

# InDesign インライン 流し込みマニュアル

社外秘！　……というわけでもない

- グリッドとインラインをきわめて流し込みに強いフォーマットを作成する方法
- Markdown 原稿をブラウザプレビューしながら編集する方法
- XML を利用して画像＆スタイル付きで一気に流し込む方法

を解説

ひとことでいうと、ソースコード（色分けあり）と画面ショットだけの 1 段組みの本を、1 日で 300 ページ仮組できる術です。

サンプルファイルは以下からダウンロードできます。

<https://github.com/lwohtsu/NagashikomiManual>

[Download ZIP] をクリックしてください。

# 目次

## 第1章 流し込みやすいフォーマットを作る

1-1 インラインとグリッドを理解する	2
一段組みなら楽勝！というフォーマット作りを目指す	2
コラム：図解タイプは例外	3
InDesign のインライン機能を見直す	3
コラム：挿入しただけで意図しないスタイルが設定されてしまう？	6
グリッドとグリッド揃えを理解しよう	6
図版の上か下にキャプションを付ける	9
複数の画像をセットにするときは？	11
図を本文の横に回り込ませる	13
コラム：グリッドフォーマットを適用せずにペースト	17
1-2 見出しやセクションタイトルをインラインで作る	18
横にはみ出す見出しをインラインで作る	18
改ページ位置の制御	20
セクションタイトルもインラインにする	22
コラム：スタイルを登録するときの注意	27
1-3 ソースコードを表組みで作る	28
ソースコード枠の特徴	28
ソースコードの文字設定	29
表組みに変換する	32
コラム：表にもインライン親行がある	37
ソースコードにタイトルを付ける	38
コラム：一部のセルを次ページに送るには	42
1-4 流し込みに適したマスターページ設計	43
マスターページはなるべく最低限に	43
マスターページ作成のポイント	44
柱の設定	46
章の色分けはスウォッチの入れ替えで対応	47
コラム：連番への対応	48

# 第2章 MarkdownからのXML流し込み

<b>2-1 本書で解説する方式の概要</b>	<b>50</b>
最初の組みの効率をアップする	50
さまざまな自動組版の方法	50
本書で説明する方式のメリット・デメリット	53
<b>2-2 編集環境の準備</b>	<b>54</b>
編集環境の準備	54
<b>2-3 Markdownで原稿を編集する</b>	<b>57</b>
Markdown記法の簡単な説明	57
Gruntの実行	58
コラム：Markdown原稿を依頼するときはガイドラインが必要	62
Markdownレビューのカスタマイズ	63
その他のMarkdownのルール	64
コラム：Gruntを複数同時に動かすには	65
コラム：キーフォントを自動設定するには	65
<b>2-4 XML変換とInDesignへの流し込み</b>	<b>66</b>
Markdown原稿をXMLに変換する	66
XML流し込みの実行	69
画像サイズをまとめて変更する	71
コラム：画像の読み込み倍率をコントロールする	74
タグとスタイルのマッピングを行う	75
<b>2-5 セクションタイトルや中見出しのパートを自動配置する</b>	<b>79</b>
自動配置の準備をする	79
コラム：後送原稿が届いたときは？	81
<b>2-6 表とソースコードを挿入する</b>	<b>82</b>
Markdownでの表とソースコードの記述	82
表とソースコードの流し込み	85
ソースコードを色分けする	89
コラム：スクリプトで画像に枠をまとめて設定する	93

# 第1章

## 流し込みやすいフォーマットを作る

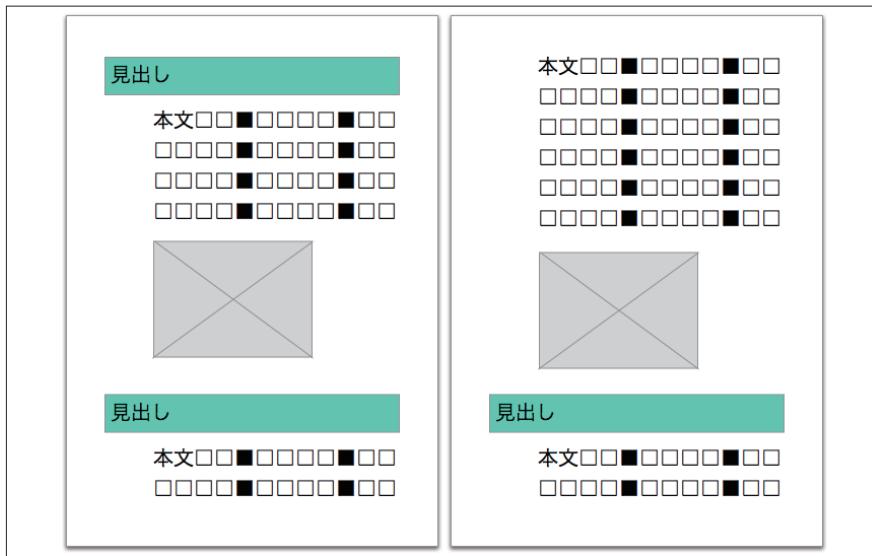
## 1-1

# インラインとグリッドを理解する

本文や画像のパートをバラバラのフレームにしていると、後からの修正対応が大変になります。ここでは本文中に画像を挿入する作業を例にして、インラインとグリッドを活用して効率よく組版する方法を説明します。

## 一段組みなら楽勝！というフォーマット作りを目指す

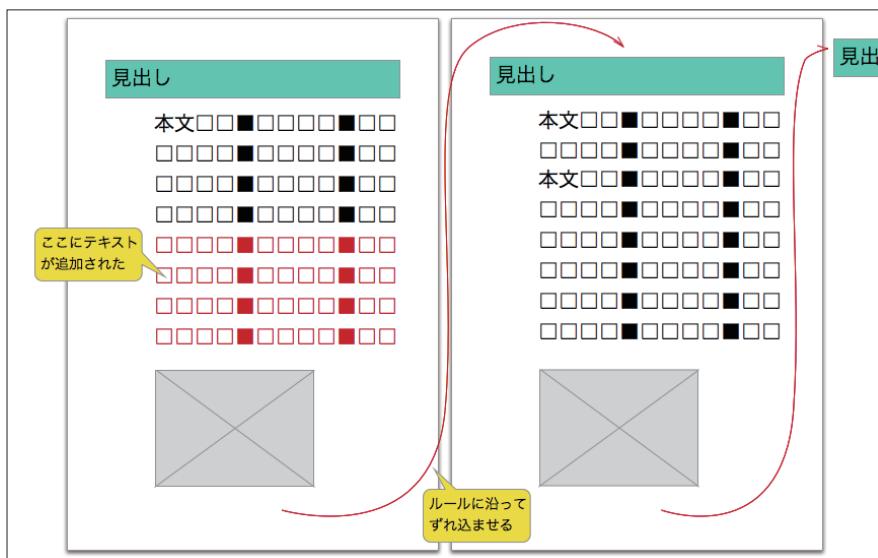
中級者向けのIT書でもっとも多いデザインは、「一段組みで文章の途中に図版やソースコードを挟みつつ、章末またはセクション末までリニアに続していくタイプ」です。このタイプがより速く組めるようになれば、その分を校正期間に回してクオリティアップを図れるようになります。



一段組みリニア型のイメージ

このタイプのデザイン指定は、たいていの場合、「各パートのサイズ」と「パート間隔」の指定になりますが、それをそのままフォーマットにしてはいけません。

そのまま各パートがバラバラの状態で組んでしまうと、後でテキストの追加や削除が発生したときに、それ以降を手作業でずらすことになってしまいます。



デザイン指定そのままのフォーマットでは修正が大変

では、組み直しが発生しない完璧な原稿をもらえばいい……というのは組む側の都合であって、作っているものが「解説書」である以上、デザインが複雑だから、DTP が大変だから、といった理由で内容に間違いがあっても直さないというのは本末転倒です。組みやすいフォーマットを作ることで、テキストや画像追加ぐらいなら楽勝！という状態に行って行きましょう。

## コラム：図解タイプは例外

この文書が対象としているのは、最初に説明したように「一段組みで……リニアに続していくタイプ」です。見開き(2ページ)単位で完結する図解タイプ(できるシリーズなどの初心者向けの解説書)の本や、凝ったレイアウトの雑誌風の書籍には適用できません。

## InDesignのインライン機能を見直す

もともと InDesign は、テキストフレームをリンクさせて複数ページに流し込む機能を持っています。しかし、小説のような文字ばかりの本でしか使えないと思っている人が多いのではないでしょうか？ 実は**インライン機能**を使いこなせば、かなり凝ったデザインの本でも、リンクしたテキストフレームを使って組むことができます。

インライン機能はテキスト中に他のオブジェクト(フレーム)を挿入する機能で、InDesign では**インラインオブジェクト**と**アンカー付きオブジェクト**の2種類に分かれます。

## ▶インラインオブジェクト

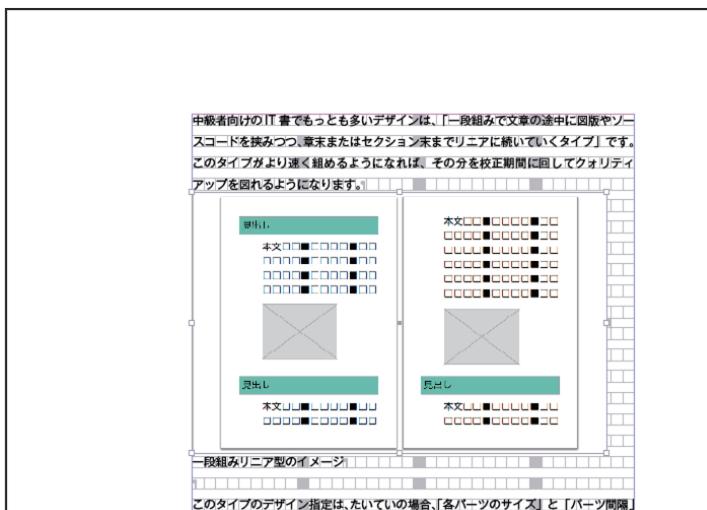
テキスト中に挿入したフレームを右クリックし、[アンカー付きオブジェクト] → [オプション] を選択すると [アンカー付きオブジェクトオプション] ダイアログボックスが表示されます。

ここで [インラインまたは行の上] の [インライン] を選んだ状態がインラインオブジェクトです。



[インライン] を選択した状態

結果を見るとわかるように、文字とまったく同じ扱いになり、オブジェクトのサイズに応じて周りの文字は追い出されます。ただし、回り込みの設定と違って、複数行のテキストがオブジェクトの横に回り込むことはありません。テキスト中の1文字を大きくした場合と同じ状態になります。



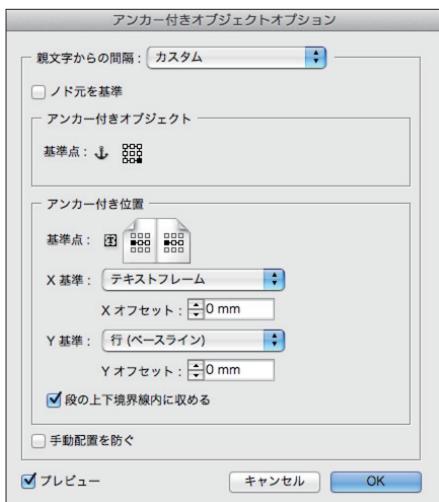
[インライン] を選択した状態

なお、「行の上」という設定もありますが、使ったことがないので割愛します。

インラインにしたときにオブジェクトが他の行に重なってしまう場合は、おそらくインラインオブジェクトが挿入されている段落のスタイルで**固定の行送りが設定されている**はずです。行送りを「自動」にしておけば、オブジェクトのサイズに合わせて行が広がります。

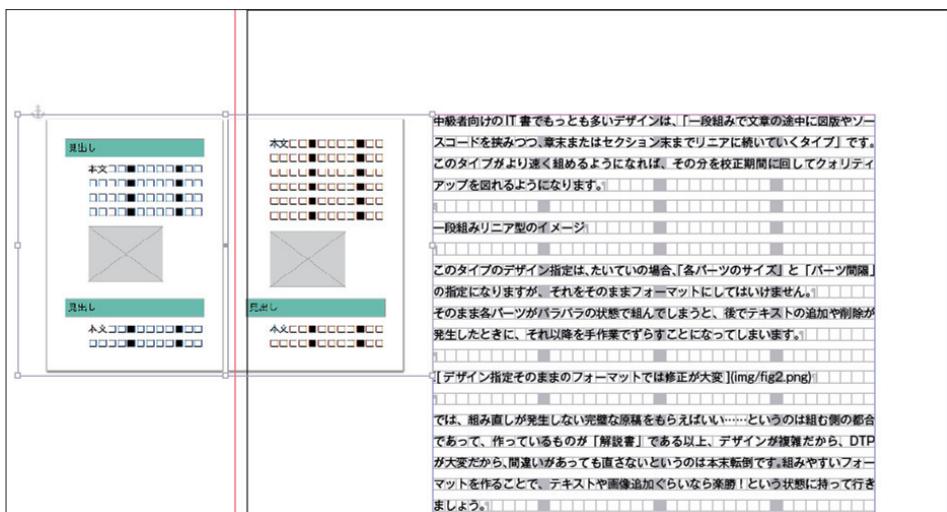
## ▶ アンカー付きオブジェクト

[アンカー付きオブジェクトオプション] ダイアログボックスで [カスタム] を選んだ状態が、アンカー付きオブジェクトです。インラインよりも設定項目が増え、配置の自由度が上がっています。



[カスタム] を選択した状態

結果を見ると、本文と図版にイカリのマークが付き、本文フレームの外に図版が飛び出しています（配置される場所はオプション設定によって変わります）。テキストの配置にはまったく影響しません。



[カスタム] を選択した状態

2つの設定のうち、メインで使用するのはインラインのほうです。**アンカー付きオブジェクトは細かい設定ができますが、その分コントロールが難しくなります。**オブジェクトと本文が重ならないようにしたい場合は、回り込み設定を組み合わせなければいけません。また、ページをまたぐ位置に来たときにオブジェクトが見えなくなったり、うっかりアンカー文字を削除してオブジェクトを消してしまったりするトラブルも起きやすくなります。

アンカー付きオブジェクトを使うのは、オブジェクトの横に本文を回り込ませたいときか、セクションタイトルもインライン扱いにしたいときぐらいです。基本的には単純なインラインを使うことにしたほうが、トラブルが少なくなります。

## コラム：挿入しただけで意図しないスタイルが設定されてしまう？

テキストに図版を挿入しただけで、何も設定していないのに段落スタイルが設定されたりアンカー付きオブジェクトになってしまうという現象に悩まされたことはないでしょうか？ その場合は、オブジェクトスタイルなどのデフォルト設定に問題があります。選択ツールを選んで何も選択していない状態にしてから、オブジェクトスタイルや段落スタイルのパネルを確認してみてください。何かのスタイルが選ばれてはいないでしょうか？ InDesignでは、未選択状態のスタイルが新たに挿入・新規作成したオブジェクトやテキストに割り当てられるようになっています。未選択状態で各スタイルパネルで「なし」（段落スタイルの場合は「基本段落」）を選んでおけば、初期設定で挿入・新規作成できるようになります。

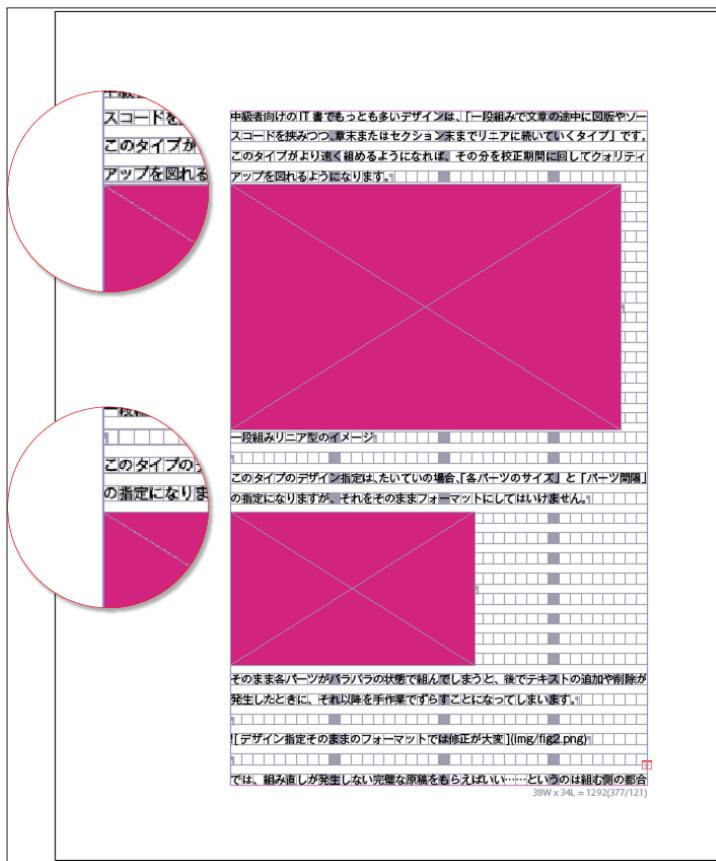
テキストボックスを作成した時点で予期しない文字スタイルが設定されてしまう場合も原因と対処方法は同じです。未選択状態で各スタイルパネルを確認してください。

## グリッドとグリッド揃えを理解しよう

図版などのパーツをインラインで挿入する場合に、もう1つ理解しておく必要があるのが**グリッド**と**グリッド揃え**です。

文章中心の本では、たいていの場合、フレームグリッドが使われていると思います。フレームグリッドの特徴は、文字やインラインオブジェクトの配置が行取りになる点です。見開きページで本文の位置を揃えやすいというメリットがあるのですが、インラインオブジェクトの場合は大きな問題が出てきます。

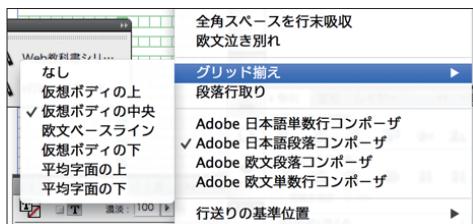
図のサイズによって、図の端と本文の空きがマチマチになってしまふのです。



図と本文の空きがマチマチになってしまします

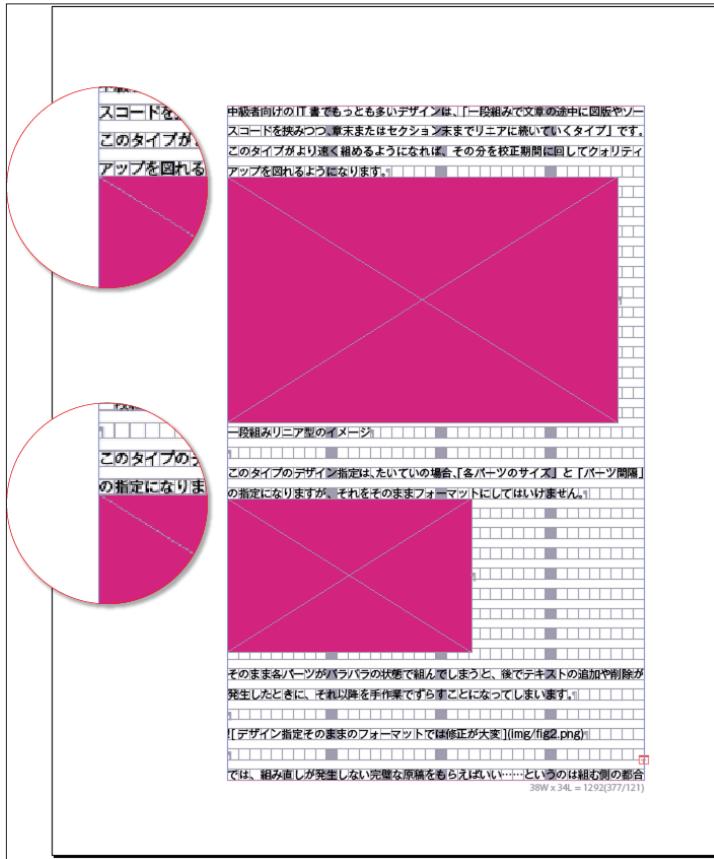
フレームグリッド内のインラインオブジェクトは、それを納めるのに10行必要なら、10行分のスペース内で中央揃えされます。オブジェクトの高さが10行分ピッタリであれば問題ありませんが、そうでなければ、空きがマチマチになってしまいます。

これを解決するには、インラインオブジェクトが挿入されている段落（以降「**インライン親行**」と呼びます）の「グリッド揃え」の設定を変更します。初期設定では「仮想ボディの中央」が選ばれているので、これをなしに変更します。



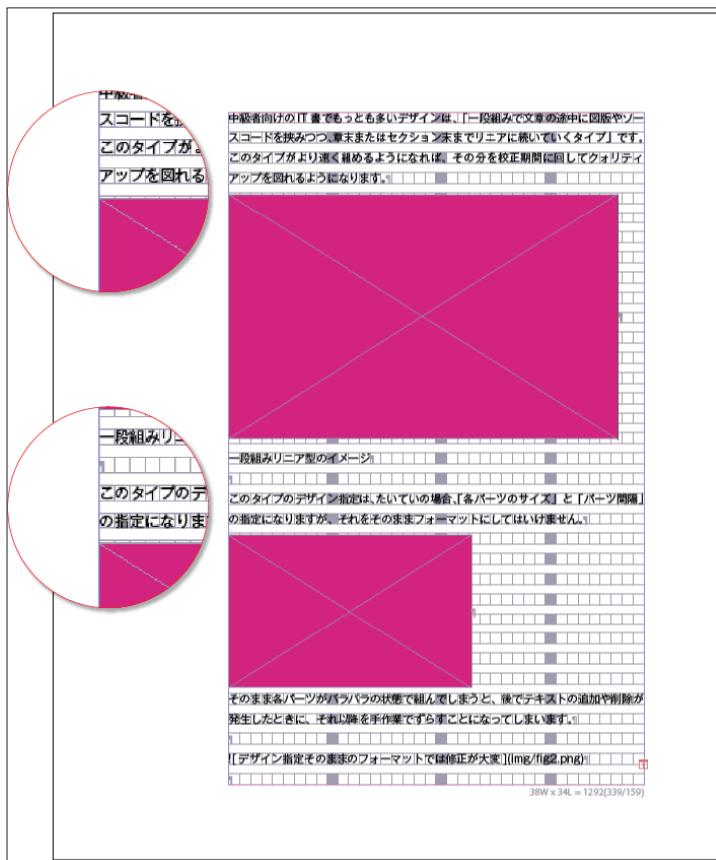
グリッド揃えをなしにする

すると、インライン親行がグリッドに揃わなくなり、上の行にピッタリくっつきます。



グリッド揃えをなしにしただけの状態

後はインライン親行の「段落間スペース」の設定で上に3mm程度の空きを作つてやれば、上の本文と図の間隔を自在に調整できるようになります。



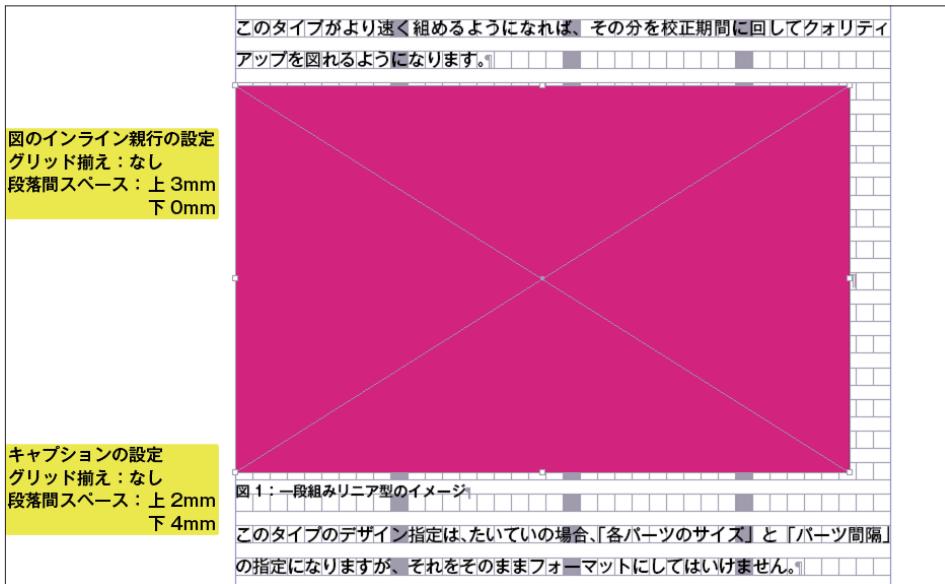
段落間スペースで上の空きを設定

図の下にある本文はグリッドに揃えられるため、下の空きはなりゆきになってしまいますが、これはフレームグリッドの目的を考えればやむを得ないでしょう。段落間スペースで段落後の空きも設定しておけば、下空きが指定量以上になるように設定することは可能です。

