Sitecore CMS 6.0 以降

インクルード ファイルのパッチ機能

Sitecore でのインクルードファイルを使用した構成ファイルの拡張とパッチ

イントロダクション

インクルード ファイル メカニズムの主な目的は、Sitecore 構成ファイルの拡張とパッチを行うことです。インクルード ファイルを使用することで、既存の設定やプロパティ、構成ノードの値をオーバーライドすることが可能になることはもちろん、新しい設定やプロパティ、構成ノードが使用できるようになります。

特別な XML シンタックスを使用することによって、次のような共通操作を行うことができます:

- 指定場所にノードを挿入する
- 属性値を変更する
- ノードを削除する

このドキュメントでは、このプロセスについて詳しく説明します。

一般概念

XML 名前空間とノード名

パッチに関連付けられたすべての属性と要素は、次の XML 名前空間内にあります。

- `patch` – http://www.sitecore.net/xmlconfig/
- `set` – http://www.sitecore.net/xmlconfig/set/

インクルードファイルで名前空間を使用するには、その名前空間を宣言する必要があります。通常はファイルの最初で宣言します。

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
. . . . .
  . . . . .
</configuration>
```

このシナリオでは、`patch:` と `set:` プレフィックスを使って、パッチをコントロールするノードにアクセスします。

用語説明:

用語	説明
ノード	XML ドキュメントはノード ツリーです。ノードには、要素ノード、属性ノード、テキスト ノードなどがあります。
要素	開始タグで始まり対応する終了タグで終わるロジカル ドキュメント コンポーネントです。または`<element />` などの空要素のみを含むロジカル ドキュメント コンポーネントです。
属性	開始タグまたは空要素のタグ内にある名前/値の組み合せによって構成されるマークアップ構築です。たとえば、`<element attribute="value" />` など。

XML ネームスペースについての追加情報は、ウィキペディア— http://en.wikipedia.org/wiki/XML_namespace を参照してください。

インクルード ディレクトリと *.config ファイル

Sitecore は次の順番で構成ファイルを読み込み、実際に使用する構成を構築します。

- `web.config` ファイル。
- `/App_config/Include` フォルダーからのすべての `*.config` ファイル。これらのファイルは、`web.config` ファイルにある `<sitecore />` ノードのコンテンツを拡張し、修正します。

/App_Config/Include フォルダーのファイルはファイル名順に適用されます。したがって、Sitecore では AName.config ファイルは ZName.config ファイルの前に適用されます。インクルード ファイルが適用される順番は特に重要ではありません。ただし、複数のインクルード ファイルで同じ設定 (またはノード) が変更される場合は、最後の変更のみが適用されます。さらに、別のインクルード ファイルから追加されるノードをパッチする場合は、パッチを持つインクルードファイルが、ノードを追加するインクルードファイルの後に適用されることを確認する必要があります。

Sitecore では*.config ファイルのみが考慮されます。特定の *.config ファイルを無効にする必要がある場合は、その拡張子を変更します。その際はファイル名を *.config.disabled に変更することを推奨します。

Sitecore はインクルードファイルを読み込む際、再帰的にファイル システムを横断します。まず Sitecore は /App_Config/Include フォルダーにあるすべての *.config ファイルを列挙し、次に再帰的にすべてのサブフォルダーを列挙します。サブ フォルダーを /App_Config/Include フォルダーに作成する場合は、この順序を考慮する必要があります。

インクルード ファイルの処理

インクルードファイルでの変更を適用する場合、Sitecore はインクルードファイルのすべての要素を再帰的に横断し、各要素を既存の構成の要素にマッチングさせます。

Sitecore は要素名とその属性をすべて使用してノードをマッチングします。インクルードファイルの要素を相対する web.config ファイルの要素とマッチングさせるには、特有の方法を使って修正が必要なノードや属性を指定する必要があります。

Sitecore では特定の要素を処理するオプションが 2 つあります。

- 既に要素が存在する場合、Sitecore はその要素を書き換えます。
- 要素が存在しない場合、Sitecore は新しく要素を挿入します。

Sitecore では次の比較のアルゴリズムを使用します。

1. Sitecore はインクルードファイルから要素を取得し、その属性とそれらの値をすべて抽出します。要素をマッチングする場合は、set と patch 名前空間からのすべてのノードと、<element xmlns:customNs="." /> などの名前空間での宣言はすべて無視されます。
2. Sitecore は、オリジナルのドキュメントに同じレベルで同じ名前と属性を持つ要素が存在するかどうかを確認します。
3. これらの制約を持つノードがひとつ以上存在する場合、Sitecore はこれらの中から最初のノードを選択します。たとえば、インクルードファイルで<processor /> 要素を指定する場合、web.config ファイルにある最初のプロセッサーのみが選択されます。
4. 制約にマッチングするノードがない場合、Sitecore は既存のノードをパッチするのではなく、新しい要素を挿入します。

Sitecore では、属性とそれらの値をマッチ条件として使用し、どの特定のノードに変更を行うかを決定します。十分な属性を指定しない場合は、誤った要素がマッチングされる場合があります。たとえば、いくつかのイベント ハンドラではタ

イプの属性値までは同じで、メソッド属性値のみ別の値を持つことがあります。タイプ属性のみを指定する場合、Sitecore は最初のイベント ハンドラを指定します。一方で、重複した属性値を指定する場合、その属性が `web.config` ファイルまたは前述のインクルードファイルなどのオリジナル ドキュメントで変更された場合はノードのマッチングが行われない場合があります。インクルード ファイルを作成するたびに、どの属性をインクルードし、どの属性を除外するのかを分析する必要があります。

サンプル

次のサンプルを使ってルールの理解を深めます。

次の構成は `web.config` ファイルで定義されます。

