationDomain` と `securityDomain` の2つの任意パラメータがあります。

スタイルモジュールと同様に、1つのアプリケーション内のリソースモジュールはすべて同一バージョンの Flex フレームワークでコンパイルする必要があります。アプリケーションがメインかサブかに関係なく、必ず同一バージョンでコンパイルしてください。同じメインアプリケーションまたはサブアプリケーション内では、異なるバージョンのフレームワークでコンパイルされたリソースモジュールを混ぜて使用することはできません。

次の例では、リソースモジュールをサブアプリケーションの子アプリケーションドメインにロードしています。

```
private function loadBundle():void {
    /* Load resource module into a child ApplicationDomain by specifying
       ApplicationDomain.currentDomain. */
    var eventDispatcher:IEventDispatcher = ResourceManager.loadResourceModule(
        "MyBundle.swf", true, false, ApplicationDomain.currentDomain);
    eventDispatcher.addEventListener(StyleEvent.COMPLETE, completeHandler);
}
```

サンドボックス化されたアプリケーション内の ToolTip オブジェクト

セキュリティドメインが異なる場合、ToolTip オブジェクトはサブアプリケーションの `SystemManager` を親とするので、メインアプリケーションクリップによってマスキングされます。ToolTipManager はサブアプリケーションの画面領域に収まるよう、サブアプリケーションのチップのスタイルと位置を決定します。ToolTip オブジェクトがサブアプリケーションの画面領域よりも大きい場合は ToolTip オブジェクトをクリップします。

サブアプリケーションの ToolTips オブジェクトがメインアプリケーションの ToolTip スタイルを引き継ぐことはありません。

サンドボックス化されたアプリケーションの List オブジェクトのエラーチップとデータチップの場合も同様です。

サンドボックス化されたアプリケーション内のレイアウト

ポップアップまたはドロップダウンメニューのあるコントロールが別々のセキュリティドメインにある場合は、まずクリップせずに表示します。これらのコントロールはサブアプリケーションのスペースに制限されているので、サブアプリケーションの境界外に出ようとするクリップされます。

サンドボックス化されたアプリケーション内のディープリンク

BrowserManager は Flex アプリケーションのディープリンクサポートを管理します。BrowserManager はそのセキュリティドメイン内のシングルトンです。兄弟アプリケーションドメインのサブアプリケーションは、サブアプリケーションが信頼できるものかどうかにかかわらず、メインアプリケーションの BrowserManager にアクセスできません。そのため、異なるセキュリティドメインまたはアプリケーションドメインにあるサブアプリケーションは、URL の変更も URL へのアクセスもできません。

信頼関係のないサブアプリケーションには、URL へのアクセスを許可しないでください。サブアプリケーションが信頼されている場合（サンドボックス化されていない、マルチバージョン化されたアプリケーションなど）は、サブアプリケーションとメインアプリケーションの BrowserManager 間のインタラクションを扱うカスタムコードを記述できます。

サブアプリケーション内から BrowserManager のインスタンスを取得しようとする、Flash Player はエラーを発生しません。

サンドボックス化されたアプリケーションでのドラッグアンドドロップ

サブアプリケーションがメインアプリケーションとは異なるセキュリティドメインにある場合、両アプリケーション間でデータをドラッグすることはできません。DragProxy を使用すると、アイテムをサブアプリケーションの画面領域の端にドラッグできます。ドラッグした時点で、カーソルは移動先のセキュリティドメインで定義されているカーソルに戻ります。

アイテムのプロキシはサブアプリケーションの境界にとどまり、場合によってはクリップされます。

サンドボックス化されたアプリケーションでのマウスイベントのリッスン

サブアプリケーションとメインアプリケーション間のマウスのインタラクションは、異なるアプリケーションドメインでイベントが発生する場合、特に複雑になる可能性があります。アプリケーションのセキュリティドメインが異なると、このインタラクションはさらにわかりにくくなります。セキュリティドメイン外のマウスイベントをリッスンするには、SandboxMouseEvent オブジェクトを使用します。SandboxMouseEvent オブジェクトを使用すると、サブアプリケーションからトリガーされたマウスイベントをメインアプリケーションでリッスンできます。その逆の場合も同様にリッスンできます。