ここでは説明の都合で、グリッド揃えと段落間スペースを個別に設定する方法で説明しましたが、実際に運用する際は当然ながら段落スタイルを使用します。

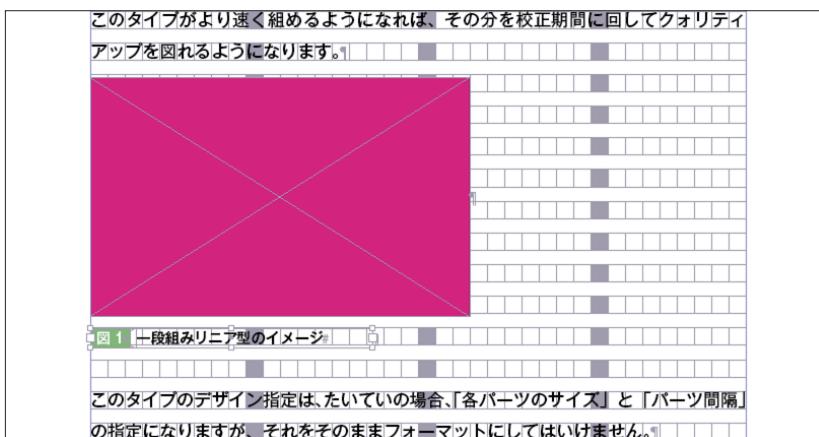
## 図版の上か下にキャプションを付ける

たいていの場合、図の前後には図タイトルかキャプションが入ります。これもグリッド揃えと段落間スペースの応用で配置できます。次の例では、図の上空きが3mm、図とキャプションの間隔が2mm、キャプションと下の本文の空きが4mm以上となるよう設定しています。



図の下にキャプションを配置

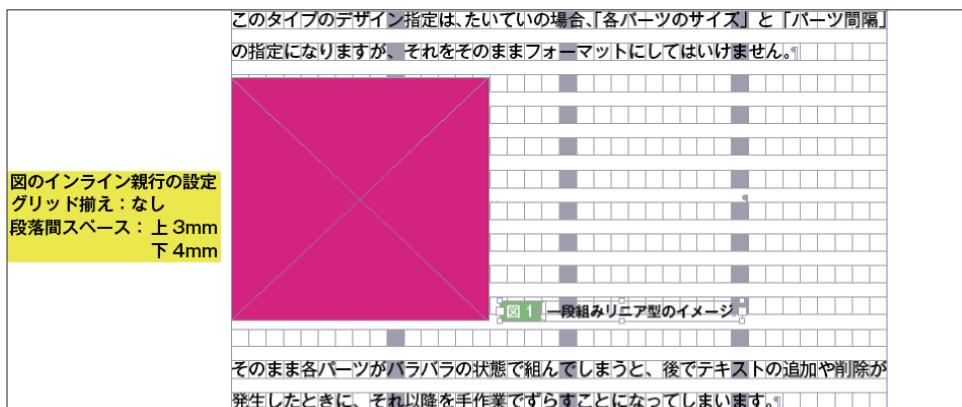
デザインによっては、キャプション文をテキストフレームに入れることもあります。その場合でも図とキャプションはグループ化しません。図とキャプションを分けておけば、後から図のサイズをすることになったときに、キャプションを気にせずに手軽にサイズ調整できます。



キャプションとグループ化しないほうが図のサイズ調整が楽

## ▶図の横にキャプションを付ける

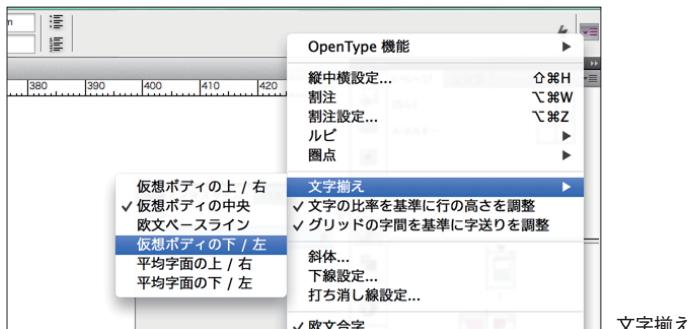
図の幅が狭い場合、キャプションを横に配置することができます。その場合は、キャプションのフレームを、図のINLINE親行の中に入れてしまいます。そうすると図の下の空きが詰まるので、段落間スペースで下空きを4mmに変更します。



横に置くときは同じ親行の中に入れてしまう

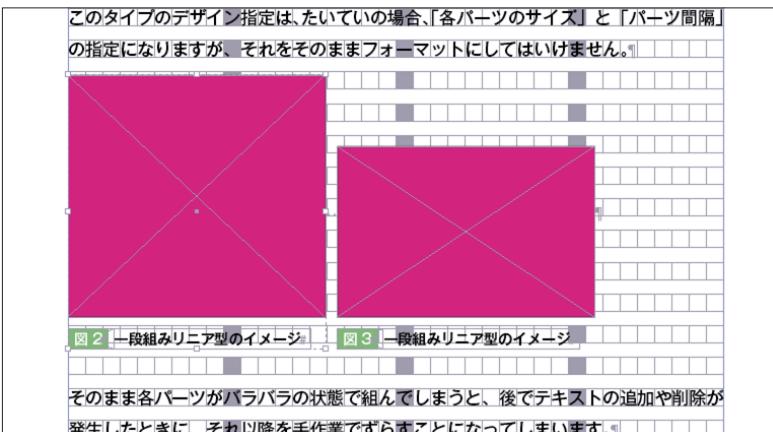
横並びの場合は、図とキャプションの間隔設定に段落間スペースが使えなくなります。ばれない程度に適当に設定するなら、半角スペースを何個か入れておけばいいでしょう。ちゃんとやるなら横並び用のキャプションパーティを用意しておきます。

図とキャプションが上下中央揃えになってしまふ場合は、段落の文字揃えの設定を「仮想ボディの下／左」に変更してみてください。これは段落内にサイズが異なる文字が混在しているときに、縦位置をどこで揃えるかを決める設定です。



## 複数の画像をセットにするときは？

横幅の狭い図が続く場合、それぞれに1行取るのはページの無駄なので、たいていは横に並べて配置することになります。こういう場合はグループ化してからインラインにします。

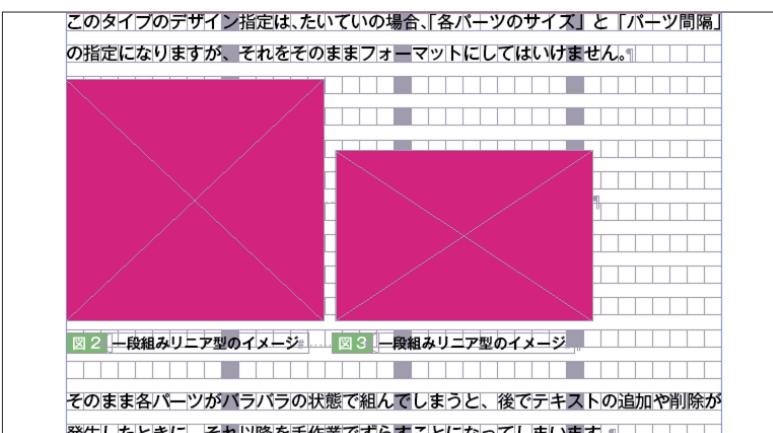


図を横に並べるときはキャプションとグループ化する

「複数の図とキャプションをまとめてグループ化」する場合と、「図とキャプションを1組みずつグループ化」して同じ行にインラインにする場合がありますが、どちらの方法でもかまいません。あえていえば後者のほうが後の修正に強いはずですが、それほど効率は変わりません。

このサンプルのようにサイドの空きが広いフォーマットでは、版面外に図版をはみ出して配置することがありますが、その場合も図とキャプションをグループ化して配置し、インライン親行に右揃えを設定してはみ出させます（P.18の「横にはみ出す見出しをインラインで作る」を参照）。

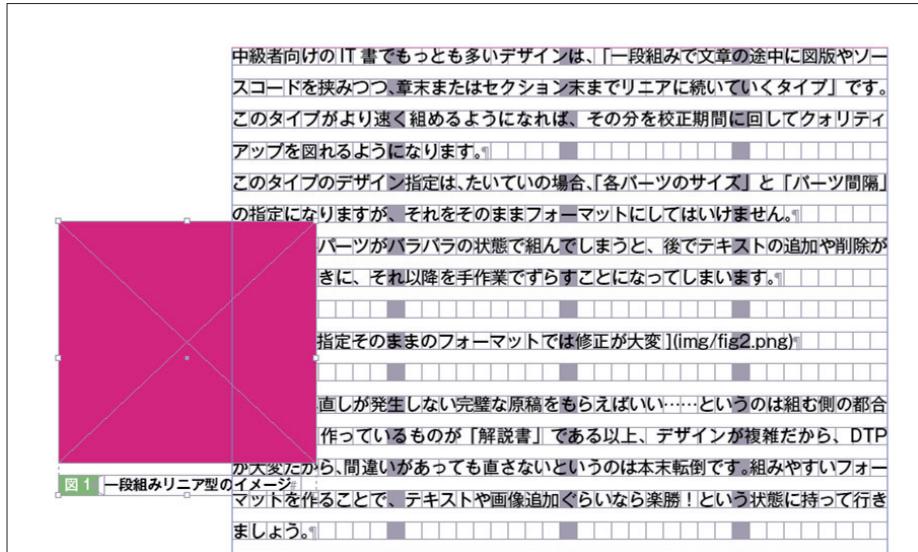
原則的に**グループ化はしないほうが効率がいい**のですが、たとえば「図とキャプションをグループ化せずに別の行に挿入して、タブやスペースで位置を揃える」といったやり方は避けるべきでしょう。図とキャプションの位置がずれやすくなります。



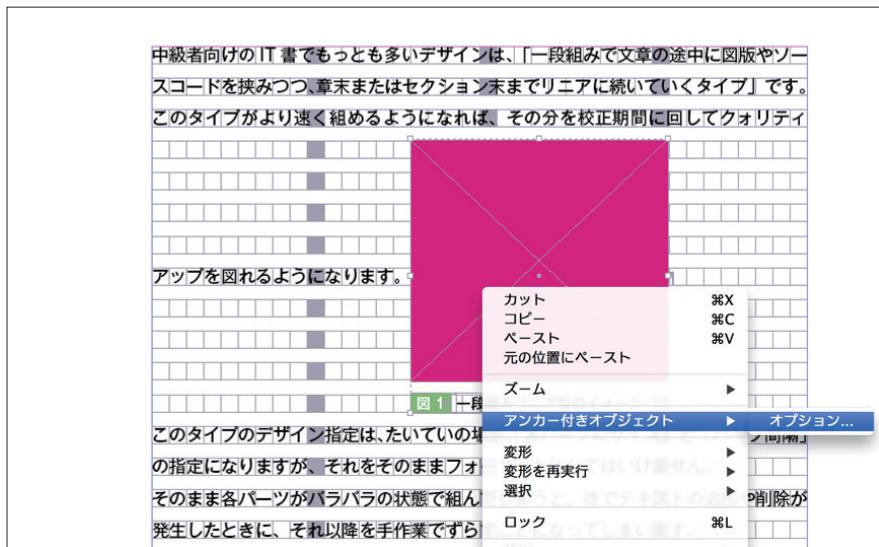
タブやスペースで図とキャプションを別々に揃えるのはかえって面倒

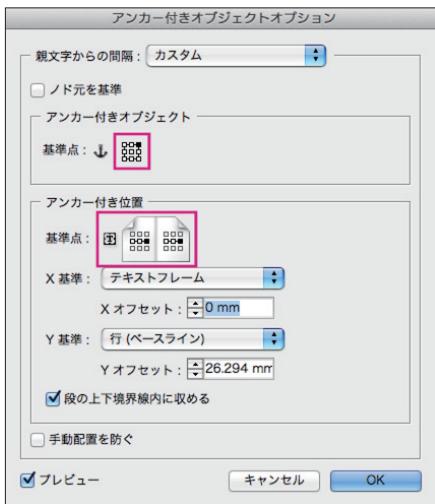
## 図を本文の横に回り込ませる

図を本文の横に回り込ませる場合は、アンカー付きオブジェクトと回り込み設定を使うしかありません。図とキャプションをセットでグループ化し、アンカーを挿入する行の1つ前の段落に挿入して、回り込み設定を行います。

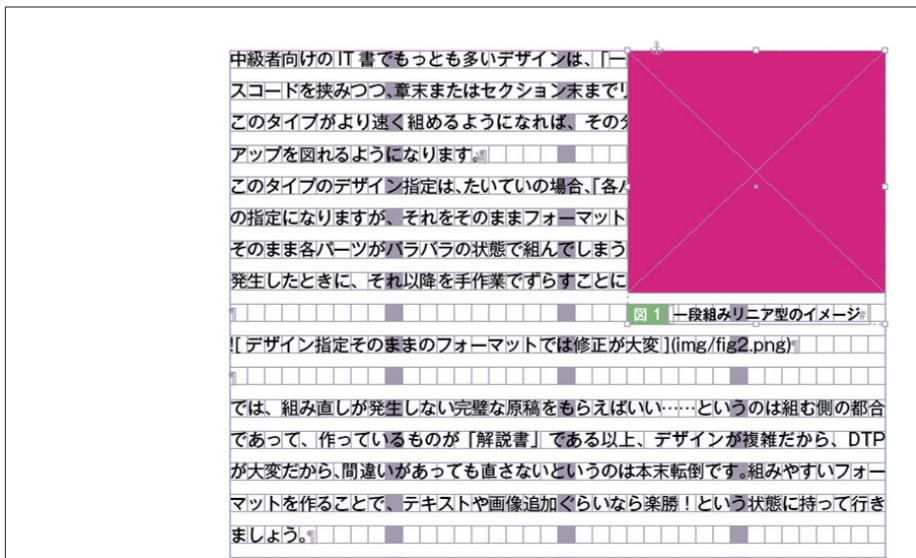


キャプションごとグループ化する





2つの基準点を設定する



アンカー付きオブジェクトになった状態



回り込みを設定する

中級者向けのIT書でもっとも多いデザインは、「一段組みで文章の途中に図版やソースコードを挟みつつ、章末またはセクション末までリニアに続いていくタイプ」です。このタイプがより早く組めるようになれば、その分を校正期間に回してクオリティアップを図れるようになります。

このタイプのデザイン指定は、たいていの場合、「各パートのサイズ」と「パート間隔」の指定になりますが、それをそのままフォーマットにしてはいけません。

そのまま各パートがバラバラの状態で組んでしまうと、後でテキストの追加や削除が発生したときに、それ以降を手作業で揃らすことになってしまいます。

【デザイン指定そのままのフォーマットでは修正が大変】(img/fig2.png)

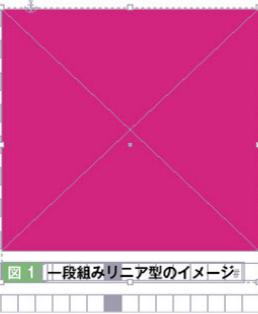


図1 一段組みリニア型のイメージ

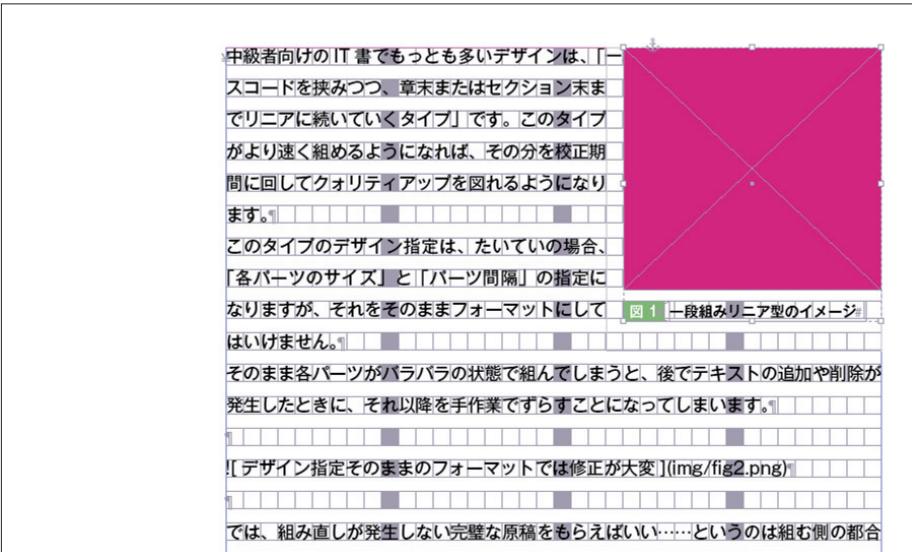
ドラッグして縦位置を揃えて完成

ちょっと面倒ですが、いったん入れてしまえばインラインオブジェクトと同様に段落と一緒に移動するようになるので、手間がかかるのは最初だけです。また、回り込み用のオブジェクトスタイルを作成しておけば、さらに簡単になります。

なお、インラインオブジェクトに戻したい場合、一番手軽なのは図とキャプションのグループ化を解除することです。アンカー付きオブジェクトや回り込みの設定はグループに対して設定されているので、グループ化を解除した時点で初期状態に戻ります。

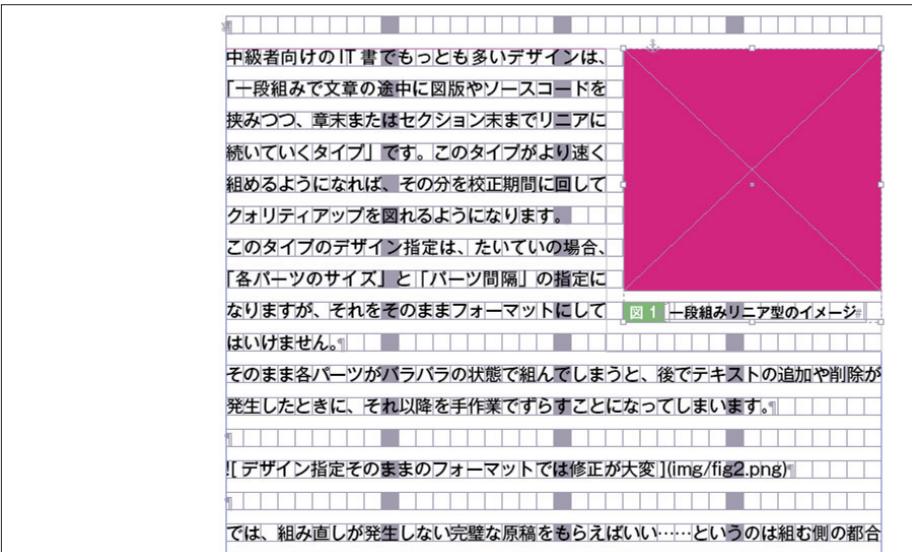
## ▶ アンカー付きオブジェクトの問題

アンカー付きオブジェクトを1つ前の段落に挿入しているのは、InDesignではアンカー親行の1行目には回り込みが効かない仕様になっているためです。



1行目に回り込みが効いていない

これが問題となってくるのは、アンカー付きオブジェクトがページの上端に来る時です。あまりいい解決方法がないので、「本文のフレームを上に1行分伸ばす」「行送りを0にした親行を挿入する」「先頭行の右インデントで回り込みの代わりにする」などの方法で対処しています。



本文フレームを伸ばして親行を作る方法で対処。ただし図版が移動したときに直すのを忘れると大変なことになる

## コラム：グリッドフォーマットを適用せずにペースト

フレームグリッドを使う時に困るのは、ペースト時にグリッドフォーマットが適用されるため、コピー＆ペーストするだけでスタイルがオーバライドされてしまう点です。

カット＆ペーストで順番を入れ替  
えただけのつもりが……

<b>現象</b> InDesign の <b>フレームグリッド</b> (原稿用紙状のマス目が表示されたフレーム) 内でテキストのコピー＆ペーストを行うと、フレームグリッドに設定された書式でオーバライドされてしまう。	<b>対処法</b> そのため、 <b>ちょっと文字直しをしたつもり</b> なのに、フォントや文字サイズなどが変わってしまい、トラブルのもとになる。
<b>現象</b> InDesign の <b>フレームグリッド</b> (原稿用紙状のマス目が表示されたフレーム) 内でテキストのコピー＆ペーストを行うと、フレームグリッドに設定された書式でオーバライドされてしまう。	<b>対処法</b> そのため、 <b>ちょっと文字直しをしたつもり</b> なのに、フォントや文字サイズなどが変わってしまい、トラブルのもとになる。

ペーストの結果がおかしくなる

一応【編集】メニューの【グリッドフォーマットを適用せずにペースト】で貼り付ければスタイルがオーバライドされることなくなるのですが、普通のペーストとの使い分けが面倒です。

これを解決するスクリプトが以下のブログで紹介されています。このスクリプトを導入してショートカットキーを設定すると、グリッドかどうかを気にせずに [command] + [V] を利用できるようになります。フレームグリッドを使うなら導入必須です。

**[InDesign] フレームグリッドではグリッドフォーマットを適用せずにペーストするスクリプト**

<http://sysys.blog.shinobi.jp/Entry/6/>

## 1-2

# 見出しやセクションタイトルを インラインで作る

凝ったデザインのタイトルパートでも、画像の応用でインラインにすることが可能です。ここでは一見オンラインでは処理しにくそうな、版面からはみだすタイトルの組み方を説明します。

## 横にはみ出す見出しをインラインで作る

中見出し・小見出しなどのパートが、段落スタイル・文字スタイルだけで表現しきれない場合、フレームを組み合わせたものを使うことになります。これも図版と同じ方法で、オンラインと段落間スペースの組み合わせで対応可能です。

では、時々ある見出しが版面の外にはみ出すデザインの場合はどうでしょうか？ 実は版面からはみ出す場合でも、オンラインなら非常に簡単に対応できます。

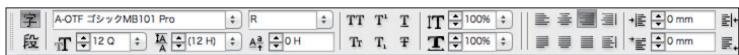
まず、普通に見出しパートをオンラインにすると、次のように右側にはみ出します。



普通にオンラインにした状態

オンライン親行にカーソルを移動して、右揃えを設定すると、左側にはみ出します。

## 1-2 見出しやセクションタイトルをインラインで作る



右揃えに設定

一段組みなら楽勝！という  
フォーマット作りを目指す

中級者向けのTT書でもっとも多いデザインは、「一段組みで文章の途中に図版やソースコードを挟みつつ、章末またはセクション末までリニアに続いていくタイプ」です。このタイプがより早く組めるようになれば、その分を校正期間に回してクオリティアップを図れるようになります。

図1：一段組みリニア型のイメージ

版面の左にはみ出した

これで完成です。

見出しがページの途中に配置されたときのために、段落間スペースで上下の空きを設定しておきましょう。段落がページ頭に配置されるときは上の段落間スペースが無視されるので、それを計算に入れて設定する必要があります。後述しますが、上の空きは改行で入れたほうが都合がいいこともあります。

見出しのオンライン親行の設定  
グリッド揃え：なし  
段落間スペース：上 13.5mm  
下 5mm

このタイプのデザイン指定は、たいていの場合、「各バーツのサイズ」と「バーツ間隔」の指定になりますが、それをそのままフォーマットにしてはいけません。そのまま各バーツがバラバラの状態で組んでしまうと、後でテキストの追加や削除が発生したときに、それ以降を手作業で直すことになってしまいます。

では、組み直しが発生しない完璧な原稿をもらえばいい……というのは組む側の都合であって、作っているものが「解説書」である以上、デザインが複雑だから、DTP

30W × 34L = 1292(373/63)

段落間スペースで上下の空きを調整

常に左にはみ出すのではなく、小口方向にはみ出させたい（右ページでは右にはみ出す）場合は、右揃えではなく「ノド元に向かって整列」を使用します。揃え方が変わるので、左右ページで見出

しのデザインが変わるのはどうにもならないですが、そういうデザインが来ないことを祈るのみです。

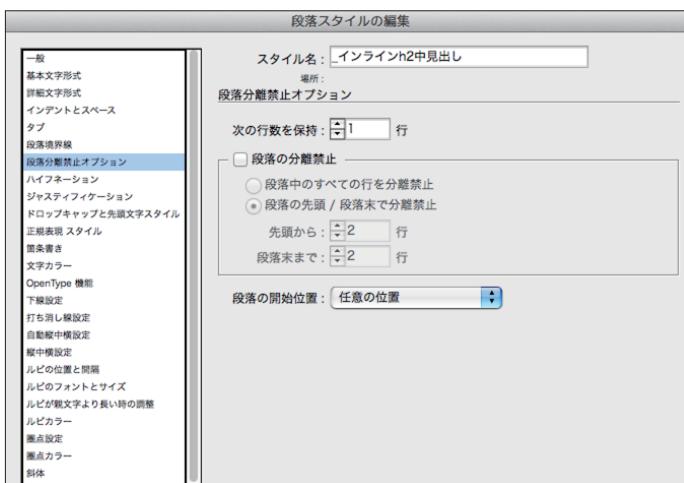
## 改ページ位置の制御

見出しがページ末に来るのはたいてい禁止です。手で直すのは面倒なので、段落スタイルの「段落分離禁止オプション」を使って自動的に次ページに送られるようにしておきましょう。



