Sitecore CMS 6.6

Sitecore MVC デベロッパー リファレンス ガイド

開発者向け Sitecore MVCガイド

目次

第 1 章

イントロダクション

この章では、Sitecore MVC の機能について概説し、お使いの開発環境の準備と Sitecore MVC のインストール方法について説明します。

この章では、Sitecore MVC のインストールの必要条件とインストール方法について説明します。

この文書には次の章があります。

- **第 1 章 ― イントロダクション**
  この章では、Sitecore MVC のインストール方法について説明します。
- **第 2 章 ―Sitecore MVC パイプラインとリクエストの処理**
  この章では、Sitecore MVC の ASP.NET への統合方法および Sitecore パイプラインプロセッサについて説明します。また、パイプライン処理中に作成され、現在のリクエストに関する情報を取得するために使用できる、重要なクラスについても説明します。
- **第 3 章 ―ビューのレンダリング**
  この章では、MVC ビューのレンダリングについて説明します。
- **第 4 章 ―MVC ルーティング**
  この章では、MVC ルーティングについて説明し、`Global.asax` モジュールにルートを登録することによって URL をマップする方法を示します。
- **第 5 章 ―Sitecore MVC に関するその他の考慮事項**
  この章では、MVC をサポートするために Sitecore に追加された機能について補足説明します。

1.1 概要

Sitecore 6.6 の Sitecore MVC では、ASP.NET MVC 3 を Sitecore のファーストクラス レンダリング エンジンとして使用できます。旧バージョンでも ASP.NET MVC の一部の機能はサポートされていましたが、Sitecore 6.6 では、ASP.NET MVC 3 のより広範な機能を利用できるようになりました。

ASP.NET WebForms と MVC の両方がサポートされており、両者を混在させることもできますが、単一のリクエストは WebForms か MVC のどちらか一方でレンダリングする必要があります。

Sitecore MVC の第二の目的は、レンダリング プロセスを制御するための、強力で柔軟性がある、拡張可能な API を提供することです。

特に明記されていないかぎり、従来の Sitecore レンダリング エンジンが影響を受けることはありません。

Sitecore MVC は、Sitecore の標準インストールでは有効化されません。Sitecore 6.6 をインストールしたら、簡単な手順に従って Sitecore MVC を有効化することができます。

Sitecore MVC のインストールに関する詳細については、「Sitecore MVC をインストールするための必要条件」および「Sitecore MVC のインストール方法」を参照してください。

Sitecore での MVC の実装方法を理解するには、3 つの重要な側面があります。それについて、以下の各セクションで説明します。3 つの側面とは、Sitecore MVC を `HttpRequestBegin` パイプラインに組み込む方法、MVC 3 ビュー レンダリングを作成する方法、MVC ルーティングを使用して URL を管理する方法の理解の 3 点です。

1.1.1 ASP.NET MVC 3 とは何か

ASP.NET MVC 3 とは、モデル-ビュー-コントローラー パターンとして知られる長年の実績のあるソフトウェアアーキテクチャ デザイン パターンの実装の 1 つです。このデザインパターンにはさまざまな実装があり、長年に渡り、さまざまなプラットフォームで、ソフトウェア開発および Web 開発フレームワークに使用されてきました。

モデル-ビュー-コントローラー デザイン パターンでは、3 つの主要なコンポーネントが次のように構成されています。

- モデルとは、お使いのアプリケーションが使用するデータを表すクラスまたはクラスセットのことです。これらのクラスには、このデータを管理するためのアプリケーション ロジックも含まれます。
- ビューは、アプリケーションのユーザー インターフェースにデータを表示する方法を決定する各種プレゼンテーション コンポーネントから成ります。
- コントローラーとは、ユーザーからのリクエストを受信し、アプリケーションロジックを実行してモデルからデータを要求し、データを表示するために使用するビューを選択するクラスまたはクラス セットのことです。