```
<configuration>
  <sitecore>
    <events timingLevel="custom">
      <event name="item:added" />
      <event name="item:removed">
        <handler type="ItemEventHandler" method="OnItemRemoved" />
        <handler type="ItemEventHandler" method="OnItemChanged" />
      </event>
    </events>
    <settings>
      <setting name="AliasesActive" value="true" />
      <setting name="AllowLogoutOfAllUsers" value="false" />
    </settings>
  </sitecore>
</configuration>
```

`OnItemChanged` イベント ハンドラを削除して `AllowLogoutOfAllUsers` 設定の値を `true` に変更します。

インクルード ファイルのコンテンツは次のようになります:

```
<configuration xmlns:patch=http://www.sitecore.net/xmlconfig/
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <events>
      <event name="item:removed">
        <handler type="ItemEventHandler" method="OnItemChanged" >
          <patch:delete />
        </handler>
      </event>
    </events>
    <settings>
      <setting name="AllowLogoutOfAllUsers" >
        <patch:attribute name="value" value="true" />
      </setting>
    </settings>
  </sitecore>
</configuration>
```

このときパッチ特有のノードは無視します。これについてはドキュメントの後半で説明します。

イベント ハンドラと設定がどのように指定されているのかを確認します。

XML ドキュメントを解析します。

- 有効な `sitecore` 要素はひとつしかないため、`sitecore` 要素はマッチ規則を要求しません。
- オリジナルの `events` 要素は `timingLevel` 属性を含みます。

events ノードはひとつしかなく、曖昧性はないのでこの属性は無視されます。さらに、web.config ファイルで timingLevel 値を変更する場合、Sitecore は後々そのマッチングを行うことができないので、timingLevel 属性は指定すべきではありません。

- 複数の event 要素があるので、name 属性を使って正確な要素を特定します。
- この例では handler 要素に、type と method 属性の両方を使用します。

  この場合 type 属性のみを使用するだけでは OnItemRemoved ハンドラがマッチングされてしまうため、不十分です。今回は、OnItemChanged メソッドを含むハンドラが他にないため、method 属性でこのメソッドを指定するだけで十分です。

  ただし、これはロジックに反します。通常、handler は type と method の組み合せで定義されます。また同じメソッド属性の値を持つ別の handler を使用する場合に問題を引き起こします。
- この例では settings 要素にいかなる属性も使用しません。このノードはひとつしか存在しないためです。
- この例では、setting 要素に name 属性値のみを指定します。

  value 属性も追加した場合、web.config ファイルまたは前述のインクルードファイル内で変更を行うまでは正確に機能しますが、変更後は機能しなくなります。

以下はパッチが適用された結果の構成です:

