初版：［2011 年 11 月 29 日］発行

RCMS テンプレート制作マニュアル

目次

1 本マニュアルについて

本マニュアルは RCMS のテンプレート制作方法に関する管理画面の操作マニュアルです。
HTML+CSS によるウェブデザインへの理解、Smarty の基本的な使い方への理解が必要です。

2 用語の説明

用語	説明
ページ	ウェブページを指します。
テンプレート CSS	ウェブサイトの見た目を構成するスタイルシートを指します。
レイアウト	ウェブページの段組を指します。
テンプレート	ウェブページに表示するコンテンツの HTML テンプレートを指します。
モジュール	あるコンテンツをウェブサイトに追加するための機能を指します。
コンテンツ	ウェブページに表示するコンテンツ自体を指します。
コンテンツタイプ	モジュールが出力出来るコンテンツの種類を指します。

3 他 CMS との違い

3-1 比較表

	Movable Type	WordPress	RCMS
テンプレート HTML		PHP	Smarty※1
レイアウト		テーマによる	11 種類から選択
機能の追加		プラグインの導入が必要	管理画面から選択
コンテンツ用 HTML		独自の開発が必要	標準で組込済み
モバイル、スマートフォン		プラグインの導入が必要	標準で組込済み

※1 レイアウト HTML は固定

3-2 ベース HTML は固定

一般的な CMS とは違い、ベースとなる HTML が固定されています。種類の違うコンテンツを複数追加してもレイアウトが崩れないのはこのためです。

3-3 標準で豊富なレイアウトが用意されている

RCMS では 11 種類ものレイアウトが用意されています。一般的な CMS のように HTML やスタイルシートを自分で書いてレイアウトを作り上げる必要はありません。

3-4 コンテンツ単位でレイアウトを変更出来る

一般的な CMS ではテンプレートはウェブサイトに対して一つのみが適用され、コンテンツ単位でレイアウトを変更することが出来ないか、出来たとしても専門的な知識や技術を必要とし、実現には多大な労力を必要とします。
RCMS では作成したコンテンツそれぞれに別のレイアウトを割り当てる機能が標準で備わっているため、トップページとお問い合わせフォームでは別のレイアウトを適用するのに特別な知識や技術は必要ありません。

3-5 コンテンツ単位で HTML を変更出来る

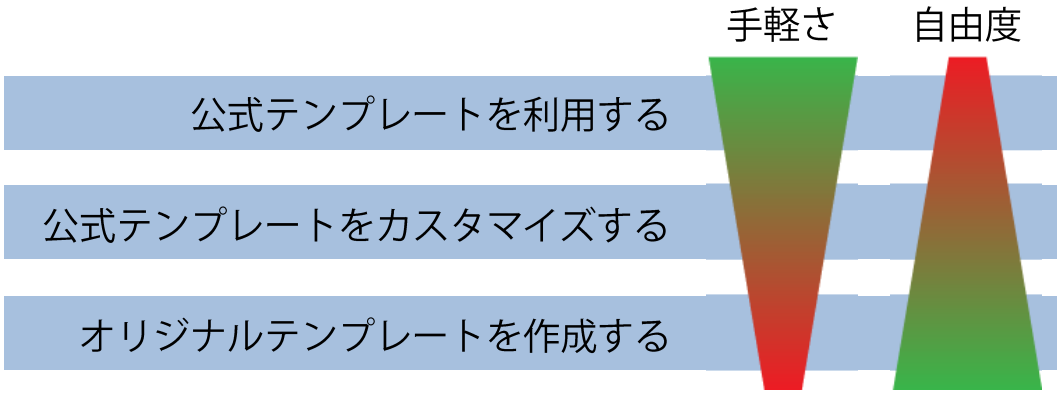
一般的な CMS でコンテンツの HTML を変更するには、記事として作成したコンテンツを記事編集画面で編集しますが、プラグインなどで追加したコンテンツについてはプラグインをカスタマイズしなければならないなど、変更には専門的な知識や技術を必要とします。
RCMS ではコンテンツごとに標準で用意されているテンプレートを編集することで HTML を変更出来ます。また、コンテンツによって複数のテンプレートが用意されています。

3-6 モバイル、スマートフォンに標準対応

従来の CMS ではプラグインでモバイルやスマートフォンへの対応を行う必要がありました。また、記事として作成したコンテンツはプラグインが提供する自動変換機能に頼るしかなく、PC 向けとモバイル向けそれぞれの HTML を独自に編集することが出来ませんでした。
RCMS では標準でモバイルとスマートフォンに対応しており、PC 向けテンプレートと同様にコンテンツごとにテンプレートを編集することが出来ます。

4 デザインの適用

RCMS のデザインは好きなところだけを好きなだけカスタマイズするだけで誰でもウェブサイトを作ることが出来ます。この方法は大きく分けて三つ。手軽さと自由度のバランスに応じて最適な方法をお選び下さい。



各手順の手軽さと自由度のバランス

4-1 テンプレート CSS の選択・作成

4-1-1 公式テンプレートを利用する

公式テンプレートは選ぶだけですぐに使える、最も手軽なデザインの適用方法です。「RCMS が提供する様々な機能を試しに使ってみたい」、「すぐにウェブサイトを作りたい」、「デザインにはあまり自信がない」などの場合にお選び下さい。

① ［デザイン］→［デザイン］を開きます。

② ［公式テンプレート CSS］から任意のテンプレートをクリックします。

(ア) 各公式テンプレート CSS はカテゴリで分類されていますので、イメージに近いカテゴリを選んでテンプレート CSS の一覧を絞り込む事も可能です。

※ プレビューではデザインの配色を変更できます。それぞれの配色のテキストボックスに直接色を指定するか、設定ボタンをクリックして配色変更ウィンドウから色をお選び下さい。

※ 配色の変更はデザイン適用後でも［デザイン］＞［デザインの基本設定］から変更できます。

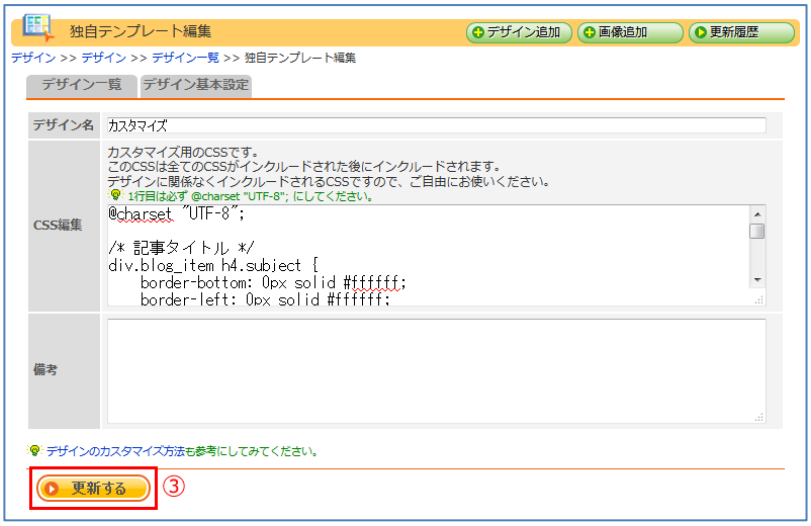
③ クリックしたテンプレート CSS を適用した場合のプレビュー画面が開きますので、よろしければ［確定］をクリックします。

4-1-2 公式テンプレートをカスタマイズする

公式テンプレートの一部をカスタマイズすることで、手軽さと自由度の両方を兼ね備えたデザインの適用方法です。「公式テンプレートよりもう少しオリジナリティをだしたい」、「ゼロからテンプレートを作るのは大変」などの場合にお選び下さい。

① ［デザイン］→［デザイン］を開きます。

② ［このシステムで使用されている CSS］にある customize.css の［編集］をクリックします。



③　編集が完了したら［更新する］をクリックして保存します。

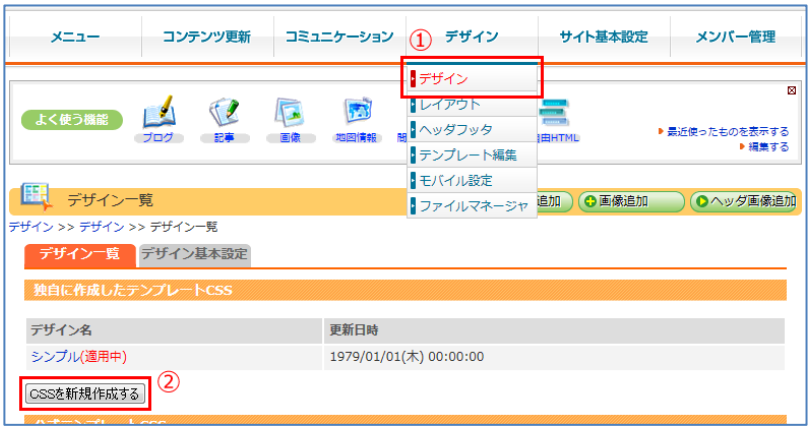
4-1-2-1 編集項目の説明

項目名	説明
デザイン名	デザインの名前です。customize.css は「カスタマイズ」に固定されています。
CSS 編集	スタイルを記述します。1 行目は必ず「@charset "UTF-8";」を記述して下さい。
備考	メモとしてお使い下さい。

※　customize.css は最後に読み込まれるスタイルシートです。公式テンプレートの一部だけをカスタマイズする他、オリジナルテンプレートを季節に応じて一時的に変更するなどの場合に便利です。

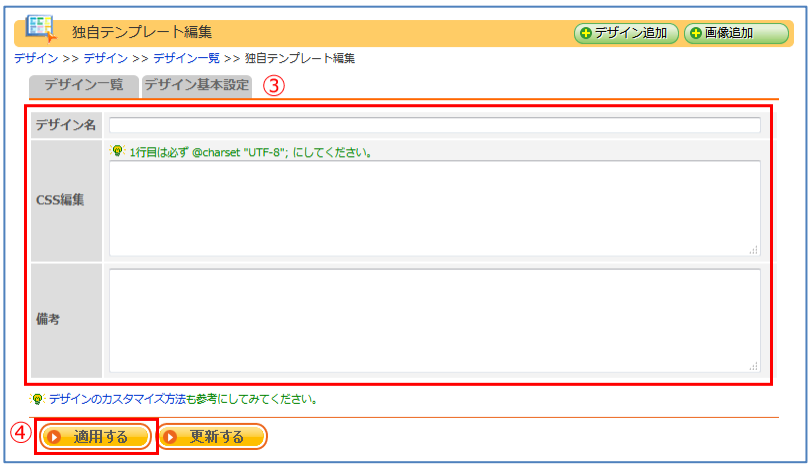
4-1-3　オリジナルテンプレート CSS を作成する

RCMS の表現力をフルに生かした、自由度の高いデザインの適用方法です。「ブランディング効果を高めたい」「独創的なコンテンツを提供したい」などの場合にお選び下さい。



①　［デザイン］→［デザイン］を開きます。

②　［独自に作成したテンプレート］にある［CSS を新規作成する］をクリックします。



③　デザイン名、CSS、備考を編集します。

④　［適用する］をクリックしてテンプレート CSS をサイトに適用します。

5 デザインのカスタマイズ

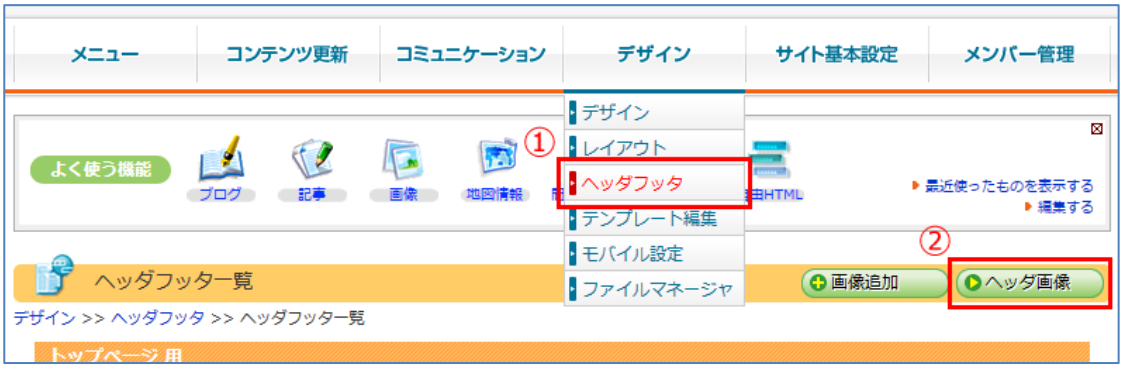
RCMS では、ほとんどのカスタマイズに専門的な知識は必要ありません。用意されたテンプレートによって、ヘッダ画像やグローバルメニュー、ぱんくずリンクなどが自動的に配置されます。
どのようなヘッダ画像を表示するか、グローバルメニューに何を表示するかは管理画面に用意された編集機能から簡単に変更する事が可能です。ご要望に応じた操作方法は以下をご覧下さい。

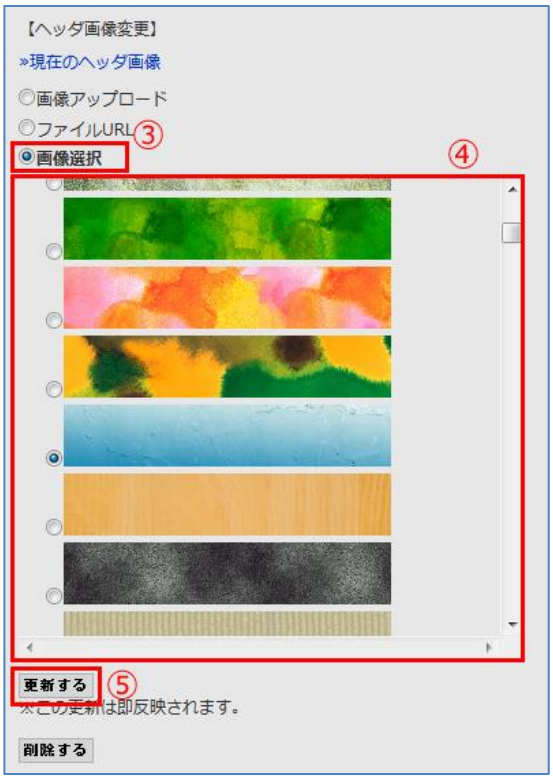
5-1 サイトの情報を変更したい

① ［サイト基本設定］→［サイト管理］を開きます。
② 該当する項目を編集します。
③ ［更新する］をクリックして変更した内容に更新します。

5-2 ヘッダ画像を変更したい

5-2-1　一覧から選択する





①　［デザイン］→［デザイン］を開きます。
②　［ヘッダ画像追加］をクリックします。
③　［画像選択］をクリックします。
④　一覧から画像を選択し、ラジオボタンをクリックします。
⑤　［更新する］をクリックして更新します。
⑥　［ヘッダ画像追加］をクリックします。
⑦　［ファイル URL］を選択します。
⑧　テキストボックスにコピーしたファイルパスを貼り付けます。
⑨　［更新する］をクリックして更新します。

5-2-2　パソコンにある画像をヘッダ画像にする

RCMS では複数のヘッダ画像用の画像を提供していますが、独自のヘッダ画像をアップロードしてご利用いただけます。

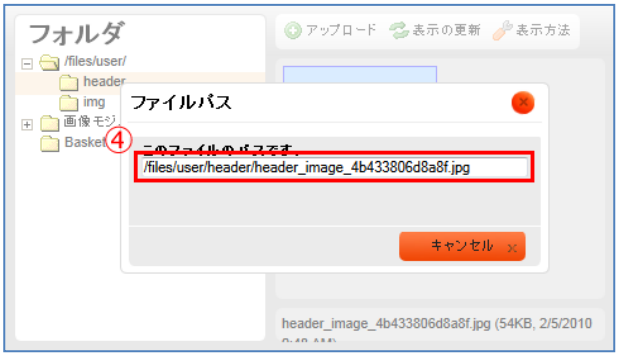
①［デザイン］→［ヘッダフッタ］を開きます。
②［ヘッダ画像］をクリックするとヘッダ画像ウィンドウが開きます。
③［画像アップロード］を選択し、アップロードするファイルを選択します。
④［更新する］をクリックします。
※ヘッダ画像はアップロード完了と同時にウェブサイトに反映されます。
※ヘッダ画像のその他の更新方法については［4-3 ヘッダ画像の変更］に記載があります。