ASP.NET MVC 3 に不慣れなデベロッパーは、マイクロソフトが公開している資料を利用して、この Web アプリケーション フレームワークの基本的な動作方法について学習することをお勧めします。このドキュメントでは、ASP.NET MVC 3 を Sitecore Web サイトに組み込むために必要となる情報のみを取り上げます。

1.1.2 Sitecore MVC をインストールするための必要条件

Microsoft MVC 3 は、Microsoft Windows XP、Windows Vista、および Windows 7 クライアント オペレーティング システムで動作します。Microsoft Visual Studio 2010 または Visual Web Developer 2010 Express も必要です。

Microsoft MVC 3 には .NET Framework 4.0 が必須であり、Sitecore システムは .NET Framework 4.0 アプリケーション プールで実行しなければなりません。

http://www.microsoft.com/web/gallery/install.aspx?appid=MVC3 から Microsoft MVC 3 Web プラットフォーム インストーラーをダウンロードおよびインストールしてください。

これにより、Visual Studio の MVC 3 プロジェクト テンプレートがインストールされます。

1.1.3 Sitecore MVC のインストール方法

Sitecore MVC は、Sitecore 6.6 以降で動作します。

Sitecore MVC をインストールするには、次の手順を実行します。

- Sitecore 6.6 をインストールします。
- Microsoft Windows IIS Manager ツールを使用して、新しい Sitecore Web サイト用のアプリケーション プールを .NET Framework 4.0 に設定します。
- 新しい Sitecore インスタンスといっしょにインストールされたデフォルトの `web.config` ファイルを確認します。

  このファイルは、`<installation folder>/Website` フォルダーに保存されています。の設定値をメモします。この値は、インストール環境によって異なります。例:

```
<sitecore database="SqlServer">
    <sc.variable name="dataFolder" value="C:\Inetpub\wwwroot\June2012\Data\"/>
    <sc.variable name="mediaFolder" value="/upload"/>
    <sc.variable name="tempFolder" value="/temp"/>
```

- `web.config.MVC` ファイルの `dataFolder` の設定を変更して、デフォルトの `web.config` ファイルの設定に一致させます。このファイルは Sitecore 6.6 といっしょにインストールされており、`<installation folder>/Website` フォルダーに保存されています。
- デフォルトの `web.config` ファイルの名前を変更してバックアップを用意し、`web.config` MVC ファイルの名前を `web.config` に変更します。
- `\App_config\Include\Sitecore.MVC.config.disabled` ファイルの名前を に変更します。その他の機能を有効にするには、すべての MVC 設定ファイルについて同じ操作 (ファイル名の変更) を繰り返します。
- すべてのファイルを `<installation folder>/Website/bin_Net4` フォルダーから `<installation folder>/Website/bin` フォルダーにコピーします。

第 2 章

Sitecore MVC パイプラインとリクエストの処理

この章では、Sitecore MVC を ASP.NET に統合する方法、および Sitecore パイプライン プロセッサについて説明します。また、パイプライン処理中に作成され、現在実行中のリクエストに関する情報を取得するために使用できる、重要なクラスについても説明します。

- MVC と HttpBeginRequest パイプライン
- PageDefinition クラス
- Sitecore MVC コンテキスト オブジェクト
- 既存のパイプラインの変更
- MVC 固有のパイプライン
- Sitecore MVC リクエストの処理

2.1 MVC と HttpBeginRequest パイプライン

Sitecore MVC は、`HttpBeginRequest` パイプラインにフックして、MVC ルートが一致するか、レイアウト ファイルが MVC ビューであるとき、`MvcSettings. ViewExtensions` プロパティによって指定されているとおりに、MVC レンダリング エンジンをリダイレクトします。このロジックは、パイプライン プロセッサ `TransferRoutedRequest` (ルートの一致) と `TransferMvcLayout` (MVC ビュー) によって実行されます。