中見出しがページ末に配置されている

[段落スタイルの編集] ダイアログボックスを表示して、[段落分離禁止オプション] を選択し、[次の行数を保持] を「1行」以上にします。



[次の行数を保持] を1行に設定

## 1-2 見出しやセクションタイトルをインラインで作る

これで中見出しの段落の後に1行以上確保できない場合は、次ページに送られるようになります。最低限2~3行はほしいといった決まりがある場合は、[次の行数を保持]を適当に変更してください。

**一段組みなら楽勝！という  
フォーマット作りを目指す**

中刷組みの「IT書」でもっとも多いデザインは、「一段組みで文書の中間に図版やソースコードを挟みつつ裏表またはセクション末までリニアに続いていくタイプ」です。このタイプがとり過度組めるようになれば、その分を校正範圍に回してクリティックを回れるようになります。

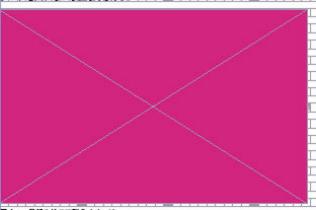


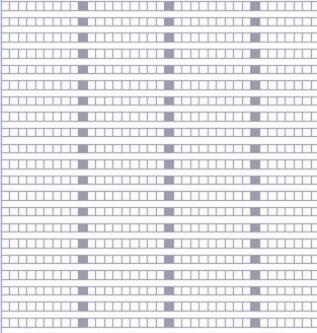
図1：一段組みリニア型のイメージ

このタイプのデザインは、たいていの場合、「各バージョンのサイズ」と「バージョンの指定」になりますが、それをそのままフォーマットにしてしまうと、そのままバージョンがバラバラの状態で組んでしまって、後でテキストの追加や削除が発生したときに、それ以降を手作業で直すことになってしまいます。そのまま各バージョンがバラバラの状態で組んでしまって、後でテキストの追加や削除が発生したときに、それ以降を手作業で直すことになってしまいます。

38W x 34L = 1292(367)

**一段組みなら楽勝！という  
フォーマット作りを目指す**

では、読み直しが発生しない完璧な構成を看らねばいい…といふのは面白くない都合であって、作っているものが「解説書」である以上、デザインは複雑だから、DTPが大家だから、難易度があっても直さないというのは本末転倒で、読みやすいフォーマットを作ることで、テキストや画像追加でらいなら楽勝！という状態に持って行きましょう。



38W x 34L = 1292(160)

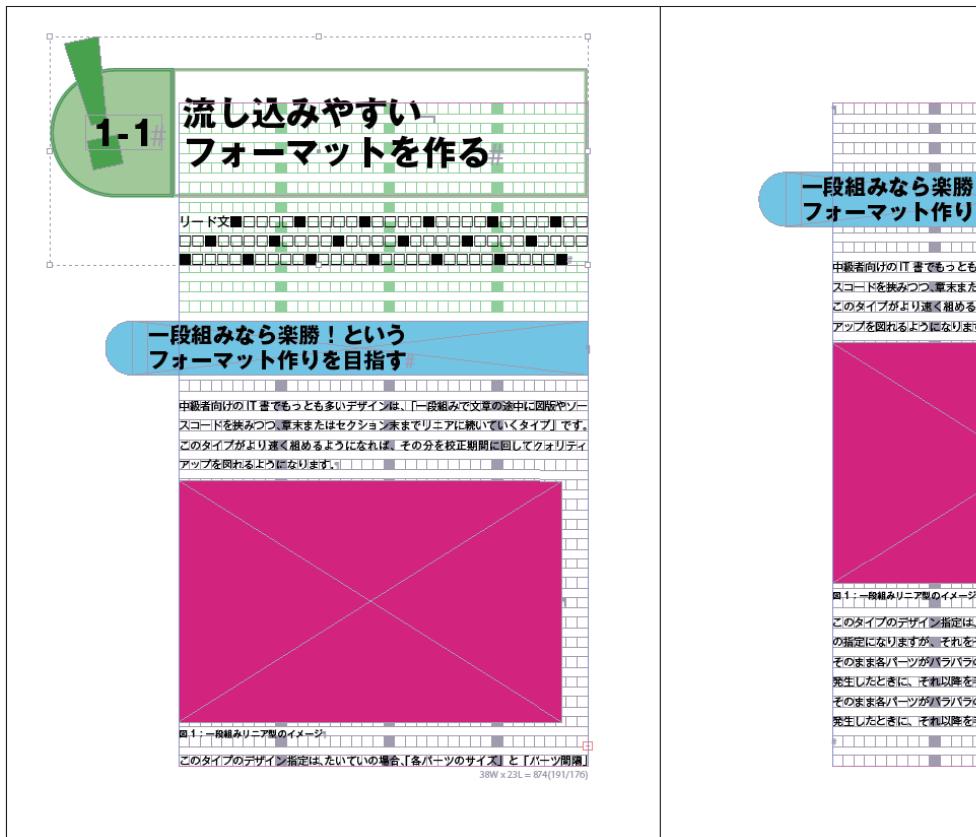
自動的に次ページに送られた

## セクションタイトルもインラインにする

セクションタイトルは、単調さを避けるために派手なデザイン（作りが複雑）になることが多く、オンラインでは扱いにくい面があります。また、中見出しに比べれば数が少ないので、手作業で配置してもさほど効率は落ちません。とはいえオンラインで配置することは可能ですし、300ページを越えるような分厚い本では必須テクニックとなります。

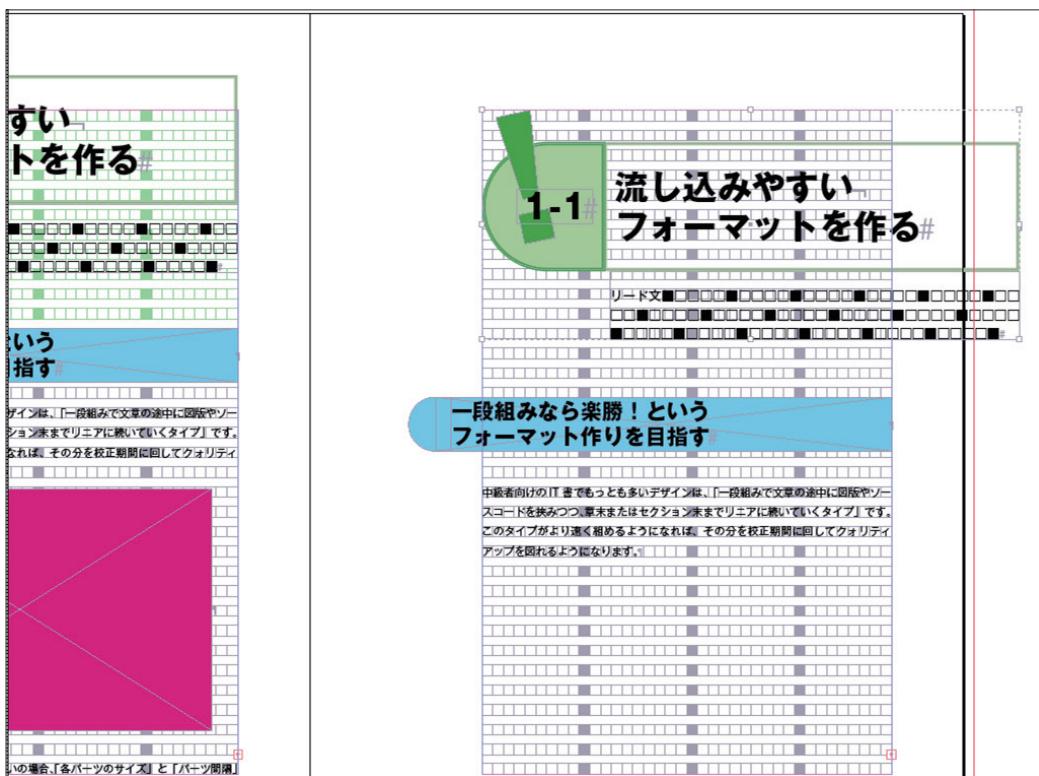
セクションタイトルは、たいていの場合ページ頭に固定されます。これは先ほど紹介した段落分離禁止オプションで対応可能です。セクションタイトルはマージンより上に突き出させることも多いので、その場合はオンラインではなくアンカー付きオブジェクトにします。

方針が決まったところで、実際にやってみましょう。まずセクションタイトルのパートを作成します。



セクションタイトルのパートを作つてグループ化する

隣に見本があると作業しやすいので、左ページの見本を見ながら右ページでセクションタイトルを設定していくことをおすすめします。右ページの本文フレーム内にセクションタイトルのパートをペーストしてオンライン状態にします。



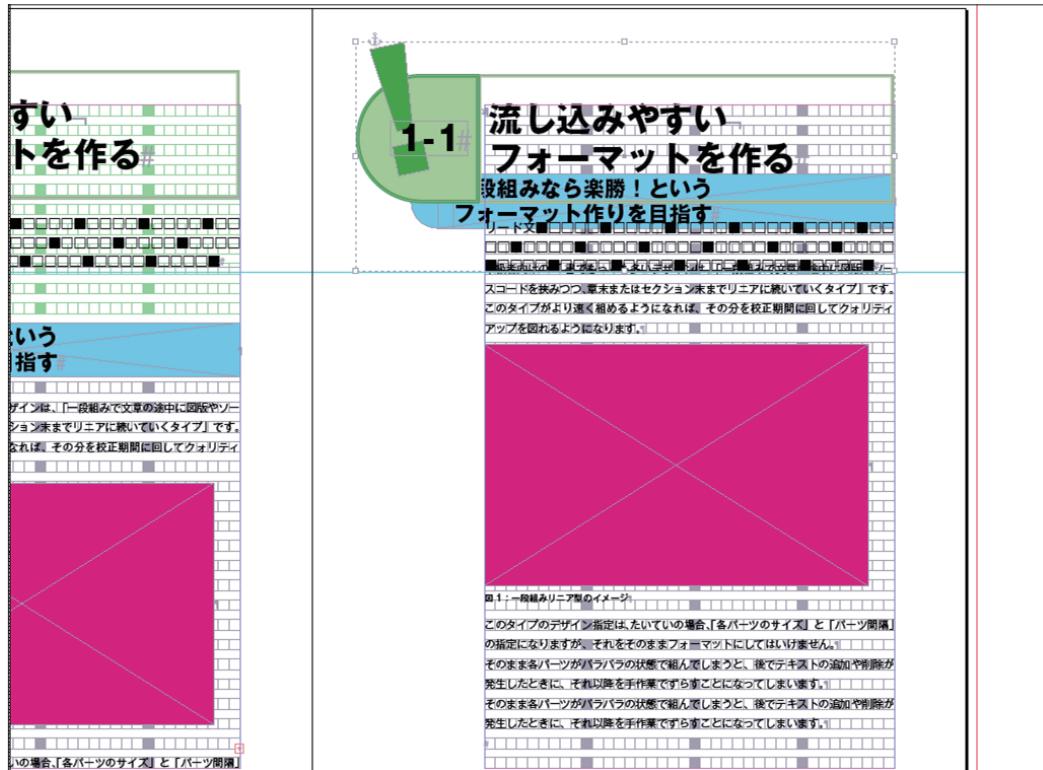
オンライン状態にする

[アンカー付きオブジェクトオプション] ダイアログボックスを表示し、[カスタム] を選んで基準点を図のように設定します。上にはみ出させるために [段の上下境界線内に収める] をオフにし、[Y オフセット] をデザイン指定の位置に来るよう調整します。



アンカー付きオブジェクトのオプションを設定

アンカー付きオブジェクトになり、版面からはみだす位置に配置されましたが、本文のテキストと重なってしまいます。

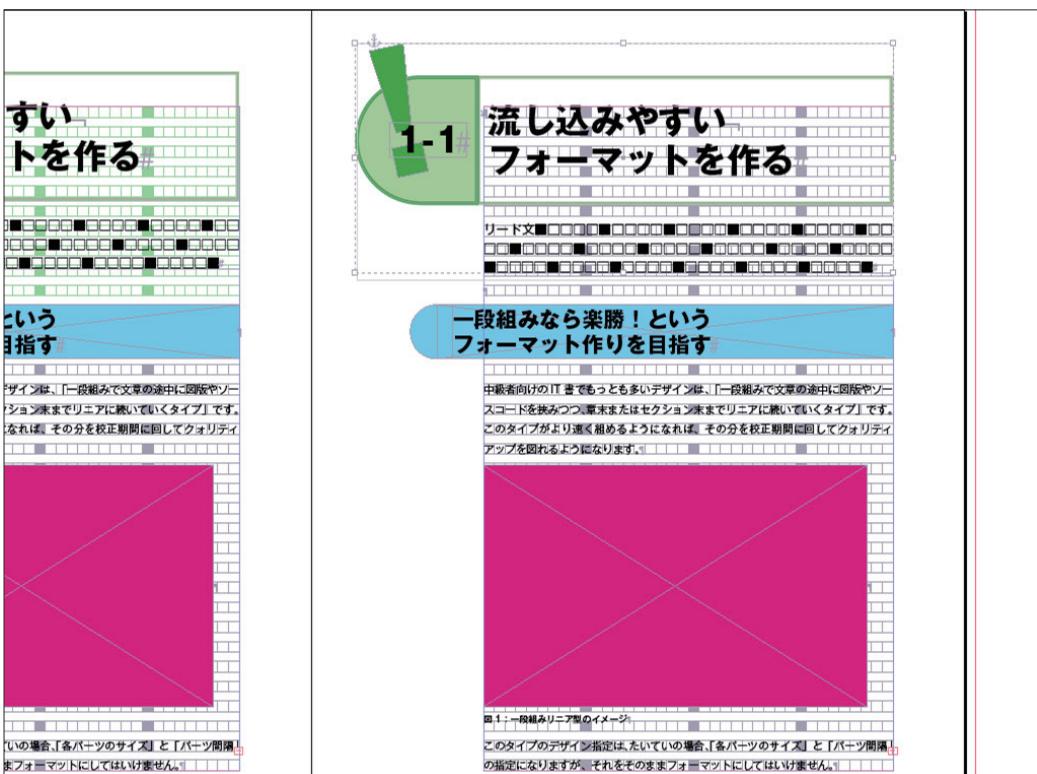


位置が調整できた

本文をどけるためにセクションタイトルパートに回り込みを設定します。このときマージンがすべて0mmだと回り込みが有効にならないようです。この例では下マージン2mmで設定します。



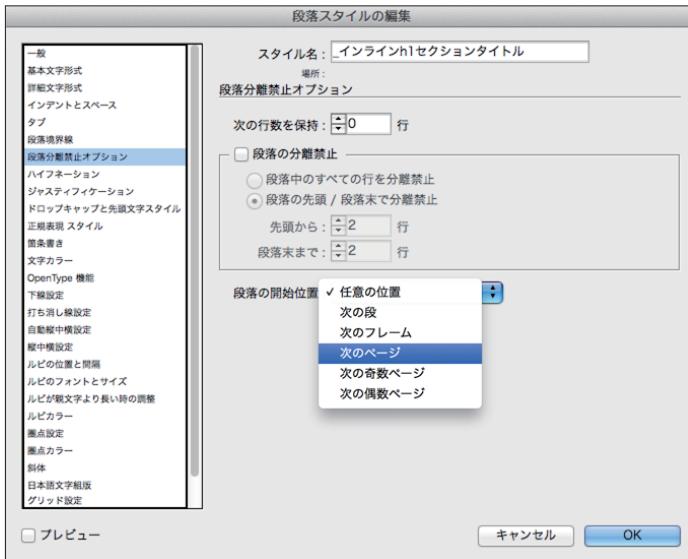
本文との重なりが解消されました。プレビューモードに切り換えて、手本の左ページと比べて違和感がないか確認しておきましょう。問題がなければオブジェクトスタイルに登録しておきます。



セクションタイトル完成

上の手順では説明しませんでしたが、実は中見出しのインライン親行の設定も変更しています。少し前に「中見出しのインライン親行は、上の段落間スペースを13.5mmにする」と説明しました(P.19参照)。しかし、その設定だとセクションタイトルパートと中見出しの間が空きすぎてしまうのです。そこで、上の段落間スペースを2mmに変更し、代わりに改行でアキを調整することにしました。手作業になるのが面倒ですが、これなら見出しの上に本文・セクションタイトル・図版のいずれが来てもケースバイケースで調整できます。

また、セクションタイトルを常にページ先頭に配置するために、セクションタイトル用のインライン親行の段落スタイルを作成して、[段落分離禁止オプション] の [段落の開始位置] を変更しておきます。



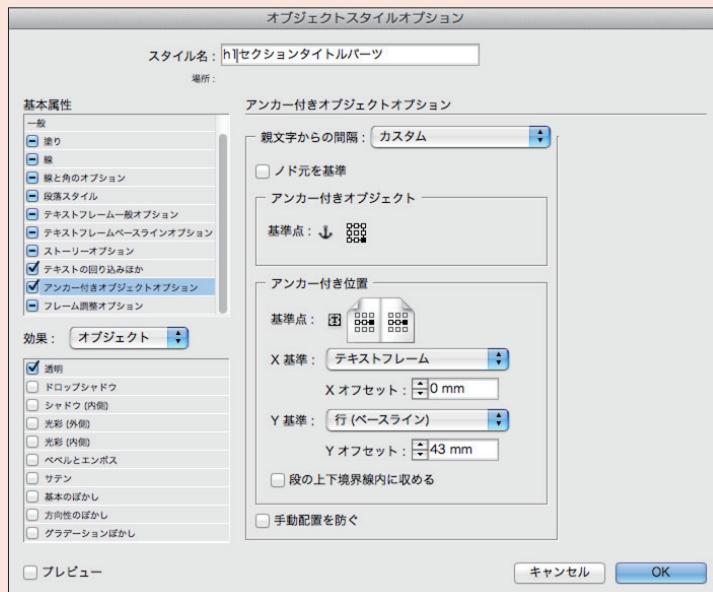
[段落の開始位置] を [次のページ] にする

これでセクションタイトルも本文と一緒に移動するようになります。前後の文章を増減させて、正しく移動することを確認してください。

なお、デザインによっては、左右ページでセクションタイトルパートの位置が微妙に違うこともあります。その場合はアンカー付きオブジェクトの設定では対応できないので、手作業で微調整するか、オブジェクトスタイルを2種類設定して切り替えながら使うことになります。ページ数が多く、セクションタイトルの数も非常に多い本ではエラーの原因になるので、デザインを変えてもらうという選択も必要でしょう。

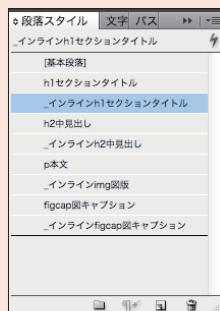
## コラム：スタイルを登録するときの注意

ここまで説明してきた設定は、すべてオブジェクトスタイルや段落スタイルに登録してから実作業に入ります。オブジェクトスタイルを登録するときは、必要な【基本属性】だけをオンにするようにしてください。そうしないと、スタイルを設定した途端に画像のトリミング範囲や倍率が狂ったり、フレームの色が変わったりしてしまいます。



不要な【基本属性】をオフにする

段落スタイルの場合は、インライン親行のせいでスタイル数が倍近く増えるため、混乱しないようにする工夫が必要です。どんなやり方でもいいのですが、個人的にはインライン親用行の段落スタイルに「\_インライン」という名前を付けるようにしています。また、スタイルグループ（フォルダ）を作って「本文・見出し」「図版」「表」などで分類するのも有効です。



段落スタイルの命名に工夫が必要

なお、この図ではスタイル名に h1、h2 などの HTML のタグ名が入っていますが、これは後で説明する Markdown からの流し込みをやりやすくするためのものです。

## 1-3

# ソースコードを表組みで作る

ソースコード枠に表組みを利用すると、複数ページにまたがるソースコード枠でも簡単に作れるようになります。ここでは合成フォントの作成や欧文組版ルールの解除といったソースコードを組むために必要な基礎知識もあわせて解説します。

### ソースコード枠の特徴

IT書だからといって必ずソースコードがあるとは限りませんが、プログラミング系の書籍を担当し始めると日常的に組むことになります。たいていのソースコードは次のような特徴を持ちます。

- ・全体が枠などで囲まれる
- ・フォントは等幅で位置が崩れないようにする
- ・行番号や罫線を入れることがある
- ・10行程度で納まることがあれば、複数ページにまたがるほど長いこともある

細かい部分を無視すると、デザイン的な外観はコラム枠に近いものといえます。ただし大きな違いは、**複数ページにまたがることがわりとよくある**という点です。ふつうにテキストフレームの組み合わせで作ると、ページ移動にともなってフレームを分割したり結合したりと大変面倒な作業になります。

<p>ここからソースコードです。</p> <pre>html2indtag.js //コード以外のlineの変換 var parseHTML = function(src, oImode){   //img関連、画像全般をfigureタグで囲む   src = src.replace(/(img[^&gt;]*&gt;)/g, '&lt;figure&gt;\$1&lt;/figure&gt;');   //src = src.replace(/&lt;img&gt;/g, '&lt;img&gt;');   //src = src.replace('&lt;/img&gt;', '&lt;/img&gt;');   //img関連、src=""&amp;gt;file:///"に   src = src.replace('&lt;a href="file:///"&gt;', '&lt;img href="file:///\$1"/&gt;');   //img関連、zoom値とclipの設定をimg要素のdata-zoom data-clip属性に   src = src.replace('&lt;img href="["?"]"&gt;zoom([0-9]+) /g, '\$1' data-zoom="\$1"');   src = src.replace('/data-zoom="[^0-9]*" /g, '\$1');   src = src.replace('&lt;(img[^&gt;]*&gt;)\?clip([0-9]+)&gt;"/g, '\$1' data-clip="\$1"');   src = src.replace('/\amp;clip([0-9]+)"/g, ' data-clip="\$1"');   //キャプションをfigcaptionに   src = src.replace('alt="["?"]"/&gt;&lt;/figure&gt;, '&gt;&lt;/figure&gt;\n&lt;caption&gt;';    alert&gt;&lt;img alt="["?"]"/&gt;&lt;/img&gt;&lt;/caption&gt;');   src = src.replace('alt="["?"]"/&gt;&lt;/p&gt;, '&gt;&lt;/p&gt;');   //リンクは文字列(url)の形に   //動作が多すぎるのでナシ   //src = src.replace('&lt;a href="(["?"]")"&gt;(\$&lt;*&gt;)&lt;/a&gt;/g, '\$2 (\$1)');   //hrは改行と一緒に、hrほどあえず独立タグに   src = src.replace('&lt;br/&gt;', '\n');   src = src.replace('&lt;br/&gt;g, '&lt;br/&gt;');   //箇条コメントや図中文字などの独自抜き（div class="***"で指定しているもの   //の処理   src = src.replace('&lt;/div class="hen"(**)&gt;&lt;/div&gt;/g,     '&lt;div_hen&gt;alert&lt;img alt="["?"]"/&gt;&lt;div_hen&gt;');   src = src.replace('&lt;/div class="zuchau"(**)&gt;&lt;/div&gt;/g,     '&lt;div_zuchau&gt;alert&lt;img alt="["?"]"/&gt;&lt;div_zuchau&gt;');   src = src.replace('&lt;/div class="(["?"]")"&gt;(**)&lt;/div&gt;/g,     '&lt;div_1\$12&gt;&lt;/div&gt;');   src = src.replace('&lt;kbd&gt; [("["?"]")] &lt;/kbd&gt;/g, '&lt;kbd&gt;1\$10&lt;/kbd&gt;');   //他のspanタグ }</pre>	<pre>src = src.replace('&lt;span class="(["?"]*)"&gt;(\$&lt;*&gt;)&lt;/span&gt;/g,   '&lt;span_1\$12\$2\$3&gt;&lt;span_1&gt;');   //テーブルをスクリプト処理しやすいよう単純化(tbody,theadをカット,セルをtdに)   src = src.replace('&lt;thead/&gt;', '');   src = src.replace('&lt;/thead/&gt;', '');   src = src.replace('&lt;tbody/&gt;', '');   src = src.replace('&lt;tr/&gt;', '');   src = src.replace('&lt;td/&gt;', '');   src = src.replace('&lt;th/&gt;', '');   src = src.replace('&lt;/th/&gt;', '');   //olモードだとolをol_1に変える   if(oImode==true){     src = src.replace('&lt;li&gt;(\$&lt;*&gt;)&lt;/li&gt;/, 'ol_li&gt;\$1&lt;ol_1&gt;');   }   //異出し1 ~ h6に&lt;alert&gt;を入れる   if(options.mdashialert == true){     src = src.replace('&lt;h1&gt;)&gt;/g, '&lt;h1&gt;alert&lt;h1&gt;alert&gt;');     src = src.replace('&lt;h2&gt;)&gt;/g, '&lt;h2&gt;alert&lt;h2&gt;alert&gt;');     src = src.replace('&lt;h3&gt;)&gt;/g, '&lt;h3&gt;alert&lt;h3&gt;alert&gt;');     src = src.replace('&lt;h4&gt;)&gt;/g, '&lt;h4&gt;alert&lt;h4&gt;alert&gt;');     src = src.replace('&lt;h5&gt;)&gt;/g, '&lt;h5&gt;alert&lt;h5&gt;alert&gt;');     src = src.replace('&lt;h6&gt;)&gt;/g, '&lt;h6&gt;alert&lt;h6&gt;alert&gt;');   }   //通常行内のcodeタグを削除   src = src.replace('&lt;code&gt;/g, '');   src = src.replace('&lt;div&gt;/g, ''');    return src; };  </pre>
---	--