5-2-3　以前アップロードしたヘッダ画像にする

①　［デザイン］→［ファイルマネージャ］を開きます。

②　新しく開いたウィンドウの左のツリーから［/files/user/］→［header］をクリックします。

③　右側に今までにアップロードしたヘッダ画像が表示されますので、変更する画像を右クリックして［ファイルパス］をクリックします。

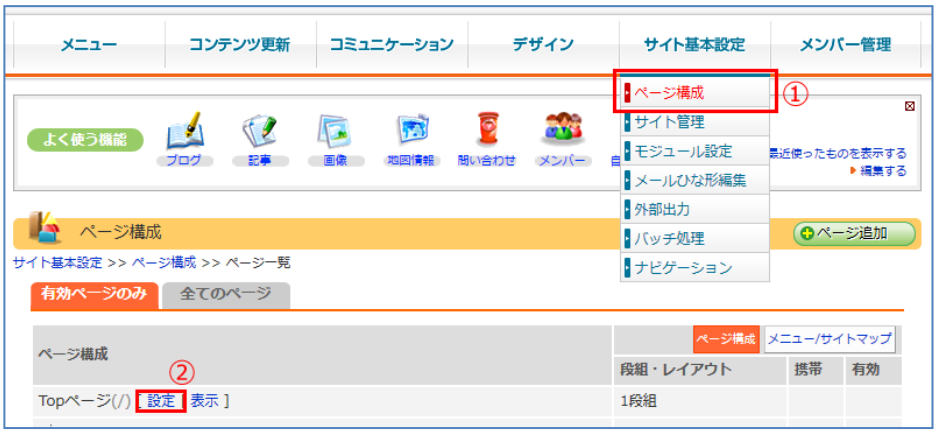


④　表示されるファイルパスをコピーしてウィンドウを閉じます。

⑤　［デザイン］→［デザイン］を開きます。

5-3 レイアウトを変更したい

5-3-1　段組を変更する



①　［サイト基本設定］→［ページ構成］を開きます。

②　レイアウトを変更したいページの［設定］をクリックします。

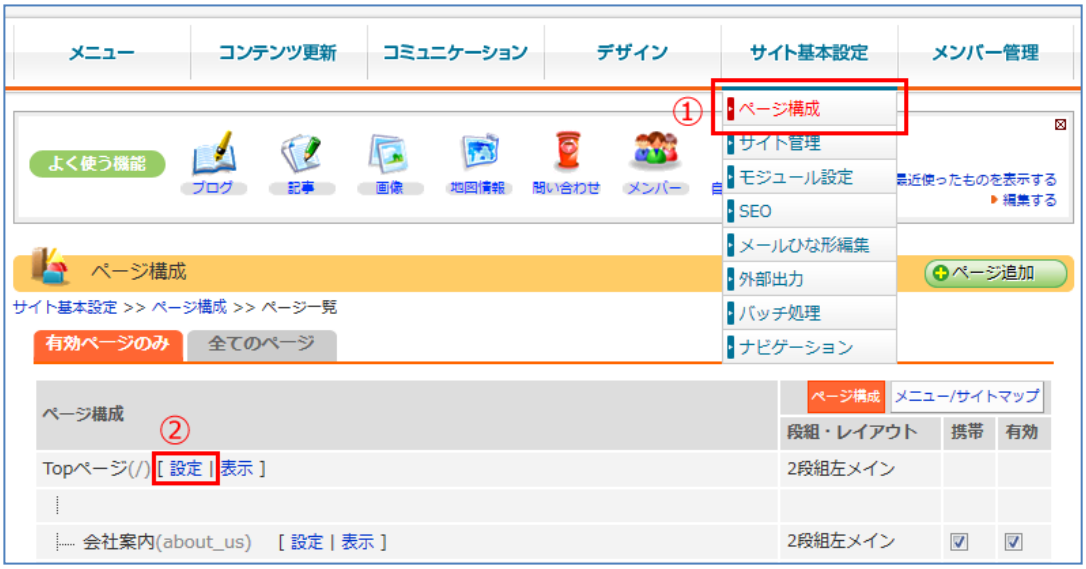
③　［基本設定］タブを開きます。

④　［段組・レイアウト］から任意の段組を選択します。

⑤　［更新する］をクリックします。

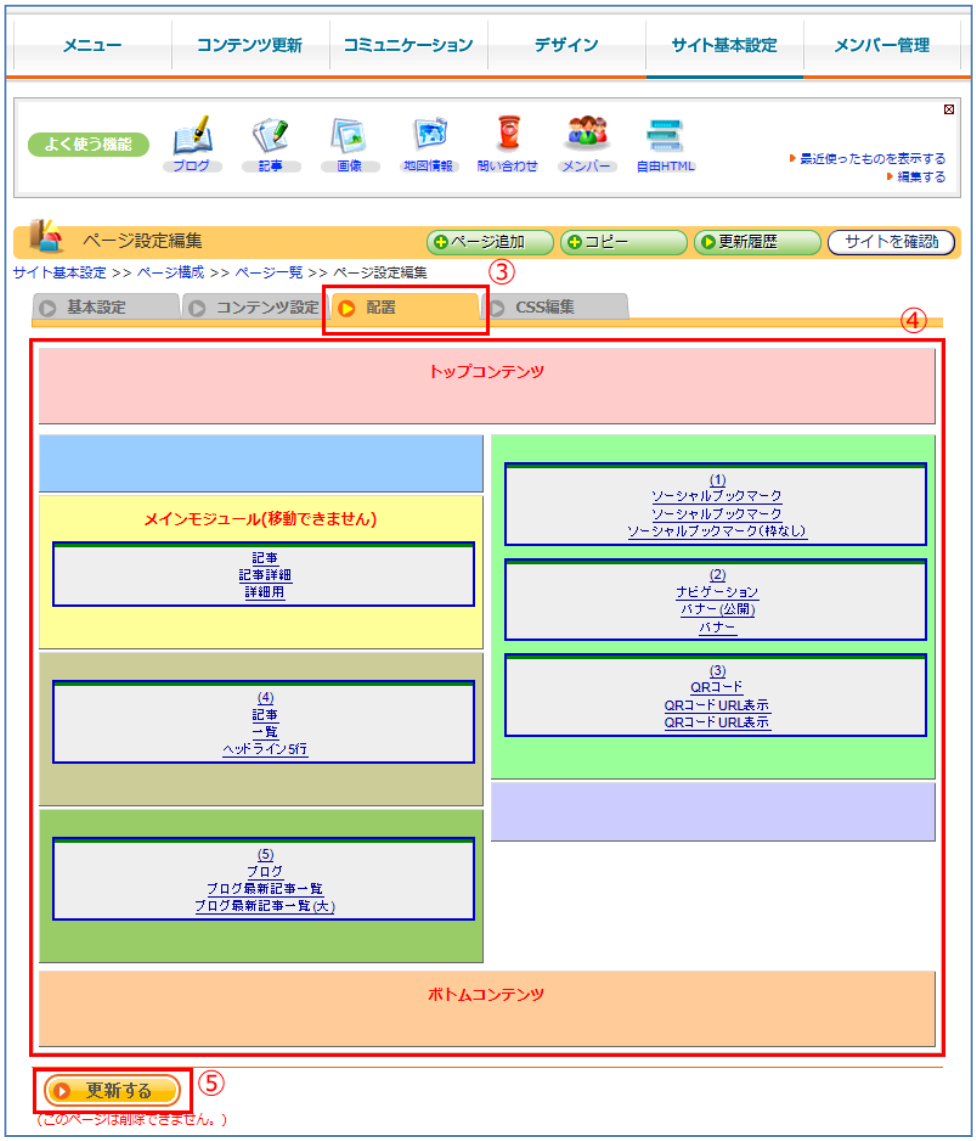
5-3-2　コンテンツの配置を変更する

ページに追加したコンテンツは自由に配置を変更する事が出来ます。



①　［サイト基本設定］→［ページ構成］を開きます。

②　配置を変更したいページの［設定］をクリックしてページの設定画面を開きます。



③　［配置］を開きます。

④　各コンテンツを任意の場所にドラッグします。

⑤　［更新する］をクリックして変更内容を更新します。

5-3-3　ヘッダとフッタを表示しない

①　［サイト基本設定］→［ページ構成］を開きます。

②　ヘッダとフッタを表示しないページの［設定］をクリックします。

③　［基本設定］→［表示設定］の［ヘッダフッタ非表示］にチェックを入れます。

④　［更新する］をクリックして更新します。

5-4 メニューを変更したい

① ［サイト基本設定］→［ページ構成］を開きます。

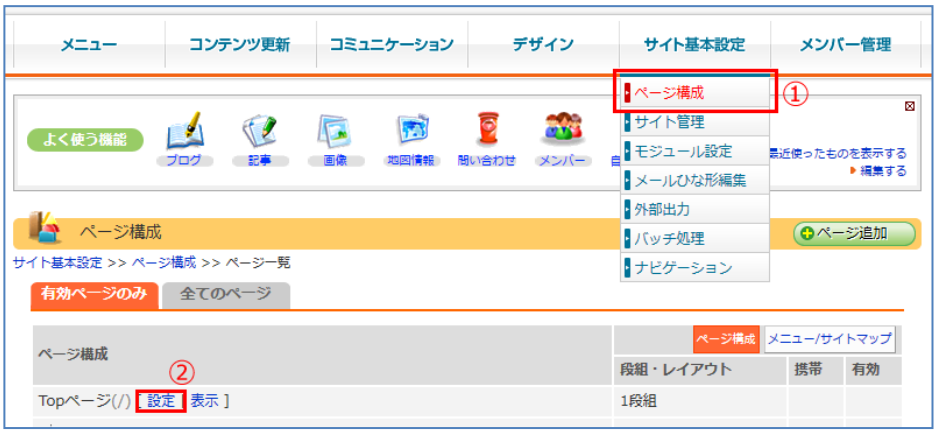
② ［メニュー/サイトマップ］をクリックします。

③ メニュー、ヘッダサブメニュー、フッターメニューそれぞれに表示したいコンテンツにチェックを入れます。

④ ［更新する］をクリックして更新します。

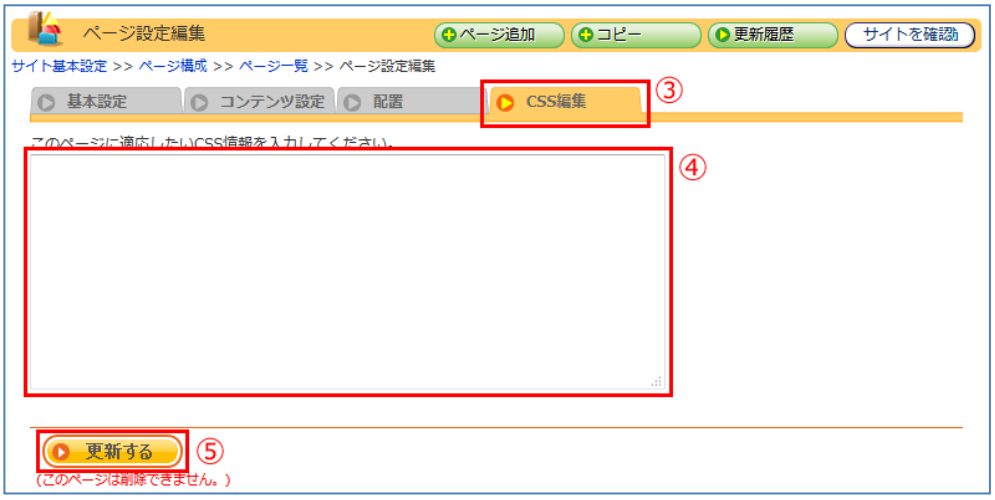
5-5 ページをカスタマイズしたい

5-5-1 特定のページだけスタイルシートを編集する



① ［サイト基本設定］→［ページ構成］を開きます。

② スタイルシートを編集したいページの［設定］をクリックします。



③ ［CSS 編集］タブを開きます。

④ スタイルシートを記述します。

⑤ ［更新する］をクリックして更新します。

5-5-2 コンテンツのテンプレート HTML をカスタマイズする

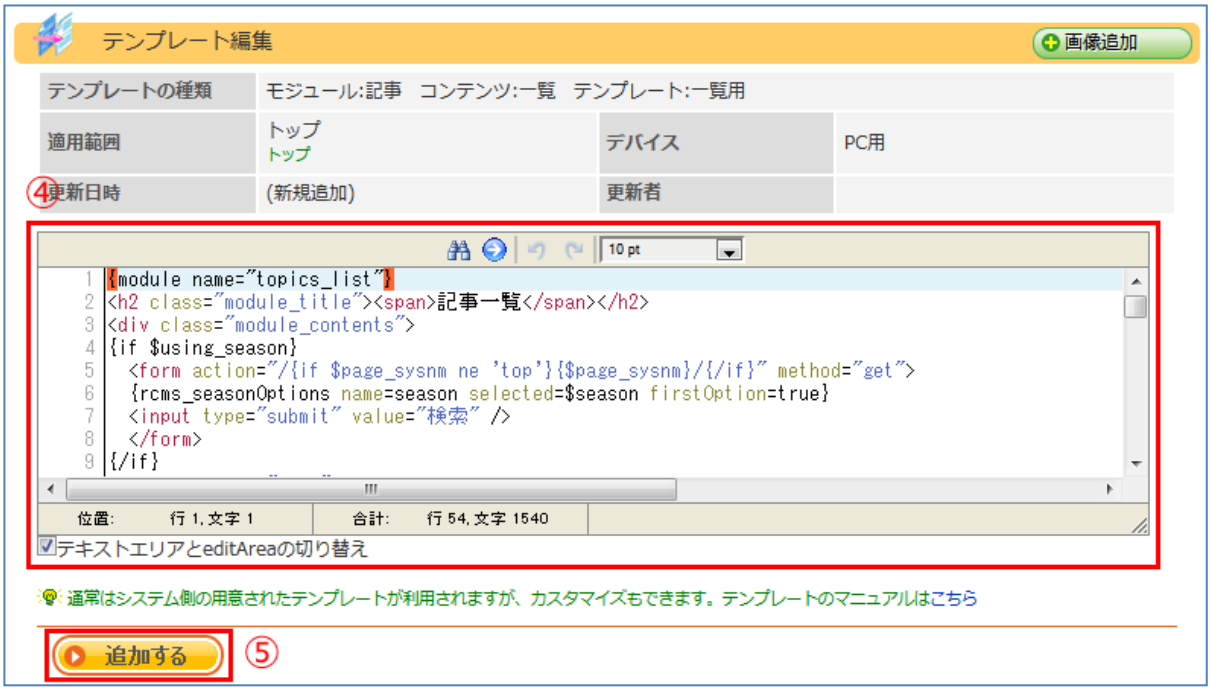
RCMS では、各モジュールに割り当てられたデフォルトテンプレートをカスタマイズすることが可能です。カスタマイズしたテンプレートは全てのページに表示されているコンテンツ、もしくは任意のページに表示されているコンテンツのみに割り当てることが可能です。

① 管理画面の［デザイン］→［テンプレート編集］を開きます。

② カスタマイズするコンテンツのモジュール、コンテンツタイプ、テンプレートを選択します。

③ ［追加する］をクリックします。

［追加する］をクリックすると、テンプレート編集フォームがポップアップします。編集フォームには選択したコンテンツのテンプレート HTML があらかじめ挿入されています。



④ テンプレート HTML を編集します。

⑤ ［追加する］をクリックしてコンテンツのテンプレート HTML を追加します。

5-5-3　サイトの幅を変更する

ウェブサイトの横幅は管理画面の[デザイン]→[レイアウト]からレイアウト毎に設定できます。横幅に指定できる単位はpx（ピクセル）と%の2種類があり、%の場合はブラウザの仕様により開いているドキュメントの幅に対する割合として適用されます。

5-5-4　段組の幅と余白を変更する

レイアウト毎に段組の左側や右側、もしくは両側の幅、余白を設定出来ます。レイアウトによっては指定出来ません。詳しくは各レイアウトテンプレートの項目を参照して下さい。
幅の単位はpxか%のいずれかを選択できますが、余白の単位はpxのみとなっています。

5-5-5　幅をコンテンツのみに指定する

レイアウト毎に指定する横幅をコンテンツにのみ反映させたい場合は、管理画面の［デザイン］→［レイアウト］にある［ヘッダ、メニュー、フッタを 100%にして、コンテンツをサイト幅に合わせる］にチェックを入れて［更新する］をクリックして下さい。
なお、この設定は全てのレイアウトに影響します。

5-5-6　配置を変更する

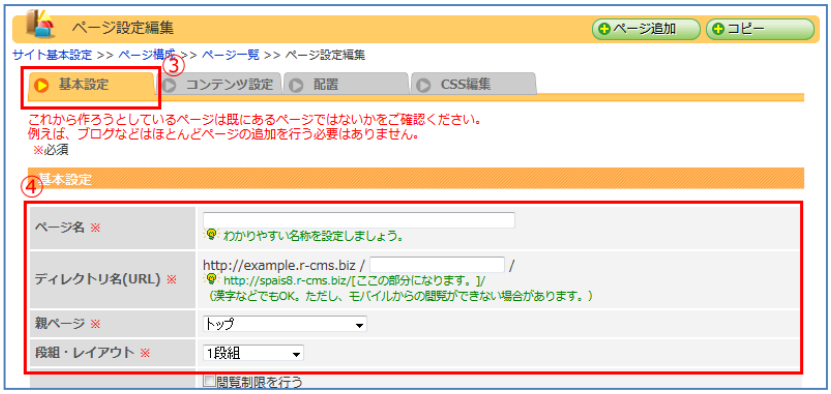
ウェブページ全体の配置を left（左寄せ）、center（真ん中寄せ）、right（右寄せ）の 3 種類から選ぶことが出来ます。