MVC ルーティングの詳細については、「MVC ルーティング」のセクションを参照してください。

MVC 処理中、Sitecore MVC は、Sitecore における高レベルのレンダリング情報を表すメモリ内モデルを構築します。このモデルは `PageDefinition` と呼ばれます。`PageDefinition` は、現在のリクエストに対するすべての応答方法を表しています。これは、特定のアイテムに対する標準の Sitecore プレゼンテーション レイアウト定義に、異なるデバイス用の複数のレンダリング コレクションを指定できるのとよく似ています。

`PageDefinition` は、デバイスに結合されるだけではありません。その他のプロパティによって、呼び出し元に返す必要がある応答が左右されることもあります。また、応答は HTML ではないことさえあります。JSON や XML も応答として指定できます。

応答を生成するために必要な情報をカプセル化するクラスは、`PageContext` と呼ばれます。実際の応答を生成するには、MVC ビュー (`IView`) を使用します。このビューは、`PageContext` のプロパティで、`PageView` と呼ばれます。

受信リクエストは、Sitecore MVC によって次のように処理されます。

- `PageContext` を作成します。
- リクエストされたアイテムを確認します。
- 呼び出す関連コントローラーを確認します。
- `PageDefinition` を作成し、`PageContext` に割り当てます。
- 応答 (通常は、リクエストされたアイテムに割り当てられるレイアウト) を生成するために使用するルート レンダリングを選択します。
- ルート レンダリングを `IView` でラップし、`PageContext.PageView` に格納します。
- `PageView` を ASP.NET MVC ランタイムに渡して、クライアントに対する応答をレンダリングします。

上記の手順はすべてパイプライン プロセッサによって処理されるため、このアーキテクチャはプラグ可能かつ拡張可能です。

ルート レンダリングは通常、出力生成処理の一部として、`Html.Sitecore().PlaceHolder` 拡張メソッドを呼び出します。このメソッドは、指定されたプレースホルダー名に対応する `PageDefinition` に定義されているレンダリングを実行します。

Html.Sitecore() オブジェクトは、レンダリング補助用メソッドを備えています。具体的には、標準の Sitecore RenderField パイプラインを呼び出す Field メソッドや、MVC ビュー、XSLT レンダリング、その他のプレゼンテーション コンポーネントを直接レンダリングする各種メソッドなどです。

2.1.1 PageDefinition クラス

PageDefinition は、すべてのデバイス、レイアウト、およびリクエストされたアイテムのレンダリングに関する高レベルの情報を保持しています。PageDefinition は、MVC リクエストで最初に実行されるアクションの 1 つとして作成されます。PageDefinition は、BuildPageDefinition パイプラインによって作成されます。

PageDefinition は Rendering オブジェクトのリストから成ります。これらのオブジェクトの 1 つが、実際の応答 (HTML、JSON、または XML) を生成するときのルート レンダリングとして使用されます。レンダリングの選択は GetPageRendering パイプライン内で行われ、選択されたレンダリングの出力は RenderRendering パイプライン内で生成されます。

大半のレンダリングには、Rendering オブジェクトの Renderer プロパティが使用されます。Renderer は、特定の種類の Rendering の実行方法を知っているオブジェクトです。

Rendering はレンダリング対象 (MVC ビュー、XSLT、静的テキスト、その他のプレゼンテーション情報) の定義、Renderer は、その定義をクライアントから要求された形式 (HTML、JSON、XML など) に変換する方法を知っているオブジェクト、と考えることができます。

RenderRendering パイプラインは、キャッシング、コンテキストなどを処理しますが、出力の実際のレンダリングでは、レンダリングの Renderer プロパティを使用するだけです。レンダラーには ViewRenderer, XsltRenderer, ItemRenderer などがあります。すべてのレンダラーは、Renderer 抽象基底クラスから導出されます。

Renderer クラスには単一のメソッド Render() が定義されており、これをカスタムのレンダラーとして実装する必要があります。

2.1.2 Sitecore MVC コンテキスト オブジェクト

Sitecore MVC リクエストの処理中には、複数のコンテキスト オブジェクトが作成されます。これらのコンテキスト オブジェクトは、現在実行中のリクエストの状態に関する情報にアクセスするときに使用します。