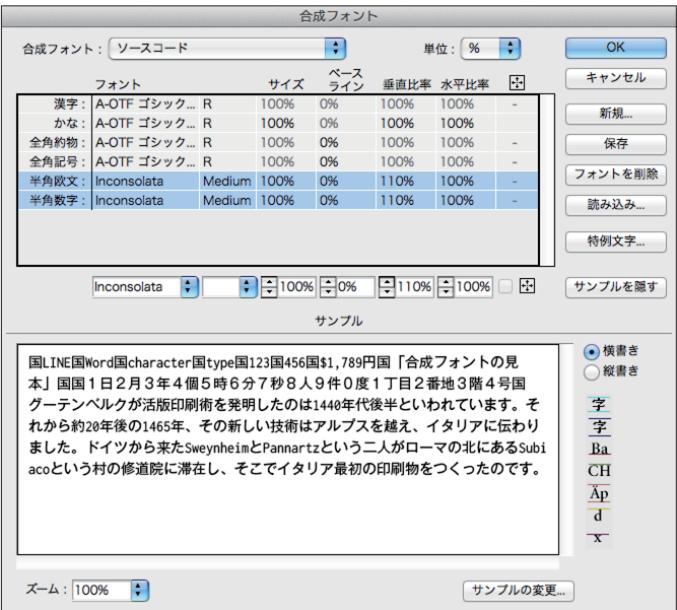
複数ページにまたがるソースコード

この問題を解決する方法が、**ソースコード枠を表組みで作る**というテクニックです。自分で考えたわけではなく、もともとは外部からもらったフォーマットで使われていたものを参考にしたのですが、その後スクリプトなどを組み合わせて、現在ではほぼ自動的に処理できるようにしています。自動的に組む方法は第2章で解説するので、ここではソースコード枠の構造を理解するために、手作業で作る方法を解説しましょう。

## ソースコードの文字設定

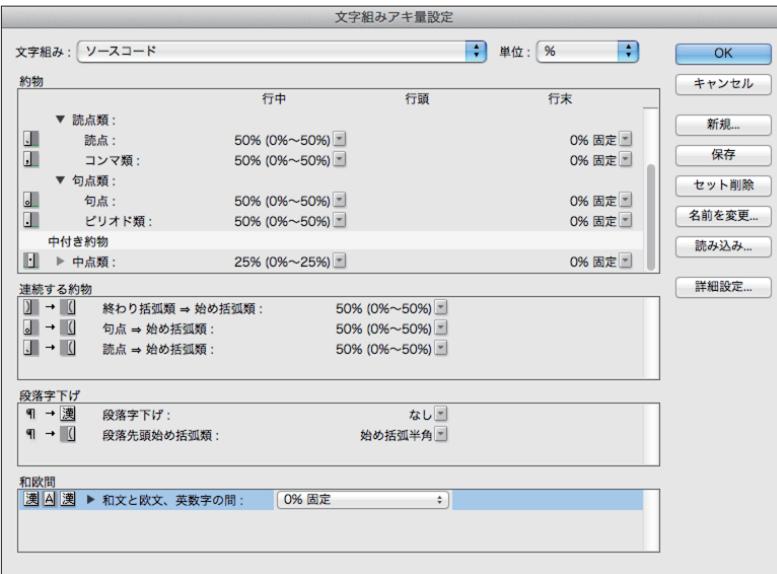
まず基本の文字スタイルについて説明します。ソースコードは原則等幅なので、フォントは、Courier New、Andale mono、Inconsolata、Source Code Proなどの等幅欧文フォントをベースにした合成フォントとなります。ただしそれで終わりではありません。これは最低限のことしかなく、半角スペースで正確に文字揃えできるように、通常の和欧混文用とは設定を変えなければいけないのです。

まず、**合成フォントを作る時に和歐の横幅を変えると揃わなくなります**。サイズを調整したい場合は、欧文を縦にちょっと引き延ばす程度がおすすめです。



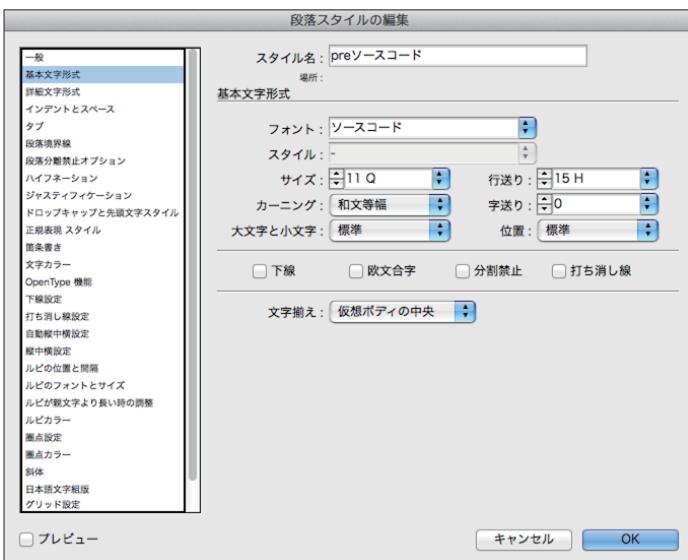
欧文を縦に 110% 引き延ばしてサイズを揃える

また、ソースコード用の文字組みアキ量設定を作成し、**和欧文間をゼロ固定**にします。



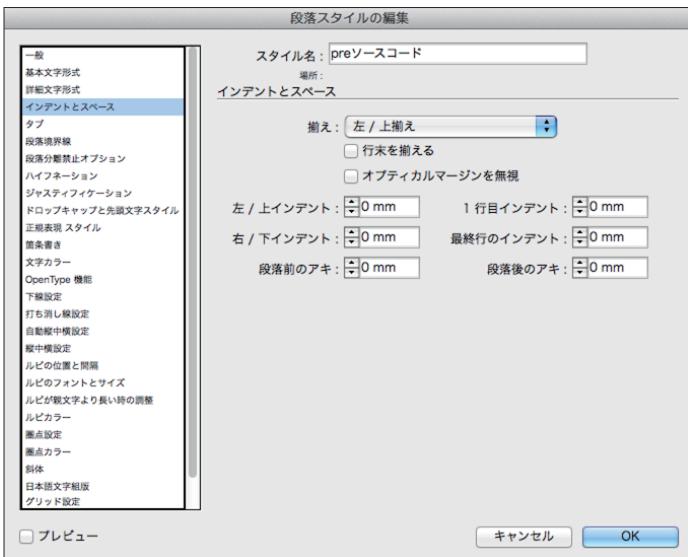
和欧文間をゼロにする

その他に、段落スタイルで **[欧文合字]** をオフにし、カーニングや字送りなどの字間を変える設定は **原則使わない** ようにします。折り返しが少ないほうがいいので、文字サイズは少し落としておくといいでしょう。



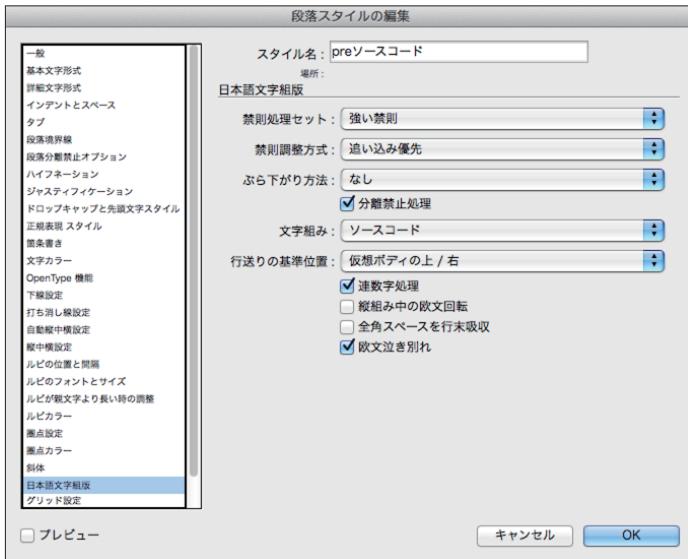
[欧文合字] をオフにし、余計な字間調整はなるべくしない

折り返しが発生した場合に字間が広がらないように、行末揃えではなく**左揃え**を使用します。



左揃えがベター

単語の途中で折り返しても問題ないようなら、[欧文泣き別れ]をオンにしてもいいでしょう。ただし、誤読の原因になる（2単語だと誤解される）こともあるので、この設定を使うかどうかはよく検討してください。



場合によっては【欧文泣き別れ】をオンに

最後に和欧混交のソースコードのサンプルを入力し、半角スペースで揃えられることを確認します。

```
//  
..... ja.=.."日本語"+today; ..... //コメント  
en = "英語" + today; //コメント  
.... afcs = "アフリカーンス語" + today; //コメント
```

半角スペースでコメントの // が揃えば OK

可能であれば、その本の対象となるプログラミング言語のサンプルを著者からもらって流し込み、問題が起きないことを確認してください。

## 表組みに変換する

それではソースコードを表に変換し、セルスタイルと表スタイルを作成して見た目を整えていきます。まずはソースコードの範囲を選択し、[表] メニューの [ソースコードを表に変換] を選択します。区切り文字はソースコード中にタブが使われていなければ、初期設定のタブと改行で OK です。使われている場合は、ソースコードの中で使われていても文字を指定してください。

ここからソースコードです。

```

//コード以外の部分の変換
var parseML = function(src, omode){
    //img開頭. 画像をfiguresタグで置く
    if(src.match(/<img\s+src=(["\u0027"]+)/g)){
        var href = src.replace(/<img\s+src=(["\u0027"]+)/g, "<figure>$1</figure>");
        //src = src.replace(/<img\s+src=(["\u0027"]+)/g, "<img>"); //imgタグを削除
        //src = src.replace(/<img\s+src=(["\u0027"]+)/g, "<img>"); //imgタグを削除
        //img閉尾. srcがhrefでfile://
        src = src.replace(/<img\s+src=(["\u0027"]+)/g, "<img href$file://$1$");
        //img開頭. zoomClipに指定する必要(要素:zoom_data-clip属性)
        src = src.replace(/<img\s+href=(["\u0027"]+)/g, "zoom$(0-9)$1$g, \"$1$ data-zoom=$2$");
    };
    src = src.replace(/<data-zoom=(0-9)*>^/g, "$1$");
    src = src.replace(/<img\s+src=(["\u0027"]+)/g, "<clip$($0-9)+$1$g, \"$1$ data-clip$1$");
    src = src.replace(/<img\s+src=(["\u0027"]+)/g, "<img$($0-9)+$1$g, \"$1$");
    //ナビゲーションリンク
    src = src.replace(/<a href=(["\u0027"]+)/g, "<figure> $1</figure> ", '</figure> $1</caption>'); //alert($1);alert($1);alert($1);
    src = src.replace(/<a href=(["\u0027"]+)/g, "<img alt=$1$> $1</img> ", '</img> $1</caption>'); //alert($1);alert($1);alert($1);
    //リンクタグは文例( url ) の形に
    //リンクタグを多くする場合
    //src = src.replace(/<a href=(["\u0027"]+)/g, "$2 $(1)$");
    //brは改行コード。brにはあまり適さないタグに
    src = src.replace(/<br>/g, "\n");
    src = src.replace(/<br>/g, "\r\n");
    //画面構成とや回し文字などの確認試験 (div class=""で指定しているもの)
    //処理
    src = src.replace(/<div class=hen>(.*)</div>/g,
        "<div_hen> $1</div_hen>"); //alert($1);
    src = src.replace(/<div class=zuchou>(.*)</div>/g,
        "<div_zuchou> $1</div_zuchou>"); //alert($1);
    src = src.replace(/<div class=(<[^>]+>)/g, "<div$1> $1</div>"); //alert($1);
    src = src.replace(/<kbd> (([^>]+)) </kbd>/g, "<kbd$1> $1</kbd>"); //alert($1);
    //その他のspanタグ
    src = src.replace(/<span class=(["\u0027"]+)/g, "<span$1> $1</span>"); //alert($1);
    //(/データをタグ化する) どうしようか実装
    src = src.replace(/<thead> /g, "<thead>"); //alert($1);
    src = src.replace(/</thead> /g, ""); //alert($1);
    src = src.replace(/<tbody> /g, ""); //alert($1);
    src = src.replace(/</tbody> /g, ""); //alert($1);
    src = src.replace(/<th> /g, "<th>"); //alert($1);
    src = src.replace(/</th> /g, "</th>"); //alert($1);
    //olmdのときはliをolに変える
    if(olmd=true){

```

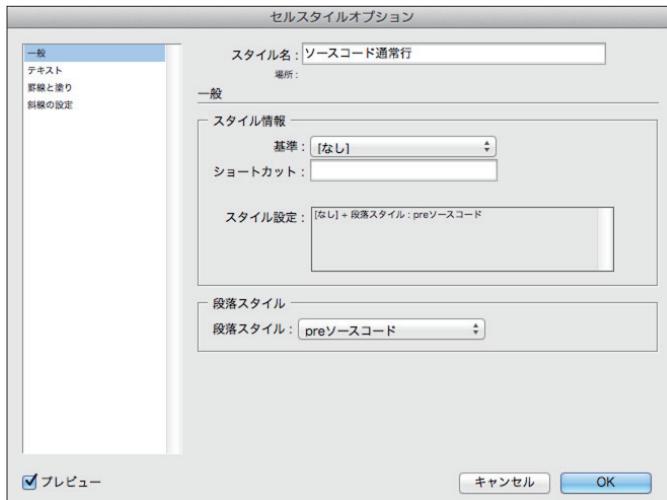
```

src = src.replace('<ol>|<ul>|<li>','');
//見出し(ol)～ulのcolorをわざる
if(options.midasableList == true){
  src = src.replace('<ol>/g','');
  src = src.replace('<ul>/g','');
  src = src.replace('<li>/g','');
  src = src.replace('<li>','');
  src = src.replace('<li>','');
}
//通常表示のcodeタグを削除
src = src.replace('<code>/g','');
src = src.replace('</code>/g','');
return src;
}

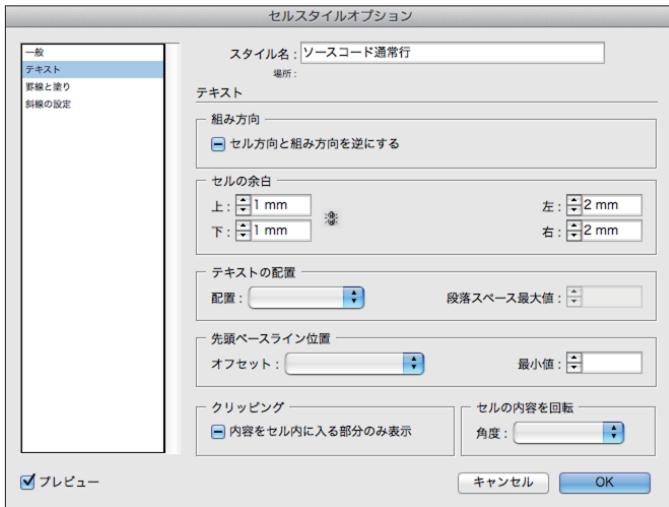
```

## ソースコードを表に変換

地味な表に変換されたので、見た目を整えていきましょう。セルスタイルを作成してセル内マージンを設定します。



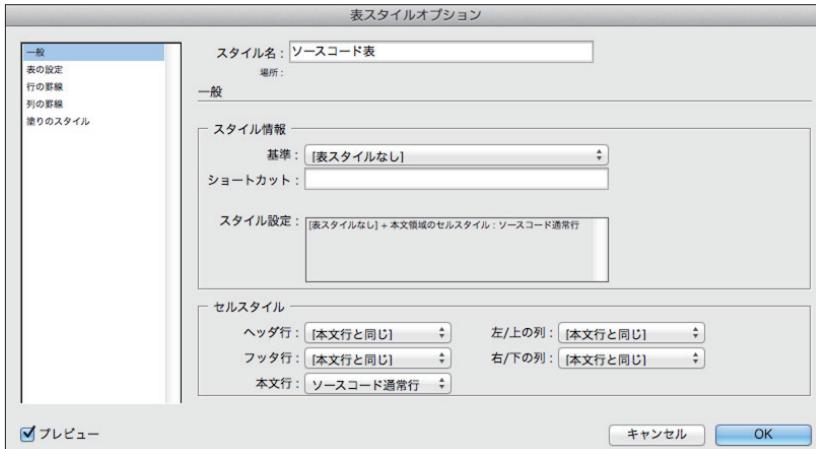
段落スタイルをソースコード用のものに指定



セルマージンを設定

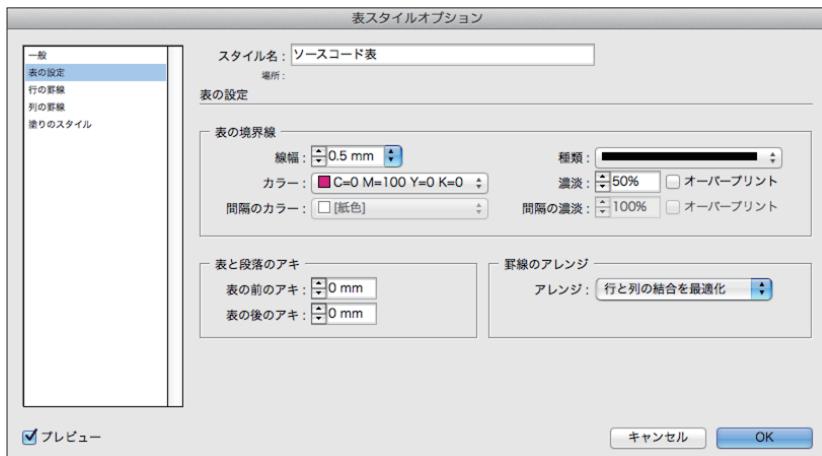
セルスタイルを使って、罫線や背景色を設定することができますが、その場合、表スタイル側の設定がオーバーライドされてしまいます。つまり、**表スタイルによる外枠の設定や交互に背景を塗りつぶす設定などが使えなくなる**ということです。今回は表スタイル側で罫線と背景色の設定を行うことにします。

表スタイルを作成し、[本文行] に先ほど作成したセルスタイルを指定しておきます。



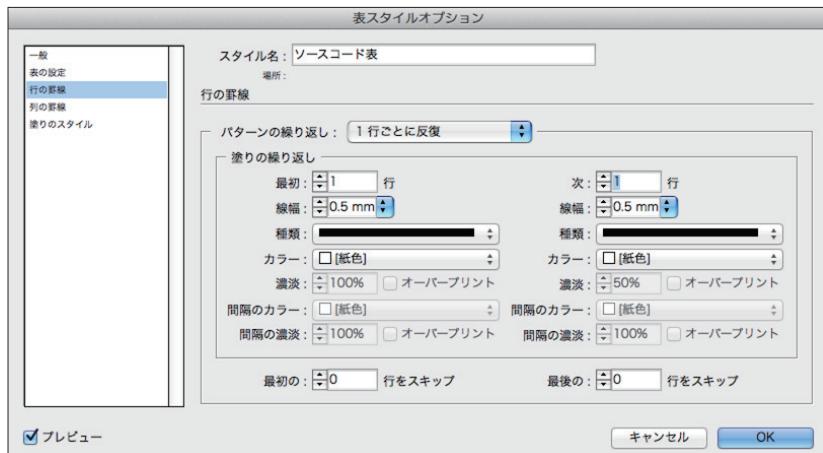
表スタイルを作成する

表の境界線（外枠の罫線）の設定を行い、[表と段落のアキ] を 0mm にします。[表と段落のアキ] は表の上下に空きを作るための設定ですが、今回はインライン親行用の段落スタイルを作る所以計算が狂わないよう空きをなくしています。



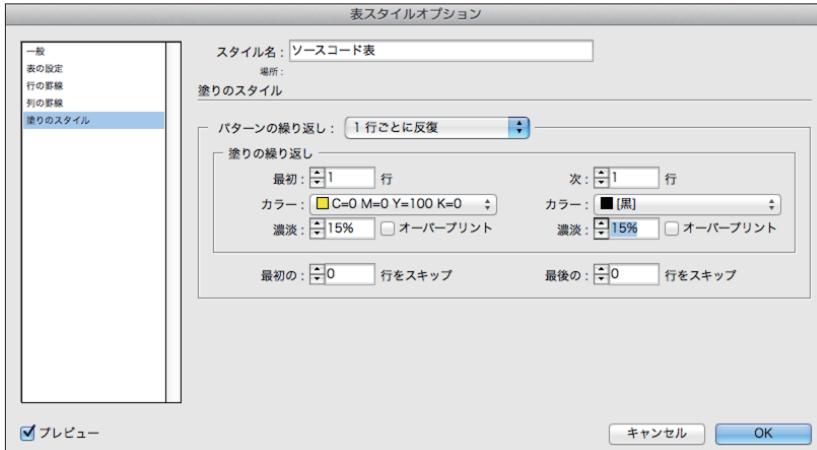
外枠と空きの設定を行う

続いて各行の間に、0.5mmの白い罫線を引きます。表スタイルでは交互にする設定しか行えないでの、2行とも同じ設定にします。



白い罫線の設定

最後に塗りのスタイルを設定します。ここでは薄い黄色とグレーで交互に背景を塗りつぶす設定とします。



背景色の設定

これで見た目がほぼできあがりました。

```

ここからソースコードです。

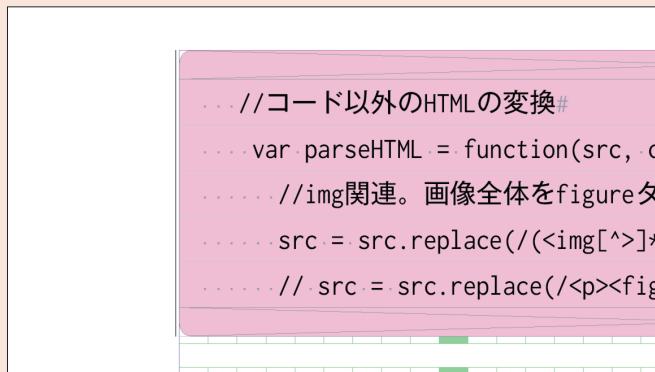
//コード以外のHTMLの変換
var parseHTML = function(src, olmode){
    //img関連。画像全体をfigureタグで囲む
    src = src.replace(/(<img[^>]*)/g, '<figure>$1</figure>');
    // src = src.replace(/<p><figure>/g, '<figure>');
    // src = src.replace(/</figure></p>/g, '</figure>');
    //img関連。src=""をhref="file://"に
    src = src.replace(/');
    //img関連。zoomとclipの指定をimg要素のdata-zoom、data-clip属性に
    src = src.replace(/(<img href="[^?]*")\?zoom=([0-9]*)/g, '$1" data-
zoom="$2" ');
    src = src.replace(/(data-zoom=[0-9]*") /g, '$1');
    src = src.replace(/(<img [^\>]*)\?clip=([0-9\+]*")/g, '$1" data-
clip="$2" ');
    src = src.replace(/\&clip=([0-9\+]*")/g, 'data-clip="$1"');
    //キャプションをfigcaptionに
    src = src.replace(/alt="([^\"]*)"\></figure>/, '/></figure>\n<figcaption>
<alert>CAP</alert>$1</figcaption>');
    src = src.replace(/alt="([^\"]*)"\><\p>/, '/><\p>\n<figcaption>alert>C
AP</alert>$1</figcaption>');
    //リンクは文字列 (url) の形に
    // 誤作動が多すぎるのでナシ
    // src = src.replace(/<a href="([^\"]*)"(<[^<]*>)\</a>/g, '$2 ($1 )');
    //brは改行コードに、hrはとりあえず孤立タグに
    src = src.replace(/<br>/g, '\n');
    src = src.replace(/<hr>/g, '<hr/>');
    //編集コメントや図中文字などの独自拡張 (div class="***"で指定しているもの
の処理

```

完成した状態

## コラム：表にもインライン親行がある

表はテキストの一部というイメージがありますが、実際にはインライン親行があり、セル内とは別に段落スタイルを設定できます。ただし直接は選択できないので、いったん表の前後の行にカーソルを置き、カーソルキーで表の親行まで移動します。表の高さサイズのカーソルが点滅したら、インライン親行が選択できている状態なので、そのままスタイルなどを設定します。



The screenshot shows a Microsoft Word document with a table containing source code. The first cell of the table has a pink background and contains the following code:

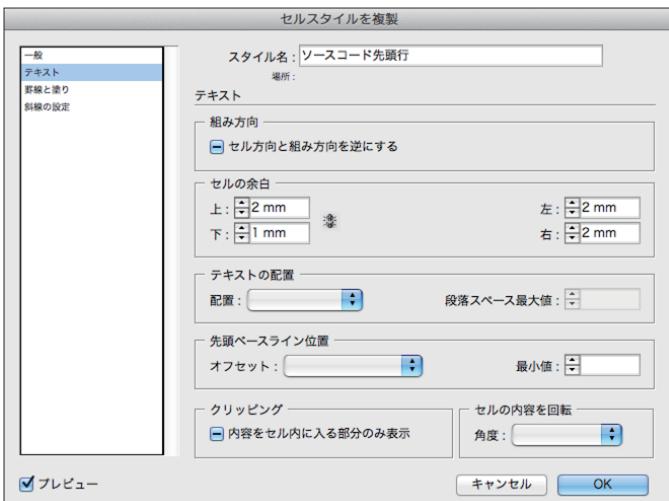
```
//コード以外のHTMLの変換#
var parseHTML = function(src, o
//img関連。画像全体をfigureタ
src = src.replace(/(<img[^>]*>)/g, <fig>
// src = src.replace(/<p><fig>
```

A green cursor is visible at the end of the third line of code. The status bar at the bottom of the screen shows "Table cell 1 of 1" and "Edit table".