```
<configuration>
  <sitecore>
    <events timingLevel="custom">
      <event name="item:added" />
      <event name="item:removed">
        <handler type="ItemEventHandler" method="OnItemRemoved" />
      </event>
    </events>
    <settings>
      <setting name="AliasesActive" value="true" />
      <setting name="AllowLogoutOfAllUsers" value="true" />
    </settings>
  </sitecore>
</configuration>
```

結果の構成 (showconfig.aspx)

Sitecore を開始すると、Sitecore は構成ファイルを読み込んで結合し、結果の XML ドキュメントを作成します。結果の構成ドキュメントはメモリに保存されて構成値が必要なときに利用されます。結果の XML ファイルを確認するには、showconfig.aspx ページを使用します。URL は http://hostName/sitecore/admin/showconfig.aspx です。

最終的な構成を確認するには、web.config ファイルやスタンド アローンのインクルードファイルではなく、showconfig.aspx ページを確認することを推奨します。

トラブルシューティング

パッチング機能のトラブルシューティングに使用可能な主なツールは、showconfig.aspx ページです。このツールを使用すると、各インクルードファイルが構成全体にどのように作用するのかを確認することができます。

より効率的にインクルード機能のトラブルシューティングを行うために、Sitecore 7.0 において `patch:source` 属性が追加されました。この属性は、現在のノードに変更を行った最新のインクルード ファイルの名前を示します。

以下は `patch:source` 属性の主なプロパティです。

- インクルードファイルによってひとつ以上の属性が追加または修正されたノードに追加されます。
- ひとつのノード上で複数のインクルードファイルが属性の修正を行った場合、`patch:source` 属性はノードに変更を加えた最新のインクルードファイル名のみを示します。
- インクルードファイルによるノードの変更が行われなかった場合、その属性は除外されます。

属性は`/sitecore/admin/showconfig.aspx` ページに表示されます。実際の構成には作用しません。

パッチの名前空間

(http://www.sitecore.net/xmlconfig/)

概要

パッチの名前空間には 2 種類のノードが含まれます。

- 属性 — `<elem patch:attribute="" … />`
- 要素 — `<patch:element …. />`

これらのノードにアクセスするには、`http://www.sitecore.net/xmlconfig/`と `http://www.sitecore.net/xmlconfig/set/` 名前空間を定義する必要があります。名前空間ではどのようなエイリアスでも指定できますが、通常 `patch` と `set` が使用されます。

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
set=”http://www.sitecore.net/xmlconfig/set/”>
```

属性ノード

通常、新しい要素が挿入されると、それは親要素の最後に追加されます。パッチの名前空間の属性ノードは、挿入される要素の相対位置を定義するために使用されます。通常これらの属性は新しい要素を追加する場合にのみ使用されます。これらの属性は既存の要素を移動させるためのものではありません。

名前空間には次の 3 つの属性があります。

- `patch:before`
- `patch:after`
- `patch:instead`

属性ノードは、現行のノードの相対位置を指定するために使用されます。アンカー ノードは XPath の属性値として指定されます。XPath はパラメーター要素を使用してコンテキスト アイテムとしてみなされます。各属性には短いエイリアスがあり、長い名前の代わりに使用できます。

patch:before

この属性を使って、現行の要素を兄弟要素の前に挿入することを指定します。

エイリアス:

シンタックス

```
<processor type=”CustomC, CustomA” patch:before=”*[@type=’Sc, Sc]’” />
<processor type=”CustomC, CustomA” patch:b=”*[@type=’Sc, Sc]’” />
```

サンプル

元の構成: b

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

インクルードファイル:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set=”http://www.sitecore.net/xmlconfig/set/”>
  <sitecore>
    <element name="d" patch:before="*[@name=b]"/>
  </sitecore>
</configuration>
```

結果の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="d"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

patch:after

この構成を使って、現行の要素を兄弟要素の後に挿入することを指定します。

エイリアス: a

シンタックス

```
<processor type=”CustomC, CustomA” patch:after=”*[@type=’Sc, Sc]’” />
<processor type=”CustomC, CustomA” patch:a=”*[@type=’Sc, Sc]’” />
```

サンプル

元の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

インクルードファイル:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set=”http://www.sitecore.net/xmlconfig/set/”>
  <sitecore>
    <element name="d" patch:after="*[@name=b]"/>
  </sitecore>
```

```
</configuration>
```