すべてのコンテキスト クラスには、現在のインスタンスを取得するための次の静的プロパティが定義されています。

- Current - 現在のコンテキストの取得を試みます。呼び出し元のコードが要求されたタイプのコンテキスト内に存在しない場合は例外を発生させます。
- CurrentOrNull - 現在のコンテキストの取得を試みます。呼び出し元のコードが要求されたタイプのコンテキスト内に存在しない場合は、null を返します。

`RequestContext` クラスには、これらのプロパティは定義されていません。その代わり、拡張メソッド `Current()` と `CurrentOrNull()` を使用できます。

RequestContext

`RequestContext` は、現在のリクエストに関する情報 (URL、各フォーム値、ユーザー、ルート) を保持している MVC クラスです。アクセスするには、`PageContext.RequestContext` プロパティを使用します。

PageContext

`PageContext` オブジェクトは、作成中のページに関する情報を保持します。具体的には、リクエストされたアイテム、現在のデバイス、`PageDefinition`、応答のレンダリングに使用するビュー (`PageView`) などです。

PlaceHolderContext

`PlaceHolderContext` は、現在のプレースホルダー情報を管理します。このオブジェクトは、ネストされたプレースホルダーをサポートしています。

RenderingContext

`RenderingContext` は、現在のレンダリングと関連データ ソース/アイテムを管理します。

2.2 既存のパイプラインの変更

2.2.1 パイプラインの初期化

Sitecore MVC は、`initialize` パイプラインに次の 3 つのプロセッサをインストールします。

1. `InitializeGlobalFilters` - このプロセッサは MVC グローバル フィルターにフック (`ActionExecuting`, `ActionExecuted`, `ResultExecuting`, `ResultExecuted` および `Exception`) を設定し、これらのイベントをフックして Sitecore パイプラインを使用できるようにします。

   詳細については、「MVC 固有のパイプライン」のセクションを参照してください。

2. `InitializeControllerFactory` - このプロセッサは、カスタムのコントローラー ファクトリ (`IControllerFactory`) を MVC ランタイムにインストールして、MVC リクエストの処理とコントローラーの作成を容易に制御できるようにします。

3. `InitializeRoutes` - デフォルトの Sitecore ルート ハンドラー (フォール スルー: `{*pathInfo}`) を設定して、既存のルートを Sitecore 固有のルート キー () で修飾します。

   詳細については、「MVC ルート」のセクションを参照してください。

2.2.2 HttpBeginRequest パイプライン

Sitecore MVC は、`HttpRequest` パイプラインに次のプロセッサをインストールします。

1. `TransferRoutedRequest` - このプロセッサは、`LayoutResolver` プロセッサの前に配置されます。リクエストが MVC ルート表に登録されているルートと一致すると、パイプラインが異常終了し、MVC リクエスト ハンドラーが処理を引き継ぎます。カスタムのルートだけが処理対象として考慮されます。Sitecore フォールスルー ルート (`{*pathInfo}`) はこのプロセッサによって無視されます。そうしないと、すべてのリクエストが一致し、MVC によって処理されることになります。

2. `TransferMvcLayout` - このプロセッサは、`LayoutResolver` プロセッサの後に配置されます。解決されたレイアウト ファイル名の拡張子が、`MvcSettings.ViewExtensions` 設定に指定されたいずれかの拡張子に一致すると、パイプラインが異常終了し、MVC リクエスト ハンドラーが処理を引き継ぎます。

3. `TransferControllerRequest` - このプロセッサは、`TransferMvcLayout` プロセッサの後に配置されます。レイアウトが解決しない場合、現在のアイテムの `Controller` フィールドが調べられます。このフィールドに値が格納されていると、パイプラインが異常終了し、MVC リクエスト ハンドラーが処理を引き継ぎます。その後、指定されたコントローラーがアイテムのレンダリングを行います。