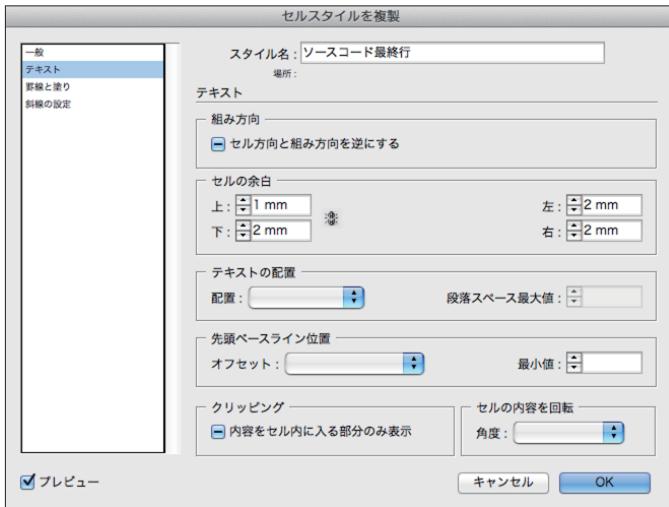
表のインライン親行にカーソルがある状態

### ▶ 表の先頭行と最終行だけスタイルを変える

先頭行と最終行は表の外枠とテキストの間隔を広げたいので、そこだけセルスタイルを作成して調整します。



先頭行では上マージンを 2mm に設定



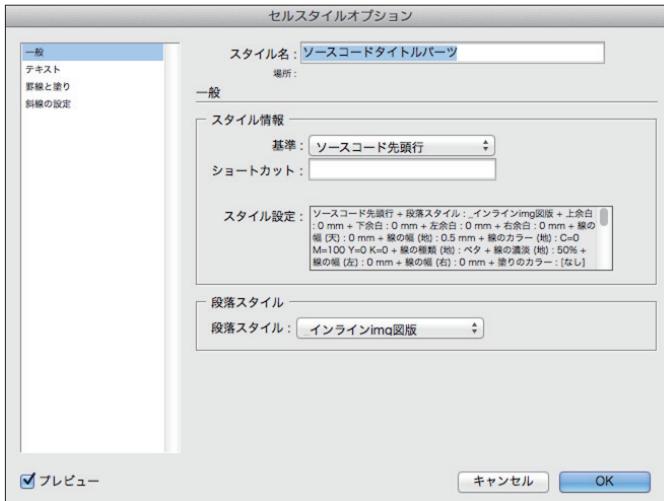
最終行では下マージンを 2mm に設定

これを手作業で設定するのは面倒なので、後半ではスクリプトで自動設定を行う方法を解説します。

## ソースコードにタイトルを付ける

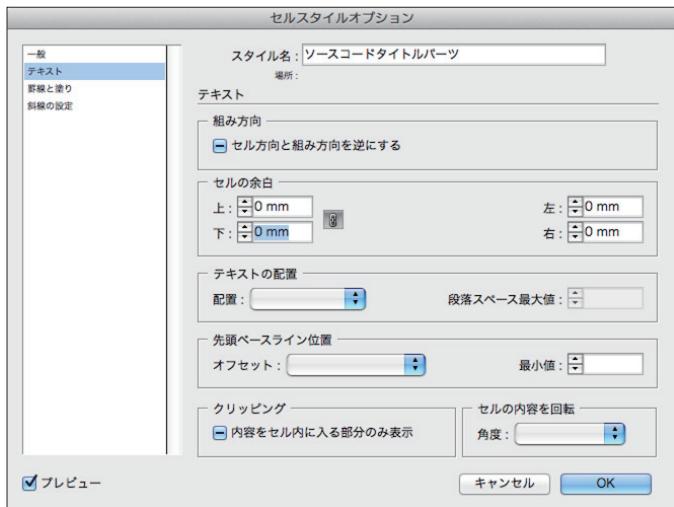
ソースコードには意外と凝ったデザインのタイトルが付くことがあります。ソースコードの枠と一緒に化していないデザインなら中見出しど同じように作成し、枠と一緒に化したデザインなら、**ソースコードの先頭行に罫線と背景が透明のセルを追加**し、そこにタイトルのパーツを挿入します。ここではちょっとわかりにくい後者の方法を説明します。

まずソースコードタイトルを入れるセルのためのスタイルを作成します。このスタイルと関連付ける段落スタイルは、タイトルパーツに合わせて高さが変わるように**行送りが「自動」になっているもの**を使ってください。図版のインライン行用のスタイルを流用しても構いません。



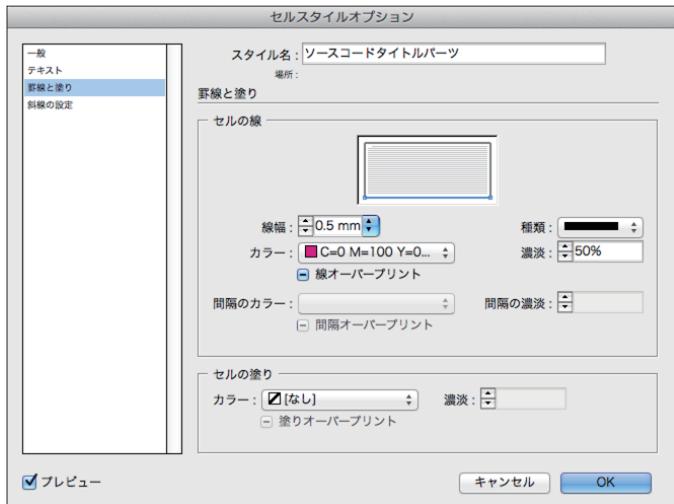
タイトルパート用のセルスタイルを作成

セルの余白を 0mm にします。



セルの余白をなしにする

罫線と塗りの設定を変更します。ここでは下罫線以外の罫線を 0mm にしてから、下罫線だけを外枠と同じマゼンタ 50% の 0.5mm 罫に設定します。そして、[セルの塗り] でカラーを「なし」(透明)にします。下罫線以外を 0mm にするのを忘れないでください。そうしないと表スタイル側の罫線設定がオーバーライドされません。



罫線と背景色の設定を行う

作成したセルスタイルを先頭行に設定し、タイトルpartsを用意します。

html2indtag.js

```
ここからソースコードです。|  
+-----+  
... //コード以外のHTMLの変換:  
... var parseHTML = function(src, olmode){#  
...   //img関連。画像全体を<figure>タグで囲む  
...   src = src.replace(/(<img[^>]*>)/g, '<figure>$1</figure>');#  
...   // src = src.replace(<p><figure>/g, '<figure>');#  
...   // src = src.replace(</figure></p>/g, '</figure>');#  
...   //img関連。src="#"をhref="file://"に#  
...   src = src.replace(/img src="#" /g, 'img href="file://$1"/');#  
...   //img関連。zoomとclipの指定をimg要素のdata-zoom、data-clip属性に#  
...   src = src.replace(/(<img href="[^>]*>)?zoom=[0-9]*"/g, '$1' data-  
zoom="$2");#  
...   src = src.replace(/data-zoom="[^>]*" /g, '$1');#  
...   src = src.replace(/(<img [^>]*>)\?clip=[0-9]+"/g, '$1' data-  
clip="$2");#  
...   src = src.replace(/\&clip=[0-9]+"/g, 'data-clip="$1"');#  
...   //キャプションをfigcaptionに#  
...   src = src.replace(/alt="[^"]*"/></figure>/, '/></figure>\n<figcaption>  
<alert>CAP</alert>$1</figcaption>');#  
...   src = src.replace(/alt="[^"]*"/></p>/, '/></p>\n<figcaption>alert>C  
AP</alert>$1</figcaption>');#  
...   //リンクは文字列 (url) の形に#  
...   // 誤動作が多すぎるのでナシ#  
...   // src = src.replace(<a href="[^"]*">[^<]*</a>/g, '$2 ($1 ') ;#
```

## タイトルパートを作成する

カット&ペーストで挿入すると完成です。

```
ここからソースコードです。』



## html2indtag.js



.... //コード以外のHTMLの変換

.... var parseHTML = function(src, olmode){#
....   //img関連。画像全体をfigureタグで囲む
....   src = src.replace(/(<img[^>]*>)/g, '<figure>$1</figure>');
....   // src = src.replace(/<p><figure>/g, '<figure>');#
....   // src = src.replace(/</figure></p>/g, '</figure>');#
....   //img関連。src="#"をhref="file://"に
....   src = src.replace(//g, '<img href="file://$/1"/>' );
....   //img関連。zoomとclipの指定をimg要素のdata-zoom、data-clip属性に#
....   src = src.replace(/(<img href="[^?]*")\?zoom=[0-9]*>/g, '$1' data-
.... zoom="$2" ');
....   src = src.replace(/(data-zoom=[0-9]* )"/g, '$1');
....   src = src.replace(/(<img [^?]*\?)\?clip=[0-9]+>/g, '$1' data-
.... clip="$2" ');
....   src = src.replace(/\&clip=[0-9]+"/g, 'data-clip="$1"');
....   //キャプションをfigcaptionに#
....   src = src.replace(/<alt="[^"]*" \/></figure>, '/></figure>\n<figcaption>
.... <alert>CAP</alert>$1</figcaption>');#
....   src = src.replace(/<alt="[^"]*" \/\></p>, '/></p>\n<figcaption>
.... AP</alert>$1</figcaption>');#
....   //リンクは文字列 (url) の形に
....   // 誤作動が多すぎるのナシ
```

タイトルパートが完成した

この方法の応用で、**角丸のソースコード枠**を作ることもできます。つまり、同じように先頭行と末尾行を背景が透明なセルにし、そこに角丸だけのパーツを挿入すればいいのです。

```
//コード以外のHTMLの変換  
var parseHTML = function(src, olmode){  
    //img関連。画像全体をfigureタグで囲む  
    src = src.replace(/(<img[^>]*>)/g, '<figure>$1</figure>');  
    // src = src.replace(/<p><figure>/g, '<figure>');
```

#### 角丸のソースコード枠

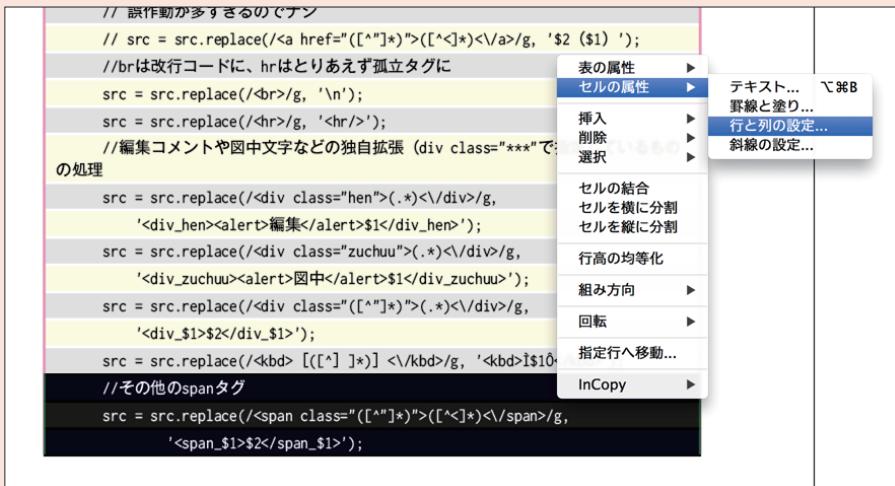
ただし、塗りがない罫線のみの角丸ソースコード枠を作るときは、この方式だとちょっと不都合があります。セルの罫線は枠境界を中心に外に広がるのに対し、セル内にインラインで挿入したパーツはセルの外にはみ出せないので、位置が微妙にずれてしまうのです。パーツをアンカー付きオブジェクトとして挿入して位置を微調整するという解決策もありますが、いずれにしても少し手間がかかります。

```
//コード以外のHTMLの変換  
var parseHTML = function(src, olmode){  
    //img関連。画像全体をfigureタグで囲む  
    src = src.replace(/(<img[^>]*>)/g, '<figure>$1</figure>');  
    // src = src.replace(/<p><figure>/g, '<figure>');
```

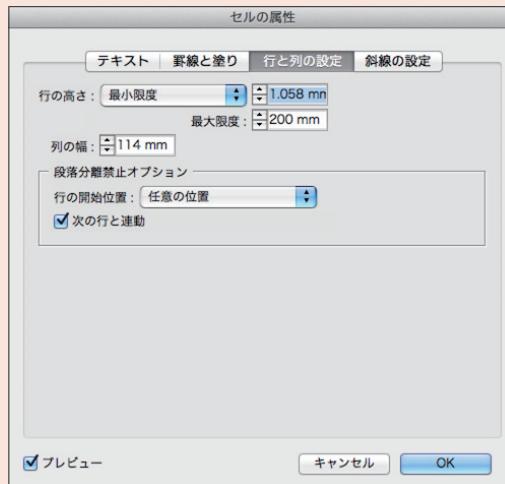
#### 罫線と角丸の両立は少し手間がかかる

## コラム：一部のセルを次ページに送るには

複数ページにまたがるソースコードでは、コードを読みやすくするためにページの最後に来たセルを次ページに送ることがあります。その場合はセルを選択して、右クリックメニューから【セルの属性】→【行と列の設定】を選択し、【次の行と連動】をオンにします。



次ページに送るセルを選択し、【行と列の設定】を選択する



【次の行と連動】をオンにする

これで選択したセルと、その次の行のセルが分離禁止になるため、同じページになるよう追い出されます。この機能の便利なところは、テキスト修正によって次ページに移動する必要がなくなった場合、いちいち解除しなくともいい点です。

## 1-4

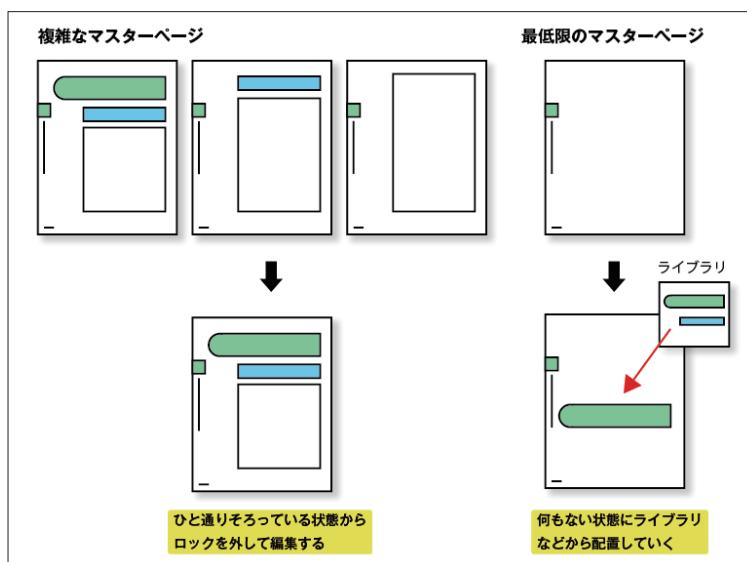
# 流し込みに適したマスターページ設計

マスターページの作成方針はデザイナー、プロダクションによってさまざまですが、この文書で解説する流し込み方式においては「シンプルなものほどいい」が原則です。ここではマスター作成のポイントをいくつか紹介します。

## マスターページはなるべく最低限に

編集プロダクションという立場柄、さまざまなデザインフォーマットを外部からもらいますが、マスターページの構造は作る人によってまったく違います。とはいっても次の3パターンのどちらかに分類できます。

1. マスターページにセクションタイトルや本文、コラムなどのパーツがひととおり配置されており、セクションタイトルあり／なしなどさまざまな種類のマスターページが用意されている。
2. マスターページは柱とノンブルなどの共通パーツのみが配置されており、種類も最低限（なしと章扉と本文の3つ）しかない。
3. マスターページを使っていない。スタイルも未設定。



3番目はDTP用のフォーマットとしては問題ですが、デザイン指定用と割り切ってDTPする前に調整する前提なら、かえって加工しやすくていいともいえます。

残る1と2のうち、DTP用フォーマットとしてどちらがいいかは、おそらく意見が分かれるところです。しかし、インラインで流し込む場合に限っていえば、2のシンプルな構造のほうが優れているといえるでしょう。インラインの流し込みではページは基本的に自動挿入するので、その際に色々なパートが配置されていると消す手間が増えるからです。

その他にも、1の方式では、「どのマスターページを使うべきか混乱する」「マスターページの再適用によって、パートが重なることがある」といった問題があります。そのあたりを考えあわせると、マスターページはなるべくシンプルにし、パートの配置についてはライブラリを使うか、別に見本ページを作つておくほうがいいでしょう。

## ▶レイヤーも少ないほうがいい

レイヤーを多数重ねたフォーマットもよく見かけますが、多くとも3つ以内にしたほうが作業しやすいです。このあたりは同じアドビ製品のIllustratorやPhotoshopと大きく異なる点です。InDesignにはマスターページもありますし、パート同士の関係がシンプルなので、レイヤーがなくても重ね順だけで解決できます。また、デザイナーとオペレーターが別人なので、レイヤー構造を複雑にすると、オペレーターがどのレイヤーに配置すればいいのか迷って、かえってエラーの原因になることがあります。

### マスターページ作成のポイント

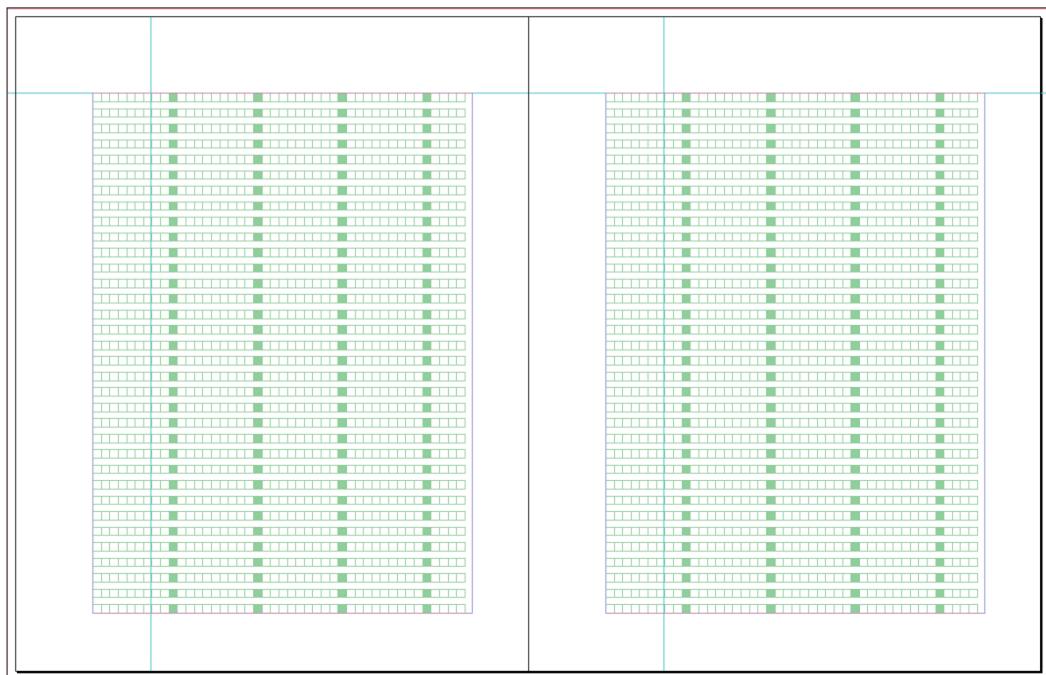
それでは自動流し込みに適したマスターページを作成するポイントを解説していきましょう。注意点のみに絞った解説となるので、細かい操作方法はInDesignの入門書を参考にしてください。

## ▶段落スタイル→文字スタイルの順で設定する

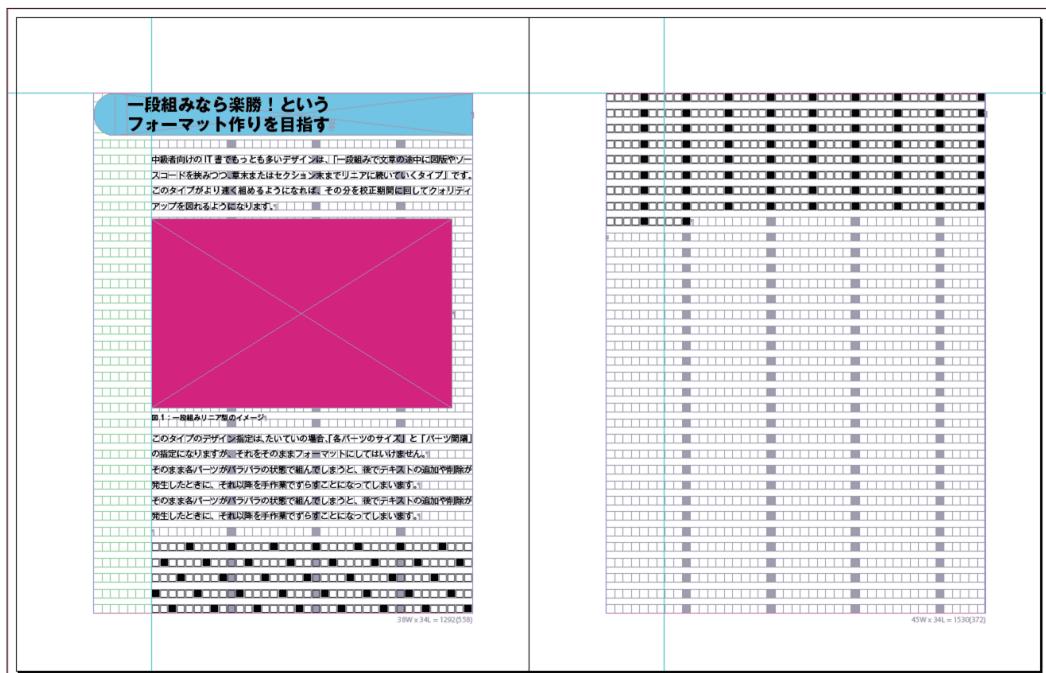
テキストに設定するスタイルには、段落スタイルと文字スタイルの2種類がありますが、なるべく段落スタイルを多めに使うよう心懸けます。段落スタイルは段落全体に1クリックで設定できるので効率が上がるからです。文字スタイルはテキストの一部だけを太字にしたい場合など、限定的に使います。

## ▶ちゃんとマージンを設定する

時折、本文の版面とマスターのマージン設定が合っていないフォーマットを見かけます。たいていはガイドラインが設定されていて手作業でフレームを配置する場合は困らないのですが、自動流し込み時はマージンに合わせてフレームが配置されるため、流し込んだ後ですべてのフレームを直すはめになってしまいます。



マージン設定が本文版面と合っていないマスターページ。版面を示すガイドラインが引かれているが……



流し込むとマージン設定に合わせてフレームが作られてしまう

ちなみに InDesign ではドキュメント作成時にマージンかレイアウトグリッドかを選びますが、マー

ジンのサイズさえ合っていれば、どちらでも大丈夫です。オンライン中心の作業だと、グリッドに合わせてレイアウトすることはほとんどありませんし、スマートガイドと「レイアウトグリッドにスナップ」機能が両立できないという問題もあります。ちなみに自分はスマートガイド多用派なので、「レイアウトグリッドにスナップ」は無効にします。

## 柱の設定

柱には一般的に章タイトルやセクションタイトルが入ります。これらはテキスト変数を使い、章タイトルやセクションタイトルの段落スタイルを参照させるのがセオリーです。章タイトルの場合は、章単位でファイルを作ることが多いので、マスターページにそのまま入力することもあるのですが、その場合は章扉などのタイトルと食い違わないように注意しなければいけません。章扉のタイトルをテキスト変数で参照しておけば、一番問題が起きにくいでしよう。