結果の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="d"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

patch:instead

この属性を使って、兄弟要素の代わりに現行の要素を挿入するよう指定します。`patch:before` や `patch:after`,とは違って、この属性は元のノードまたは要素を完全に置換えることができます。

エイリアス: `i`

シンタックス

```
<processor type="CustomC, CustomA" patch:instead="*[@type='Sc, Sc]'" />
<processor type="CustomC, CustomA" patch:i="*[@type='Sc, Sc]'" />
```

サンプル

元の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

インクルードファイル:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <element name="d" patch:instead="*[@name=b]"/>
  </sitecore>
</configuration>
```

結果の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="d"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

要素ノード

要素ノードとは、`patch` 名前空間のタグで、その親に適用されます。これは `<patch:delete/>` と `<patch:attribute …/>` 要素は、それらを囲む親要素のみに作用することを意味します。これらのノードは子の要素を含むことができません。子の要素が含まれている場合は、それらは無視されます。

パッチの名前空間には 2 つの要素ノードが含まれます。

- `patch:delete`
- `patch:attribute`

存在しない要素名を指定する場合、その要素は無視されます。これらのノードにはフルネームの代わりに使用できるエイリアスがあります。

patch:delete

この要素は、親要素の移動が必要なことを示すマーカーです。いかなる属性または内部要素も指定できません。

エイリアス: `d`

シンタックス

```
<elementToRemove>
  <patch:delete />
  <!-- or -->
  <patch:d />
</elementToRemove>
```

サンプル

元の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

インクルードファイル:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set=”http://www.sitecore.net/xmlconfig/set/”>
  <sitecore>
    <element name="b">
      <patch:delete />
    </element>
  </sitecore>
</configuration>
```

結果の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

この例では、削除する必要がある要素を特定するために、インクルードファイルの `element` ノードに `name` 属性を指定する必要があります。`name` 属性を除外する場合は、最初の要素が削除されます。この例では、`a` という要素が削除されます。

patch:attribute

この要素は親要素の属性の値を変更するために使用されます。通常は設定の値を変更したり、またはタイプ属性を変更するために使用されます。デフォルトの Sitecore 機能の一部をオーバーライドする場合に、パイプライン プロセッサーまたはイベント ハンドラのタイプ属性を変更するのに便利です。

エイリアス: `a`

シンタックス

```
<elementToAlter>
  <patch:attribute name="parentAttributeName" value="newAttributeValue" />
<!-- or -->
  <patch:a name="parentAttributeName" value="newAttributeValue" />

</elementToAlter>
```

代替シンタックス:

```
<elementToAlter>

  <patch:attribute name="parentAttributeName">
    newAttributeValue
  </patch:attribute>
<!-- or -->
  <patch:a name="parentAttributeName">
    newAttributeValue
  </patch:a>
</elementToAlter>
```

属性	説明
`name`	値を変更する親属性の名前。
`value`	親属性に割り当てる新しい値。

多くの場合、`patch:attribute` よりも `set: prefix` を使用するほうが好ましいです。

`set:prefix` についての追加情報は、Set 名前空間 (http://www.sitecore.net/xmlconfig/set/) の章を参照してください。

サンプル

元の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

インクルードファイル:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <element name="b">
      <patch:attribute name="name" value="d" />
    </element>
  </sitecore>
</configuration>
```

結果の構成:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="d"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

Set 名前空間

(http://www.sitecore.net/xmlconfig/set/)

概要

`set` 名前空間は、属性値をオーバーライドするために使用されます。この名前空間には定義済みの要素または属性は含まれず、属性ノードのみに適用できます。

`set` 名前空間は `patch:attribute` 要素の機能を重複します。しかし、`set` の名前空間は、既にノードに属性が存在するか否かに関わらず特定の値に属性を設定できるため、より有用です。

`set` 名前空間プレフィックスは、現行の要素の現行の属性値をオーバーライドすることを示します。以前の状態にかかわらず、この属性の値を指定された値に設定することを示します。この属性がまだない場合は追加します。この属性は指定された値といっしょに取得します。

Sitecore は要素と属性を比較してパッチを適用するノードを特定する際、`set` 名前空間プレフィックスで始まる属性は無視されます。ひとつの要素には同じ名前を持つ複数の属性が含まれることがあるため、この名前空間を使用する際は注意してください。この場合、`set:` プレフィックスを使用せずに属性と値を指定し、パッチを適用するノードを特定する必要があります。次に `set:` プレフィックスを使って属性を指定し、属性値を追加します。たとえば、`<element attr="a" set:attr="b" />`のようになります。

`set` 名前空間を使用する前に、`http://www.sitecore.net/xmlconfig/set/` XML 名前空間を定義する必要があります。

ノードのマッチングについての追加情報は、インクルード ファイルの処理セクションを参照してください。

シンタックス

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/
xmlns:set=http://www.sitecore.net/xmlconfig/set/>
  <element attr1="value1" attr2="value2" set:attr3="newValue" />
</configuration>
```