2.3 MVC 固有のパイプライン

設定ファイルでは、Sitecore 標準のパイプラインとの名前の衝突を避けるため、MVC 固有のパイプラインの名前はすべて "mvc." で始まる点に注意してください。

2.3.1 RequestBegin

このパイプラインは、メインの HttpRequest パイプラインから渡されてきた MVC リクエストの最初のアクションとして実行されます。

`PageContext` が作成され、現在のスレッドに割り当てられます。

リクエストがフォームのポストである場合は、関連付けられたフォーム ハンドラー (存在する場合) が実行されます。

詳細については、フォームのセクションを参照してください。

2.3.2 RequestEnd

このパイプラインは、MVC リクエストの最後のアクションとして実行されます。

現時点では、このパイプラインでは何も実行されません。

2.3.3 CreateController

このプロセッサは `SitecoreControllerFactory` によって呼び出され、MVC リクエスト ハンドラーによって選択された現在のルートのコントローラーを作成します。

リクエストされたアイテム (`PageContext.Item`) の `__Controller Name` フィールドにコントローラーが指定されている場合は、そのコントローラーがインスタンス化され返されます。同フィールドにコントローラーが指定されていない場合は、デフォルトのコントローラー `SitecoreController` のインスタンスが返されます。

2.3.4 ActionExecuting

このプロセッサは、MVC コントローラー アクションが実行される前に実行されます。

現時点では、このパイプラインでは何も実行されません。

2.3.5 ActionExecuted

このプロセッサは、MVC コントローラー アクションが実行された後に実行されます。

現時点では、このパイプラインでは何も実行されません。

2.3.6 ResultExecuting

このプロセッサは、MVC 結果が実行される前に実行されます。

現時点では、このパイプラインでは何も実行されません。

2.3.7 ResultExecuted

このプロセッサは、MVC 結果が実行された後に実行されます。

現時点では、このパイプラインでは何も実行されません。

2.3.8 Exception

このプロセッサは、MVC ランタイムが未処理の例外を捕捉した後に実行されます。

現時点では、このパイプラインでは何も実行されません。

このパイプラインにカスタムのプロセッサを追加する場合は、プロセッサに渡される `ExceptionArgs` オブジェクトに `ExceptionContext.ExceptionHandled` プロパティを設定する必要があります。そうしないと、Sitecore MVC によって、標準のエラー処理が実行されます。

2.3.9 GetPageItem

このパイプラインは、ルート情報を使用して、リクエストされたアイテムを解決します。ルートからアイテムを解決できない場合は、`Context.Item` を使用します。

詳細については、「MVC ルート」のセクションを参照してください。

2.3.10 BuildPageDefinition

このパイプラインは、通常、ページ アイテムの `Renderings` フィールドに指定されている XML ベースのレイアウト定義を使用して、最初の `PageDefinition` を作成します。

2.3.11 GetPageRendering

このパイプラインは、`Rendering` を選択し、それをページのルート レンダリングとして使用して現在のリクエストの出力を作成します。

現時点では、ルート レンダリングは、現在のデバイス (`PageContext.Device`) のみに基づいて選択されます。

2.3.12 GetRenderer

このパイプラインは、抽象レンダリング定義 (`Rendering`) を、出力をレンダリングしてクライアントに返すオブジェクト (`Renderer`) に変換します。

2.3.13 GetModel

このパイプラインは、MVC ビューをレンダリングするときに使用するモデル オブジェクトを作成します。通常、APS.NET MVC 3 Razor が作成されます。このパイプラインによってモデルが返されない場合、MVC ビューは `EmptyModel` のインスタンスを受け取ります。これにより、`.cshtml` ファイルに `@model` ディレクティブが指定されており、モデルを要求しているときは、意味のあるエラー メッセージが生成されます。

2.3.14 RenderPlaceHolder

このパイプラインは、`Html.Sitecore().PlaceHolder` 拡張メソッドの一部として呼び出されます。必要に応じて、ネストされたプレースホルダーを処理します。