ウェブページの配置は管理画面の［デザイン］→［レイアウト］からレイアウト毎に設定できます。

5-6 新しいページを作成したい

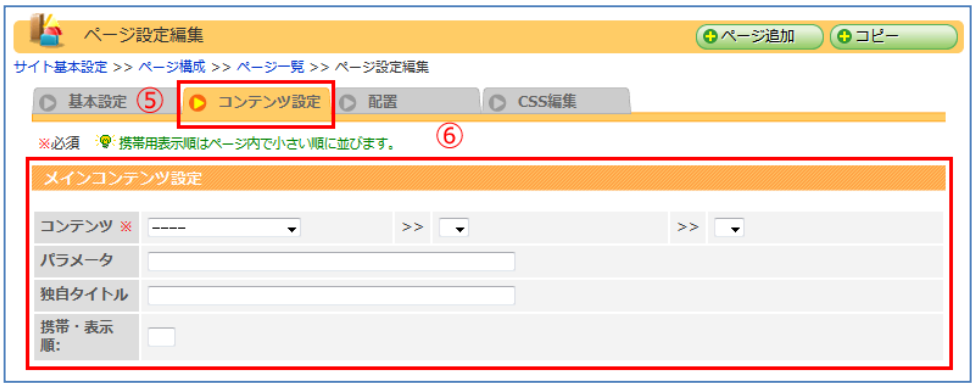
①　［サイト基本設定］→［ページ構成］を開きます。

②　［ページ追加］をクリックします。



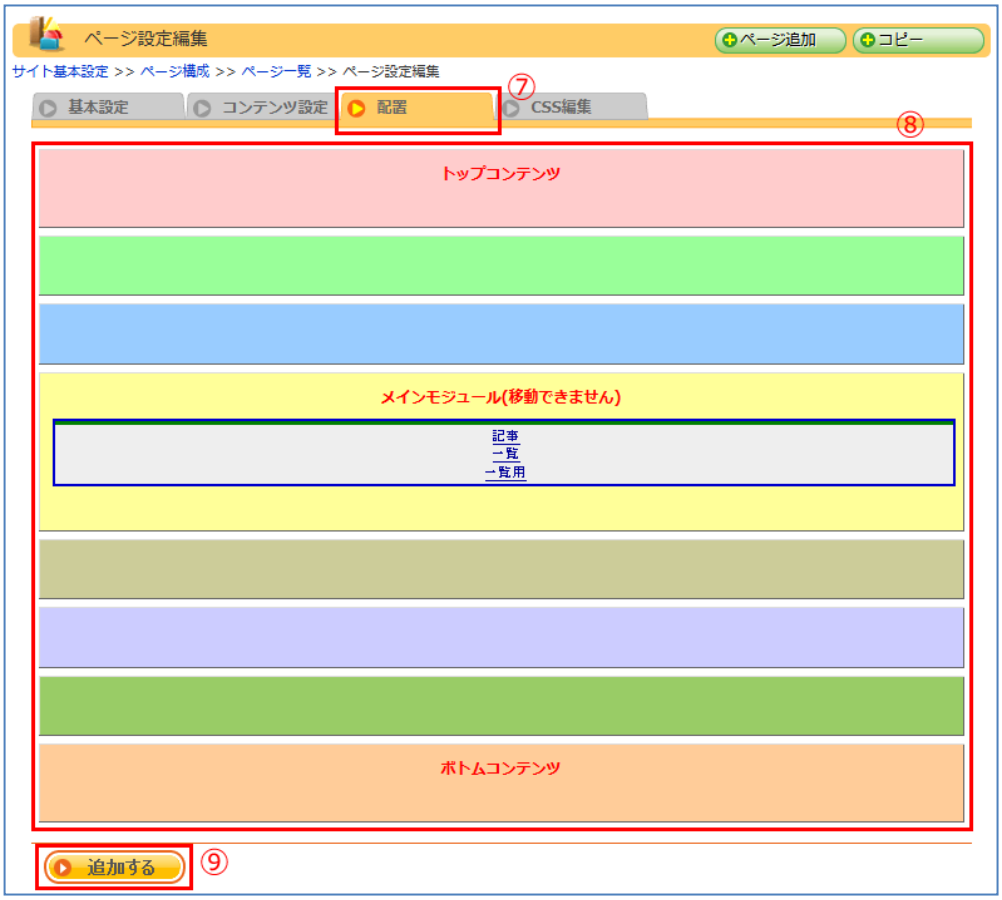
③　［基本設定］タブを開きます。

④　［ページ名］、［ディレクトリ名（URL）］をそれぞれ入力し、［親ページ］、［段組・レイアウト］をそれぞれ選択します。



⑤　［コンテンツ設定］タブを開きます。

⑥　［メインコンテンツ設定］の［コンテンツ］を選択します。



⑦　［配置］タブを開きます。

⑧　コンテンツをドラッグして配置します。

⑨　［追加する］をクリックしてページを追加します。

5-6-1　表示設定の内容

設定項目	内容
サイトマップ	サイトマップに表示する場合にチェックします。
メニュー	メニューに表示する場合にチェックします。
ヘッダサブメニュー	ヘッダサブメニューに表示する場合にチェックします。
フッターメニュー	フッターメニューに表示する場合にチェックします。
有効	チェックしない場合ページにアクセス出来ません。
携帯	携帯電話からのアクセスを可能にする場合にチェックします。
スマートフォン	スマートフォンからのアクセスを可能にする場合にチェックします。
ヘッダフッタ非表示	ページ内のヘッダとフッタを非表示にする場合にチェックします。

6 スタイルシート

RCMS テンプレートは(X)HTML と CSS によってデザインおよび修飾がなされています。原則としてコンテンツの構造を変更するには HTML を、修飾を変更するには CSS を編集します。
RCMS の修飾はスタイルシートで定義されています。修飾だけならばページのカスタマイズを行わずともスタイルシートの変更のみで行えるようになっています。ただし、デザインテンプレートによってはページのカスタマイズが必要な場合があります。

6-1 各スタイルシートの説明

1 種類のデザインテンプレートには複数の CSS が定義されており、システムによって編集可能な CSS とそうではない CSS に分類されています。以下は各 CSS の役割と編集の可否を一覧したものです。

6-1-1 layout[1-13].css

デザインのレイアウトを構成しているスタイルシートで、変更することは出来ません。
[1-6]の部分には 1 から 13 の数字が入り、各段組に対応しています。

スタイルシート名	対応する段組
layout1.css	1 段組
layout2.css	2 段組左メイン
layout3.css	2 段組右メイン
layout4.css	3 段組中メイン
layout5.css	逆 L 右メイン
layout7.css	3 段組左メイン
layout8.css	2 段組右メイン 2
layout9.css	逆 L 右メイン
layout10.css	2 段組左メイン 2
layout11.css	2 段組中メイン
layout12.css	3 段組中メイン 2
layout13.css	2 段組左メイン 3

※各レイアウトの詳細は［2-3-4 レイアウトテンプレート］を参照してください。

6-1-2 default.css

全てのデザイン、レイアウトに共通したログイン時に表示されるヘッダーやエラー表示などの基本的なスタイルが含まれているスタイルシートで、変更は出来ません。

6-1-3 parts.css

公式テンプレート、もしくは独自に作成したテンプレート css が定義されているスタイルシートです。ウェブサイト全体のビジュアルデザインを構成するスタイルが定義されています。

6-1-3-1 公式テンプレートの一部をカスタマイズしたい

公式テンプレートの一部をカスタマイズして利用する場合、parts.css ではなく customize.css を編集してください。

6-1-4 modules.css

モジュールやコンテンツ内で利用するスタイルが定義されているスタイルシートで、変更は出来ません。

6-1-4-1 コンテンツのスタイルを変更したい

コンテンツ別にスタイルを変更したい場合はページ別 CSS に定義してください。複数のページに表示するコンテンツのスタイルは customize.css に定義します。

6-1-5 customize.css

ウェブサイト全体に一番最後に適用されるスタイルシートです。公式テンプレートの一部をカスタマイズする場合や、各ページに共通して表示するコンテンツのスタイルなどを定義します。

6-1-5-1 複数のテンプレート CSS を運用する場合

customize.css はテンプレート CSS の内容とは関係なく固定したスタイルシートですので、複数のテンプレート CSS を運用する際、各テンプレートで共通して適用したいスタイルはテンプレート CSS ではなく customize.css にて定義した方が効率的です。

6-1-6 ページ別 CSS

各ページ単位にスタイルを定義出来るスタイルシートです。管理画面の［サイト基本設定］→［ページ構成］→［対象のページ設定］→［CSS 編集タブ］で表示されるテキストエリアにスタイルを定義してください。
インラインスタイルとしてページ内に読み込まれますので、スタイルの継承レベルが同等であった場合、customize.css などの外部スタイルシートの定義を上書きします。
※詳しくは［4-6-1 特定のページだけスタイルシートを編集する］を参照して下さい。

6-1-7 ブログ用 CSS

ブログ毎にブログ記事用にスタイルを定義出来るスタイルシートです。ページ別 CSS と同様、インラインスタイルとしてページ内に読み込まれますが、ページ別 CSS の後に定義されるため、ページ別 CSS と同じセレクタにスタイルを定義している場合はブログ用 CSS のスタイルが適用されます。
また、ブログ用 CSS はブログ記事専用のスタイルですので、ブログトップやブログ一覧ページなどには反映されません。

6-2 継承ルール

指定したスタイルの優先順位が同様であった場合、以下の順に優先されます。

1 ブログ用 CSS
2 ページ別 CSS
3 default.css
4 layout[1-13].css
5 parts.css
6 modules.css
7 customize.css

このうちブログ用 CSS とページ別 CSS は指定されたページでのみ有効となり、インラインスタイルとして読み込まれます。

6-3 デザインテンプレート

上記 CSS のうち「公式テンプレート CSS」もしくは「独自に作成した CSS」として parts.css がデザインテンプレートにあたります。独自に作成した CSS は名前を付けて保存する事が可能ですので、キャンペーン期間中だけデザインを変更する場合になどにワンタッチで切り替えるなど、複数のデザインを活用する際にご利用下さい。
公式テンプレートの一部だけを変更したい場合などには、customize.css にスタイルを記述します。ただし、customize.css は上記の parts.css の種類に依存しない点についてご注意下さい。

7 HTML テンプレートの構造

RCMS は(X)HTML によりコンテンツの構造が、CSS によりコンテンツの修飾が定義されています。そのため、カスタマイズの内容によって修正する箇所が違います。

7-1 基本となるテンプレート HTML

RCMS の基本となるテンプレート HTML はデザインやコンテンツの内容によらず以下のように固定となっています。body 要素の id に指定されたレイアウト番号に対応する段組が、固定されたテンプレート HTML のスタイルによって定義されます。

```
<body id="layout[1-13]">
    <div id="top" class="wrapper1">
        <div id="main_module_id_" class="wrapper2">
            <div id="container">
                <div id="header">
                    <ul id="snavi">ヘッダサブメニュー</ul>
                    <h1 class="siteName">サイト名</h1>
                    <p class="mainPhoto">ヘッダー画像</p>
                </div>
                <div id="navi">ヘッダメニュー</div>
                <div id="footpath">パンくずリスト</div>
                <div id="contents">
                    <div id="top_contents">コンテンツ</div>
                    <div id="main_contents">コンテンツ</div>
                    <div id="bottom_contents">コンテンツ</div>
                </div>
                <div id="footer">フッター</div>
            </div>
        </div>
    </div>
</body>
```

7-2 ページ別のテンプレート HTML

RCMS では追加するモジュール毎に一つ、もしくは複数のデフォルトテンプレートが用意されています。カスタマイズしない場合はデフォルトのテンプレートをそのまま利用してください。

7-2-1　適用範囲

適用範囲は対象となるコンテンツを表示しているページによって適用するテンプレート HTML を変更する場合に指定します。例えば「トップページに表示する記事一覧だけ、別のテンプレートを適用したい」という場合に、適用範囲にトップページを指定したテンプレート HTML を追加します。
「あるページにのみ」という条件が無い場合、適用範囲は［全てのページで適用］として下さい。

7-2-2　デバイス

RCMS では標準でモバイルやスマートフォンに対応しています。PC 向けと同様にテンプレート HTML をカスタマイズすることが可能です。モバイル、スマートフォン向けのテンプレートをカスタマイズする場合は［デバイス］でモバイルやスマートフォンを選択して下さい。

7-3 ヘッダとフッタ

ウェブページのヘッダとヘッダは管理画面の［デザイン］→［ヘッダフッタ］から編集します。
ヘッダとフッタではナビゲーションメニューやサイトのヘッダ画像、メタ情報などをカスタマイズ出来ます。
また、ヘッダとフッタはそれぞれトップページと一般ページを別々にカスタマイズ出来ますので、例えば「トップページにのみヘッダ画像を表示したい」などの場合に、トップページ用のヘッダテンプレートにのみヘッダ画像を表示するようカスタマイズすることが可能です。

7-3-1　ナビゲーションメニュー

RCMS では標準で［ヘッダメニュー］、［ヘッダサブメニュー］、［フッタメニュー］と、3 種類のナビゲーションメニューが用意されています。
各ナビゲーションメニューに表示する項目を変更したい場合は「4-5 メニューを変更したい」を参照して下さい。

7-3-2　各メニューの定義

	格納されている変数名	デフォルトの定義場所
ヘッダメニュー	$global_menu_list	グローバルメニュー
ヘッダサブメニュー	$global_menu_head_sub_list	ヘッダ（サイト上部）
フッタメニュー	$global_menu_footer_list	フッタ

7-3-3　変数の構造

```
$global_menu_list = array(array(
    'this' => 今表示しているページならば 1,
    'sys_nm' => ページの定義名,
    'page_nm' => ページの表示名
));
```

※ナビゲーションメニューが格納されている変数の構造はいずれのメニューも同じです。

7-3-4　ヘッダ画像

ヘッダ画像はデフォルトのテンプレートでは［ヘッダ（サイト上部）］に定義されています。
ヘッダ画像が指定されている場合、$smarty.const.HEADER_IMAGE_URL もしくは <!--%header_image_url%-->に URL が格納されていますので、img 要素の src にそのまま指定して下さい。

7-3-4-1 定義例

```
<img src="<!--%header_image_url%-->">
```

7-3-5　メタ情報

デフォルトのテンプレートではウェブサイトのメタ情報は［ヘッダ（HEAD タグなど）］に定義されています。［ヘッダ（HEAD タグなど）］はウェブページの先頭で読み込まれますので、head 要素の定義はここでのみ行って下さい。

7-3-5-1 メタ情報に関連するテンプレート変数

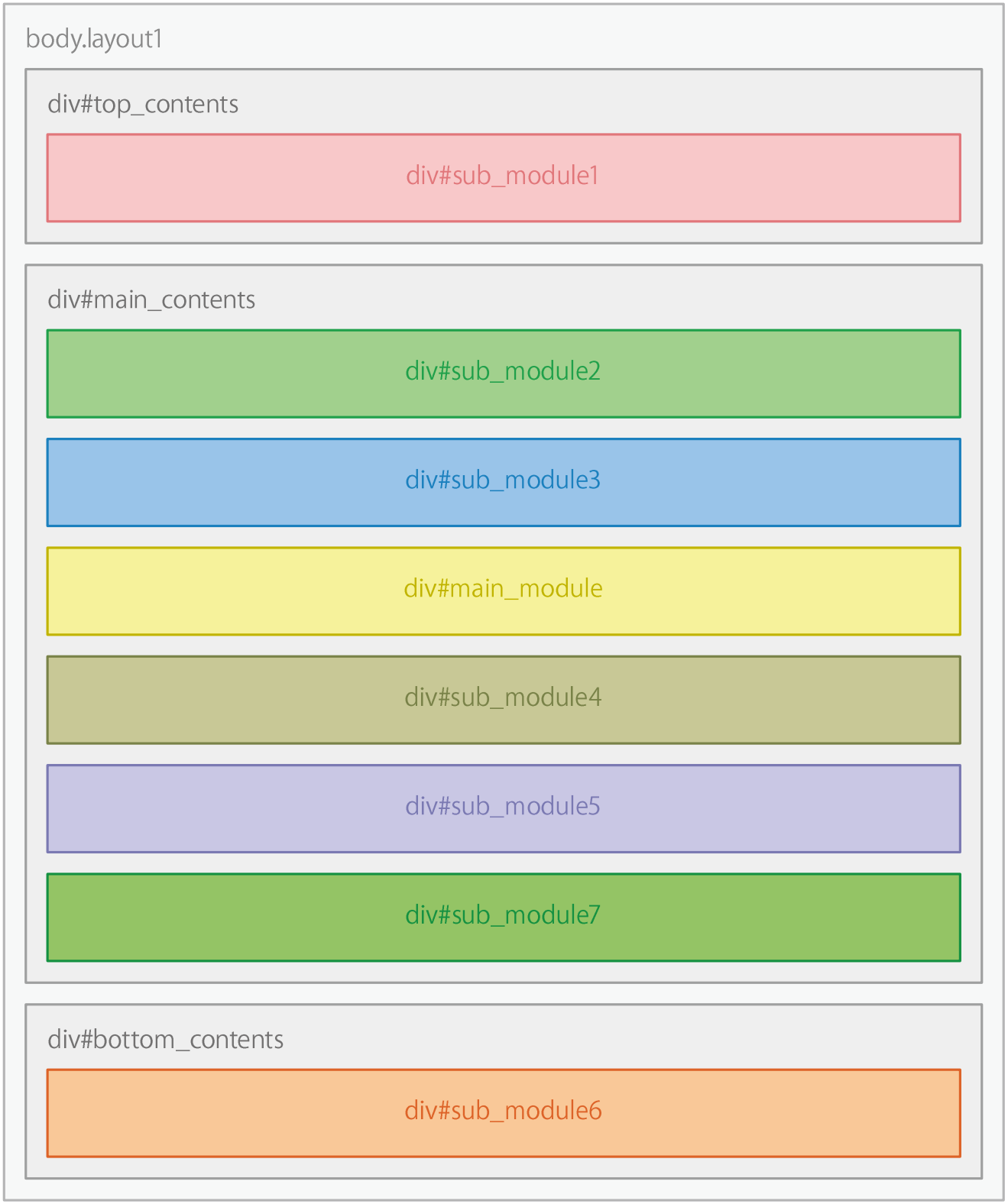
変数名	内容
<!--%title%-->	ウェブページのタイトル
<!--%keywords%-->	ウェブページに指定したメタキーワード
<!--%description%-->	ウェブページに指定したメタディスクリプション
<!--%site_nm%-->	ウェブサイト名
<!--%template_id%-->	レイアウトテンプレート番号（1 から 13）
<!--%author%-->	コンテンツの制作者名

8 レイアウト

RCMS では段組をレイアウトテンプレートとして提供しており、作成したページ毎にレイアウトテンプレートを指定して任意の段組のレイアウトを利用出来ます。
また、レイアウトテンプレートはそれぞれ横幅や余白などを設定する事が出来ます。横幅や余白の設定については［5-5-3 サイトの幅を変更する］を参照して下さい。
RCMS ではレイアウトテンプレート毎に出力する HTML が違います。以下の図は出力する HTML の構造とスタイルシートでの段組の模式図となります。レイアウトを選択する際、またカスタマイズする際に参考として下さい。