次のアプリケーションは一部の例外はありますが、「43 ページの「マルチバージョン化されたアプリケーションでのマウスイベントのリッスン」」で紹介したアプリケーションとほぼ同じです。例えば、MouseEvent.MOUSE_MOVE と MouseEvent.MOUSE_UP イベントを登録する際に、SandboxMouseEvent.MOUSE_UP も登録する点が異なります。このイベントはセキュリティドメイン内のすべての SystemManager で登録可能です。このイベントがトリガーされた場合、すべてのアプリケーションで通知を受け取ることができます。このアプリケーションでは、globalToLocal() メソッドでマウス位置を取得します。ステージへのアクセスがない場合に、サブアプリケーションのオブジェクトの絶対位置を決定する方法が示されます。

```

<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/ZoomerPattern4.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="setup()"
    height="250"
>
    <mx:Script>
    <![CDATA[
        import mx.core.UIComponent;
        import mx.events.SandboxMouseEvent;
        import mx.managers.PopUpManager;

        [Bindable]
        public var data:Array = ["Ice Cream", "Fudge", "Whipped Cream", "Nuts"];

        public var zoomTool:UIComponent;

        public function setup():void {
            // Draw the zoom rectangle.
            zoomWidget.graphics.lineStyle(1);
            zoomWidget.graphics.beginFill(0, 0);
            zoomWidget.graphics.drawRect(0, 0, 17, 17);
            zoomWidget.graphics.endFill();

            // Listen for mouse down events.
            zoomWidget.addEventListener(MouseEvent.CLICK, zoom_mouseDownHandler);
        }

        private var lastX:int;
        private var lastY:int;

        private function zoom_mouseDownHandler(event:MouseEvent):void {
            // When the mouse is down, listen for the move and up events.
            // The getSandboxRoot() method lets you listen to all mouse activity in your
            // SecurityDomain.

            systemManager.getSandboxRoot().addEventListener(
                MouseEvent.CLICK, zoom_mouseClickHandler, true);
            systemManager.getSandboxRoot().addEventListener(
                MouseEvent.CLICK, zoom_mouseClickHandler, true);

            // The SandboxMouseEvents provide you with some mouse information,
            // but not its position
            systemManager.getSandboxRoot().addEventListener(
                SandboxMouseEvent.CLICK, zoom_mouseClickHandler);

            // Update last position of the mouse.
            lastX = event.stageX;
            lastY = event.stageY;

            // Create and pop up the zoomTool. This is the rectangle that is dragged around.
            // It must be a popup so that it can float over other content.
            zoomTool = new UIComponent();
            PopUpManager.addPopUp(zoomTool, this);

            var pt:Point = new Point(zoomWidget.transform.pixelBounds.x,
                zoomWidget.transform.pixelBounds.y);
            pt = zoomTool.parent.globalToLocal(pt);
            zoomTool.x = pt.x;
            zoomTool.y = pt.y;
        }
    ]]>
    </mx:Script>
</mx:Application>

```

```
zoomTool.graphics.lineStyle(1);
zoomTool.graphics.beginFill(0, 0);
zoomTool.graphics.drawRect(0, 0, 17, 17);
zoomTool.graphics.endFill();

// Hide the rectangle that was the target.
zoomWidget.visible = false;
}

private function zoom_mouseMoveHandler(event:MouseEvent):void {
    // Update the position of the dragged rectangle.
    zoomTool.x += event.stageX - lastX;
    zoomTool.y += event.stageY - lastY;
    lastX = event.stageX;
    lastY = event.stageY;

    var bm:BitmapData = new BitmapData(16, 16);

    // Capture the bits on the screen.
    // Use the globalToLocal() method to get the coordinates of the rectangle.
    // Untrusted sub applications do not have access to the stage so you have
    // to call the globalToLocal() method on a point.
    var pt:Point = new Point(zoomTool.transform.pixelBounds.x + 2,
        zoomTool.transform.pixelBounds.y + 2);
    pt = DisplayObject(systemManager.getSandboxRoot()).globalToLocal(pt);
    bm.draw(DisplayObject(systemManager.getSandboxRoot()),
        new Matrix(1, 0, 0, 1, -pt.x, -pt.y));

    // Create a Bitmap to hold the bits.
    if (zoomed.numChildren == 0) {
        var bmp:Bitmap = new Bitmap();
        zoomed.addChild(bmp);
    } else
        bmp = zoomed.getChildAt(0) as Bitmap;

    // Set the bits.
    bmp.bitmapData = bm;

    // Zoom in on the bits.
    bmp.scaleX = bmp.scaleY = 8;
}

private function zoom_mouseUpHandler(event:Event):void {
    // Remove the listeners.
    systemManager.getSandboxRoot().removeEventListener(
        MouseEvent.MOUSE_MOVE, zoom_mouseMoveHandler, true);
    systemManager.getSandboxRoot().removeEventListener(
        MouseEvent.MOUSE_UP, zoom_mouseUpHandler, true);
    systemManager.getSandboxRoot().removeEventListener(
        SandboxMouseEvent.MOUSE_UP, zoom_mouseUpHandler, true);

    // Replace the target rectangle.
    zoomWidget.visible = true;

    // Remove the dragged rectangle.
    PopupManager.removePopup(zoomTool);
}
```

```

    }

]]>
</mx:Script>
<mx:HBox>
    <mx:HBox backgroundColor="0x00e000" height="140" paddingTop="4" paddingRight="4">
        <mx:Label text="Drag Rectangle"/>
        <mx:UIComponent id="zoomWidget" width="17" height="17"/>
        <mx:Canvas id="zoom"
            borderStyle="solid"
            borderThickness="2"
            width="132"
            height="132"
        >
            <mx:UIComponent id="zoomed" width="128" height="128"/>
        </mx:Canvas>
    </mx:HBox>
    <mx>List dataProvider="{data1}"/>
</mx:HBox>
</mx:Application>