デフォルトでは、このパイプラインは、現在の `PageDefinition` に指定されているプレースホルダー名に一致するすべてのレンダリングを検索し、それらのレンダリングを実行します。子レンダリングが実行されるのは、レンダリングが明示的に行われる場合だけです。明示的なレンダリングの実行には通常、`Html.Sitecore().Inner()` が呼び出されます。

2.3.15 RenderRendering

このパイプラインは、指定されたレンダリングを実行します。このパイプラインはキャッシングを処理します。また、レンダリング ソース アイテムにアクセスし、ネストされたレンダリングをサポートするために、`IRenderingContext` を設定します。

2.4 Sitecore MVC リクエストの処理

2.4.1 MVC Request の転送

Sitecore MVC は、次の 2 つのケースに該当する場合に、リクエスト処理を ASP.NET MVC レンダリング エンジンに転送します。どちらも、カスタムのプロセッサによって、`httpRequestBegin` パイプラインで検出および処理されます。

1. リクエストが MVC ルートに一致すると、Sitecore コンテキストが設定された後、Sitecore レイアウトが解決される前に、そのリクエストは MVC に転送されます。
2. Sitecore レイアウトが解決されると、レイアウト ファイルの拡張子が調べられ、それが `MvcSettings.ViewExtensions` 設定に指定されたいずれかの拡張子に一致すると、リクエストは MVC に転送されます。
3. 現在のアイテムにレイアウトは関連付けられていないが、アイテムのコントローラー フィールドにコントローラーが指定されている場合、リクエストは MVC に転送されます。

上記、いずれの条件も満たされない場合、リクエストは WebForms リクエストとして処理されます。

2.4.2 プレースホルダーの処理

`Html.Sitecore().PlaceHolder` 拡張メソッドは、指定されたプレースホルダーのコンテンツをレンダリングします。この処理は、`RenderPlaceHolder` パイプラインで実行されるので、出力をプレースホルダー レベルで変更することができます。

2.4.3 レンダリング処理

レンダリングの実行中は、`RenderRendering` パイプラインが実行されます。これにより、デベロッパーは、レンダリングの出力を変更できます。たとえば、出力が HTML 標準に準拠しているかどうかを検証できます。

ページ エディターでは、このパイプラインを使用して、出力にページ エディター固有のタグを追加できます。

2.4.4 アイテムのレンダリング

Sitecore MVC では、アイテム レンダリングの概念を導入しています。アイテム レンダリングは、レンダリングのデータ ソース プロパティに基づいてアイテムをレンダリングする新しいタイプのレンダリングです。このレンダリングでは、アイテムのレンダリングに、`Renderings` フィールドではなく、`Renderer` フィールドを使用します。

アイテム レンダリングの例として、Home ページの頻繁に変更される情報があります。ユーザーは、`Home` アイテムのレイアウト定義にアイテム レンダリングを追加します。アイテム レンダリングのデータ ソースは、固有の `Spot` アイテムに設定されます。`Spot` アイテムの `Renderer` フィールドには、`Spot1` レンダリングへの参照が格納されます。`Home` アイテムがレンダリングされるとき、`Spot` アイテムの `Renderer` フィールドが解析され、そこに指定されているレンダリング (この場合は、`Spot1` レンダリング) が実行されます。

アイテム レンダリングの各要素は Sitecore レンダリングを指している必要はありません。デフォルトでは、Sitecore レンダリングを指していない要素は、MVC ビューによってレンダリングする必要があるものと見なされます。この MVC ビューは、要素名と `path` および `folder` プロパティの組み合わせを使用して解決されます。

以下に、アイテム レンダリングの例を示します。

```
<modes>
  <mode name="mobile" screen-size="small">
    <view path="spots/spot" color="red">
      <text>
        <header>Header: {{:title}}</header>
        <text>
          <![CDATA[<div>This is embedded HTML. Value of 'text' field is:
              {{:text}}</div>]]>
        </text>
        <field name="title" cachekey="{{:__style}}" cache_timeout="00:03:02"
            cac="1"/>
        <field datasource="/sitecore/content/home" vbd="1">{{:title}}</field>
      </text>
    </view>
    <copyright id="{493B3A83-0FA7-4484-8FC9-4680991CF743}"/>
    <spot folder="spots" color="{{:__style}}"/>
  </mode>
</modes>
```