元の要素ノードが `attr3` 属性を含む場合、その値は `newValue` によって置換えられます。元の要素にこの属性が含まれない場合には追加されます。

サンプル

元の構成:

```
<configuration>
  <sitecore>
    <element name="a" color="blue" />
    <element name="b" color="gray"/>
    <element name="c" color="red" />
    <element name="c" color="blue"/>
  </sitecore>
```

```
</configuration>
```

b 要素の色要素を変更します。

インクルードファイルは、ノード自体を特定し、新しい値を持つ色属性を含む必要があります。color 属性を持つ set プレフィックスを使用する必要があります。

インクルードファイル:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <element name="b" set:color="yellow"/>
  </sitecore>
</configuration>
```

結果の構成:

```
<configuration>
  <sitecore>
    <element name="a" color="blue" />
    <element name="b" color="yellow"/>
    <element name="c" color="red" />
    <element name="c" color="blue"/>
  </sitecore>
</configuration>
```

name="b" 属性は現行の要素を特定するために使用されます。set 名前空間プレフィックスと組み合わされた、set:color="yellow" 属性は、既存の color 属性の値を変更するために使用されます。

高度な変更を行います。

次の要素で blue を violet に変更します。

```
    <element name="c" color="blue" />
```

name 属性のみ、または color 属性のみを使用することはできません。これだけでは曖昧性を引き起こし、Sitecore が特定の属性値をマッチングする最初のノードを選択してしまうためです。

ノードを特定するには、name と color 属性の両方を使用する必要があります。インクルードファイルの要素は color 属性の新しい値を持つ set 名前空間も含む必要があります。

インクルードファイル:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <element name="c" color=”blue” set:color="violet"/>
  </sitecore>
</configuration>
```

結果の構成:

```
<configuration>
  <sitecore>
    <element name="a" color="blue" />
    <element name="b" color="gray"/>
    <element name="c" color="red" />
    <element name="c" color="violet"/>
  </sitecore>
</configuration>
```

インクルードファイルは name="c" と color="blue" 属性を使ってノードを特定し、set:color="violet" 属性を使って属性値を変更しました。

同じタスクの処理を行う複数のアプローチ

set 名前空間についてのセクションで、すべての patch:attribute 機能は set 名前空間によって実現できることを説明しました。さらに、複数の異なる方法で同じタスクを実現できることも説明しました。これについて確認します。

サンプル

次は Email Campaign Manager での構成です。

```
<configuration>
  <sitecore>
    <TypeResolver
type="Sitecore.Modules.EmailCampaign.Core.TypeResolver,Sitecore.EmailCampaign"
singleInstance="true" />
  </sitecore>
</configuration>
```

デフォルトの TypeResolver をオーバーライドし、type 属性にカスタムの値を設定します。

このタスクには複数のアプローチがあります:

アプローチ #1: patch:delete の使用と、新しく挿入

最もダイレクトなアプローチは、TypeResolver ノード全体を削除してカスタムのノードを新たに指定します。これは次のインクルードファイルで行うことができます。

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/" >
  <sitecore>
    <TypeResolver>
      <patch:delete />
    </TypeResolver>
    <TypeResolver type="CustomTypeResolver, Custom" singleInstance="true" />
  </sitecore>
</configuration>
```

アプローチ #2: patch:instead の使用

patch:instead 属性を使って、デフォルトの TypeResolver 要素ではなくカスタムの TypeResolver 要素を挿入します。

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/" >
  <sitecore>
    <TypeResolver type="CustomTypeResolver, Custom" singleInstance="true"
patch:instead="TypeResolver" />
  </sitecore>
</configuration>
```