```

次の例では、前述の例のアプリケーションをメインアプリケーションにロードしています。trustContent プロパティの値を false に設定し、リモートでロードされた信頼関係がないサブアプリケーションの動作を模倣しています。また、サブアプリケーションが PopUpManager を使用しているため、PopUpManager にもリンクします。

```

<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/MainZoomerPattern4.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" backgroundColor="#FFFFFF">
    <mx:Script>
        <![CDATA[
            /* The PopUpManager must be linked in to all applications that are
               at the same security sandbox root. Because the sub application
               in this example uses the PopUpManager, the main application must also
               link it it. */
            import mx.managers.PopUpManager; PopUpManager;
        ]]>
    </mx:Script>
    <mx:Text text="Portal (untrusted versioning application):"/>
    <mx:SWFLoader id="swf1"
        compatibleLoad="true"
        trustContent="false"
        source="ZoomerPattern4.swf"
    />
</mx:Application>

```

サンドボックス化されたアプリケーションでの flashVars 変数へのアクセス

flashVars 変数としてサブアプリケーションに渡されたアプリケーションパラメータにアクセスできます。これを行うには、getSandboxRoot() メソッドを使用してアプリケーションオブジェクトのパラメータの配列にアクセスします。

次の例では、ルートアプリケーションの参照を取得して、パラメータにアクセスします。

```

var app:DisplayObject = DisplayObject(
    SystemManager.getSandboxRoot()["application"]);
var parameters:Object = app["parameters"];

```

マルチバージョン化されたアプリケーションの開発

非常に大きいアプリケーションであっても、リリース前にはほぼすべてのアプリケーションでソースコード全体のベースがコンパイルされます。この再コンパイルにより、アプリケーション全体で確実に同じ API を使用することができます。この方法は、デスクトップソフトウェアおよびリリース管理が厳格なソフトウェアに適していますが、アプリケーションが頻繁に更新されるアプリケーション開発のインターネットモデルには適さない場合があります。

インターネットアプリケーションの開発およびリリースは段階的に行われることが多く、新しい機能が常に追加されています。こうしたアプリケーション開発を行う場合は、マルチバージョン化されたアプリケーションを作成できる Flex を使用すると便利です。複数のサブアプリケーションを異なるバージョンの Flex フレームワークでコンパイルした場合でも、1つのアプリケーションにこれらのサブアプリケーションをロードすることができます。サンドボックス化されたアプリケーションで複数の開発者グループが別々に機能開発している場合、マルチバージョン化されたアプリケーションを使用することはよくあり、継続的に改善やリリースが行われる非常に大型のアプリケーションでも広く使用されています。

マルチバージョン化されたアプリケーションでの RPC および DataServices 関連機能の使用は制限されています。制限内容は使用するリモートデータアクセスによって異なります。具体的には、プロキシサーバを介した RPC クラスおよびプロキシサーバを介さない RPC クラス、DataServices と RemoteObject の機能によって異なります。大半の場合、マルチバージョンではなくサンドボックス化されたアプリケーションを開発した方が賢明です。詳細については、「40 ページの「[マルチバージョン化されたアプリケーションでの RPC および DataService クラスの使用](#)」を参照してください。

Flex は、バージョン 3.2 以降の Flex フレームワークでコンパイルされた、マルチバージョン化されたアプリケーションをサポートしています。例えば、メインアプリケーションは Flex 4 アプリケーション、またロードされるアプリケーションはフレームワークのバージョン 3.2 または 4 のどちらでもコンパイル可能です。3.2 以前のバージョンのフレームワークでコンパイルしたアプリケーションをロードしようとする場合、両方のアプリケーションが Flex フレームワークのバージョン間で API が変更した共有リソースを使用しているとランタイムエラーが発生します。

一般的には、Flash Player 10 用にコンパイルされたメインアプリケーションは、Flash Player 10 または Flash Player 9 用にコンパイルされたサブアプリケーションをロードできます。ただし、Flash Player 9 用にコンパイルされたメインアプリケーションは、Flash Player 10 用にコンパイルされたサブアプリケーションをロードできません。

異なるバージョンの Flex フレームワークでコンパイルしたアプリケーションは、ローカル（同じドメインまたは信頼関係を持つサブアプリケーションドメインから）またはクロスドメイン（デフォルトで信頼関係のない異なるドメインから）のいずれかでロード可能です。信頼関係を持つマルチバージョン化されたアプリケーションのメインアプリケーションとの相互運用性は、信頼関係を持つシングルバージョン化されたアプリケーションとほぼ同レベルです。Style と ResourceBundle は共有されませんが、それ以外の相互運用性は同レベルです。信頼関係を持たないマルチバージョン化されたアプリケーションは、信頼関係を持たないシングルバージョン化されたアプリケーションがロードされた場合よりも相互運用性が若干高くなります。信頼関係を持たないマルチバージョン化されたアプリケーションの場合、フォーカス管理とマウスイベントはメインアプリケーションで動作します。