2.4.5 内部レンダリング

`Rendering` は、レンダリングの再帰データ構造体で構成されています。各レンダリングには、任意の数のネストされたレンダリング (`ChildRenderings` で指定) を含めることができます。

あるレンダリングの子レンダリングは内部レンダリングとも呼ばれ、ASP.NET MVC 3 Razor ビューの `Html.Sitecore().Inner` メソッドを使用してレンダリングできます。今後のバージョンで、内部レンダリングを実行するための XSLT 関数も用意される予定です。

内部レンダリングは、レンダリング要素の再利用に有用です。外部レンダリングは内部レンダリングをカプセル化するため、内部レンダリングを修飾することができます。この考え方は、Xml コントロールのプレースホルダーに似ています。

内部レンダリングも単なるレンダリングなので、今後、XSLT レンダリング内で Razor ビューをレンダリングすることも可能になります。

内部レンダリングの UI はサポートされていません。また、WebForms では、内部レンダリングはサポートされていません。

第 3 章

ビューのレンダリング

この章では、MVC ビューのレンダリングについて説明します。

この章には次のセクションがあります。

- レンダリング

3.1 レンダリング

MVC では、ASP.NET WebForms とは異なるコントロールのセットを使用してレンダリングを行います。そのため、一部の Sitecore レンダリングはサポートされていません。

サポートされているレンダリングの種類は次のとおりです:

- コンテンツ レンダリング - 静的テキスト
- コントローラー レンダリング - MVC コントローラーを実行し、結果を出力
- アイテム レンダリング - アイテムによって指定されたレンダリングを実行
- メソッド レンダリング
- URL レンダリング
- ビュー レンダリング - MVC ビュー
- XSLT レンダリング

次のレンダリングはサポートされていません。

- Web コントロール
- サブレイアウト

通常、`Sublayouts` は、簡単にビュー レンダリングに変換できます。

MVC では、Sitecore データ テンプレートを再利用して、XSLT レンダリング、URL レンダリング、およびメソッド レンダリングを実現していますが、デベロッパーは、MVC のレンダラー クラスが、`WebControl` からではなく、`Renderer` から継承していることを認識する必要があります。

第 4 章

MVC ルーティング

この章では、MVC ルーティングについて説明し、`Global.asax` モジュールにルートを登録することによって URL をマップする方法を示します。

この章には次のセクションがあります。

- MVC ルート

4.1 MVC ルート

Sitecore では、MVC ルート表を使用して、MVC ランタイムに送信して処理するリクエストを決定します。

ルート表は通常、`global.asax` 内に設定します。Sitecore では、ルートに Sitecore 固有の情報を追加するときに使用するカスタムのルート値を用意しています。

ASP.NET MVC 3 プロジェクトのデフォルトの global.asax には、次のコードが含まれています。

```
public class MvcApplication : System.Web.HttpApplication
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new HandleErrorAttribute());
        }

        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Home", action = "Index", id =
               UrlParameter.Optional } // Parameter defaults
                );
        }

        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();

            RegisterGlobalFilters(GlobalFilters.Filters);
            RegisterRoutes(RouteTable.Routes);
        }
    }
```

ルートを追加することもできます。

Sitecore のルート拡張の例を以下に示します。

```
// bypasses Sitecore and is handled by the 'MyController' custom controller
routes.MapRoute("example1", "special/{id}", new { controller = " MyController", });

// matches the item '/home/blogPosts/{blogNo}'
routes.MapRoute("example2", "blog/{blogNo}", new { scItemPath =
              "/home/blogPosts/{blogNo}", });

// sets language to {scLanguage} and matches the item '/home/{itemName}'
routes.MapRoute("example3", "home/{scLanguage}/{itemName}");

// matches the item '/home/{itemName}'
routes.MapRoute("example4", "home/{something}/{itemName}",
               new { scKeysToIgnore = "something", });

// matches the item '/home/{itemName}'
routes.MapRoute("example5", "home/{something}/{orOther}/{itemName}",
               new { scKeysToIgnore = new[] { "something", "orOther" }, });
```