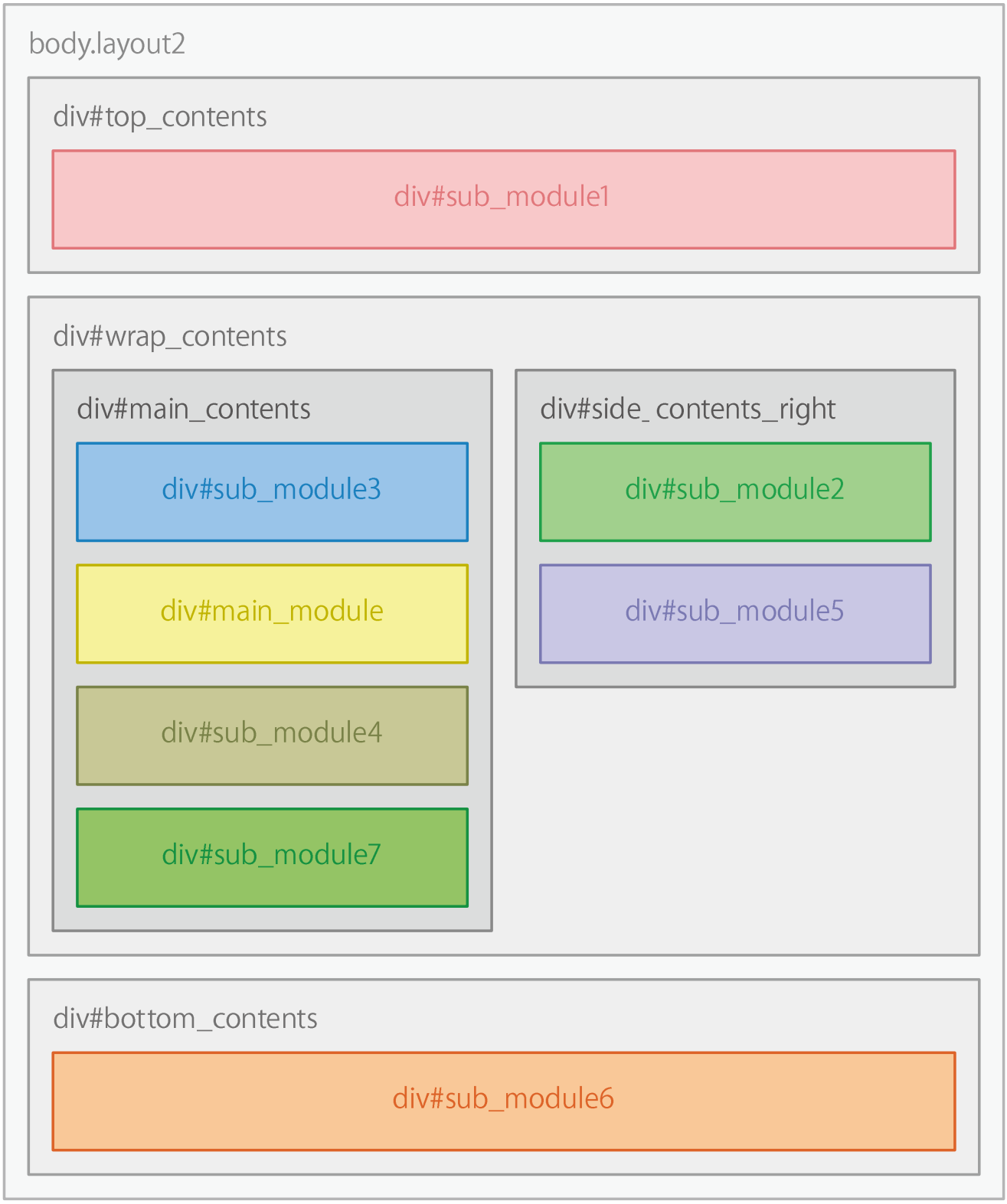
8-1 1 段組

全てのコンテンツを 1 段組で表示します。1 段組とは段組の無い状態です。
段組毎に幅を指定することは出来ません。また余白も設定出来ません。



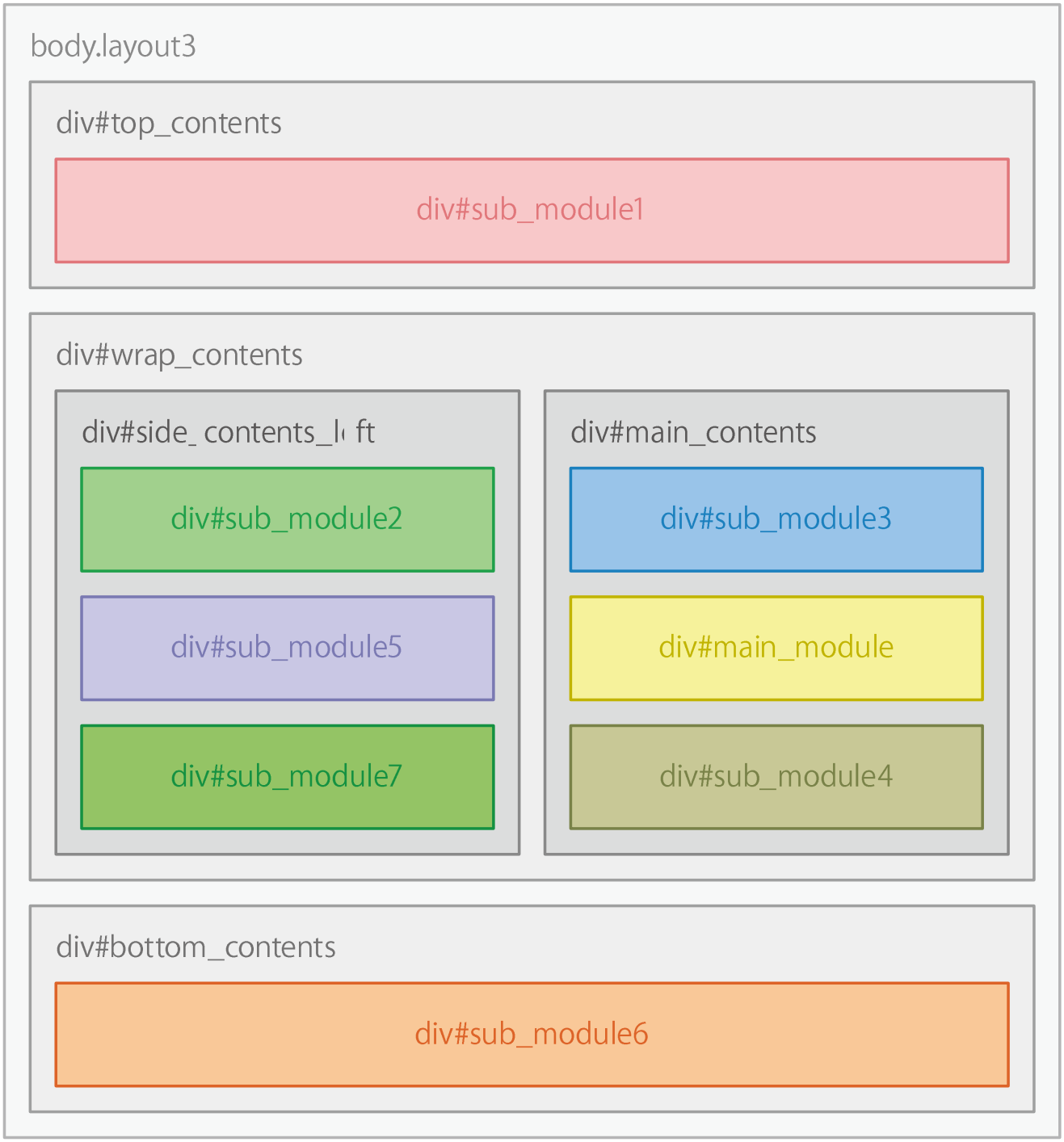
8-2 2 段組左メイン

メインのコンテンツを左側に配置した 2 段組のレイアウトです。右側段組の幅と、余白を指定出来ます。左側段組の幅は［ウェブサイトの幅－右側段組の幅－余白］となります。



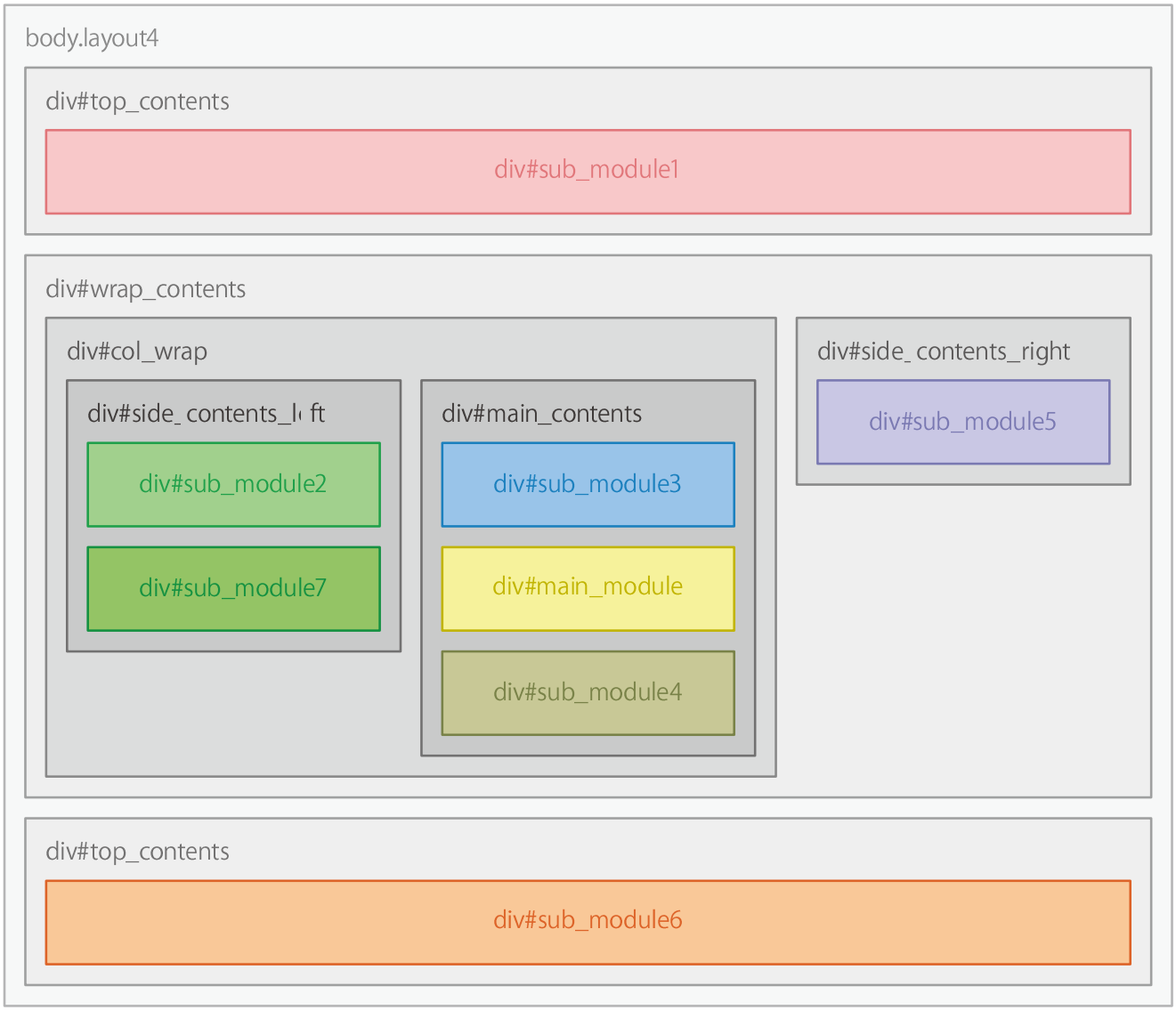
8-3 2 段組右メイン

メインのコンテンツを右側に配置した 2 段組のレイアウトです。左側段組の幅と、余白を指定出来ます。右側段組の幅は［ウェブサイトの幅－左側段組の幅－余白］となります。



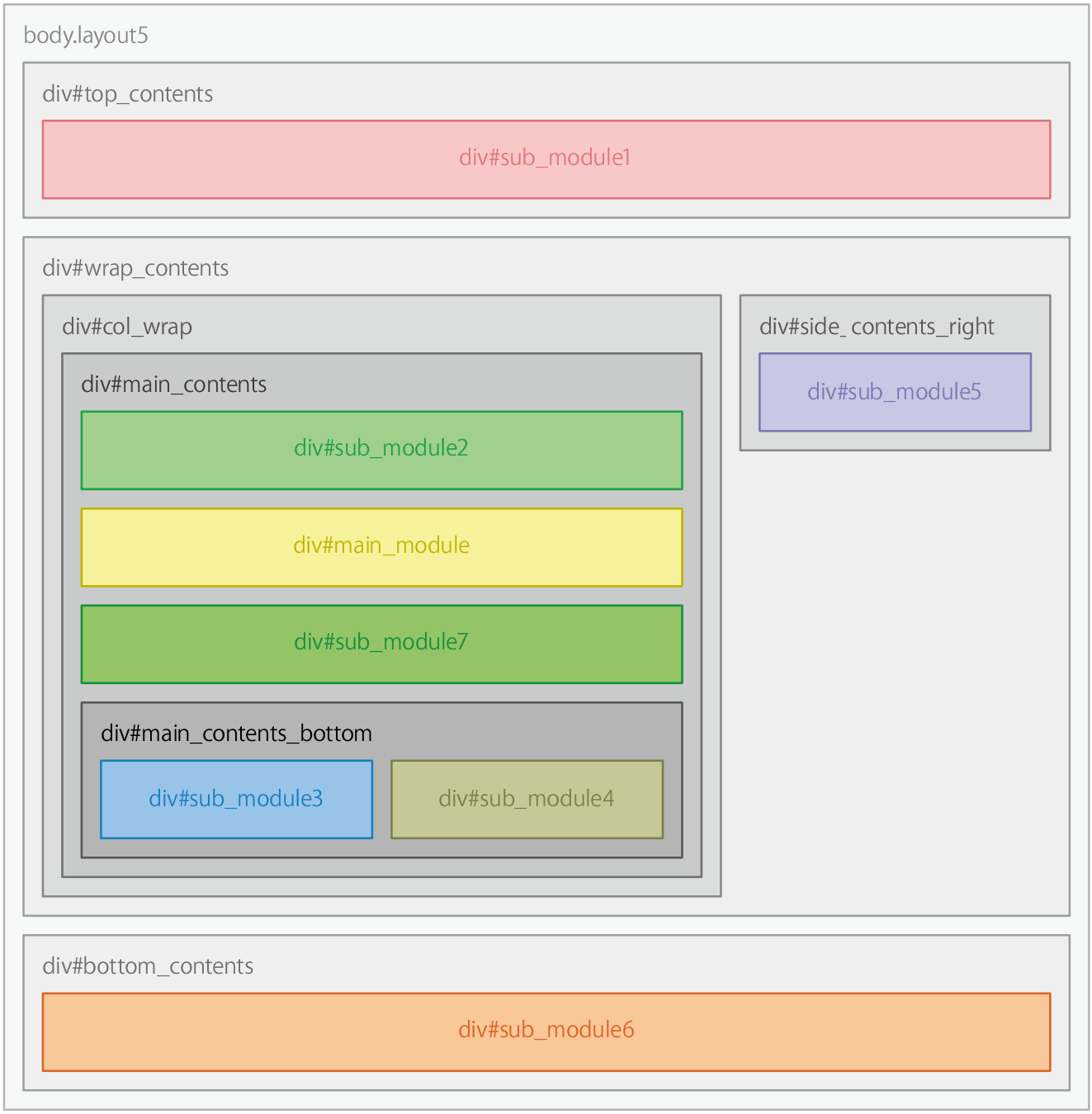
8-4 3 段組中メイン

メインのコンテンツを真ん中に配置した 3 段組のレイアウトです。左右段組の幅と、余白を指定出来ます。
真ん中段組の幅は［ウェブサイトの幅－左右段組の幅－余白］となります。



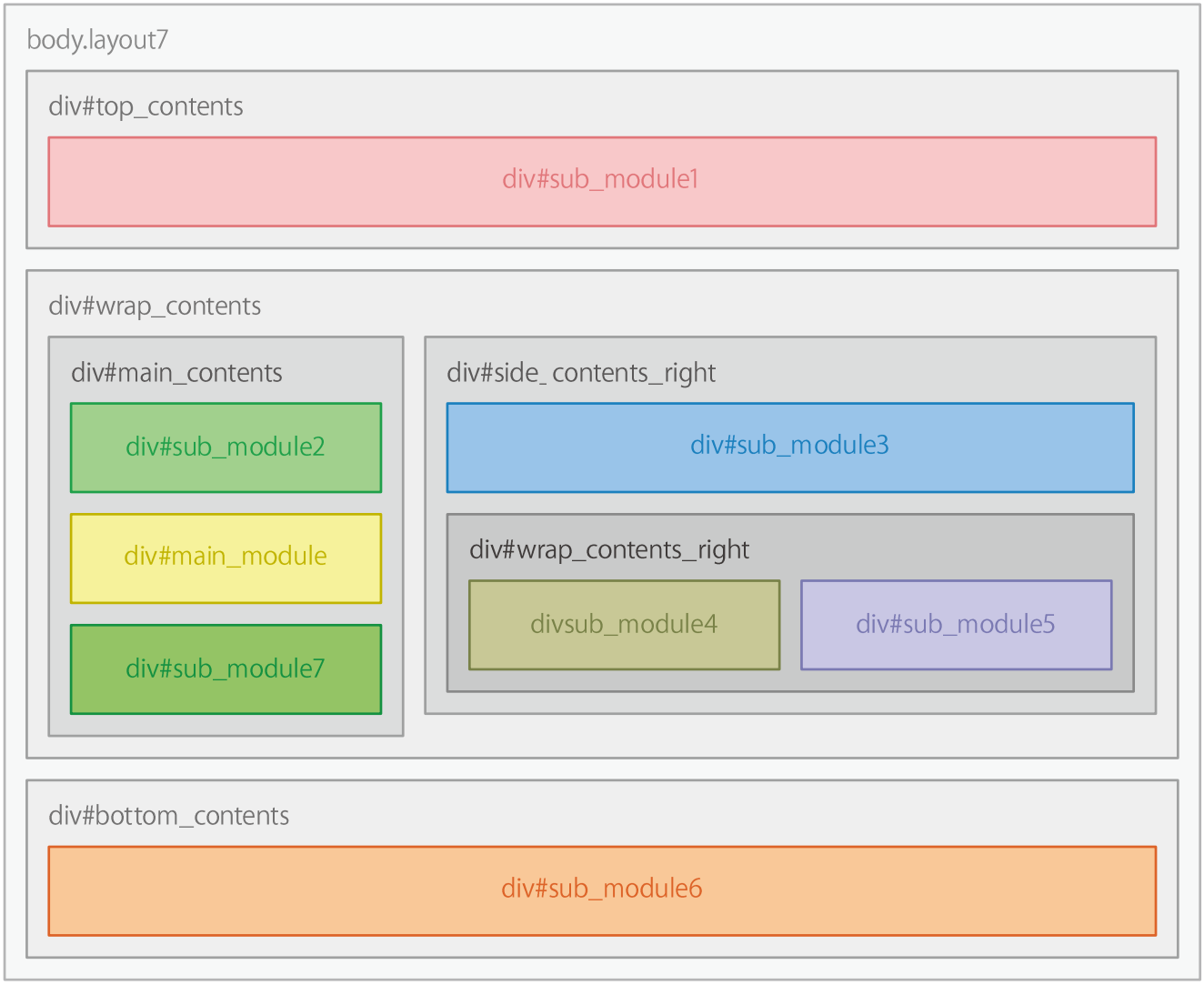
8-5 逆 L 左メイン

メインのコンテンツを左側に配置した、左側段組下部に 2 段組のレイアウトを内包した変則 2 段組のレイアウトです。右側段組の幅と、余白を指定出来ます。
左側段組の幅は［ウェブサイトの幅－右側段組の幅－余白］となります。