注意が必要なのは、**テキスト変数は改行できない**という点です。ですから、柱のセクションタイトルが複数行になるフォーマットだと、テキスト変数が使えなくなつて各ページに手作業で配置することになります。なるべく柱は1行にしてもらうよう、デザイナーと交渉することをおすすめします。

### ▶セクションタイトル中で改行せずに任意位置で折り返す

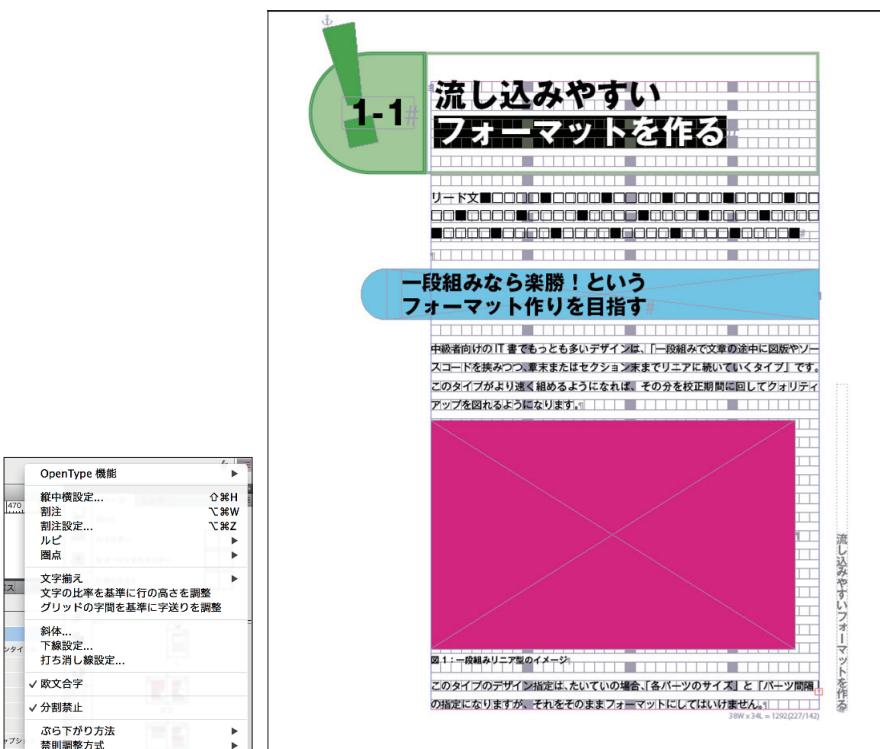
参照するセクションタイトルが改行されている場合、**1行目しか表示されなくなってしまいます。**



セクションタイトルの途中で改行されているため、柱が1行目までしか表示されない

## 1-4 流し込みに適したマスターページ設計

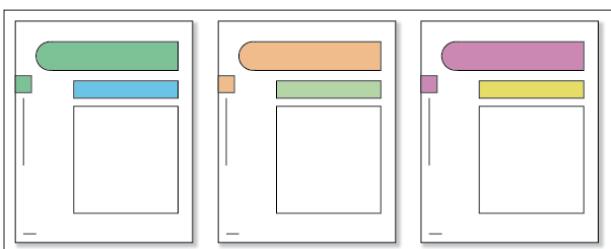
セクションタイトルの改行への対処法としては、改行ではなく「分離禁止」を使うというものがあります。2行目にしたい範囲を選択して分割禁止を掛けると、まとめて次行に送られるというしくみです。



分割禁止を使って自動的に折り返させる

## 章の色分けはスウォッチの入れ替えで対応

章の色分けのために、**色違いのマスターページを作っているフォーマット**を見かけることがありますが、マスターページを極力シンプルにするという方針からはあまり望ましいとはいえないかもしれません。というのは、章が長すぎたので初校後に複数の章に分割したり、章の順番を変更したりすることは、わりとよくあるからです。マスターページレベルで色が違う場合、その章はいちから作り直すか、各パートの色を1つずつ変更することになってしまいます。



章ごとにカラーが変わるデザイン

後から簡単に変更できるようにするには、マスターページは1つにして章のメインカラーのスウォッチを作っておき、それを変更するだけで全体の色を変えられるように設計しておきます。当然ながら、各種スタイルやパーツの色はメインカラーのスウォッチだけを使って設定しておかなければいけません。スウォッチのサムネイル部分をドラッグ＆ドロップすると、簡単に色設定を変更することができます。



ドラッグ & ドロップでメインカラーを変更する

## コラム：連番への対応

解説書ではセクション番号や図番号、表番号など、数多くの連番を使いますが、InDesignにはページ番号と箇条書きぐらいしか連番を作る機能がありません。そこで筆者は連番用のスクリプトで対応しています。

しくみは、特定の文字スタイルを設定した数字を正規表現で検索し、ページ・Y座標・X座標の順番で並べ替えてから連番を振るというものです。図番号や表番号など連番が数種類ある場合は、それぞれ異なる文字スタイルを設定しておけば個別に振ることができます。

縦書きの書籍や段組みでは正しく番号を振ることはできませんが、IT書はたいてい横書きなので大部分はフォローできます。



スーパー連番 (<https://github.com/lwohtsu/InddRenban>)

## 第 2 章

# Markdown からの XML 流し込み

## 2-1

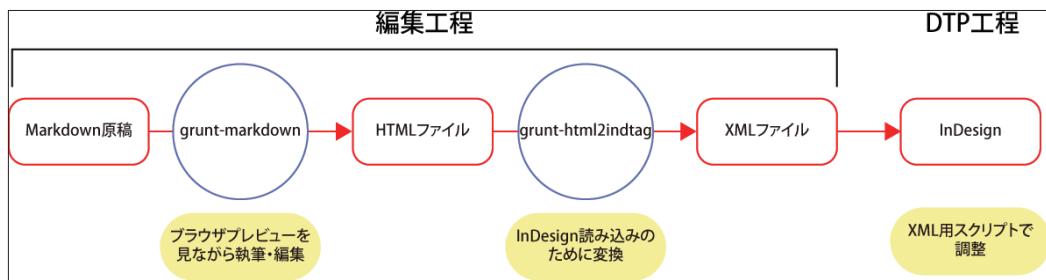
# 本書で解説する方式の概要

ここからは Markdown 原稿を HTML や XML に変換し、InDesign に流し込んで各スタイルの設定や画像の読み込み、ソースコード枠・表組みを自動で処理する方法を解説します。まずはその概要から説明しましょう。

## 最初の組みの効率をアップする

ここまで説明してきたインラインを活用した方法で変更・修正は効率化できますが、最初に組むときの手間は変わりません。ここからは **Markdown と XML 読み込み機能**を利用して、最初の組みを大幅に効率アップする方法を解説します。

筆者はこの方法で、300 ページ越えの仮組ゲラを 1 ~ 2 日で作成したことが何度かあります（テキスト編集や作図の時間を除く）。元原稿の状態や、フォーマットデザインの複雑さ、作図が必要かどうかなどによって、作業時間は変わってきてしまうのですが、**スタイル設定と図版挿入、表作成、コードの色分け**がほぼ自動で完了するため、大幅な改善が見込めるのは間違ひありません。



全体のワークフロー

## さまざまな自動組版の方法

スタイルなどを自動的に行ういわゆる自動組版は、TeX や md2iniao などすでに色々な方式が存在しますが、この文書で解説する方式は、もともとオープンソース系開発者の方から Markdown 原稿をもらう比率が増えてきたことからスタートしています。Markdown はテキストファイルにシンプルな記号を組み合わせて書式を指定する記法で、GitHub などのオープンソース向けサービスで使われてお

り、利用者も対応ツールも多いというメリットがあります。

## Markdown の例

# 大見出し：ソースコードを表組みで作る
## 中見出し：ソースコード枠の特徴
本文：IT 書だからといって必ずソースコードがあるとは限りませんが、プログラミング系の書籍を担当し始めると日常的に組むことになります。たいていのソースコードは次のような特徴を持ちます。
- 箇条書き：全体が枠などで囲まれる
- フォントは等幅で位置が崩れないようにする
- 行番号や罫線を入れることがある
- 10 行程度で納まることもあれば、複数ページにまたがるほど長いこともある
![画像：複数ページにまたがるソースコード](img/s4-f1g31.png)

筆者がはじめて Markdown 原稿をもらった頃（2012 年後半）は扱い方がわからず、「\* 強調 \*」などの記号をそのまま InDesign に流し込んで著者さんに注意されていたのですが、四苦八苦しているうちに HTML への変換までは自在にできるようになり、過渡的に HTML ファイルを組み指示として使いはじめました。InDesign の作業工程は変わりませんが、従来はプレーンテキスト内に書いた画像ファイル名を見ながら手作業で画像を配置してもらっていたので、仕上がりに近いイメージをオペレータに見せられるだけでも多少のメリットがあります。

その後、Markdown 原稿を InDesign のスタイルにダイレクトに変換できるという「md2iniao」の存在を知って採用を検討したのですが、WEB+DB プレスを前提に設計されているのがネックとなりました（Perl が苦手というのもあったのですが……）。編集プロダクションでは複数の出版社からさまざまなデザインフォーマットをもらうので、それらに汎用的に対応できなければ仕事になりません。

さらにしばらく試行錯誤を続けた結果、HTML にちょっと手を加えれば簡単に InDesign 用の XML が作れるということに気が付きました。

## HTML

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>doc</title>
</head>
<body>
<p>（書名）InDesign インライン流し込み道</p>
<div class="chapter">chapter1 流し込みやすいフォーマットを作る </div>
</body>
<h1 id="-"> インラインとグリッドを理解する </h1>
<h2 id="-"> 一段組みなら楽勝！というフォーマット作りを目指す </h2>
<p>中級者向けの IT 書でもっとも多いデザインは、「一段組みで文章の途中に図……中略……</p>

```
<p></p>
```

## InDesign 用の XML

```
<?xml version="1.0" encoding="UTF-8"?><story xmlns:aid5="http://ns.adobe.com/AdobeInDesign/5.0/" xmlns:aid="http://ns.adobe.com/AdobeInDesign/4.0/"><body>
<p>（書名）InDesign インライン流し込み道</p>
<div_chapter>chapter1 流し込みやすいフォーマットを作る</div_chapter>
<h1><alert>h1</alert> インラインとグリッドを理解する</h1>
<h2><alert>h2</alert> 一段組みなら楽勝！というフォーマット作りを目指す</h2>
<p>中級者向けのIT書でもっとも多いデザインは、「一段組みで文章の途中に図……中略……」</p>
<p><figure><img href="file:///img/fig1.png" /></figure>
<figcaption><alert>CAP</alert>一段組みリニア型のイメージ</figcaption></p>
```

Markdown からの HTML 変換には、たまたま **grunt-markdown** というツールを使用していました。これが書き出す HTML には **highlight.js** で色分けした結果が静的に反映される という特徴があり、そのおかげでソースコードを自動的に色分けする道も開けました。

## 色分け対応したソースコード部分の HTML

```
<h6 id="html1">HTML</h6>
<pre><code class="lang-html"><span class="hljs-doctype">&lt;!DOCTYPE html&gt;</span>
<span class="hljs-tag">&lt;<span class="hljs-title">html</span>&gt;</span>
<span class="hljs-tag">&lt;<span class="hljs-title">head</span>&gt;</span>
<span class="hljs-tag">&lt;<span class="hljs-title">meta</span> <span
class="hljs-attribute"> charset</span>=<span class="hljs-value">"utf-8"</span>&gt;</
span>
<span class="hljs-tag">&lt;<span class="hljs-title">title</span>&gt;</
span>doc<span class="hljs-tag">&lt;<span class="hljs-title">title</span>&gt;</span>
```

ソースコードの色分けは、従来工程だと非常に手間がかかる作業で、印刷したコードをマーカーで塗り分けてオペレータに手作業で色を設定してもらったり、一回 rtf に変換してから読み込むといった方法で処理していました。それがスウォッチや文字スタイルの設定を済ませれば完全自動で処理できるようになったのは画期的です。

後は InDesign のスクリプトを組み合わせて、少しずつ組みやすく改良していく、2014 年の夏頃にようやくこの文書で説明する方式が固まりました。XML に簡単に変換できることに気付くまでに 1 年以上、そこから仕事の合間に Grunt プラグインや InDesign スクリプトの書き方を覚えたり、フォーマットの作り方を工夫したりしていたので、Markdown 原稿をもらいはじめた時期から計算すると 2 年近くかかっています。

## 本書で説明する方式のメリット・デメリット

本書で説明する方式のメリットとデメリットを挙げておきます。

### ▶ メリット

- Markdown が開発者に広く普及しているので、著者に覚えてもらいやすい。プレビュー用のツールも色々ある。
- 画像も自動配置できる。
- ソースコードを色分けできる。
- 表も自動的に組める。
- 最終的に InDesign で組み上げるので従来工程との相性がよく、ほとんどのデザインフォーマットに対応できる。

### ▶ デメリット

- Markdown → InDesign の一方通行なので、一回組んだ後は Markdown 原稿を修正しても対応できない。元原稿が変わったら、いちから組み直すことになる。
- マルチユースは目的としていない。途中に HTML を挟むので EPUB などに変換することは可能だが、InDesign 側で文字などを編集してしまった場合は、手作業で同期することになる。
- Markdown ではタグが足りないことがある。
- 段組みは苦手。2段組み以上のフォーマットだと手作業でのレイアウトが必要。

ひと言でいうと従来工程（InDesign DTP）との相性重視、電子書籍などのマルチユース軽視という感じでしょうか。ただ、編集工程で Markdown や HTML を扱うことになるので、今後電子書籍が中心となった場合でも、乗り換えの助けにはなるだろうと考えています。

## 2-2

# 編集環境の準備

Markdown を HTML や XML に変換するために、タスクランナーの Grunt といいくつかのプラグインをインストールします。この文書に必要な話にしぼって解説するので、各ツールの正式な使い方は割愛します。

## 編集環境の準備

Markdown をブラウザでプレビューする処理や、XML に変換する処理のためにタスクランナーの **Grunt** (<http://gruntjs.com/>) を使用しています。タスクランナーというのは主に Web 制作などで使われている一種のバッチ処理ツールで、「フォルダ内にファイルが保存されたら自動的にコンパイルや変換処理を行って仕上がりファイルを作成する」といった処理を自動化するために使われています。

色々な方法の中から Grunt を選んだ理由は次の 3 点です。

- grunt-markdown プラグインが書き出す HTML は、highlight.js によるコード色分けもされていて加工しやすい。
- プレビュー機能を含む Markdown 専用エディタは重い。
- 変換処理はすべて Grunt が担当するため、エディタは自分が使いやすいものを選べる。
- JavaScript ならある程度自分でカスタマイズできる。

## ▶ grunt-cli のインストール

さて、インストールの手順を解説していきましょう。この文書では Mac OS X へのインストール方法のみを解説します。Windows でも Grunt の利用は可能ですが、色々とインストールしないといけないものが増えるため、やや面倒になります。

Grunt を利用するには **node.js** が必要です。node.js は、ブラウザ外で JavaScript を実行できるシステムで、Mac OS X には標準でインストールされています。ただし、バージョンが違うとうまく動作しないことがあるので、ターミナルから「`node -v`」と入力してバージョンを確認してください。この文書では OS X 10.9.5 に node.js の v0.12.0 をインストールした環境で解説します。

node.js の用意ができたら、次のコマンドを入力して、**grunt-cli** というものをインストールします。これでこの Mac 内で Grunt を利用できるようになります。この作業は一回だけ行えば OK です。

```
sudo npm install -g grunt-cli
```

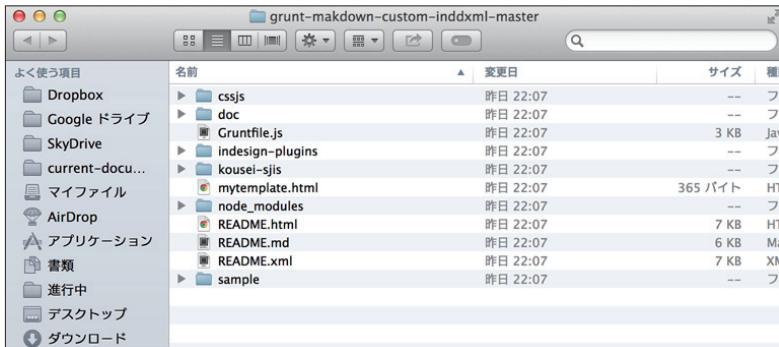
## ▶プロジェクトフォルダの作成

この後は書籍ごとに作業用のフォルダ（以降プロジェクトフォルダと呼びます）を作成し、その中に Grunt とプラグインをインストールしていくのですが、書籍を作るたびにインストール作業を行うのも面倒ですし、他人にも教えにくいので、筆者はインストール済みの作業フォルダを GitHub や NAS に保存しておき、それをコピーして使うようにしています。

公式に推奨されている Grunt の使用方法ではない（プラグインのアップデートができないなどの問題がある）ので、正しい使い方は Grunt について解説している Web サイトや書籍を参照してください。

GitHub 上の公開場所 (<https://github.com/lwohtsu/grunt-markdown-custom-inddxml>)

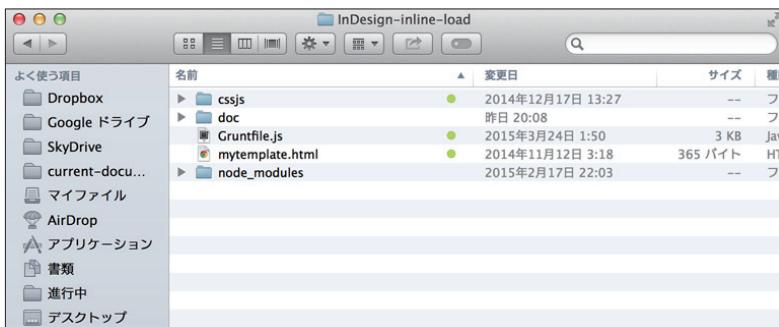
GitHub のページの右側に「Download ZIP」というボタンがあるので、それをクリックしてファイルをダウンロードしたら、**日本語を含まないファイルパス内**に解凍し、プロジェクトに合わせてリネームします。**ファイルパスに日本語が含まれていると Grunt の実行が失敗する**ので注意してください。



解凍直後のフォルダ

プロジェクトフォルダ内の各ファイル・フォルダの内容は次のとおりです。

ファイル・フォルダ名	説明
cssjs	プレビュー用のCSSとJavaScriptファイルを入れておくフォルダ(P.63参照)。
doc	Markdown原稿を入れるフォルダだが、リネームしてもいいし、このフォルダ外に原稿を置いても問題ない。
Gruntfile.js	Gruntの設定ファイル。
indesign-plugins	InDesign用のプラグイン(P.■■■参照)。インストール後は削除しても問題ない。
kousei-sjis	Just-Rightという校正ツールを使用するために、Shift JISに変換したHTMLファイルを保存しておくフォルダ。削除しても問題ない。
mytemplate.html	プレビュー用HTMLの元になるテンプレートファイル。CSSへのリンクなどを記述しておく(P.63参照)。
node_modules	Gruntやプラグインがインストールされているフォルダ。
README.html～xml	使用説明ファイル。削除しても問題ない。
sample	テスト用のファイル。削除しても問題ない。



書籍名に合わせてリネームし、最低限必要なファイル・フォルダに絞った状態

これで準備は完了です。続いて使い方を説明していきましょう。

## 2-3

# Markdownで原稿を編集する

簡単な Markdown の説明と、HTML プレビューを確認しながら編集作業を進める方法を解説します。

## Markdown記法の簡単な説明

Markdown はプレーンテキストに「#」などの記号を入れたものです。プレーンテキストで原稿を書く場合でも、見出しの行頭に「■」を入れるといったルールにすることが多いので、それを規則化したものだと考えればなじみやすいでしょう。HTML の h1 などのタグと一一で対応しており、簡単に変換することができます。また、Markdown の記号だけで対応しきれないときは、HTML タグを直接書き込むことも可能です。

ただし、Markdown にはいくつかの方言があります。後であらためて説明しますが、原稿を依頼するときは「Markdown をお願いします」だけではトラブルが起るので注意してください。この文書の方式では GFM (GitHub Flavored Markdown) という GitHub 上で使われている記法をベースにしています。

The screenshot shows the GitHub Help page for "Writing on GitHub / GitHub Flavored Markdown". The page title is "GitHub Flavored Markdown". It explains that GitHub uses "GitHub Flavored Markdown," or GFM, across the site--in issues, comments, and pull requests. It differs from standard Markdown (SM) in a few significant ways, and adds some additional functionality. It links to "Markdown Basics" for more features. A sidebar on the right lists "Article versions" for GitHub.com, Enterprise 2.2, Enterprise 2.1, Enterprise 2.0, and Enterprise 11.10.340.

**Differences from traditional Markdown**

**Multiple underscores in words**

Where Markdown transforms underscores (\_) into italics, GFM ignores underscores in words, like this:

- > `wow_great_stuff`
- > `do_this_and_do_that_and_another_thing`.

This allows code and names with multiple underscores to render properly. To emphasize a portion of a word, use asterisks (\*).

**URL autolinking**

GitHub Flavored Markdown (<https://help.github.com/articles/github-flavored-markdown/>)

練習も兼ねて次の Markdown 原稿を入力してみてください。テキストエディタは何を使ってもかまいません。保存するときに拡張子を「.md」にすれば、Markdown ファイルとなります。

## test.md

```
## h1 見出し  
## h2 見出し  
### h3 見出し  
#### h4 見出し  
##### h5 見出し  
##### h6 見出し
```

記号を付けずに入力した行は本文（p 要素）になります。

ただし、改行は無視されます。

改行したい場合は 1 行空けます。

br 要素相当の段落内改行をしたい場合は、行末に 2 個以上の段落スペースを入れます。ただし InDesign の強制改行になるわけではないので、使いどころに注意が必要です。

- 箇条書きは行頭に「-」か「\*」を入れ、半角スペースを空けます。
- 半角スペースは忘れがちなので注意してください。
- 箇条書きの前にアキ行を入れないと箇条書きになりません。

1. 番号付き箇条書きの場合は「1. 」とします。

2. ピリオドの後はやはり番号が必要です。

3. 番号は自動で振り直されるので、元原稿と InDesign 上でずれることがあります。注意してください。

リンクを入力するには、「[リンク文字列] (URL)」の形式で入力します。このように本文中に Markdown や HTML タグで使われる記号を入れる時は、「` `」（バッククオート）で囲みます。

[google] (<https://www.google.co.jp>)

画像を挿入したい場合は、リンクの前に ! を付けます。

![alt 属性になる文字列] (test.JPG)

なお、この例にはリンク文字列の入力方法がありますが、実際に書籍原稿でリンクの記法を使うことはありません。書籍に掲載してほしいとおりに入力するよう依頼します。

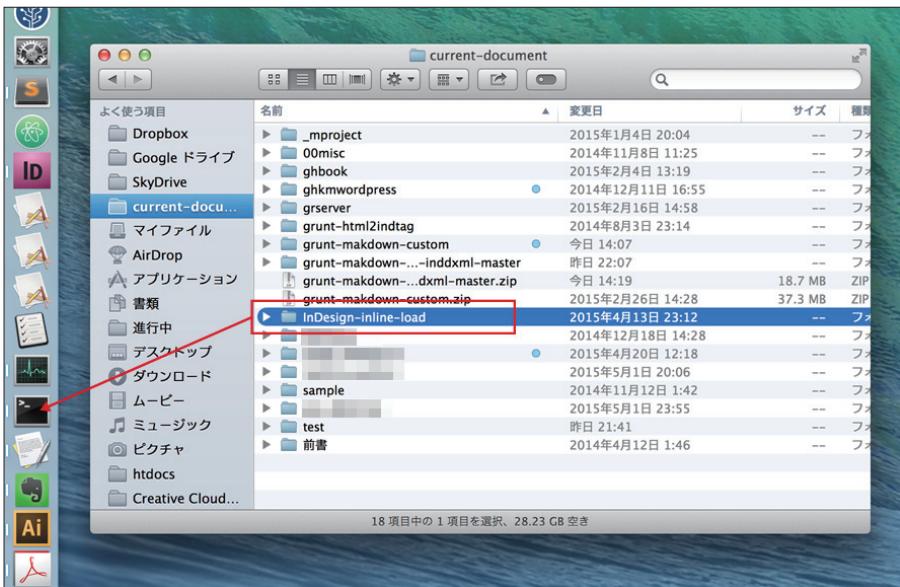
## Gruntの実行

入力し終わったら、test.md などの適当なファイル名でプロジェクトフォルダ内に保存します。

続いて Grunt を実行します。プロジェクトフォルダをターミナルのアイコンにドラッグ＆ドロップ

## 2-3 Markdownで原稿を編集する

すると、プロジェクトフォルダをカレントフォルダとしてターミナルが開かれます。



プロジェクトフォルダをターミナルのアイコンにドラッグ&ドロップ

\$ というプロンプトが表示されているので、その後で「grunt」と入力して [return] キーを押します。

```
~/current-document/InDesign-inline-load — node — 80x24
Last login: Thu May  7 21:27:21 on ttys003
OhtsuYuchiro-no-Macbook-Pro:InDesign-inline-load ohtsuy1$ grunt
Running "markdown:all" (markdown) task
File "doc/InDesign-inline-load-chap1-2.html" created.
File "doc/InDesign-inline-load-chap3.html" created.
File "doc/test.html" created.

Running "html2indtag:dist" (html2indtag) task
>> File "doc/InDesign-inline-load-chap1-2.xml" created.
>> File "doc/InDesign-inline-load-chap3.xml" created.
>> File "doc/test.xml" created.

Running "watch" task
Waiting...
```

gruntと入力すると、Gruntによってファイルが変換される

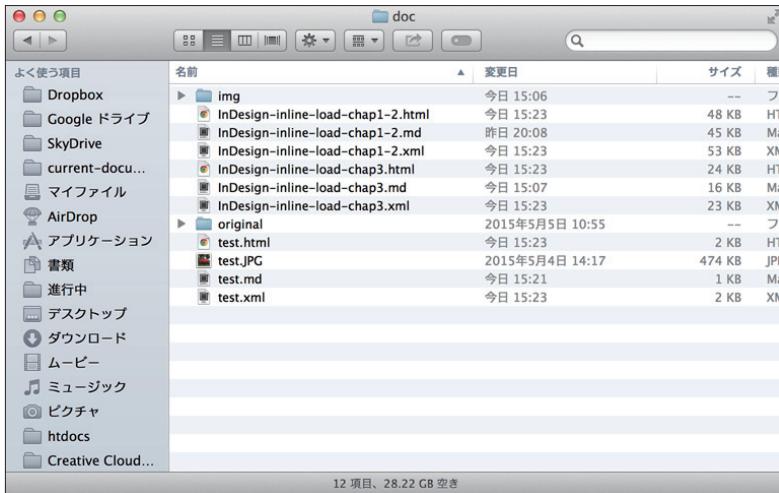
「File ○○○ created.」というメッセージが表示され、HTML ファイルと XML ファイルが生成されたことが確認できます。後で説明しますが、XML ファイルが生成されるのは grunt コマンドを実行した直後だけです。