カスタムのルート キーについて簡単に説明します。

- `scLanguage` - URL のこのキーに一致する部分は、リクエストのコンテキスト言語として使用されます。
- `scItemPath` - アイテムは、ルート URL に関係なく、この要素の値を使用して解決されます。上の例のようにハード コーディングすることもできますし、`routes.MapRoute("myRoute", "/special/{*scItemPath}")` のように、ルート URL の一部として指定することもできます。
- `scKeysToIgnore` - Sitecore が無視しなければならない 1 つ以上のルート URL キー。これらのキーがルート内に現れる場合は、URL からアイテムを解決する前に削除されます。

カスタム ルートにコントローラーが指定されていない場合は、デフォルトの Sitecore コントローラーがリクエストを処理します。同様に、アクションが指定されていない場合は、デフォルトのアクション `'Index'` が使用されます。

デフォルト ルート (`{*pathInfo}`) を `global.asax` 内に設定してはなりません。Sitecore では、`InitializeRoutes` 内で、`initialize` パイプラインの一部として、デフォルト ルートをインストールします。これにより、`global.asax` に設定されているどのカスタム ルートにも一致しないすべてのルートが、Sitecore によって処理されます。

ルート解析が行われる `GetPageItem` パイプラインにプロセッサを追加することによって、ルート処理にフックすることも可能です。

第 5 章

Sitecore MVC に関するその他の考慮事項

この章では、MVC をサポートするために Sitecore に追加された機能について補足説明し、Sitecore で ASP.NET MVC 3 を実装するときの考慮事項を示します。

この章には次のセクションがあります:

- Sitecore データ テンプレート
- デバッグ機能
- 条件付きレンダリング

5.1 Sitecore MVC に関するその他の考慮事項

以下の各セクションでは、Sitecore MVC 機能をサポートするために Sitecore 6.6 で実装されたいくつかの変更点について説明します。

5.1.1 Sitecore データ テンプレート

以下の内容が、標準のテンプレートに追加されました。

- `__Controller Name`: アイテムがリクエストされたときに実行する MVC コントローラーの名前。このフィールドに GUID を指定することもできます。GUID は、`Controller` テンプレートに基づくアイテムを指していなければなりません。
- `__Controller Action` - 実行するコントローラー アクション。
- `__Renderers` フィールドのフィールド型が、"text" から "memo" に変更されました。

次のテンプレートがシステムに追加されました。

- `Controller` - コントローラーの名前とアクションを指定します。
- `Model` - MVC ビューのモデルとして使用するタイプを指定します。
- `Controller rendering` - 実行するコントローラーの名前とアクションを指定します。そのコントローラーが出力を返す場合は、それが出力ストリームに与えられます。
- `Item rendering` - レンダリングのデータ ソースによって指定されたアイテムを、`__Renderers` フィールドを使用したアイテムに格納されたレンダリング定義を使用してレンダリングする必要があることを示します。
- `View rendering` - レンダリングされる MVC ビューを指定します。

5.1.2 デバッグ機能

Sitecore MVC は、標準の Sitecore トレースおよびプロファイル機能を使用しています。

また、Sitecore MVC は、`Sitecore.Glimpse.dll` アセンブリを介して Glimpse をサポートしています。これは、MVC とは別に出荷されます。このプラグインでは、Glimpse ビューに 2 つのタブ、[Sitecore トレース] と [Sitecore プロファイル] が追加されています。ここに表示される情報は、標準の Sitecore トレースおよびプロファイルと同じです。

5.1.3 条件付きレンダリング

条件付きレンダリングは、MVC ではサポートされていません。