マルチバージョン化されたアプリケーションをロードする際は、SWFLoader コントロールの loadForCompatibility プロパティを true に設定します。このプロパティを設定すると、ローダはアプリケーションを兄弟アプリケーションドメインにロードするよう指示されます。兄弟アプリケーションドメインにあるアプリケーションはそれぞれのクラス定義を維持し、必要に応じてイベントを受け渡します。例えば、最上位のアプリケーションのマネージャのみが実行可能な内容をサブアプリケーションが行う必要がある場合は、イベントの送出によりメインアプリケーションと通信します。

次の節では、マルチバージョン化されたアプリケーションの開発方法を説明します。ここでは、すべてのアプリケーションが信頼関係を持ち（同じセキュリティドメインにロードされている）、メインアプリケーションの兄弟アプリケーションドメインにロードされていると仮定します。別々のセキュリティドメインにあるアプリケーションの開発については、「31 ページの「[サンドボックス化されたアプリケーションの開発](#)」を参照してください。

マルチバージョン化されたアプリケーションでの RPC および DataService クラスの使用

RPC クラスおよび DataService クラスをマルチバージョン化されたサブアプリケーションで使用するのには、特殊なケースです。メッセージ送信を機能させるには、同じセキュリティドメインにあるすべてのアプリケーションが RPC クラス定義と DataService クラス定義を共有する必要があり、カスタムバリューオブジェクトクラスの定義も共有する必要があります。ただし、メインアプリケーションとサブアプリケーションに同じ定義のセットを使用させるには、サブアプリケーションをメインアプリケーションの子アプリケーションドメインにロードするしかなく、その結果、マルチバージョン化されたアプリケーションの動作を妨げる場合があるため、この方法では問題が生じる可能性があります。このため、マルチバージョン化されたアプリケーションで RPC クラスを使用するには、次のいずれかの方法を使用します。

サンドボックス化されたアプリケーションとしてロードする この方法の場合、SWFLoader コントロールの complete イベントハンドラのサブアプリケーションのドメイン上で、メインアプリケーションから Security.allowDomain() メソッドを呼び出す必要があります。また、メインアプリケーションのドメイン上でサブアプリケーションの Security.allowDomain() メソッドも呼び出す必要があります。これによって、マルチバージョン化されたアプリケーションと同レベルの相互運用性が提供されます。サブアプリケーションでは、初期化サイクルの早い段階で allowDomain() メソッドを呼び出す必要があります。アプリケーションの preinitialize イベントを使用してください。

ブートストラップローダを使用する まず、ブートストラップローダでメッセージクラスを定義してから、メインアプリケーションをブートストラップローダにロードします。これで、これらのクラス定義をメインアプリケーションとそのサブアプリケーションで共有できるようになります。

プロキシサーバを介さずに RPC クラス (WebService や HTTPService など) を使用する場合は、サブアプリケーションをサンドボックス化されたアプリケーションとしてロードしたり、ブートストラップクラスローダのクラスを外部化する必要はありません。プロキシサーバを介さずにこれらのクラスを使用するアプリケーションは、マルチバージョン化されたアプリケーションと同様に動作します。必ずターゲットサービスの URL をメインアプリケーションと同じサーバ上に置くか、crossdomain.xml ファイルをアクセスを許可するターゲットサービス上に置いてください。

関連項目

31 ページの「[サンドボックス化されたアプリケーションの開発](#)」

45 ページの「[ブートストラップローディング](#)」

マルチバージョン化されたアプリケーション内のポップアップコントロール

サブアプリケーションによって起動されたポップアップコントロールは、アプリケーション全体に浮遊し、ステージ上の任意の場所にドラッグできます。ポップアップコントロールはサブアプリケーションのアプリケーションドメインに作成され、メインアプリケーションの PopUpManager に渡されます。ポップアップコントロールは IUIComponent のように強い型付けを行うことはできないので、サブアプリケーションからメインアプリケーションにマーシャリングして表示します。

ポップアップウィンドウはマルチバージョン化されたサブアプリケーション内で次のように動作します (これらの動作はシングルバージョンサブアプリケーションの場合も同じです)。

- モーダルダイアログボックスを起動すると、モーダルダイアログを起動したサブアプリケーションだけでなく、メインアプリケーション全体がグレー表示になります。
- ポップアップをセンタリングすると、サブアプリケーションだけでなく、メインアプリケーションの前面中央にポップアップを表示します
- ポップアップコントロールは、サブアプリケーションだけでなく、アプリケーション全体でドラッグできます。
- サブアプリケーションからポップアップを初めて起動すると、フォーカスがポップアップに移動します。