アプローチ #3: patch:attribute の使用

patch:attribute ノードを使用すると、タイプ属性のみを修正することがでるので、singleInstance などの他の属性に影響を及ぼすことはありません。したがってそれを使用することもできます。

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/" >
  <sitecore>
```

```
    <TypeResolver>
      <patch:attribute name="type">
        CustomTypeResolver, Custom
      </patch:attribute>
    </TypeResolver>
  </sitecore>
</configuration>
```

アプローチ #4: Set 名前空間の使用

`patch:attribute` ノードを使ってこの結果を取得できる場合は、同じことを `set` 名前空間を使用して行うことができます。インクルードファイルは次のようになります。

```
<configuration xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <TypeResolver set:type="CustomTypeResolver, Custom" />
  </sitecore>
</configuration>
```

どのメソッドを選択しても、最終的な構成は同じになります。

```
<configuration>
  <sitecore>
    <TypeResolver type="CustomTypeResolver, Custom" singleInstance="true" />
  </sitecore>
</configuration>
```

メソッドの選択

ゴールを達成するために、このドキュメントで説明するメソッドをどれでも使用することができます。ただし、これらのメソッドすべてを推奨するものではありません。

`patch:delete`/`patch:insert` または `patch:instead` を使用する最初の 2 つのアプローチは、`TypeResolver` 要素を完全にオーバーライドします。`singleInstance` 属性値をオーバーライドし、古い値を無視します。

`web.config` ファイルまたは先に読み込まれるインクルードファイルでオリジナルの属性値を変更しても、上記のアプローチをとった場合、最終的な値に影響を受けません。このような特定の変更は比較的まれですが、意図的にこのアプローチを使用するかどうかは個人の選択となります。

ただし、このアプローチの使用が不適切な場合があります。たとえば、データベース タイプをオーバーライドする場合です。

```
<database id="master" singleInstance="true" type="Sitecore.Data.Database, Sitecore.Kernel">
. . . . . .
</database>
```

この特定の場合において、ノード全体をインクルードファイルに挿入する必要があります。前述のインクルードファイルの中にはデータベースの設定に影響を及ぼすものもあります。たとえば、ノード全体を置換えする場合、キャッシュ サイズを調整したり、またはカスタム エンジンを指定してもこれらの変更は無視されます。

上記の例において、`patch:attribute` または `set` を使用する最後の 2 つのアプローチは、タイプ属性のみに作用するため、より優れたアプローチです。`set` 名前空間メソッドはより少ないテキストを要求するのでよりシンプルに見えます。したがって、最後の 2 つのアプローチのどちらかを使用することを推奨します。

アプローチを選択する前に、常に別のアプローチも検討することを推奨します。使用する属性/要素のみに作用し、他の値への依存が最小数なアプローチを選択することが理想的です。

ベスト プラクティス

CMS 構成を追加する

Sitecore の関連付けられた構成設定の大多数は、アプリケーション構成ファイルの `<sitecore />` セクションにあります。Sitecore は次々に `*.config` ファイルを `<sitecore />` セクションに適用するため、`<sitecore />` セクション内にあるノードは後に適用されるインクルードファイルによって簡単にオーバーライドできます。CMS 構成の変更は、直接 `web.config` または `Sitecore.Analytics.config` ファイルなどのプリインストールされた構成ファイルで修正するのではなく、パッチング機能を使って修正を行うことを推奨します。

インクルードファイルの使用には多くのメリットがあります。

- CMS 構成を変更している場所の特定が容易で、また必要に応じてそれらの簡単に復元できます。

  カスタムの `*.config` ファイルで構成が変更されている場合、その変更を無効にするために必要な操作はファイルの拡張子を変更するだけです。

- カスタム設定をクリーンな CMS インストール 環境に展開することが容易なので、CMS のアップグレードが非常に簡単です。
- 特定の CMS 機能に影響を及ぼすカスタマイズの検証とトラブルシューティングが非常に簡単です。
- CMS モジュールの機能を壊す危険性はほとんどありません。

  たとえば、モジュールのなかには独自のカスタム `*.config` ファイルを展開して、カスタム構成を使って CMS 機能を拡張するものがあります。これまで確認したとおり、パッチのコマンドは正確に機能するために、正確なノード値 (たとえばタイプ属性値など) を頻繁に要求します。デフォルト値を直接 `web.config` ファイルで修正すると、パッチ機能は想定通りには機能せず、モジュールが壊れる恐れがあります。もちろんパッチング機能を使用してもこれらの問題を回避することはできませんが (値が前述のインクルードファイルによって変更されている場合があるため)、構成を変更するために `*.config` ファイルのみを使用する場合は、それらを特定することがより簡単になります。

これらのメリットから、次の場合にインクルードファイルを使用して構成変更を行うことを推奨します。

- デフォルトの CMS またはモジュール設定の値、オブジェクト タイプなど を変更する場合
- 新しいプロセッサー、設定、プロバイダーなどを挿入する場合

カスタム `*.config` ファイルに会社またはプロジェクトの名前を付け、ファイルによる変更点に関する簡単な説明を加えることを推奨します。

たとえば、Sitecore ログイン ページのタイトルを変更する場合、インクルードファイルは次のようになります。

インクルードファイル名: `MyCompany.WelcomeTitle.config`

インクルードファイルのコンテンツ:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
          <sitecore>
            <settings>
              <setting name="WelcomeTitle" set:value="Welcome to MyCompany" />
            </settings>
          </sitecore>
        </configuration>
```