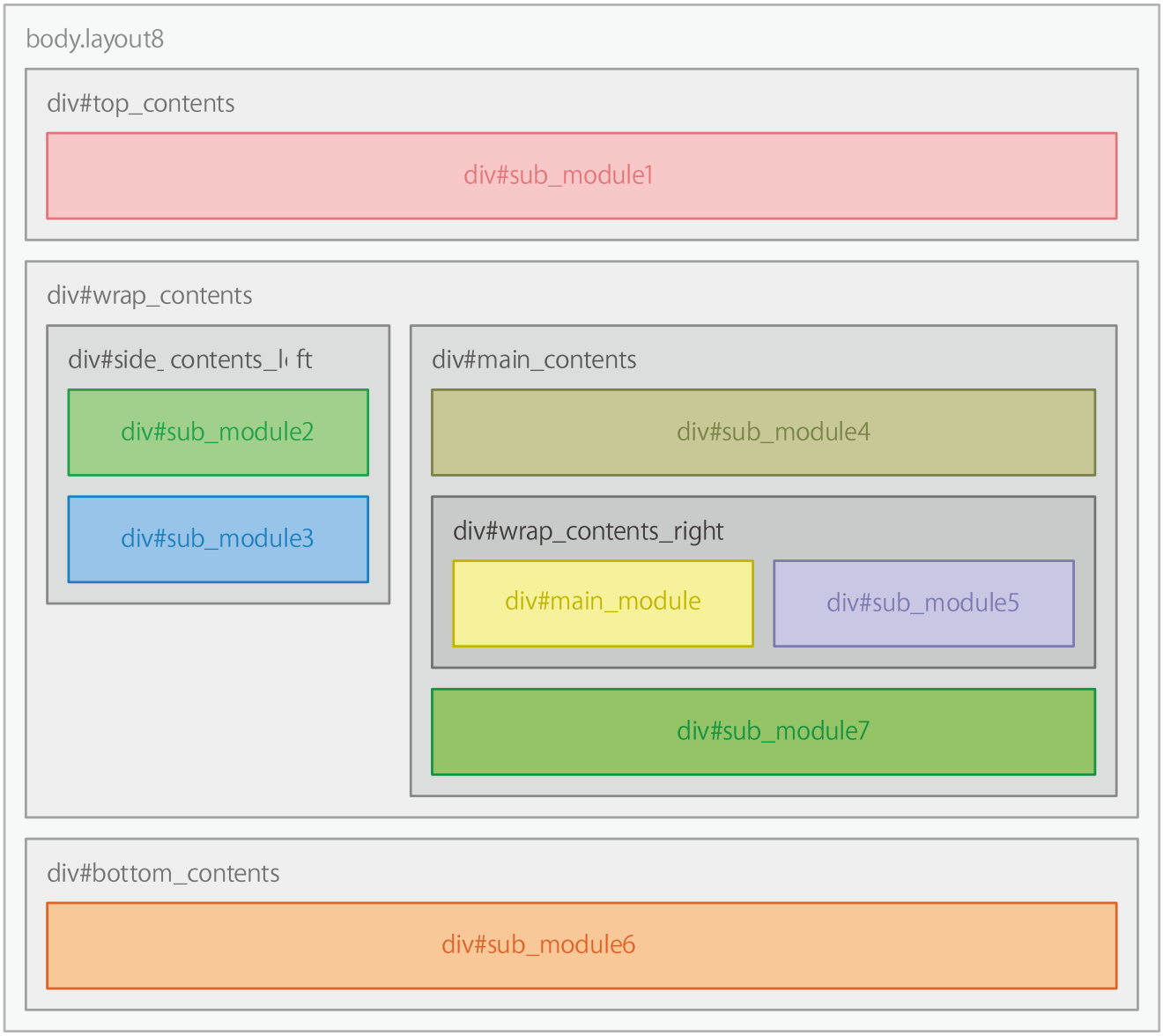
8-6 3 段組左メイン

メインのコンテンツを左側に配置した 3 段組のレイアウトです。右側段組の幅と、余白を指定出来ます。左側段組の幅は［ウェブサイトの幅－右側段組の幅－余白］となります。



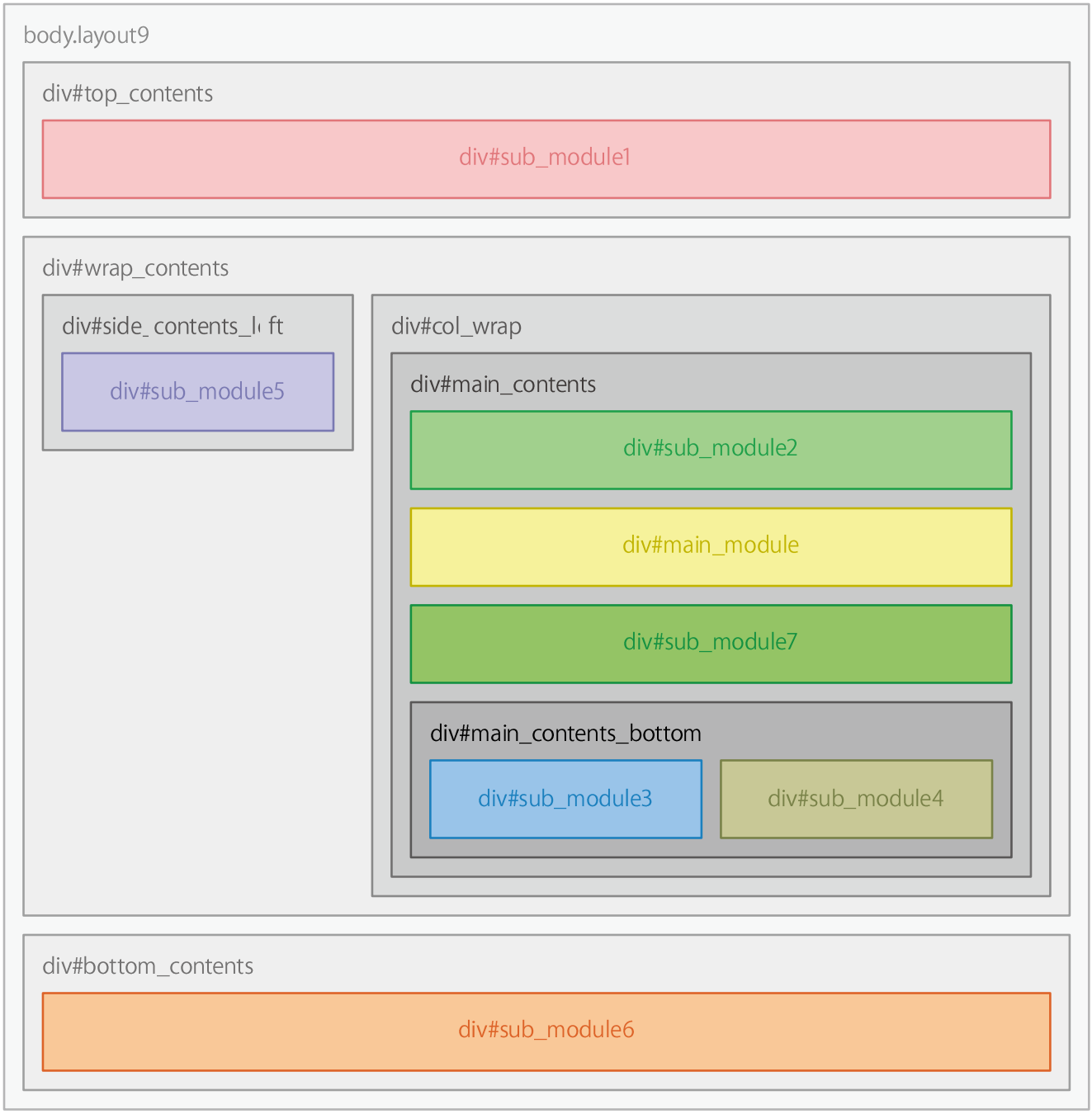
8-7 2 段組右メイン 2

メインのコンテンツを右側に配置した、右側段組中心部に 2 段組のレイアウトを内包した変則 2 段組のレイアウトです。左側段組の幅と、余白を指定出来ます。
右側段組の幅は［ウェブサイトの幅－左側段組の幅－余白］となります。



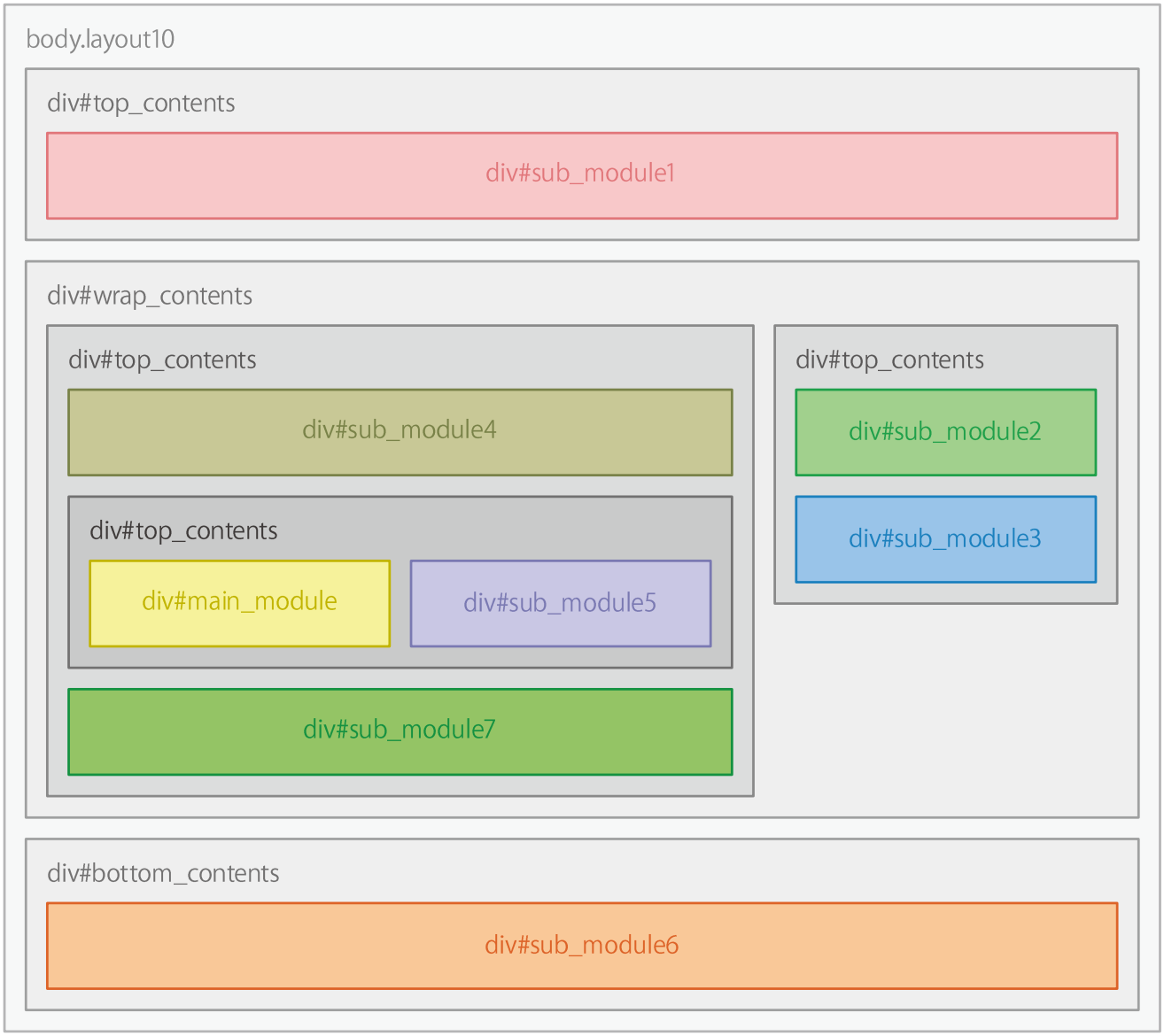
8-8 逆 L 右メイン

メインのコンテンツを右側に配置した、右側段組下部に 2 段組のレイアウトを内包した変則 2 段組のレイアウトです。左側段組の幅と、余白を指定出来ます。
右側段組の幅は［ウェブサイトの幅－左側段組の幅－余白］となります。



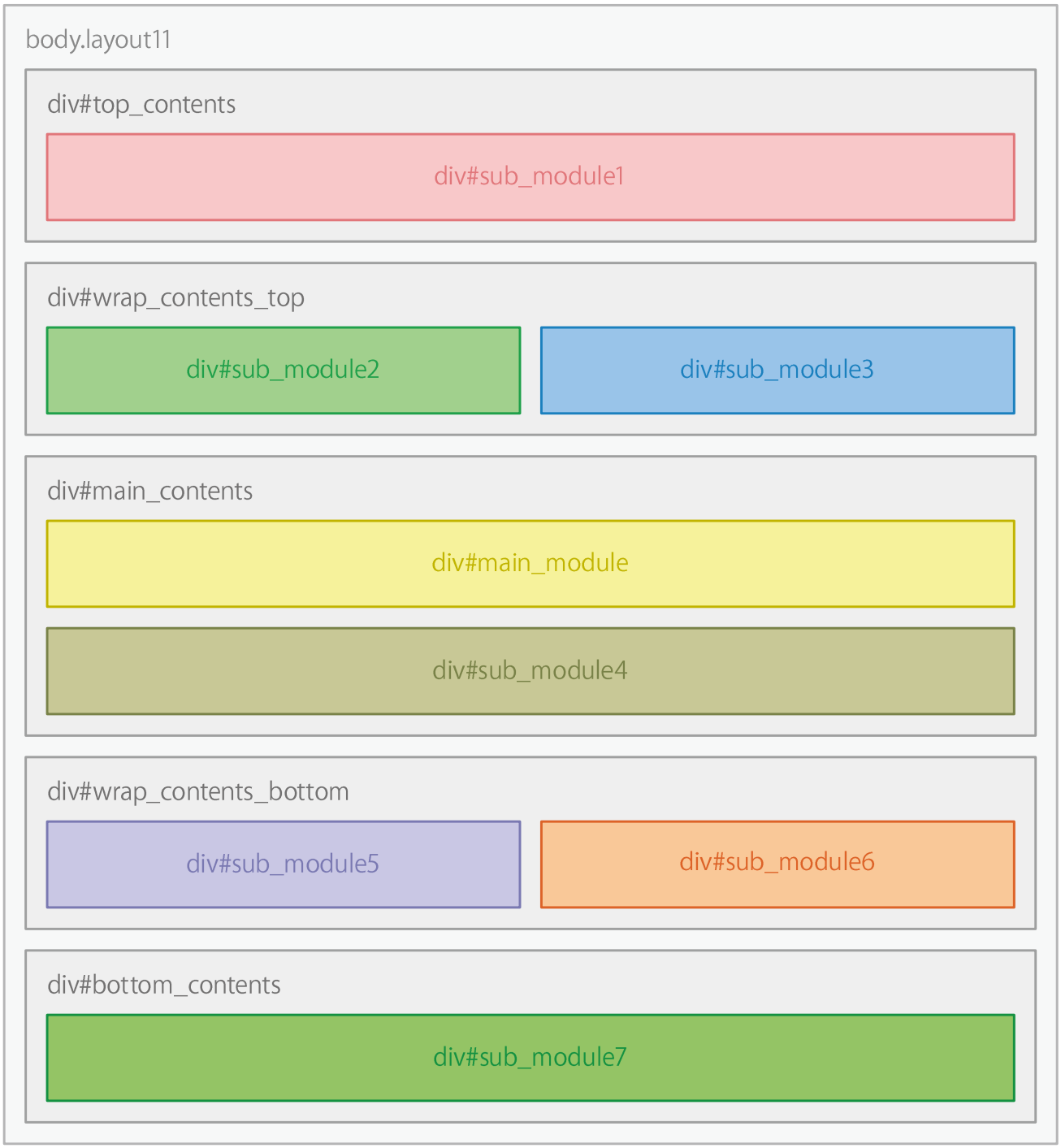
8-9 2 段組左メイン 2

メインのコンテンツを左側に配置した、左側段組中心部に 2 段組のレイアウトを内包した変則 2 段組のレイアウトです。右側段組の幅と、余白を指定出来ます。
左側段組の幅は［ウェブサイトの幅－右側段組の幅－余白］となります。