PopUpManager は、サンドボックスのルートであるすべてのアプリケーションにリンクしている必要があります。子アプリケーションで PopUpManager を使用する場合は、メインアプリケーションにも PopUpManager のインスタンスが存在する必要があります。PopUpManager を使用していないメインアプリケーションに PopUpManager をリンクするには、次のコードを追加します。

```
import mx.managers.PopUpManager;  
private var popupManager1:PopUpManager;
```

PopUpManager をアプリケーションにリンクしていない場合、サンドボックス内のサブアプリケーションは、モーダルダイアログボックスの表示をリクエストします。そのため、次のようなエラーが表示される可能性があります。

```
No class registered for interface 'mx.managers::IPopUpManager'.
```

サブアプリケーションでリストコントロールを使用するには、追加コードが必要です。サブアプリケーションのポップアップに List コントロールがあり、メインアプリケーションにない場合、List クラスをメインアプリケーションにもリンクする必要があります。また、List コントロールにデータプロバイダがある必要があります。この場合、次の例のように空の配列を設定できます。

```
<mx:List visible="false" includeInLayout="false" dataProvider="" />
```

マルチバージョン化されたアプリケーション内の埋め込みフォント

同じアプリケーションドメインに置かれた各アプリケーションは、フォント名を指定すれば同じ組み込みフォントを使用できます。ただし、サブアプリケーションが異なるアプリケーションドメインにロードされている場合（マルチバージョン化されたアプリケーションと同様）は、サブアプリケーションにフォントを組み込む必要があります。

マルチバージョン化されたアプリケーション内のスタイルとスタイルモジュール

マルチバージョン化されたサブアプリケーションでのスタイルとスタイルモジュールの使用は、サンドボックス化されたサブアプリケーションで使用する場合と同じです。サブアプリケーションでのスタイルとスタイルモジュールの使用については、「32 ページの「[サンドボックス化されたアプリケーション内のスタイルとスタイルモジュール](#)」を参照してください。

マルチバージョン化されたアプリケーション内のフォーカス

マルチバージョン化されたサブアプリケーションでの FocusManager の使用は、サンドボックス化されたサブアプリケーションで使用する場合と同じです（「33 ページの「[サンドボックス化されたアプリケーション内のフォーカス](#)」を参照してください）。

マルチバージョン化されたアプリケーション内のカーソル

メインアプリケーションおよびサブアプリケーションにある CursorManager クラスは PopupManager クラスと同じように通信します。サブアプリケーションでカーソルをカスタムカーソルに変更した場合、カーソルをサブアプリケーションからメインアプリケーションに移動しても、そのカスタムカーソルが表示されます。サブアプリケーションで行った CursorManager のメソッド呼び出しやプロパティ設定はメインアプリケーションにも適用されます。また、その逆も同様です。

サブアプリケーション上に表示されているビジーカーソルをメインアプリケーションに移動すると、カーソルはビジーカーソルのまま表示されます。

マルチバージョン化されたアプリケーションのローカライズ

マルチバージョン化されたサブアプリケーションでのリソースバンドルの使用は、サンドボックス化されたサブアプリケーションで使用する場合と同じです。サブアプリケーションでの FocusManager の使用については、「34 ページの「[サンドボックス化されたアプリケーションのローカライズ](#)」を参照してください。

マルチバージョン化されたアプリケーション内の ToolTip オブジェクト

ToolTip オブジェクトはサブアプリケーションのアプリケーションドメインに作成され、メインアプリケーションの ToolTipManager に渡されます。これは、複数のアプリケーション境界にまたがる、クラスのマーシャリングの一例です。

これは、マルチバージョン化されたアプリケーションの List オブジェクトのエラーチップとデータチップの場合も同様に動作します。

マルチバージョン化されたアプリケーション内のレイアウト

サブアプリケーションが画面レイアウトに使用できる画面領域は、メインアプリケーションによって決定されます。

各アプリケーションの LayoutManager は、それぞれ独立して実行されます。アプリケーションはクリップされません。

マルチバージョン化されたアプリケーション内のディープリンク

異なるアプリケーションドメインにあるサブアプリケーションから、メインアプリケーションの BrowserManager に直接アクセスすることはできません。サブアプリケーションの BrowserManager は無効になっています。これはサンドボックスでもマルチバージョン化されたサブアプリケーションでも同様です。詳細については、「35 ページの「[サンドボックス化されたアプリケーション内のディープリンク](#)」」を参照してください。

マルチバージョン化されたアプリケーションでのドラッグアンドドロップ

サブアプリケーションがメインアプリケーションとは異なるアプリケーションドメインにあっても、セキュリティドメインが同じであれば、サブアプリケーションのリストアイテムをメインアプリケーションのリストにドラッグすることができます。また、その逆も可能です。複数のアプリケーション間でこの動作が実行できるようにするには、ドロップターゲットを定義するクラスがパブリックである必要があります。