その後「Running "watch" task」「Waiting...」と表示されているのは、grunt-contrib-watch というプラグインが実行中でプロジェクトフォルダ内を監視していることを示しています。Markdown ファイルを保存し直したり新規作成したりすると、自動的に HTML ファイルが更新されます (XML ファイルは更新されません)。

Gruntによる監視を終了したい場合は、[control] + [C] を押します。

## ▶ プレビュー結果を確認する

Finder でプロジェクトフォルダを見ると、Markdown ファイルと同じフォルダ内に HTML ファイルが作成されていることがわかります。元が「test.md」なら「test.html」というファイルになります。これをブラウザで開くと、結果が確認できます。



HTML ファイルを開く

grunt-contrib-watch によるファイル更新の監視と LiveReload というブラウザの自動リロードが有効になっているので、テキストエディタ上で Markdown 原稿を書き換えて保存すると、ブラウザ側のプレビューも更新されます。

なお、毎回全部の Markdown 原稿が変換されるので、ファイル数が増えると変換にかかる時間も長くなります。**ignore** という名前のフォルダを作ってそこに Markdown ファイルを移動すれば、grunt-contrib-watch に無視させることができます。

# ●-●h1見出し

## h2見出し

### h3見出し

#### h4見出し

##### ●h5見出し

###### h6見出し

記号を付けずに入力した行は本文（p要素）になります。ただし、改行は無視されます。

改行したい場合は1行空けます。

br要素相当の段落内改行をしたい場合は、行末に2個以上の段落スペースを入れます。ただしInDesignの強制改行になるわけではないので、使いどころに注意が必要です。

- 箇条書きは行頭に「-」か「\*」を入れ、半角スペースを空けます。
- 半角スペースは忘れないで注意してください。
- 箇条書きの前にアキ行を入れないと箇条書きになりません。

1. 番号付き箇条書きの場合は「1.」とします。

2. ピリオドの後はやはり番号が必要です。

3. 番号は自動で振り直されるので、元原稿とInDesign上ですれることがあります。注意してください。

リンクを入力するには、[リンク文字列](URL)の形式で入力します。このように本文中にMarkdownやHTMLタグで使われる記号を入れる時は、「（バッククォート）で囲みます。

[google](#)

画像を挿入したい場合は、リンクの前に!を付けます。

!(test.JPG)



図■ : alt属性になる文字列

ブラウザによるプレビュー

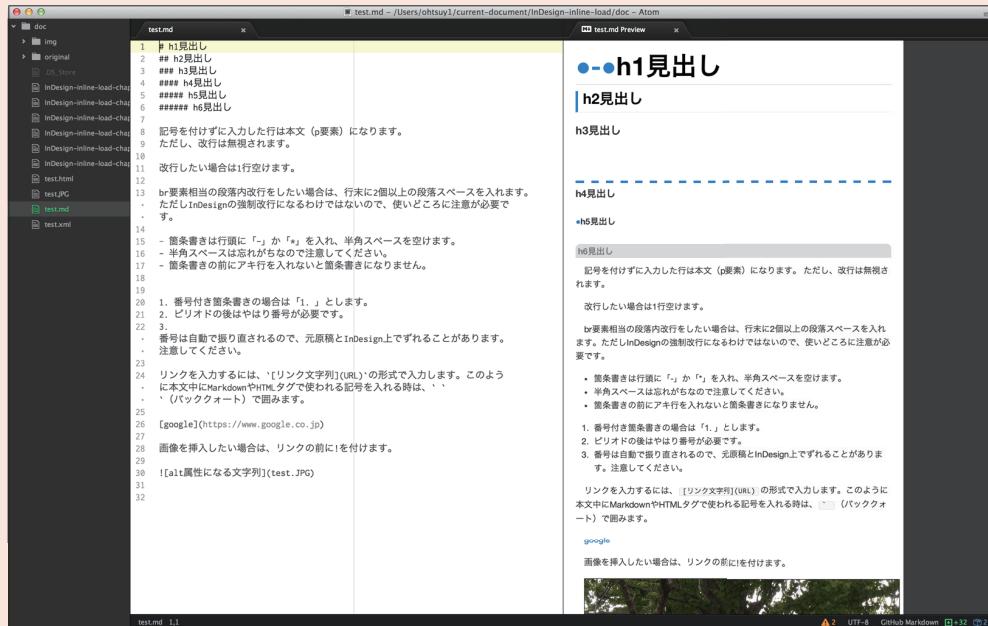
## コラム：Markdown 原稿を依頼するときはガイドラインが必要

Markdown での原稿を外部に依頼する場合は、先にガイドラインを決めておいてすり合わせておかなければいけません。Markdown に方言があるのも理由の 1 つですが、本ごとに見出しは何ランクまで許されるのか、図やソースコードにタイトルが必要かどうかといったフォーマット上のルールが異なるからです。

また、InDesign の制限で、**画像ファイル名は日本語不可（XML 読み込みできなくなる）** ということも伝えておく必要があります。

筆者は、たいていガイドライン兼原稿見本の Markdown ファイルと、それを元に組んだ PDF ファイルをセットで渡します。そうしておけば「#」を付けた見出しが、書籍ではどんなデザインになるのかが理解しやすくなります。

また、その際に Markdown を手軽に確認できるツールとして、GitHub の ATOM (<https://atom.io/>) という無料のテキストエディタを紹介するようにしています。というのは、プレビューを見ずに Markdown を間違えずに入力するのはほぼ不可能で、画像のリンク切れや見出しランクの間違いなどが多発するからです。ATOM は Grunt に比べて導入が容易ですし、当然ながら GFM 対応なので方言の問題も解決できます。また、Themes の CSS を編集すれば、書籍の仕上がりに近くなるようプレビューを調整することも可能です。ATOM で執筆するよう強要する必要はありませんが、時々プレビューを確認してもうようお願いすることは大変重要です。



ATOM で Markdown をプレビューする

## Markdownプレビューのカスタマイズ

Markdown プレビューの外観を書籍の仕上がりに近づけておけば、いちいち InDesign で組まなくともブラウザ上で仕上がりイメージを把握しやすくなります。

プロジェクトフォルダ内の「**mytemplate.html**」を開いてみてください。この HTML ファイルは grunt-markdown が HTML 変換するときのテンプレートで、ここで参照する CSS ファイルや JavaScript ファイルなどを指定します。Markdown 原稿と CSS・JS ファイルの相対位置が変わった場合は、正しく参照されるようパスを変更しておく必要があります。

### mytemplate.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">

    <title>doc</title>
    <script src="../cssjs/jquery-2.0.3.min.js"></script>
    <script src="../cssjs/addimagename.js"></script>
    <link rel="stylesheet" href="../cssjs/myrule.css">
    <link rel="stylesheet" href="../cssjs/hljsstyles/xcode.css">
  </head>
  <body>
    <%=content%>
  </body>
</html>
```

**cssjs** というフォルダ内にある myrule.css という CSS ファイルが、プレビューの外観を決めていきます。書籍のデザインに合わせて適当に書き換えてください。

### myrule.css

```
div.chapter{
  margin-top: 3em;
  font-size: 2.5em;
  border-top:solid 4px #F08;
  border-bottom:solid 4px #F08;
}

div.hen{background: #ff0; color:#D00; font-size: 0.7em;}

img {
  display:block;
  border: solid 1px #444;
  max-width:600px;
```

```

}
blockquote{
  font-weight: bold;
  background:#ff0;
  margin: 1em 0;
  padding-left: 1em;
}
body {
  font-family:sans-serif;
  width:36em;
}
.....後略.....

```

なお、この CSS ファイルの設定は、InDesign の組みにはまったく影響しないことに注意してください。InDesign に持ち込まれるのは HTML から変換した XML なので、見た目を合わせるには CSS 側と InDesign 側それぞれで調整する必要があります。

## ■ その他の Markdown のルール

Markdown にはここまでに説明したもの以外にもさまざまなルールがあります。たとえば「\*\*」で文字列を囲むと重要 (strong 要素) になる、行頭に「>」を入れると引用 (blockquote 要素) になる、「|」(パイプ) と「-」(ハイフン) の組み合わせで表を作成できる、ソースコード (pre、code 要素) を挿入したい場合は、「```」(バッククオート 3つ) で囲む、といったものです。

その他にパーソナルルールとして、著者やオペレータ向けのコメントは「<div class="hen"></div>」で囲むことになっています。毎回タグを打ち込むのは面倒なので、スニペットなどを登録しておくと便利です。

ソースコードを挿入する「```」の後に「```javascript」のように言語を表すキーワードを入れておくと、より正確に色分けされるようになります。キーワードを知りたい場合は、プロジェクトフォルダ内の「node\_modules/grunt-markdown/node\_modules/highlight.js/lib」の中にある「index.js」というファイルを開いてください。registerLanguage 関数の最初の引数が、言語のキーワードとなります。

### index.js

```

var Highlight = require('./highlight');
var hljs      = new Highlight();

hljs.registerLanguage('1c', require('./languages/1c'));
hljs.registerLanguage('actionscript', require('./languages/actionscript'));
hljs.registerLanguage('apache', require('./languages/apache'));
hljs.registerLanguage('applescript', require('./languages/applescript'));
hljs.registerLanguage('xml', require('./languages/xml'));
hljs.registerLanguage('asciidoc', require('./languages/asciidoc'));
.....後略.....

```

## コラム：Grunt を複数同時に動かすには

ターミナルのウィンドウを複数同時に開けば、Grunt (grunt-contrib-watch) を複数同時に実行できます。ただし、ブラウザの自動リロードを行う LivePreview が使用するポート番号が重複しないようずらす必要があります。Gruntfile.js を開き、プロジェクトごとに「livereload:」の後の数値を変更してください。

### Gruntfile.js

```
html: {
  files: ['**/*.html', '!kousei-sjis/**/*.html', '!**/ignore/**/*.html'],
  tasks: [ ],
  options: {
    livereload: 35732
  }
},
```

ただし、複数の Grunt を実行するとかなり重くなるので、基本的には使っていない grunt-contrib-watch を停めて、実行するのは一度に 1 つにすることをおすすめします。

## コラム：キーフォントを自動設定するには

grunt-html2indtag では kbd タグを利用して、キーフォント (PIXymbolsKanakey) を自動設定できます。[Ctrl] + [C] などです。Markdown 側で次のように記述しておくと、[] の部分がキーフォントの両横の囲みの文字に置換されます。後は kbd タグをキーフォントを使った文字スタイルにマップします。

```
<kbd> [Ctrl] </kbd> + <kbd> [C] </kbd> を押します。
```

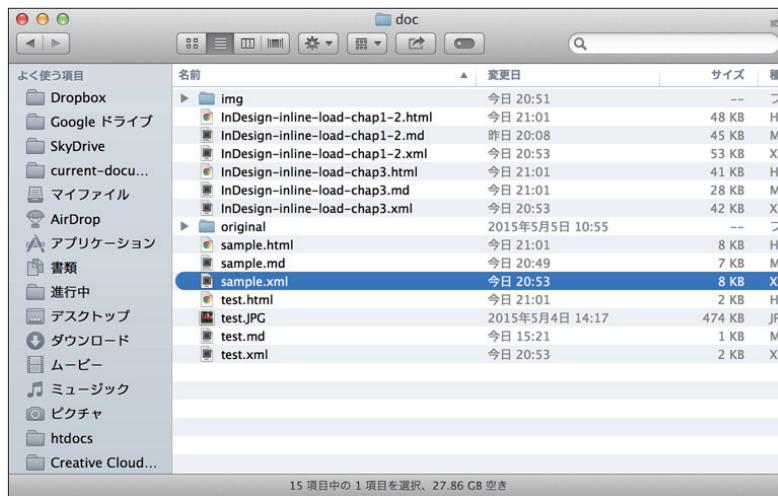
## 2-4

# XML変換とInDesignへの流し込み

いよいよ流し込みの本番です。Grunt を使って HTML を XML に変換し、InDesign に読み込みます。タグを適切なスタイルにマップすることが重要です。

## Markdown原稿をXMLに変換する

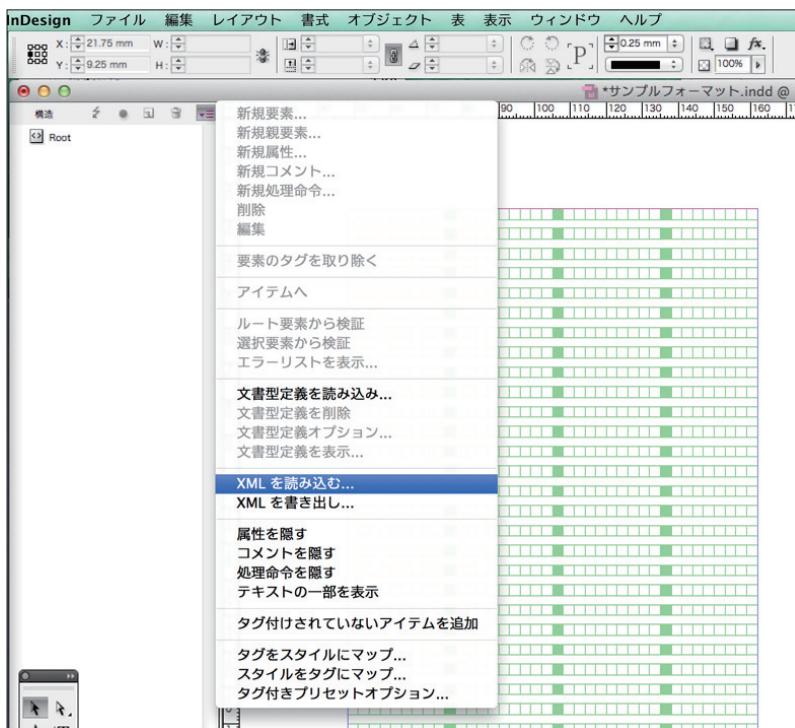
Markdown 原稿を XML に変換するには、ターミナルに「grunt xml」というコマンドを入力します。これで Markdown から HTML に変換し、さらに XML への変換が行われます。このコマンドでは grunt-contrib-watch による監視は行われず、すぐに Grunt が終了します。XML 変換は連続で行うには重すぎたためです。「grunt」コマンドによる変換では最初の一回しか XML 変換しないため、InDesign に流し込む前に必ず「grunt xml」を実行してください。



XML ファイルが生成された

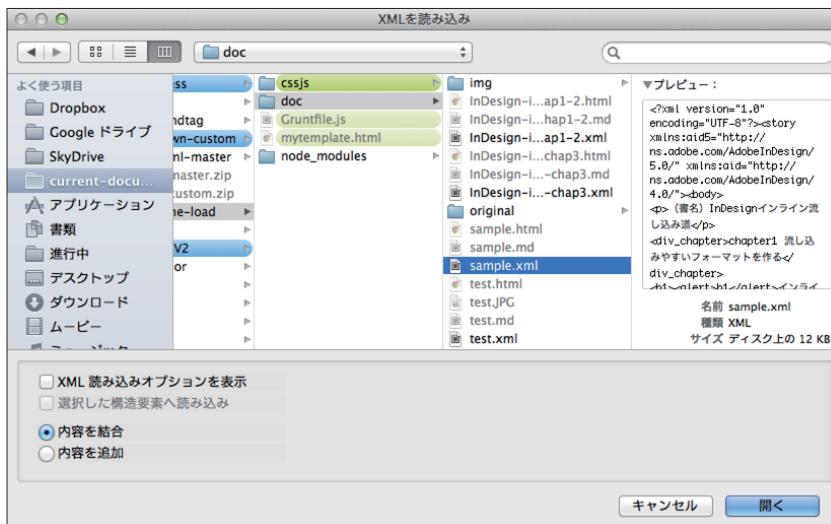
Markdown 原稿と同じフォルダに XML ファイルが作成されるので、これを InDesign に読み込みます。

[command] + [option] + [1] (テンキーは不可) を押すと、ドキュメントウィンドウの左に [構造] パネルが表示されます。そこからメニューを選択して [XML を読み込む] を選択します。



[構造] パネルのメニューから [XML を読み込み] を選択

ファイルダイアログボックスが表示されるので XML ファイルを選んで読み込みます。ここでは sample.xml という名前の、この文書の冒頭部分を抜き出した文書を読み込んでいます。



XML ファイルを選択する

ここで次のようなメッセージが表示される場合、XML ファイルにエラーがあります。



XML ファイルが読み込めない

「行：」がエラーのある行を表しているので、XML ファイルを開いてその場所を探します。よくあるエラーの原因は、**HTML 関連の書籍で HTML タグをそのまま入力してしまった場合**や、画像挿入 (![]()) を空白行を入れずに連続して書いた場合などです。元の Markdown 原稿を開いて XML のエラー部分と対応する行を探し、HTML タグなら前後を「`」(バッククオート) で囲み、画像挿入の場合は前後に改行を入れてみてください。