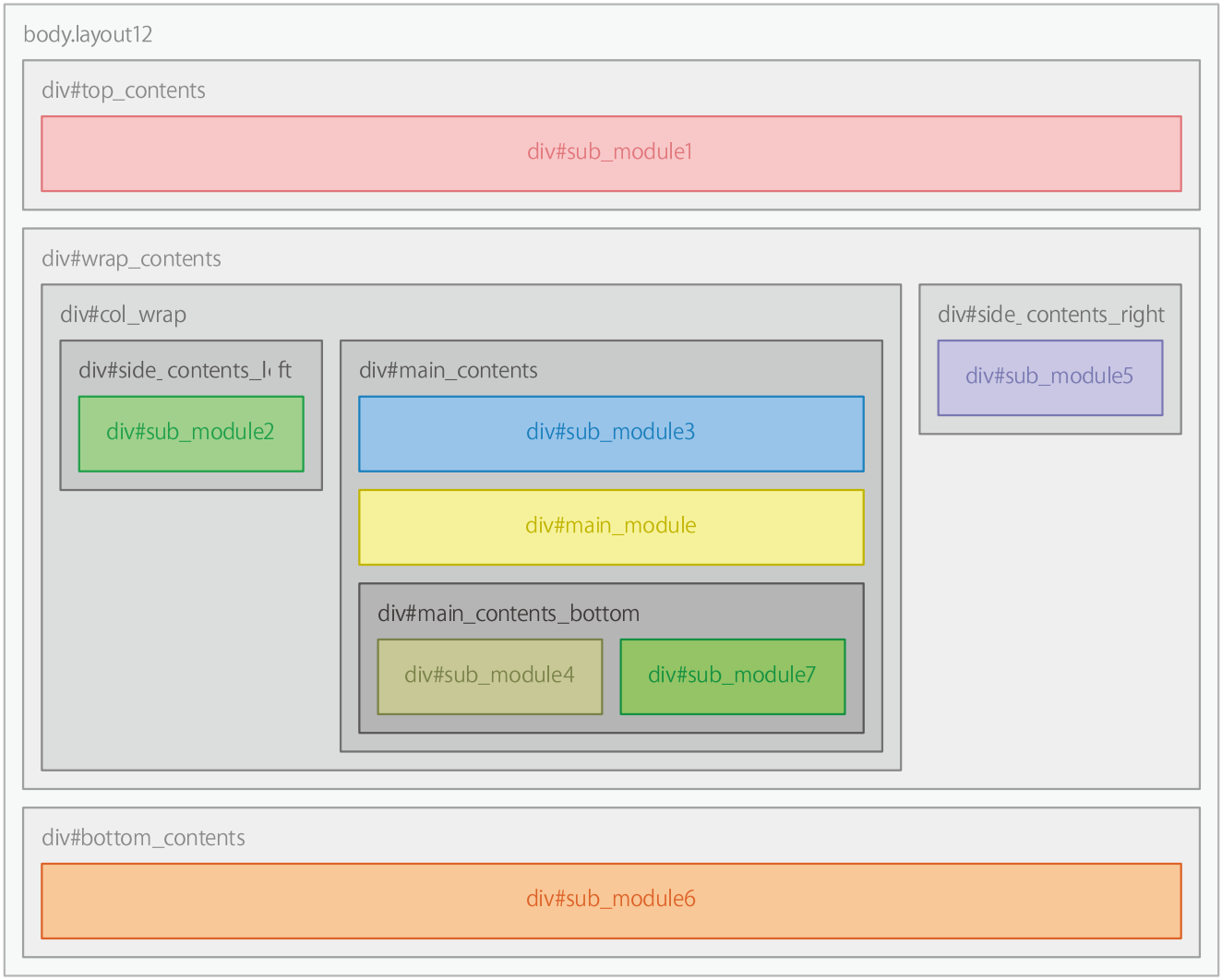
8-102 段組中メイン

メインのコンテンツを中心に配置した 2 段組のレイアウトです。左右の段組はそれぞれ半分の大きさが指定されるため、段組の幅や、余白を指定することは出来ません。



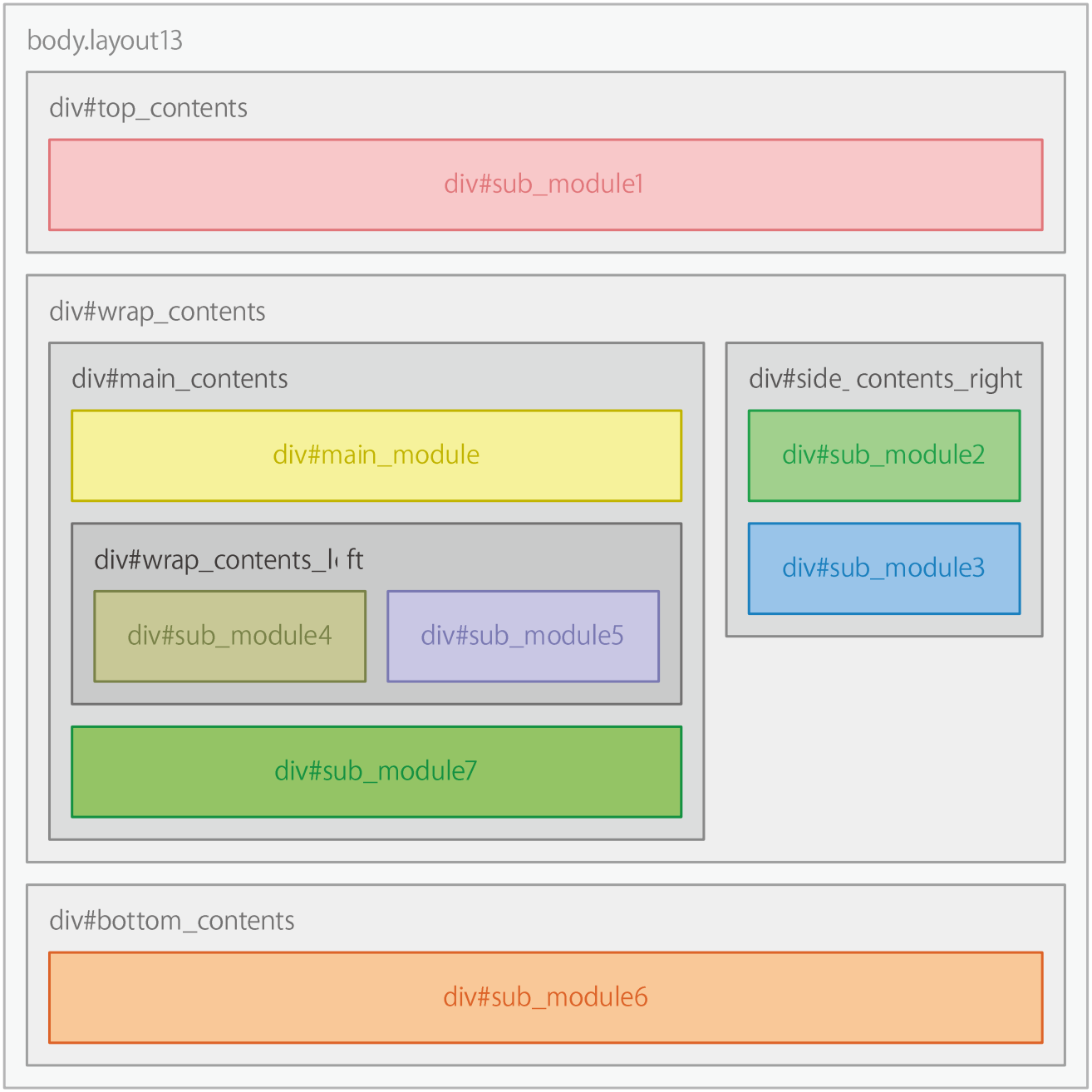
8-113 段組中メイン 2

メインのコンテンツを左側に配置した、真ん中段組下部に 2 段組のレイアウトを内包した変則 3 段組のレイアウトです。左右段組の幅と、余白を指定出来ます。
真ん中段組の幅は［ウェブサイトの幅－左右段組の幅－余白］となります。



8-122 段組左メイン 3

メインのコンテンツを左側に配置した、左側段落中心部に 2 段組のレイアウトを内包した変則 2 段組のレイアウトです。右側段組の幅と、余白を指定出来ます。
左側段組の幅は［ウェブサイトの幅－右側段組の幅－余白］となります。



9 テンプレート

RCMS のテンプレートには Smarty を採用しています。サイトタイトルや記事データなどの動的なデータを Smarty によってテンプレートへ組み込み、テンプレート CSS によって見た目の装飾を定義しています。

9-1 Smarty とは

Smarty は PHP のテンプレートエンジンです。PHP のプログラムとデザインを分離するために使われています。これにより RCMS で提供している様々なコンテンツを作るプログラムを理解せずともデザインをカスタマイズ出来るようになっています。

9-1-1 テンプレート変数

RCMS から提供するデータは全てテンプレート変数に格納されています。テンプレート変数の名称と内容は表示するコンテンツに依存しており、それぞれ文字列か配列が格納されています。
文字列が格納されているテンプレート変数へは{$arg}で、配列が格納されているテンプレート変数へは{$args.arg}でアクセスが可能です。
Smarty は色々な種類の変数を持っています。変数の種類は接頭辞の記号によって決まります (記号によって囲まれる場合もあります)。
Smarty 変数は、その値を直接表示したり 関数 の引数や 属性、 修飾子、 そして条件式の内部などで使用されたりします。 変数の値を表示するには、それを単純に デリミタ で囲み、デリミタ内に変数のみが含まれるようにします。

9-1-1-1 変数の例

```
{$Name}

{$product.part_no} <b>{$product.description}</b>

{$Contacts[row].Phone}

<body bgcolor="{#bgcolor#}">
```

PHP から 割り当てられた 変数は、(php と同様に) 先頭にドル記号 ($) をつける事で参照できます。 テンプレート内で {assign} 関数を用いて割り当てられた変数もこの方法で表示されます。

9-1-1-2 割り当てられた変数

php script

```
<?php

$smarty = new Smarty();

$smarty->assign('firstname', 'Doug');
$smarty->assign('lastname', 'Evans');
```

```
$smarty->assign('meetingPlace', 'New York');

$smarty->display('index.tpl');

?>
```

一方、index.tpl の内容はこのようになります。

```
Hello {$firstname} {$lastname}, glad to see you can make it.
<br />
{* これは動作しません。変数名は大文字小文字を区別するからです。 *}
This weeks meeting is in {$meetingplace}.
{* こちらは動作します *}
This weeks meeting is in {$meetingPlace}.
```

出力は次のようになります。

```
Hello Doug Evans, glad to see you can make it.
<br />
This weeks meeting is in .
This weeks meeting is in New York.
```

9-1-1-3 連想配列

PHP から割り当てられた連想配列を参照することもできます。 この場合は、'.'(ピリオド) 記号の後にキーを指定します。

9-1-1-3-1 連想配列の値にアクセスする

```
<?php
$smarty->assign('Contacts',
    array('fax' => '555-222-9876',
          'email' => 'zaphod@slartibartfast.example.com',
          'phone' => array('home' => '555-444-3333',
                           'cell' => '555-111-1234')
                           )
        );
$smarty->display('index.tpl');
?>
```

一方、index.tpl の内容はこのようになります。

```
{$Contacts.fax}<br />
{$Contacts.email}<br />
{* you can print arrays of arrays as well *}
{$Contacts.phone.home}<br />
{$Contacts.phone.cell}<br />
```

出力は次のようになります。

```
555-222-9876<br />
zaphod@slartibartfast.example.com<br />
```

```
555-444-3333<br />
555-111-1234<br />
```

9-1-1-4 配列のインデックス

配列に対してインデックスでアクセスすることもできます。 これは PHP 本来の構文と同じです。

9-1-1-4-1 インデックスによって配列にアクセスする

```
<?php
$smarty->assign('Contacts', array(
                        '555-222-9876',
                        'zaphod@slartibartfast.example.com',
                         array('555-444-3333',
                               '555-111-1234')
                         ));
$smarty->display('index.tpl');
?>
```

一方、index.tpl の内容はこのようになります。

```
{$Contacts[0]}<br />
{$Contacts[1]}<br />
{* you can print arrays of arrays as well *}
{$Contacts[2][0]}<br />
{$Contacts[2][1]}<br />
```

出力は次のようになります。

```
555-222-9876<br />
zaphod@slartibartfast.example.com<br />
555-444-3333<br />
555-111-1234<br />
```

9-1-1-5 オブジェクト

PHP から割り当てられた オブジェクト のプロパティにアクセスするには、-> 記号の後にプロパティ名を指定します。

9-1-1-5-1 オブジェクトのプロパティにアクセスする

```
name:  {$person->name}<br />
email: {$person->email}<br />
```

出力は次のようになります。

```
name:  Zaphod Beeblebrox<br />
email: zaphod@slartibartfast.example.com<br />
```

9-1-2　関数

Smarty にはいくつかの組み込み関数があります。 これらはテンプレートエンジンにとって必要不可欠なものです。これらと同じ名前の カスタム関数 を作成したり、組み込み関数を修正したりする事はできません。

これらの関数の一部は assign 属性を持っており、 結果を出力せずにここで指定した名前のテンプレート変数に格納します。これは {assign} 関数と似ています。

{capture} は、タグの間のテンプレートの出力を集め、 それをブラウザに表示する代わりに変数に受け渡します。 {capture name='foo'} と {/capture} の間のあらゆるコンテンツは、name 属性で指定した変数に格納されます。

キャプチャされたコンテンツは、特別な変数 $smarty.capture.foo ("foo" は name 属性で指定した変数) によって利用できます。 name 属性を指定しない場合は "default" が使われ、$smarty.capture.default のようになります。

{capture}'s はネスト可能です。

属性名	型	必須	デフォルト	概要
name	string	no	*default*	キャプチャされるブロックの名前
assign	string	No	*n/a*	キャプチャされた出力を割り当てるための変数名

注意: {insert} の出力をキャプチャする際には注意が必要です。 $caching が有効の時に、実行したい {insert} コマンドがもしキャッシュされたコンテンツ内にあるのなら、そのコンテンツはキャプチャされません。

9-1-2-1 name 属性を使用した {capture}

```
{* コンテンツが表示されない限り、テーブルの行を表示しません *}
{capture name=banner}
  {include file='get_banner.tpl'}
{/capture}

{if $smarty.capture.banner ne ''}
<div id="banner">{$smarty.capture.banner}</div>
{/if}
```

9-1-2-2 {capture} をテンプレート変数に格納

この例は、 {popup} 関数の使用法を示すものです。

```
{capture name=some_content assign=popText}
The server is {$smarty.server.SERVER_NAME|upper} at
{$smarty.server.SERVER_ADDR}<br>
Your ip is {$smarty.server.REMOTE_ADDR}.
{/capture}
<a href="#" {popup caption='Server Info' text=$popText}>help</a>
```

$smarty.capture、 {eval}、 {fetch}、 fetch() および {assign} も参照してください。

9-1-2-3 {foreach}, {foreachelse}

{foreach} を使用して、通常の数値添字配列と同じように 連想配列 をループします。{section} のように、数値添字の配列のみ をループさせるということはありません。{foreach} の構文は {section} よりずっと簡単ですが、その代わりに 1 つの配列 しか扱えません。すべての {foreach} タグは、 終了タグ {/foreach} とペアである必要があります。
Smarty の様々な機能を利用するための関数がいくつか利用可能です。

属性名	型	必須	デフォルト	概要
from	array	Yes	n/a	ループに使用する配列
item	string	Yes	n/a	現在の要素を示す変数の名前
key	string	No	n/a	現在のキーを示す変数の名前
name	string	No	n/a	foreach プロパティにアクセスするための foreach ループ名

- 必須の属性は from と item です。
- {foreach} ループの name は、英数字とアンダースコアを使用して自由に命名できます。これは PHP の変数 と同じです。
- {foreach} ループはネスト可能で、ネストした {foreach} の name はお互いにユニークである必要があります。
- from 属性は、通常は値の配列で、{foreach} のループ回数を決定するために使われます。
- {foreachelse} は、 from 変数の値が存在しない場合に実行されます。
- {foreach} ループは、プロパティを操作する変数を自身で持っています。 これらは次のように表されます。 {$smarty.foreach.name.property} ここで、“name” は name 属性の値となります。
- {foreach} のプロパティには index、 iteration、 first、 last、 show、 total があります。

注意

name 属性が必要となるのは {foreach} のプロパティにアクセスする必要がある場合のみです。 これは {section} の場合とは異なります。{foreach} のプロパティに対して 定義されていない name でアクセスしてもエラーは発生しませんが、 結果は予測できない値になります。

- {foreach}, {foreachelse}, {if}, {elseif}, {else}, {ldelim}, {rdelim}, {literal}, {section}, {sectionelse}など、関数の仕様については Smarty マニュアルをご参照下さい。

9-1-2-3-1 Item 属性

```
<?php
$arr = array(1000, 1001, 1002);
$smarty->assign('myArray', $arr);
?>
```

$myArray を順序なしリストで出力するテンプレート

```
<ul>
{foreach from=$myArray item=foo}
    <li>{$foo}</li>
```

```
{/foreach}
</ul>
```

出力

```
<ul>
    <li>1000</li>
    <li>1001</li>
    <li>1002</li>
</ul>
```

9-1-2-3-2item および key 属性の説明

```
<?php
$arr = array(9 => 'Tennis', 3 => 'Swimming', 8 => 'Coding');
$smarty->assign('myArray', $arr);
?>
```

$myArray を キー/値 のペアで出力するテンプレート。PHP の foreach と似ています。

```
<ul>
{foreach from=$myArray key=k item=v}
   <li>{$k}: {$v}</li>
{/foreach}
</ul>
```

出力

```
<ul>
    <li>9: Tennis</li>
    <li>3: Swimming</li>
    <li>8: Coding</li>
</ul>
```

9-1-2-3-3{foreach} で連想配列の item 属性を指定する例

```
<?php
$items_list = array(23 => array('no' => 2456, 'label' => 'Salad'),
                    96 => array('no' => 4889, 'label' => 'Cream')
                    );
$smarty->assign('items', $items_list);
?>
```

$items と $myId を url に出力するテンプレート

```
<ul>
{foreach from=$items key=myId item=i}
  <li><a href="item.php?id={$myId}">{$i.no}: {$i.label}</li>
{/foreach}
</ul>
```

出力

```
<ul>
```

```
  <li><a href="item.php?id=23"&#62;2456: Salad&#60;/li&#62;
  &#60;li&#62;&#60;a href="item.php?id=96">4889: Cream</li>
</ul>
```

9-1-2-3-4 {foreach} で item と key をネストする例

配列を Smarty に割り当てます。key にはループする値のキーが含まれます。

```
<?php
 $smarty->assign('contacts', array(
                         array('phone' => '1',
                               'fax' => '2',
                               'cell' => '3'),
                         array('phone' => '555-4444',
                               'fax' => '555-3333',
                               'cell' => '760-1234')
                         ));
?>
```

$contact を出力するテンプレート

```
{foreach name=outer item=contact from=$contacts}
  <hr />
  {foreach key=key item=item from=$contact}
    {$key}: {$item}<br />
  {/foreach}
{/foreach}
```

出力

```
<hr />
  phone: 1<br />
  fax: 2<br />
  cell: 3<br />
<hr />
  phone: 555-4444<br />
  fax: 555-3333<br />
  cell: 760-1234<br />
```

9-1-2-3-5データベースを使用する {foreachelse} の例

データベース (PEAR や ADODB など) を検索する例で、クエリの結果を Smarty に割り当てます。

```
<?php
  $search_condition = "where name like '$foo%' ";
  $sql = 'select contact_id, name, nick from contacts
'.$search_condition.' order by name';
  $smarty->assign('results', $db->getAssoc($sql) );
?>
```