DragProxy は、マウスがどのアプリケーションにあるかに関係なく、イベント全体を通じてアイテムの外観を維持します。単一アプリケーション内で動作しているかのように、ドラッグイベントがあらゆるソースおよびデスティネーションコントロールに対してトリガーされます。

ドラッグ操作中には、同時に DragManager がサブアプリケーションのアプリケーションドメインに DragProxy を生成し、メインアプリケーションの DragManager に渡しています。

メインアプリケーションからサブアプリケーションに強く型付けされたオブジェクトを渡すことは不可能です。また、その逆も同様です。結果として、DragManager は強く型付けされていない DragProxy を受け付けます。

IDragManager インターフェイスで定義したすべてのプロパティとメソッドは、メインアプリケーションの DragManager によって処理されます。

ジェネリックタイプのオブジェクトをドラッグすると、Flash Player のアプリケーションドメイン内のジェネリックタイプ Object になります。Flex はカスタムコードを記述しなくてもドラッグ&ドロップ操作ができるよう、プロパティをマーシャリングします。

ドロップデータの一部として使用しているカスタムクラスは、強く型付けされたものとして共有することはできません。その場合はドロップイベントをオーバーライドし、データがドラッグソースからドロップターゲットにマーシャリングされるようにロジックを追加します。

マルチバージョン化されたアプリケーションでのマウスイベントのリッスン

メインアプリケーションとサブアプリケーション間のマウスのインタラクションは、異なるアプリケーションドメインでイベントが発生する場合に特に複雑になる可能性があります。例えば、サブアプリケーションのオブジェクトをクリックしてドラッグし、メインアプリケーション上でドロップした場合、MOUSE_UP や MOUSE_DOWN_OUTSIDE、MOUSE_LEAVE など Flex の代表的なマウスイベントがトリガーされますが、これらイベントをサブアプリケーションで直接リッスンすることはできません。

複数のアプリケーションドメイン間でマウスイベントをリッスンするには、セキュリティドメインにあるすべてのマウスアクティビティをリッスンしてください。リッスンするには、サンドボックスのルートへの参照を取得し、イベントリスナーを SystemManager に登録します。

別のアプリケーションドメインにロードする場合、メインアプリケーションの SystemManager が別のアプリケーションドメインにあるので、topLevelSystemManager プロパティはサブアプリケーションの SystemManager を参照します。したがって、メインアプリケーションの SystemManager への参照を取得するために topLevelSystemManager プロパティを使用することはありません。代わりに systemManager.getSandboxRoot() メソッドを使用して、セキュリティドメイン内の最上位レベルの SystemManager への参照を取得してください。この SystemManager からは、現在のセキュリティドメインにあるすべてのマウスアクティビティにアクセス可能です。

次の例では、セキュリティドメインにある最上位の SystemManager への参照を取得するのに systemManager.getSandboxRoot() メソッドの呼び出しを使用しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/ZoomerPattern3.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical"
  creationComplete="setup()"
  height="250"
>
  <mx:Script>
  <![CDATA[
    import mx.core.UIComponent;
    import mx.managers.PopUpManager;

    [Bindable]
    public var data:Array = ["Ice Cream", "Fudge", "Whipped Cream", "Nuts"];

    public var zoomTool:UIComponent;

    public function setup():void {
      // Draw the zoom rectangle.
      zoomWidget.graphics.lineStyle(1);
      zoomWidget.graphics.beginFill(0, 0);
      zoomWidget.graphics.drawRect(0, 0, 17, 17);
      zoomWidget.graphics.endFill();

      // Listen for mouse down events.
      zoomWidget.addEventListener(MouseEvent.MOUSE_DOWN, zoom_mouseDownHandler);
    }

    private var lastX:int;
    private var lastY:int;

    private function zoom_mouseDownHandler(event:MouseEvent):void {
      // When the mouse is down, listen for the move and up events.
      // The getSandboxRoot() method lets you listen to all mouse activity in your
      // SecurityDomain.
      systemManager.getSandboxRoot().addEventListener(
        MouseEvent.MOUSE_MOVE, zoom_mouseMoveHandler, true);
      systemManager.getSandboxRoot().addEventListener(
```

```

        MouseEvent.MOUSE_UP, zoom_mouseUpHandler, true);

// Update last position of the mouse.
lastX = event.stageX;
lastY = event.stageY;

// Create and pop up the zoomTool. This is the rectangle that is dragged around.
// It must be a popup so that it can float over other content.
zoomTool = new UIComponent();
PopUpManager.addPopUp(zoomTool, this);

var pt:Point = new Point(zoomWidget.transform.pixelBounds.x,
    zoomWidget.transform.pixelBounds.y);
pt = zoomTool.parent.globalToLocal(pt);
zoomTool.x = pt.x;
zoomTool.y = pt.y;
zoomTool.graphics.lineStyle(1);
zoomTool.graphics.beginFill(0, 0);
zoomTool.graphics.drawRect(0, 0, 17, 17);
zoomTool.graphics.endFill();