既存のプロセッサーを変更するのではなく、新しいプロセッサーを追加する

パイプラインの動作を変更する必要がある場合は、パイプラインの引数の値をいくつか変更するだけで十分です。このような場合は、既存のプロセッサーを書換えるよりも、カスタムのプロフェッサーを挿入するほうが好ましいです。

このアプローチには次のようなメリットがあります。

- 機能を壊す恐れがほとんどないため、ソリューションのアップグレードがより簡単です。
- 同じプロセッサーを書換えるコンポーネントとの互換性に優れています。

  時々モジュールはプロセッサーをカスタムのプロセッサーと置き換えることがあります。プロセッサーを書換えるとモジュールが壊れる危険性があります。

たとえば、コンテキスト アイテムがデフォルトの `ItemResolver` で解決されない場合、特別な方法を使って解決する必要があります。単にカスタムの `ItemResolver` をデフォルト版のすぐ後に追加することが好ましいです。コンテキスト アイテムが NULL の場合 (そのアイテムは解決されていない)、機能を解決します。

Web.config ファイルコンテンツ:

```
<httpRequestBegin>
 ...
  <processor type="Sitecore.Pipelines.HttpRequest.ItemResolver, Sitecore.Kernel" />
  ...
</httpRequestBegin
```

インクルード ファイル名:

```
\App_Config\Include\MyCompany.ItemResolver.config
```

インクルード ファイル コンテンツ:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <pipelines>
      <httpRequestBegin>
        <processor type="MyCompany.ItemResolver, MyCompany"
patch:after="*[@type='Sitecore.Pipelines.HttpRequest.ItemResolver, Sitecore.Kernel']" />
      </httpRequestBegin>
    </pipelines>
  </sitecore>
</configuration>
```

デフォルトのタイプを修正するのではなく、常に新しいプロバイダーを追加する

たとえば、`Sitecore.Security.Authentication.FormsAuthenticationProvider` などの既存のプロバイダーのロジックを拡張する必要がある場合、デフォルトのプロバイダーを置換えるのではなく、変更済みのロジックを使って新しいプロバイダーを追加することを推奨しました。これによってプロダクトの支援性が向上し、デフォルトのプロバイダーへの置換が簡単になります。

新しいプロバイダーを追加するには、次のアプローチを使用することを推奨します。

初期構成:

```
<configuration>
  <sitecore>
    <authentication defaultProvider="forms">
      <providers>
        <clear/>
        <add name="forms" type="Sitecore.Security.Authentication.FormsAuthenticationProvider,
Sitecore.Kernel"/>
      </providers>
    </authentication>
  </sitecore>
</configuration>
```

インクルード ファイル コンテンツ:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <authentication set:defaultProvider="custom">
      <providers>
        <add name="custom" type="CustomType, CustomAssembly"/>
      </providers>
    </authentication>
  </sitecore>
</configuration>
```

このアプローチによってさらにプロバイダーが追加され、Sitecore はデフォルト版ではなく、強制的にこのプロバイダーを使用します。