結果がない場合に、{foreachelse} を使用して "見つかりません" と表示するテンプレート

```
{foreach key=cid item=con from=$results}
    <a href="contact.php?contact_id={$cid}">{$con.name} -
{$con.nick}</a><br />
{foreachelse}
    検索結果が見つかりませんでした
{/foreach}
```

9-1-2-3-6. index

index には、現在の配列のインデックスをゼロから数えた値が含まれます。

9-1-2-3-6-1 index の例

```
{* ヘッダブロックを５行おきに出力します *}
<table>
{foreach from=$items key=myId item=i name=foo}
  {if $smarty.foreach.foo.index % 5 == 0}
     <tr><th>タイトル</th></tr>
  {/if}
  <tr><td>{$i.label}</td></tr>
{/foreach}
</table>
```

9-1-2-3-7. iteration

iteration は現在のループが反復された回数を表示します。 index とは異なり、常に 1 から始まります。 各ループごとに 1 ずつ加算されます。

9-1-2-3-7-1 iteration および index の例

```
{* この出力は 0|1, 1|2, 2|3, ... のようになります *}
{foreach from=$myArray item=i name=foo}
{$smarty.foreach.foo.index}|{$smarty.foreach.foo.iteration},
{/foreach}
```

9-1-2-3-8. first

first は、現在の {foreach} の反復が最初のものであるときに TRUE となります。

9-1-2-3-8-1 first プロパティの例

```
{* 最初の項目には「最新」、それ以外は id を表示します *}
<table>
{foreach from=$items key=myId item=i name=foo}
<tr>
```

```
  <td>{if $smarty.foreach.foo.first}最新{else}{$myId}{/if}</td>
  <td>{$i.label}</td>
</tr>
{/foreach}
</table>
```

9-1-2-3-9. last

last は、現在の {foreach} の反復が最後のものであるときに TRUE となります。

9-1-2-3-9-1 last プロパティの例

```
{* 一覧の最後に横罫線を追加します *}
{foreach from=$items key=part_id item=prod name=products}
  <a href="#{$part_id}">{$prod}</a>{if
$smarty.foreach.products.last}<hr>{else},{/if}
{foreachelse}
  ... コンテンツ ...
{/foreach}
```

9-1-2-3-10. show

show は {foreach} のパラメータとして使用します。 show は boolean 値です。 FALSE の場合は {foreach} は表示されず、 もし {foreachelse} が存在すれば、それが代わりに表示されます。

9-1-2-3-11. total

total には、 {foreach} がループするトータル回数が含まれます。 これは、{foreach} の内部だけではなく ループを抜けた後でも使用できます。

9-1-2-3-11-1total プロパティの例

```
{* 返された行の総数を最後に表示します *}
{foreach from=$items key=part_id item=prod name=foo}
{$prod.name><hr/>
{if $smarty.foreach.foo.last}
  <div id="total">{$smarty.foreach.foo.total} items</div>
{/if}
{foreachelse}
 ... 別の内容 ...
{/foreach}
```

{section} および $smarty.foreach も参照してください。

9-1-2-4 {if}, {elseif}, {else}

Smarty における {if} ステートメントは、PHP の if と同等の柔軟性を持っています。 さらに、テンプレートエンジンのための機能をいくつか追加しています。 全ての {if} は、対応

する {/if} とペアである必要があります。{else} と {elseif} も使用できます。|| や or、&&、and、 is_array() など、PHP の条件演算子や関数はすべて利用可能です。

$security が有効な場合は、 $security_settings の配列 IF_FUNCS に含まれる PHP の関数のみが利用可能となります。

以下は認識される条件演算子の一覧です。 これらはスペースによって周りの要素から分離される必要があります。 [] 内に記載された項目は任意である事に注意して下さい。 "PHP 相当" には、PHP において当てはまるものが示されます。

条件演算子	代替	構文例	意味	PHP 相当
==	eq	$a eq $b	等しい	==
!=	ne, neq	$a neq $b	等しくない	!=
>	gt	$a gt $b	より大きい	>
<	lt	$a lt $b	より小さい	<
>=	gte, ge	$a ge $b	以上	>=
<=	lte, le	$a le $b	以下	<=
===		$a === 0	同一性のチェック	===
!	not	not $a	否定（単項）	!
%	mod	$a mod $b	剰余	%
is [not] div by		$a is not div by 4	割り切れる	$a % $b == 0
is [not] even		$a is not even	偶数である［ない］（単項）	$a % 2 == 0
is [not] even by		$a is not even by $b	偶数番目のグループである［ない］	($a / $b) % 2 == 0
is [not] odd		$a is not odd	奇数である［ない］（単項）	$a % 2 != 0
is [not] odd by		$a is not odd by $b	奇数番目のグループである［ない］	($a / $b) % 2 != 0

9-1-2-4-1 {if} ステートメント

```
{if $name eq 'Fred'}
   Welcome Sir.
{elseif $name eq 'Wilma'}
   Welcome Ma'am.
{else}
   Welcome, whatever you are.
{/if}
```

```
{* 論理演算子 "or" の例 *}
{if $name eq 'Fred' or $name eq 'Wilma'}
   ...
{/if}

{* 上と同じ *}
{if $name == 'Fred' || $name == 'Wilma'}
   ...
{/if}



{* 括弧は使用可能 *}
{if ( $amount < 0 or $amount > 1000 ) and $volume >= #minVolAmt#}
   ...
{/if}



{* PHP 関数を埋め込むことも可能 *}
{if count($var) gt 0}
   ...
{/if}

{* 配列のチェック *}
{if is_array($foo) }
   .....
{/if}

{* null でないことのチェック *}
{if isset($foo) }
   .....
{/if}



{* 値が偶数か奇数か *}
{if $var is even}
   ...
{/if}
{if $var is odd}
   ...
{/if}
{if $var is not odd}
   ...
```

```
{/if}



{* 値が 4 で割り切れるかどうか *}
{if $var is div by 4}
   ...
{/if}



{*
  ふたつずつグループ化したときに、値が even であるかどうか
  0=even, 1=even, 2=odd, 3=odd, 4=even, 5=even, etc.
*}
{if $var is even by 2}
   ...
{/if}

{* 0=even, 1=even, 2=even, 3=odd, 4=odd, 5=odd, etc. *}
{if $var is even by 3}
   ...
{/if}
```

9-1-2-4-2 {if} のその他の例

```
{if isset($name) && $name == 'Blog'}
    {* 何かを行います *}
{elseif $name == $foo}
    {* 何かを行います *}
{/if}

{if is_array($foo) && count($foo) > 0)
    {* foreach ループを実行します *}
{/if}
```

9-1-2-5 {include}

{include} タグを使用して、 現在のテンプレートに他のテンプレートをインクルードします。現在のテンプレートにて利用可能なあらゆる変数は、 インクルードされたテンプレートでも同じく利用可能です。

- {include} タグには、テンプレートリソースのパスを含んだ file 属性を必ず指定する必要があります。
- {include} の出力をブラウザに表示する代わりに変数に格納したい場合は、 オプションの assign 属性にその変数名を定義します。 {assign} と同等です。

- インクルードされたテンプレートに変数を渡すには、 attributes を使用します。インクルードされたテンプレートに明示的に渡された変数は、 インクルードされたファイルのスコープでのみ有効となります。 そのテンプレートに同じ名前の変数が存在する場合は、渡された変数がそれをオーバーライドします。
- 全ての割り当て変数の値は、インクルードされたテンプレートのスコープが閉じた後に元に戻ります。 これは、インクルードされたテンプレート内で全ての変数を使用可能であるということです。 しかし、インクルードされたテンプレート内での変数の変更は {include} の後でインクルードしている側のテンプレート内では見ることはできません。
- $template_dir ディレクトリ外にあるファイルを {include} するには、 テンプレートリソース を指定します。

属性名	型	必須	デフォルト	概要
file	string	Yes	*n/a*	インクルードするテンプレートファイル名
assign	string	No	*n/a*	インクルードしたコンテンツの出力を格納する変数名
[var ...]	[var type]	No	*n/a*	ローカルからテンプレートに渡す変数

9-1-2-5-1 シンプルな {include} の例

```
<html>
<head>
  <title>{$title}</title>
</head>
<body>
{include file='page_header.tpl'}

{* ここにテンプレートの本体を記述します。変数 $tpl_name
   はたとえば 'contact.tpl' などに置き換えられます。
*}
{include file="$tpl_name.tpl"}

{include file='page_footer.tpl'}
</body>
</html>
```

9-1-2-5-2 {include} に変数を渡す

```
{include file='links.tpl' title='Newest links' links=$link_array}
{* ここにテンプレートの本体を記述します *}
{include file='footer.tpl' foo='bar'}
```

このテンプレートは、以下のような links.tpl をインクルードします。

```
<div id="box">
<h3>{$title}{/h3>
```

```
<ul>
{foreach from=$links item=l}
.. 何かを行います  ...
</foreach}
</ul>
</div>
```

9-1-2-5-3 {include} と変数への割り当て

この例は、nav.tpl の内容を変数 $navbar に割り当て、 ページの最初と最後に出力させるものです。

```
<body>
  {include file='nav.tpl' assign=navbar}
  {include file='header.tpl' title='Smarty is cool'}
    {$navbar}
    {* テンプレートの本体をここへ記述します *}
    {$navbar}
  {include file='footer.tpl'}
</body>
```

9-1-2-5-4 さまざまな {include} リソースの例

```
{* ファイルの絶対パス *}
{include file='/usr/local/include/templates/header.tpl'}

{* ファイルの絶対パス (結果は上と同じ) *}
{include file='file:/usr/local/include/templates/header.tpl'}

{* Windows 環境のファイルの絶対パス (接頭辞の"file:"は必須) *}
{include file='file:C:/www/pub/templates/header.tpl'}

{* "db"と名付けられたテンプレートリソースからインクルード *}
{include file='db:header.tpl'}

{* 変数名に格納された名前のテンプレートをインクルード - 例 $module =
'contacts' *}
{include file="$module.tpl"}

{* この例は、シングルクォートでは変数が展開されないため、動作しません *}
{include file='$module.tpl'}

{* 複数の可変テンプレートをインクルード - 例 amber/links.view.tpl *}
{include file="$style_dir/$module.$view.tpl"}
```

{include_php}、 {insert}、 {php}、 テンプレートリソース および コンポーネント化したテンプレート も参照してください。

9-1-2-6 {include_php}

テクニカルノート: {include_php} は Smarty ではほとんど推奨されていません。 カスタムテンプレート関数を使用すれば、同等の機能を実現できます。 {include_php} を使用する理由がもしあるとすれば、 plugins/ ディレクトリやアプリケーションのコードから PHP 関数を完全に隔離したい場合などです。 詳細は コンポーネント化したテンプレートの例 を参照してください。

属性名	型	必須	デフォルト	概要
file	string	Yes	*n/a*	インクルードする PHP ファイル名
once	boolean	No	*TRUE*	同じ PHP ファイルが複数回インクルードされた場合に、一度だけインクルードするかどうか
assign	string	No	*n/a*	include_php の出力を格納する変数名

{include_php} タグを使用して、PHP スクリプトをテンプレートにインクルードします。 $security が有効な場合は、PHP スクリプトは $trusted_dir で指定されたディレクトリに存在する必要があります。{include_php} タグには file 属性が必須で、 ここにはインクルードする PHP ファイルへのパスを指定します。 このパスは $trusted_dir からの相対パスか絶対パスのいずれかとなります。

デフォルトでは、PHP ファイルはテンプレート内で複数回呼ばれても一度しかインクルードしません。 once 属性によって毎回インクルードするべきかどうかを指定できます。 この属性を FALSE に設定すると、テンプレート内でインクルードの指示がある毎に PHP スクリプトをインクルードします。

オプションで assign 属性を渡すこともできます。 これは、{include_php} の出力をブラウザに表示させる代わりに 変数に格納したい場合に、その変数名を指定します。

Smarty オブジェクトは、インクルードした PHP スクリプト内で $this として使用可能です。

9-1-2-6-1 {include_php} 関数

load_nav.php ファイル

```
<?php

// mysql データベースから変数の値を読み込み、それをテンプレートに割り当てます
require_once('database.class.php');
$db = new Db();
$db->query('select url, name from navigation order by name');
$this->assign('navigation', $db->getRows());
```

```
?>
```

テンプレート

```
{* 絶対パス、あるいは $trusted_dir からの相対パスか *}
{include_php file='/path/to/load_nav.php'}

{foreach item='nav' from=$navigation}
  <a href="{$nav.url}">{$nav.name}</a><br />
{/foreach}
```

{include}、 $security、 $trusted_dir、 {php}、{capture}、テンプレートリソース および コンポーネント化したテンプレート も参照してください。

9-1-2-7 {insert}

{insert} タグは {include} タグと似た動作をします。ただ {insert} タグは、テンプレートのキャッシュ が有効であってもキャッシュされません。テンプレートが呼び出されるたびに実行されます。

属性名	型	必須	デフォルト	概要
name	string	Yes	*n/a*	呼び出す insert 関数の名前 (insert_name)
assign	string	No	*n/a*	出力を格納するテンプレート変数名
script	string	No	*n/a*	insert 関数を呼び出す前にインクルードされる PHP スクリプト名
[var ...]	[var type]	No	*n/a*	insert 関数に渡す変数

例えば、ページの上部にバナーを表示するテンプレートを持っているとします。 バナーには HTML, images, flash 等が混合して含まれます。 したがってここに静的リンクを用いる事はできないので、 バナーコンテンツをキャッシュの対象にしたくありません。 そのためには、あらかじめ設定ファイルから取得した #banner_location_id# と #site_id# の値を渡し、バナーコンテンツを表示するために {insert} タグを呼び出す必要があります。

9-1-2-7-1 {insert} 関数

```
{* バナーを取得する例 *}
{insert name="getBanner" lid=#banner_location_id# sid=#site_id#}
```

この例では、name 属性に "getBanner" を指定し、 パラメータに #banner_location_id# と #site_id# を渡しています。Smarty は PHP アプリケーション内の insert_getBanner() 関数を探し、第 1 パラメータとして #banner_location_id# と #site_id# の値を格納した連想配列を渡します。 アプリケーションにおける全ての {insert} 関数の名前は、 ネームスペースの衝突を避けるために "insert_" によって始まる必要があります。 insert_getBanner() 関数は、渡された値によって何らかの処理を行い、結果を返すべきです。 この結果はテンプレートの {insert} タグに置換されて表示されます。 この例では、Smarty は insert_getBanner(array("lid" => "12345","sid" => "67890")); という関数を呼び出し、返された結果が {insert} タグの位置に表示されます。

- assign 属性を指定すると、 {insert} タグの出力は ブラウザに表示される代わりにテンプレート変数に格納されます。

注意: 出力をテンプレート変数に格納するのは、 キャッシュ が有効な状態ではあまり有益ではありません。

- script 属性を与えると、この PHP スクリプトは {insert} 関数が実行される前に (一度だけ) インクルードされます。 これは、insert 関数がまだ存在しないかもしれない場合や、insert 関数の動作のために PHP スクリプトを最初にインクルードする必要がある場合に指定します。
  パスには、絶対パスかあるいは $trusted_dir からの相対パスを指定します。 $security が有効な場合は、スクリプトは $trusted_dir 内にある必要があります。 Smarty オブジェクトは第２パラメータとして渡されます。 これにより、{insert} 関数から Smarty オブジェクトの情報の参照や修正が可能です。
  テクニカルノート: テンプレートには、キャッシュの対象外となる部分を持たせる事が可能です。 キャッシュ が有効の場合でも、 {insert} タグによる出力はキャッシュされません。 そのページが呼び出される度に動的に実行されます。 この動作は、バナー・投票・天気予報・検索結果・ユーザーフィードバックエリア等に向いています。

{include} も参照してください。

9-1-2-8 {ldelim}, {rdelim}

{ldelim} および {rdelim} は、テンプレートのデリミタを エスケープ します。 デフォルトでは、これは { および } となります。 Javascript や CSS のようなテキストのあつまりをエスケープするためには {literal}{/literal} を使用することもできます。{$smarty.ldelim} も参照してください。

9-1-2-8-1 {ldelim}, {rdelim}

```
{* これは、テンプレートからデリミタのリテラルを出力します *}

{ldelim}funcname{rdelim} is how functions look in Smarty!
```

上の例の出力

```
{funcname} is how functions look in Smarty!
```

Javascript を使用する別の例

```
<script language="JavaScript">
function foo() {ldelim}
    ... コード ...
{rdelim}
</script>
```

出力

```
<script language="JavaScript">
function foo() {
    .... コード ...
}
```

```
</script>
```

9-1-2-8-2別の Javascript の例

```
<script language="JavaScript" type="text/javascript">
    function myJsFunction(){ldelim}
        alert("The server
name¥n{$smarty.server.SERVER_NAME}¥n{$smarty.server.SERVER_ADDR}"
);
    {rdelim}
</script>
<a href="javascript:myJsFunction()">Click here for Server Info</a>
```

{literal} および Smarty の構文解析を回避 も参照してください。

9-1-2-9 {literal}

{literal} タグに囲まれたデータのブロックは、 リテラルとして認識されます。これは一般的に、Javascript やスタイルシートなどで 中括弧がテンプレートの デリミタ として解釈されるとまずい場合に使用します。 {literal}{/literal} タブの内部は解釈されず、 そのままで表示されます。{literal} ブロック内にテンプレートタグを含める必要がある場合は、代わりに {ldelim}{rdelim} で個々のデリミタをエスケープしてください。

9-1-2-9-1 {literal} タグ

```
{literal}
<script type="text/javascript">
<!--
  function isblank(field) {
    if (field.value == '')
      { return false; }
    else
      {
      document.loginform.submit();
      return true;
    }
  }
// -->
</script>
{/literal}
```

9-1-2-9-2 Javascript の関数の例

```
<script language="JavaScript" type="text/javascript">
{literal}
function myJsFunction(name, ip){
   alert("The server name¥n" + name + "¥n" + ip);
```

```
}
{/literal}
</script>
<a
href="javascript:myJsFunction('{$smarty.server.SERVER_NAME}','{$s
marty.server.SERVER_ADDR}')">Click here for the Server Info</a>
```