// Hide the rectangle that was the target.
zoomWidget.visible = false;
}

private function zoom_mouseMoveHandler(event:MouseEvent):void {
    // Update the position of the dragged rectangle.
    zoomTool.x += event.stageX - lastX;
    zoomTool.y += event.stageY - lastY;
    lastX = event.stageX;
    lastY = event.stageY;

    var bm:BitmapData = new BitmapData(16, 16);

    // Capture the bits on the screen.
    // You must use the getSandboxRoot() method here, too, because it gets
    // the parent of all of the pixels you are allowed to access.
    bm.draw(DisplayObject(systemManager.getSandboxRoot()),
        new Matrix(1, 0, 0, 1, -zoomTool.transform.pixelBounds.x - 2,
            -zoomTool.transform.pixelBounds.y - 2));

    // Create a Bitmap to hold the bits.
    if (zoomed.numChildren == 0) {
        var bmp:Bitmap = new Bitmap();
        zoomed.addChild(bmp);
    } else
        bmp = zoomed.getChildAt(0) as Bitmap;

    // Set the bits.
    bmp.bitmapData = bm;

    // Zoom in on the bits.
    bmp.scaleX = bmp.scaleY = 8;
}

private function zoom_mouseUpHandler(event:Event):void {
    // Remove the listeners.
    systemManager.getSandboxRoot().removeEventListener(
        MouseEvent.MOUSE_MOVE, zoom_mouseMoveHandler, true);
    systemManager.getSandboxRoot().removeEventListener(
        MouseEvent.MOUSE_UP, zoom_mouseUpHandler, true);
}

```

```

        // Replace the target rectangle.
        zoomWidget.visible = true;

        // Remove the dragged rectangle.
        PopupManager.removePopUp(zoomTool);
    }

    ]]>
</mx:Script>
<mx:HBox>
    <mx:HBox backgroundColor="0x00eeee" height="140" paddingTop="4" paddingRight="4">
        <mx:Label text="Drag Rectangle"/>
        <mx:UIComponent id="zoomWidget" width="17" height="17"/>
        <mx:Canvas id="zoom"
            borderStyle="solid"
            borderThickness="2"
            width="132"
            height="132"
        >
            <mx:UIComponent id="zoomed" width="128" height="128"/>
        </mx:Canvas>
    </mx:HBox>
    <mx>List dataProvider="{data1}"/>
</mx:HBox>
</mx:Application>

```

次の例では、前述の例のアプリケーションをメインアプリケーションにロードしています。loadForCompatibility プロパティを true に設定して、サブアプリケーションがマルチバージョン化されたアプリケーションをロードするようにしています。

```

<?xml version="1.0" encoding="utf-8"?>
<!-- apploading/MainZoomerPattern3.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" backgroundColor="#FFFFFF">
    <mx:Text text="Cross-Versioning (trusted versioning application):"/>
    <mx:SWFLoader id="swf1" compatibleLoad="true" source="ZoomerPattern3.swf"/>
</mx:Application>

```

ブートストラップローディング

ブートストラップローダは、アプリケーション間で共有するクラスを定義する、小さいアプリケーションです。

ブートストラップローダは実際のメインアプリケーションに代わって最初にロードされ、その後、自身の子アプリケーションドメインにメインアプリケーションをロードします。メインアプリケーションは次にサブアプリケーションを兄弟アプリケーションドメインにロードします。つまり、どちらのアプリケーションもブートストラップローダのアプリケーションドメインの子アプリケーションドメインにロードされます。その結果、メインアプリケーションとサブアプリケーションは、ブートストラップで定義したすべてのクラスを共有し、それ以外については独自のクラス定義を使用します。

通常、1つのセキュリティドメインに持てるブートストラップローダは1つのみです。ブートストラップされたクラスは異なるセキュリティドメイン間で共有されません。

ブートストラップローダのクラスを変更すると、他のサブアプリケーションが壊れる可能性があります。したがって、ブートストラップローダに追加するクラスが変更されることのないようにしてください。

プロキシサーバを介して RPC クラス (Blaze DS や LCDS など) を使用する場合、マルチバージョン化されたアプリケーションのブートストラップローダの RPC クラスを外部的化する必要があります。ブートストラップローダには RPC クラスのみが含まれるようにします。基本的に、サーバとデータをやり取りするために、カスタムバリューオブジェクトクラスをブートストラップローダに追加することはありません。マーシャリングを減らすためにこのようなクラスを追加する場合は Array や ArrayCollection など、Flex のフレームワークのクラスにリンクしない、単純なクラスにする必要があります。

DataService または RemoteObject 機能を使用する際は、通常、ブートストラップローダを使用してマルチバージョン化されたアプリケーションをロードします。ブートストラップローダには必要な RPC クラスおよび DataService 関連クラスが含まれる必要があります。また、サーバとのデータ送受信用のバリューオブジェクトクラスも含まれていなければなりません。カスタムバリューオブジェクトクラスをブートストラップローダに追加する場合は、Array や ArrayCollection など、Flex のフレームワークのクラスにリンクしない、単純なクラスにする必要があります。複雑なバリューオブジェクトクラスをブートストラップローダに追加することはできません。

次の例は、RPC クラスのブートストラップローダの一例です。このようなブートストラップローダは、プロキシサーバを介して HTTPService や WebService などのクラスをサブアプリケーションで使用する際に一般的に使用されています。

MainApp.swf がどのようにロードされているかに着目してください。ブートストラップローダは Loader クラスを使用し、LoaderContext を定義しません。その結果 Loader はデフォルトの動作を行います。デフォルトでは、サブアプリケーションが子アプリケーションドメインにロードされ、セキュリティドメインのデフォルトが使用されます。

各インポートにはクラスの import ステートメントが存在し、その直後にクラスの定義が続く点にも注意が必要です。これにより、クラス定義がブートストラップローダにリンクされます。ロードされたアプリケーションおよびそれに続いてロードされるすべてのアプリケーションに同じ定義を適用します。