9-1-2-9-3テンプレート内での css style

```
{* included this style .. as an experiment *}
<style type="text/css">
{literal}
/* this is an intersting idea for this section */
.madIdea{
    border: 3px outset #ffffff;
    margin: 2 3 4 5px;
    background-color: #001122;
}
{/literal}
</style>
<div class="madIdea">With smarty you can embed CSS in the
template</div>
```

{ldelim}{rdelim} および Smarty の構文解析を回避 のページも参照してください。

9-1-2-10 {php}

{php} タグで、PHP コードを直接テンプレートに埋め込むことができます。 $php_handling の設定にかかわらず、これはエスケープされません。 このタグは上級ユーザのためのものなので通常は必要とされません。

テクニカルノート: {php} ブロック内の PHP 変数にアクセスするには、PHP の global キーワードを使う必要があります。

9-1-2-10-1 {php} タグ内での PHP コード

```
{php}
  // PHP スクリプトをテンプレートから直接インクルードします
  include('/path/to/display_weather.php');
{/php}
```

9-1-2-10-2 {php} タグで global を使用して変数を代入する

```
{* このテンプレートは {php} ブロックを含み、その中で変数 $varX を割り当てます
*}
{php}
  global $foo, $bar;
  if($foo == $bar){
```

```
    echo 'This will be sent to browser';
  }
  // 変数を Smarty に割り当てます
  $this->assign('varX','Toffee');
{/php}
{* 変数を出力します *}
<strong>{$varX}</strong> is my fav ice cream :-)
```

$php_handling、{include_php}、{include}、{insert} および コンポーネント化したテンプレート も参照してください。

9-1-2-11 {section}, {sectionelse}

{section} は、 データの配列 をループするために使用します。 これは、{foreach} が 1 つの連想配列 をループするのとは異なります。すべての {section} タグは、終了タグ {/section} とペアになっている必要があります。

属性名	型	必須	デフォルト	概要
name	string	Yes	*n/a*	セクション名
loop	mixed	Yes	*n/a*	ループ回数を決定する値
start	integer	No	*0*	ループを開始するインデックス位置。この値が負の場合は、 配列の最後尾から開始位置が算出されます。 例えばループ配列に 7 つの値があり、そして start が-2 であるならば、 開始インデックスは 5 になります。 ループ配列の長さを超えるような無効な値は、自動的に最も近い値に切り捨てられます。
step	integer	No	*1*	ループインデックスを進めるために使われるステップ値。 例えば step=2 なら、インデックスは 0, 2, 4 をループします。 step の値が負の場合は、配列の前方に向かって進みます。
max	integer	No	*n/a*	セクションがループする最大の回数
show	boolean	No	*TRUE*	このセクションを表示するかどうか

- 必須の属性は name と loop です。
- {section} の name は、 英数字とアンダースコアを使って自由に命名できます。これは PHP の変数 と同様です。
- {section} はネスト可能で、その場合の {section} の名前はお互いにユニークである必要があります。
- loop 属性で指定されたループ変数（たいていは配列）は、{section} のループ回数を決定するために使用されます。 loop の値として、整数値を渡すこともできます。
- {section} 内で値を表示するには、 変数名に続けてブラケット {} で囲んだセクション名を指定します。
- ループ変数に値が存在しない場合は {sectionelse} が実行されます。

- {section} には、そのプロパティを操作するための 自身の変数があります。これらには {$smarty.section.name.property} としてアクセスできます。"name" は、name 属性の値です。
- {section} のプロパティには、index、index_prev、index_next、iteration、first、last、 rownum、 loop、 show、 total があります。

9-1-2-11-1 {section} でのシンプルな配列のループ

配列を Smarty に assign() します。

```
<?php
$data = array(1000,1001,1002);
$smarty->assign('custid',$data);
?>
```

配列を出力するテンプレート

```
{* この例は $custid 配列のすべての値を表示します *}
{section name=customer loop=$custid}
  id: {$custid[customer]}<br />
{/section}
<hr />
{* $custid 配列のすべての値を逆順に表示します *}
{section name=foo loop=$custid step=-1}
  {$custid[foo]}<br />
{/section}
```

上の例の出力

```
id: 1000<br />
id: 1001<br />
id: 1002<br />
<hr />
id: 1002<br />
id: 1001<br />
id: 1000<br />
```

9-1-2-11-2 {section} で配列を割り当てない例

```
{section name=foo start=10 loop=20 step=2}
  {$smarty.section.foo.index}
{/section}
<hr />
{section name=bar loop=21 max=6 step=-2}
  {$smarty.section.bar.index}
{/section}
```

上の例の出力

```
10 12 14 16 18
<hr />
```

```
20 18 16 14 12 10
```

9-1-2-11-3 {section} の名前

{section} の name は自由につけることができます。PHP の変数 を参照してください。これは、{section} 内のデータを参照する際に使用します。

```
{section name=anything loop=$myArray}
  {$myArray[anything].foo}
  {$name[anything]}
  {$address[anything].bar}
{/section}
```

9-1-2-11-4 {section} での連想配列のループ

これは、データの連想配列を {section} で出力する例です。 次に示すのは、配列 $contacts を Smarty に渡す PHP スクリプトです。

```
<?php
$data = array(
        array('name' => 'John Smith', 'home' => '555-555-5555',
             'cell' => '666-555-5555', 'email' =>
'john@myexample.com'),
        array('name' => 'Jack Jones', 'home' => '777-555-5555',
             'cell' => '888-555-5555', 'email' =>
'jack@myexample.com'),
        array('name' => 'Jane Munson', 'home' => '000-555-5555',
             'cell' => '123456', 'email' => 'jane@myexample.com')
      );
$smarty->assign('contacts',$data);
?>
```

$contacts を出力するテンプレート

```
{section name=customer loop=$contacts}
<p>
  name: {$contacts[customer].name}<br />
  home: {$contacts[customer].home}<br />
  cell: {$contacts[customer].cell}<br />
  e-mail: {$contacts[customer].email}
</p>
{/section}
```

上の例の出力

```
<p>
  name: John Smith<br />
  home: 555-555-5555<br />
  cell: 666-555-5555<br />
  e-mail: john@myexample.com
```

```
</p>
<p>
  name: Jack Jones<br />
  home phone: 777-555-5555<br />
  cell phone: 888-555-5555<br />
  e-mail: jack@myexample.com
</p>
<p>
  name: Jane Munson<br />
  home phone: 000-555-5555<br />
  cell phone: 123456<br />
  e-mail: jane@myexample.com
</p>
```

9-1-2-11-5 {section} での loop 変数の使用

この例では、$custid、$name および $address にはすべて配列が割り当てられ、 その要素数は同じであるものとします。まず、Smarty に配列を割り当てる PHP スクリプトです。

```
<?php

$id = array(1001,1002,1003);
$smarty->assign('custid',$id);

$fullnames = array('John Smith','Jack Jones','Jane Munson');
$smarty->assign('name',$fullnames);

$addr = array('253 Abbey road', '417 Mulberry ln', '5605 apple st');
$smarty->assign('address',$addr);

?>
```

loop 変数は、ループの回数を決定するためにのみ使用します。 {section} 内ではあらゆるテンプレート変数にアクセス可能です。

```
{section name=customer loop=$custid}
<p>
  id: {$custid[customer]}<br />
  name: {$name[customer]}<br />
  address: {$address[customer]}
</p>
{/section}
```

上の例の出力

```
<p>
  id: 1000<br />
```

```
  name: John Smith<br />
  address: 253 Abbey road
</p>
<p>
  id: 1001<br />
  name: Jack Jones<br />
  address: 417 Mulberry ln
</p>
<p>
  id: 1002<br />
  name: Jane Munson<br />
  address: 5605 apple st
</p>
```

9-1-2-11-6 ネストした {section}

{section} は無制限にネスト可能です。{section} をネストすることで、 多次元配列のような複雑なデータ構造にアクセスすることが可能です。 これは、配列を割り当てる .php スクリプトの例です。

```
<?php

$id = array(1001,1002,1003);
$smarty->assign('custid',$id);

$fullnames = array('John Smith','Jack Jones','Jane Munson');
$smarty->assign('name',$fullnames);

$addr = array('253 N 45th', '417 Mulberry ln', '5605 apple st');
$smarty->assign('address',$addr);

$types = array(
         array( 'home phone', 'cell phone', 'e-mail'),
         array( 'home phone', 'web'),
         array( 'cell phone')
       );
$smarty->assign('contact_type', $types);

$info = array(
         array('555-555-5555', '666-555-5555',
'john@myexample.com'),
         array( '123-456-4', 'www.example.com'),
         array( '0457878')
      );
```

```
$smarty->assign('contact_info', $info);

?>
```

このテンプレートでは、$contact_type[customer] は現在の顧客の連絡手段を格納した配列となります。

```
{section name=customer loop=$custid}
<hr>
  id: {$custid[customer]}<br />
  name: {$name[customer]}<br />
  address: {$address[customer]}<br />
  {section name=contact loop=$contact_type[customer]}
    {$contact_type[customer][contact]}:
{$contact_info[customer][contact]}<br />
  {/section}
{/section}
```

上の例の出力。

```
<hr>
  id: 1000<br />
  name: John Smith<br />
  address: 253 N 45th<br />
    home phone: 555-555-5555<br />
    cell phone: 666-555-5555<br />
    e-mail: john@myexample.com<br />
<hr>
  id: 1001<br />
  name: Jack Jones<br />
  address: 417 Mulberry ln<br />
    home phone: 123-456-4<br />
    web: www.example.com<br />
<hr>
  id: 1002<br />
  name: Jane Munson<br />
  address: 5605 apple st<br />
    cell phone: 0457878<br />
```

9-1-2-11-7データベースを使用する {sectionelse} の例

データベース（ADODB や PEAR）の検索結果を Smarty に格納します。

```
<?php
$sql = 'select id, name, home, cell, email from contacts '
     ."where name like '$foo%' ";
$smarty->assign('contacts', $db->getAll($sql));
?>
```

データベースの結果を HTML のテーブルに出力するテンプレート

```
<table>
<tr><th> </th><th>Name></th><th>Home</th><th>Cell</th><th>Em
ail</th></tr>
{section name=co loop=$contacts}
  <tr>
    <td><a
href="view.php?id={$contacts[co].id}"&#62;view&#60;a&#62;&#60;/td
&#62;
    &#60;td&#62;{$contacts[co].name}&#60;/td&#62;
    &#60;td&#62;{$contacts[co].home}&#60;/td&#62;
    &#60;td&#62;{$contacts[co].cell}&#60;/td&#62;
    &#60;td&#62;{$contacts[co].email}&#60;/td&#62;
  &#60;tr&#62;
{sectionelse}
  &#60;tr&#62;&#60;td colspan="5">No items found</td></tr>
{/section}
</table>
```

9-1-2-11-8. index

index は現在のループインデックスを表示します。 0(又は start 属性の値)から始まり、1(又は step 属性の値)ずつ増加します。

> **テクニカルノート**: step と start 属性が変更されていない場合は、セクションのプロパティ iteration と同じ動作をします。ただ、1 ではなく 0 から始まるという点が異なります。

9-1-2-11-9 {section} の index プロパティ

ちなみに……: $custid[customer.index] と $custid[customer] は同じ意味です。

```
{section name=customer loop=$custid}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}
```

上の例の出力

```
0 id: 1000<br />
1 id: 1001<br />
2 id: 1002<br />
```

9-1-2-11-10. index_prev

index_prev は前回のループインデックスを表示します。最初のループでは-1 がセットされます。

9-1-2-11-11. index_next

index_next は次回のループインデックスを表示します。 ループの最後でもやはり現在のインデックスの次回の値を返します (step 属性の設定に従います)。

9-1-2-11-11-1 index、index_next および index_prev プロパティ

```
<?php
$data = array(1001,1002,1003,1004,1005);
$smarty->assign('rows',$data);
?>
```

上の配列をテーブルに出力するテンプレート

```
{* $rows[row.index] と $rows[row] は同じ意味です *}
<table>
  <tr>
    <th>index</th><th>id</th>
    <th>index_prev</th><th>prev_id</th>
    <th>index_next</th><th>next_id</th>
  </tr>
{section name=row loop=$rows}
  <tr>
    <td>{$smarty.section.row.index}</td><td>{$rows[row]}</td>

<td>{$smarty.section.row.index_prev}</td><td>{$rows[row.index_
prev]}</td>

<td>{$smarty.section.row.index_next}</td><td>{$rows[row.index_
next]}</td>
  </tr>
{/section}
</table>
```

上の例の出力するテーブルは次のようになります。

```
index  id    index_prev prev_id index_next next_id
0      1001  -1              1         1002
1      1002  0         1001    2         1003
2      1003  1         1002    3         1004
3      1004  2         1003    4         1005
4      1005  3         1004    5
```

9-1-2-11-12. iteration

iteration は現在のループが反復された回数を表示します。

注意: index プロパティとは異なり、これは {section} のプロパティ start、step および max の影響を受けません。 iteration も 1 から始まります。これは index が 0

から始まるのとは異なります。rownum は iteration の別名で、全く同じ働きをします。

9-1-2-11-12-1セクションのプロパティ iteration

```
<?php
// 3000 から 3015 までの配列
$id = range(3000,3015);
$smarty->assign('arr',$id);
?>
```

$arr 配列の要素を step=2 で出力するテンプレート

```
{section name=cu loop=$arr start=5 step=2}
  iteration={$smarty.section.cu.iteration}
  index={$smarty.section.cu.index}
  id={$custid[cu]}<br />
{/section}
```

上の例の出力

```
iteration=1 index=5 id=3005<br />
iteration=2 index=7 id=3007<br />
iteration=3 index=9 id=3009<br />
iteration=4 index=11 id=3011<br />
iteration=5 index=13 id=3013<br />
iteration=6 index=15 id=3015<br />
```

もうひとつの例は、iteration プロパティを使用して 5 行おきにテーブルのヘッダ部を出力します。 {if} 関数を mod 演算子とともに使用します。

```
<table>
{section name=co loop=$contacts}
  {if $smarty.section.co.iteration % 5 == 1}

<tr><th> </th><th>Name></th><th>Home</th><th>Cell</th><th
>Email</th></tr>
  {/if}
  <tr>
    <td><a href="view.php?id={$contacts[co].id}">view<a></td>
    <td>{$contacts[co].name}</td>
    <td>{$contacts[co].home}</td>
    <td>{$contacts[co].cell}</td>
    <td>{$contacts[co].email}</td>
  <tr>
{/section}
</table>
```

9-1-2-11-13. first

first は、現在 {section} の一回目の処理を行っている場合に TRUE となります。

9-1-2-11-14. last

last は、現在 {section} の最後の処理を行っている場合に TRUE となります。

9-1-2-11-14-1 {section} プロパティ first と last

この例は $customers 配列をループし、 ループの最初でヘッダブロック、そしてループの最後でフッタブロックを出力します。 total プロパティも使用します。

```
{section name=customer loop=$customers}
  {if $smarty.section.customer.first}
    <table>
    <tr><th>id</th><th>customer</th></tr>
  {/if}

  <tr>
    <td>{$customers[customer].id}</td>
    <td>{$customers[customer].name}</td>
  </tr>

  {if $smarty.section.customer.last}
    <tr><td></td><td>{$smarty.section.customer.total}
customers</td></tr>
    </table>
  {/if}
{/section}
```

9-1-2-11-15. rownum

rownum は現在のループが反復された回数を表示します(1 から開始)。 これは iteration の別名で、同じ動作をします。

9-1-2-11-16. loop

loop は、この {section} ループの最後のインデックス番号を表示します。 {section} の内部だけでなく、外部で使用することもできます。

9-1-2-11-16-1 {section} プロパティ loop

```
{section name=customer loop=$custid}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}
There are {$smarty.section.customer.loop} customers shown above.
```

上の例の出力

```
0 id: 1000<br />
1 id: 1001<br />
2 id: 1002<br />
There are 3 customers shown above.
```

9-1-2-11-17. show

show は、セクションのパラメータとして使用する boolean 値です。FALSE の場合はこのセクションは表示されません。 {sectionelse} があれば、それが代わりに表示されます。

9-1-2-11-17-1 show プロパティ

Boolean $show_customer_info を PHP アプリケーションから渡し、このセクションを表示するかどうかを調整します。

```
{section name=customer loop=$customers
show=$show_customer_info}
  {$smarty.section.customer.rownum} id:
{$customers[customer]}<br />
{/section}

{if $smarty.section.customer.show}
  the section was shown.
{else}
  the section was not shown.
{/if}
```

上の例の出力

```
1 id: 1000<br />
2 id: 1001<br />
3 id: 1002<br />

the section was shown.
```

9-1-2-11-18. total

total は {section} がループしたトータル回数を表示します。これは {section} の内部だけでなく外部でも使うことができます。

9-1-2-11-18-1 total プロパティの例

```
{section name=customer loop=$custid step=2}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}
   There are {$smarty.section.customer.total} customers shown
above.
```

{foreach} および $smarty.section も参照してください。

9-1-2-12 {strip}

Web デザイナーの方は、HTML コードに含まれたホワイトスペースとキャリッジリターンがブラウザの表示に影響を及ぼす問題に何度も遭遇した事があると思います。 問題を回避するには、テンプレートの全てのタグを連ねて記述する必要があります。 しかしこれでは大変読みづらく管理しにくいテンプレートになってしまいます。

{strip}{/strip} タグに囲まれたコンテンツは、 ブラウザに表示される前に、各行の先頭と終端にある 余分なホワイトスペースやキャリッジリターンが除去されます。これによってテンプレートは可読性を維持し、 余分なホワイトスペースによって問題を引き起こす心配もありません。

注意: {strip}{/strip} はテンプレート変数の内容に影響しません。 詳細は strip 修飾子 を参照してください。

9-1-2-12-1 {strip} タグ

```
{* 次の例は全て 1 行に出力されます  *}
{strip}
<table border='0'>
 <tr>
  <td>
   <a href="{$url}"&#62;
    &#60;font color="red">This is a test</font>
   </a>
  </td>
 </tr>
</table>
{/strip}
```

上の例の出力

```
<table border='0'><tr><td><a href="http://.
snipped...</a></td></tr></table>
```

上記の例は、全ての行が HTML タグで始まり HTML タグで終わる事に注意して下さい。 全ての行は連ねて出力されます。行の始めか終わりにプレーンテキストがある場合は、 連続して出力されたその結果は望むものではないかもしれません。

strip 修飾子も参照してください。