```
package {

import flash.display.Loader;
import flash.display.Sprite;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;
import flash.net.URLRequest;
import flash.system.ApplicationDomain;

/**
 * Classes used by the networking protocols go here. These are the classes
 * whose definitions are added to the bootstrap loader and then shared
 * by the main application and all sub applications.
 */
import mx.messaging.config.ConfigMap; ConfigMap;
import mx.messaging.messages.AcknowledgeMessage; AcknowledgeMessage;
import mx.messaging.messages.AcknowledgeMessageExt; AcknowledgeMessageExt;
import mx.messaging.messages.AsyncMessage; AsyncMessage;
import mx.messaging.messages.AsyncMessageExt; AsyncMessageExt;
import mx.messaging.messages.CommandMessage; CommandMessage;
import mx.messaging.messages.CommandMessageExt; CommandMessageExt;
import mx.messaging.messages.ErrorMessage; ErrorMessage;
import mx.messaging.messages.HTTPRequestMessage; HTTPRequestMessage;
import mx.messaging.messages.MessagePerformanceInfo; MessagePerformanceInfo;
import mx.messaging.messages.RemotingMessage; RemotingMessage;
import mx.messaging.messages.SOAPMessage; SOAPMessage;
import mx.messaging.channels.HTTPChannel; HTTPChannel;
import mx.core.mx_internal;

[SWF(width="600", height="700")]
public class RPCBootstrapLoader extends Sprite {
    /**
     * The URL of the application SWF that this bootstrap loader loads.
     */
    private static const applicationURL:String = "MainApp.swf";

    /**
     * Constructor.
     */
    public function RPCBootstrapLoader() {
        super();
    }
}
```

```

    if (ApplicationDomain.currentDomain.hasDefinition("mx.core::UIComponent"))
        throw new Error("UIComponent should not be in the bootstrap loader.");

    stage.scaleMode = StageScaleMode.NO_SCALE;
    stage.align = StageAlign.TOP_LEFT;

    if (!stage)
        isStageRoot = false;

    root.loaderInfo.addEventListener(Event.INIT, initHandler);
}

/**
 * The Loader that loads the main application's SWF file.
 */
private var loader:Loader;

/**
 * Whether the bootstrap loader is at the stage root or not.
 * It is only the stage root if it was the root
 * of the first SWF file that was loaded by Flash Player.
 * Otherwise, it could be a top-level application but not stage root
 * if it was loaded by some other non-Flex shell or is sandboxed.
 */
private var isStageRoot:Boolean = true;

/**
 * Called when the bootstrap loader's SWF file has been loaded.
 * Starts loading the application SWF specified by the applicationURL property.
 */
private function initHandler(event:Event):void {
    loader = new Loader();
    addChild(loader);
    loader.load(new URLRequest(applicationURL));
    loader.addEventListener("mx.managers.SystemManager.isBootstrapRoot",
        bootstrapRootHandler);
    loader.addEventListener("mx.managers.SystemManager.isStageRoot",
        stageRootHandler);

    stage.addEventListener(Event.RESIZE, resizeHandler);
}

private function bootstrapRootHandler(event:Event):void {
    // Cancel event to indicate that the message was heard.
    event.preventDefault();
}

private function stageRootHandler(event:Event):void {
    // Cancel event to indicate that the message was heard.
    if (!isStageRoot)
        event.preventDefault();
}

private function resizeHandler(event:Event):void {
    loader.width = stage.width;
    loader.height = stage.height;
    Object(loader.content).setActualSize(stage.width, stage.height);
}
}
}

```

次の図は、mx.messaging* クラスを定義するブートストラップローダのクラス定義を、各ドメインがどこから取得するかを示しています。