```

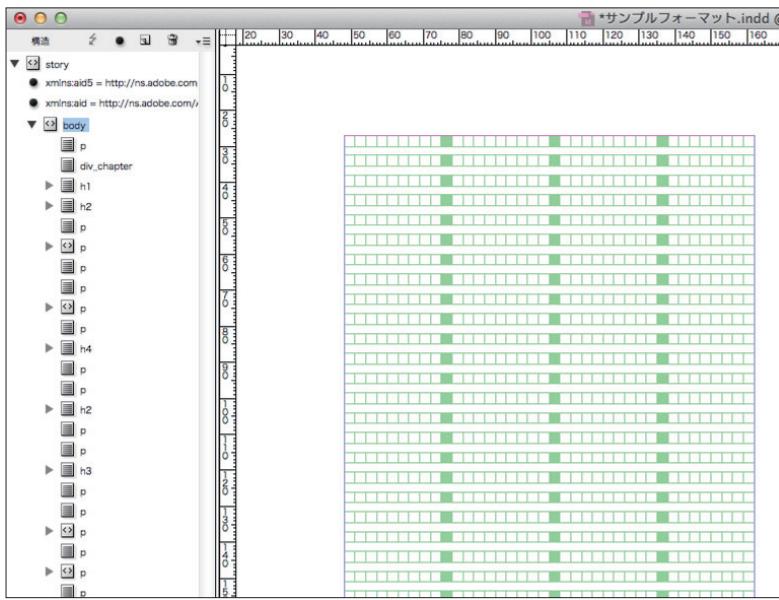
<?xml version="1.0" encoding="UTF-8"?><story xmlns:aid5="http://ns.adobe.com/AdobeInDesign/5.0/" xmlns:aid="http://ns.adobe.com/AdobeInDesign/4.0/"><body>
<p>(書名) InDesignインライン流し込み道</p>
<div _chapter>chapter1 流し込みやすいフォーマットを作る</div _chapter>
<h1><alert>h1</alert>インラインとグリッドを理解する</h1>
<h2><alert>h2</alert>一段組みなら楽勝！というフォーマット作りを目指す</h2>
<p>中級者向けのIT書でもっとも<div>多いデザインは、「一段組みで文章の途中に図版やソースコードを挟みつつ、草木またはセクション末までリニアに統いていくタイプ」です。このタイプがより早く組めるようになれば、その分を校正期間に回して品質アップを図れるようになります。</p>
<p><figure><img href="file:///img/fig1.png" /></figure>
<figcaption><alert>CAP</alert>一段組みリニア型のイメージ</figcaption></p>
<p>このタイプのデザイン指定は、たいていの場合、「各パーティのサイズ」と「パーティ間隔」の指定になりますが、それをそのままフォーマットにしてはいけません。</p>
<p>そのまま各パーティがバラバラの状態で組んでしまうと、後でテキストの追加や削除が発生したときに、それ以降を手作業で直すことになってしまいます。</p>
<p><figure><img href="file:///img/fig2.png" /></figure>
<figcaption><alert>CAP</alert>デザイン指定そのままのフォーマットでは修正が大変</figcaption></p>
<p>では、組み直しが発生しない完整な原稿をもらえばいい.....というのは組む側の都合であって、作っているものが「解説書」である以上、デザインが複雑だから、DTPが大変だから、内容に間違いがあっても直さないというのは本末転倒です。組みやすいフォーマットを作ることで、テキストや画像追加ぐらいなら楽勝！という状態に持って行きましょう。</p>
<h4><alert>h4</alert>コラム：図解タイプは例外</h4>
<p>この文書が対象としているのは、最初に説明したように「一段組みで.....リニアに統いていくタイプ」です。見開き（2ページ）単位で完結する図解タイプ（できるシリーズなどの初心者向けの解説書）の本や、凝ったレイアウトの雑誌風の書籍には適

```

XML ファイルを開いてエラーの箇所を探す

ファイルが正しく読み込まれると、[構造] パネルに XML のツリーが表示されます。

## 2-4 XML変換とInDesignへの流し込み

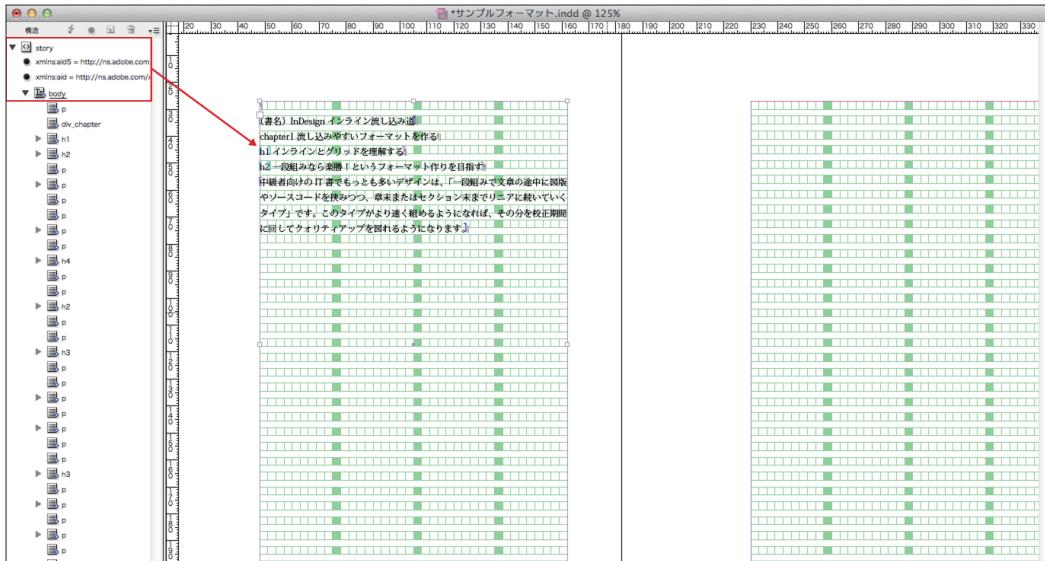


[構造] パネルで XML ツリーを確認

なお、画像のファイル名が日本語の場合も読み込みエラーになるので注意してください。

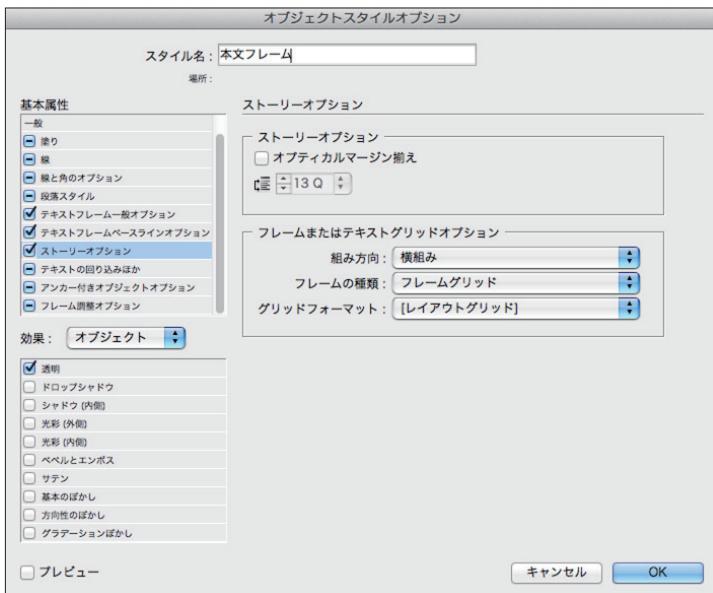
## XML流し込みの実行

[構造] パネルから story 要素か body 要素をドキュメントにドラッグ＆ドロップすると、XML の文書が配置できます。ただし、レイアウトグリッドになってしまんし、よく見ると途中までしか流し込まれていません。



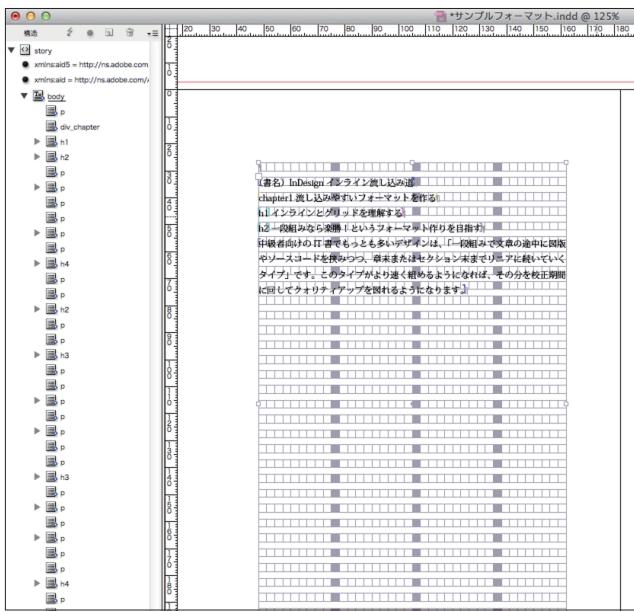
ドラッグ＆ドロップで文書を配置

フォーマットとして作成済みのページに切り替え、それを元に本文フレーム用のオブジェクトスタイルを作成します。このとき、レイアウトグリッド付きでドキュメントを作成した場合はそれをオブジェクトスタイルに指定し、マージン設定で作成した場合は先にグリッドフォーマットを作成してから指定します。



本文用のオブジェクトスタイルを作成する

このオブジェクトスタイルを設定して、フレームグリッドにします。

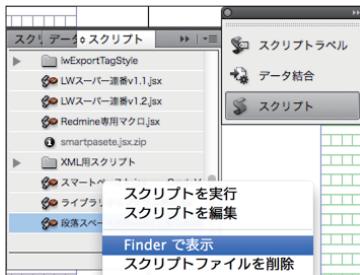


オブジェクトスタイルを設定

## 画像サイズをまとめて変更する

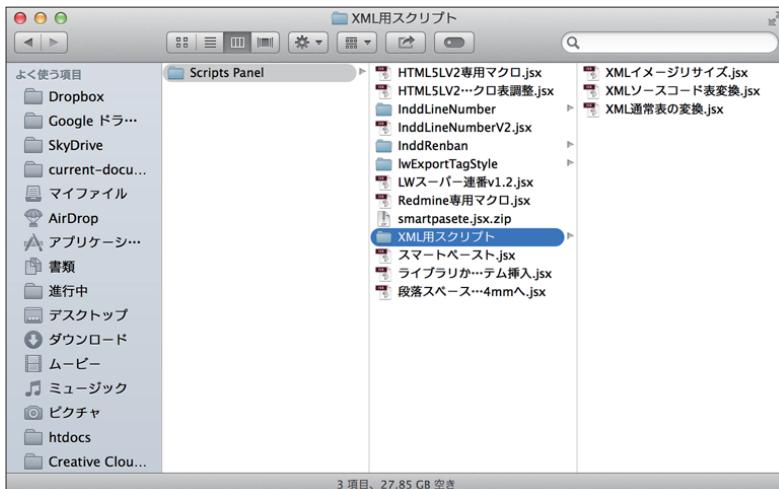
テキストが途中までしか流し込まれていないのは、画像が原寸サイズで読み込まれているためです。この問題を解決するために画像サイズをまとめて変更するスクリプトを用意しています。

まずスクリプトをインストールします。InDesignで【スクリプト】パネルを表示し、スクリプトのいずれかを右クリックして【Finderで表示】を選択します。



[スクリプト] パネル

Finderで【Scripts Panel】フォルダが表示されます。そこにGruntと一緒にダウンロードしたスクリプトファイルをコピーします。ファイルは3つあるので、フォルダを作ってその中に入れるといいでしょう。



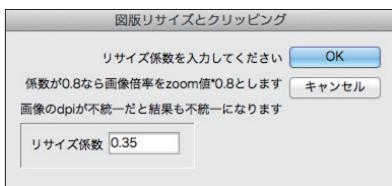
[Scripts Panel] フォルダにファイルをコピーする

スクリプトがインストールできたら、InDesignに戻って「XML イメージリサイズ」をダブルクリックして実行します。



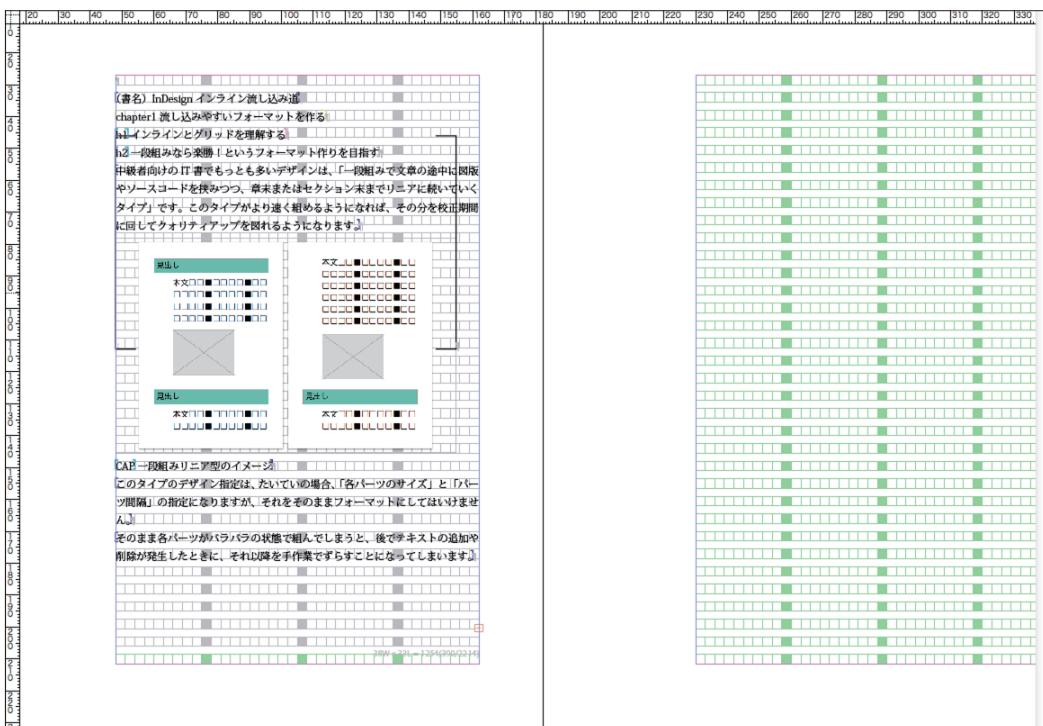
XML イメージリサイズを実行する

ダイアログボックスが表示されるのでリサイズ係数を入力します。0.5なら画像の倍率は50%となります。ここでは0.35と入力して35%にします。



リサイズ係数を入力

画像がまとめてリサイズされました。画像ごとに倍率を変えたい場合は、後で個別に調整してください。



画像がリサイズされた

## 2-4 XML変換とInDesignへの流し込み

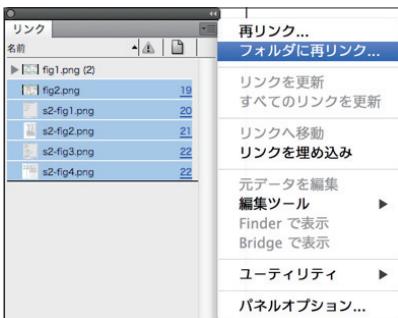
続きのテキストがあるので、オーバーフローハンドルをクリックし、右ページを $\text{Shift} + \text{クリック}$ して自動流し込みします。



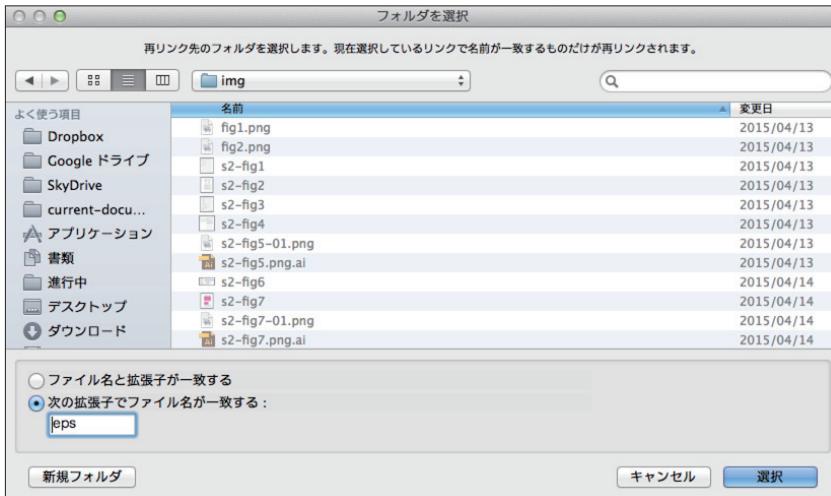
オーバーフローしたテキストを流し込む

なお、画像のリサイズは何度でもやり直せますが、XML タグを削除した後は変更できない——つまり、基本的には流し込んだ直後しか変更できないので、リサイズ係数を慎重に決める必要があります。できれば印刷して画像サイズが適切かを確認してから作業を進めることをおすすめします。

また、挿入された画像は RGB モードの PNG 画像なので、当然ながらそのまま印刷用では使えません。InDesign の「フォルダに再リンク」機能を使って、EPS や PSD、AI、PDF などの画像に置き換えてください。



[リンク] パネルで画像を選択して、メニューから [フォルダに再リンク] を選択



ファイル名が一致するファイルにまとめて置き換える

## コラム：画像の読み込み倍率をコントロールする

Markdown の仕様ではありませんが、次のように「?zoom=パーセンテージ」と記述しておくことで、「XML イメージリサイズ」を実行したときの結果をコントロールできます。

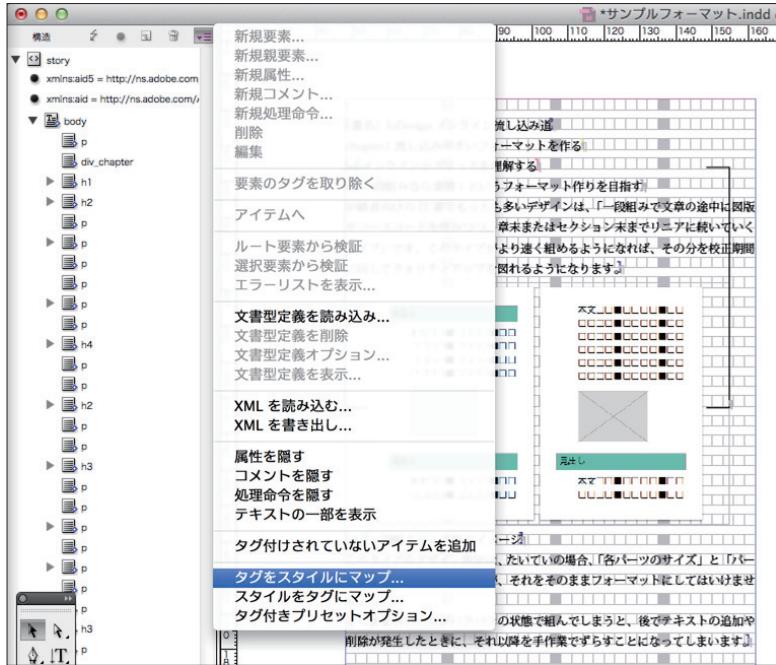
```
! [キャプション] (img/sample.png?zoom=40)
```

この画像は他よりも 40% 縮小した上で、さらにリサイズ係数を掛けた倍率になります。画面ショットと図版でサイズの基準が大きく異なる場合などに使うと便利です。

## タグとスタイルのマッピングを行う

次にタグとスタイルのマッピングを行い、特定の要素のテキストに段落スタイルや文字スタイルが設定されるようにします。

[構造] パネルのメニューから [タグをスタイルにマップ] を選択します。



[タグをスタイルにマップ] を選択

ダイアログボックスが表示されるので、スタイルを割り当てていきます。



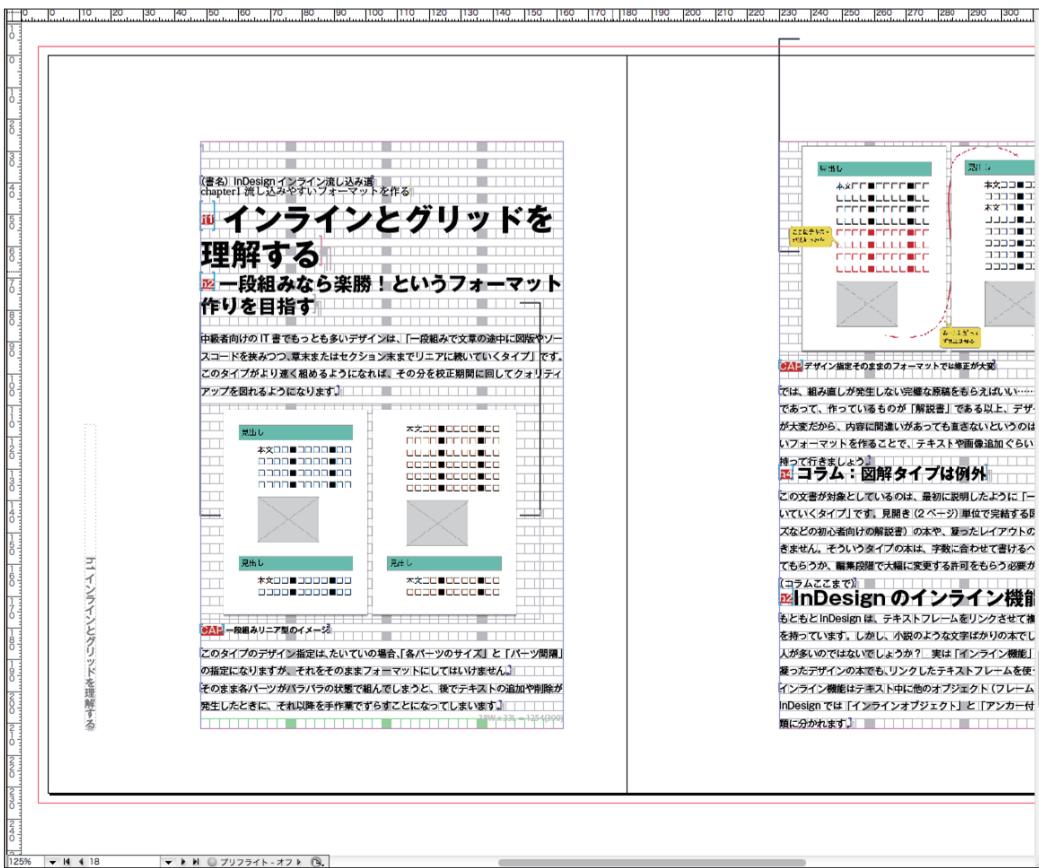
タグに対応する段落・文字スタイルを選択していく

いくつかHTMLにないタグがありますが、それらはMarkdownをXML変換する際に編集しやす

いよう追加したものです。

タグ	説明
alert	見出しやキャプションの前に自動挿入される「h1～h6」「CAP」という文字に設定される。赤下線+白文字の文字スタイルを割り当てる。
div_hen	<div class="hen">として挿入しておいた編集コメントに設定される。黄色下線の段落スタイルを割り当てる。
figcaption	画像のalt属性を抜き出してfigcaptionタグではさんだもの。キャプション用の段落スタイルを割り当てる。
figure	画像のimgタグをさらに囲む形で自動追加しているタグ。HTML5のfigureとは用法が合っていない。imgタグと同じ画像のインライン用行の段落スタイルを割り当てる。

[OK] をクリックすると、一気にスタイルが設定されます。



スタイルが一気に設定された

タグとスタイルのマッピングは**一度設定するとドキュメントに保存されます**。この設定をきちんと済ませておけば、他の章は XML ファイルを読み込むだけでスタイルが設定されるわけです。ただし、サンプルとして読み込む XML ファイルに含まれていないタグは、[タグをスタイルにマップ] ダイアログボックスに表示されないので、たいていの場合、何度も原稿ファイルとスタイル設定を調整しながら

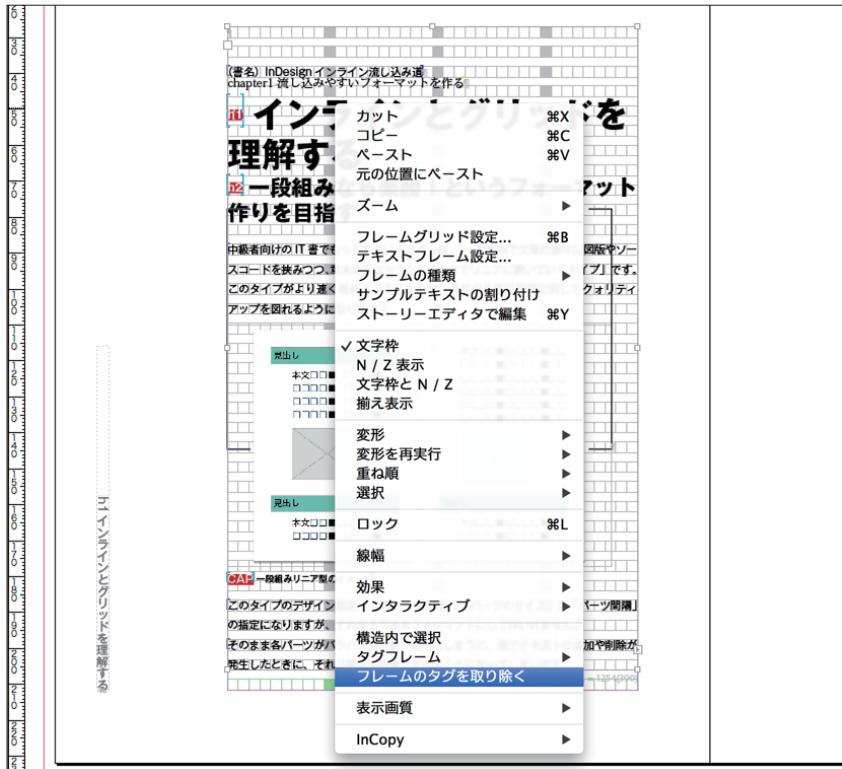
試行錯誤することとなるでしょう。

また、XMLファイルが長いとマッピング処理に時間がかかるので、なるべく短いサンプル原稿で設定を済ませ、後から本番の原稿を読み込むことをおすすめします。

## ▶ XMLタグを削除する

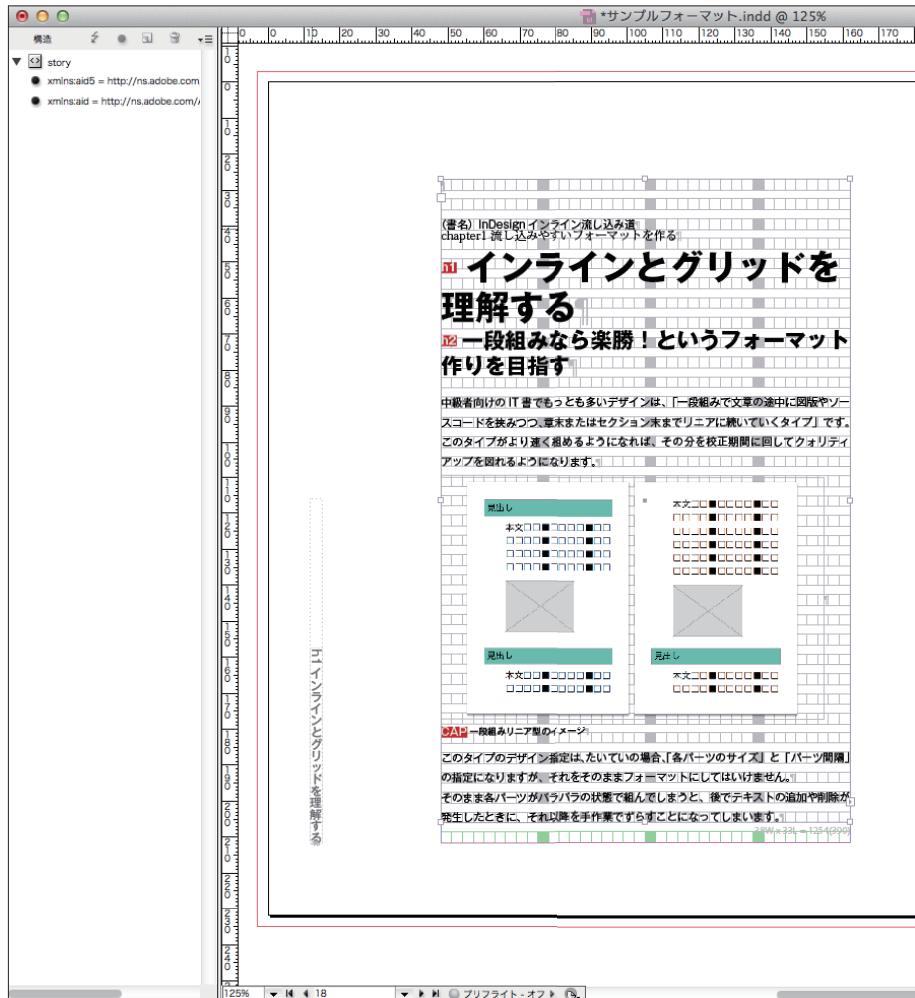
続いてセクションタイトルや中見出しをパーツに収めていく作業に移りますが、その前に XML タグを削除する必要があります。タグを削除していない場合、他のフレームへのカット＆ペーストに失敗したり、書式が綺麗に揃わなくなることがあるためです。なお、後で説明する表とソースコードの処理が必要な場合は、それが済んでから XML タグを削除するようにしてください。**XML タグを削除してしまうと、もう表やソースコードに変換することはできません。**

XML タグを削除するには、本文のフレームを選択ツールで右クリックして [フレームのタグを取り除く] を選択します。



[フレームのタグを取り除く] を選択

XML タグを表す枠が消え、[構造] パネルの中が空（story のみ残ります）になります。これで配置されたテキストは、XML と関係ないスタイルが設定されただけの通常のテキストになります。



タグが削除された

[構造] パネルのゴミ箱ボタンをクリックしてもタグを削除できますが、その際はタグが付けられているテキストや画像を残すかどうかを確認するダイアログボックスが表示されます。

## 2-5

# セクションタイトルや中見出しのパートを自動配置する

デザイン上、セクションタイトルや中見出しが別パートになっている場合は、それを自動挿入してテキストをカット&ペーストしていきます。

## 自動配置の準備をする

今回のサンプルフォーマットではセクションタイトルと中見出しが別パートです。手作業で配置していくとせっかくの自動流し込みのメリットが薄れるので、スクリプトを使って一部自動化します。

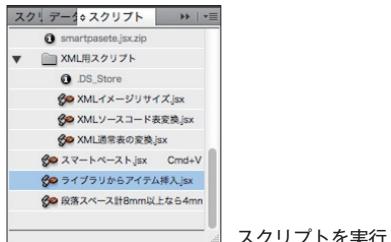
まず、以下の URL からスクリプトをダウンロードし、解凍した中から「ライブラリからアイテム挿入.jsx」というファイルを探して、[Scripts Panel] フォルダにコピーしておきます。

- <https://github.com/lwohtsu/lwItemAutoInsert>

ライブラリを作成し、パートを登録しておきます。ここでは h1 のセクションタイトル用と h2 の中見出し用のパートを登録します。



パートを挿入したい本文フレーム内にカーソルを置き、[スクリプト] パネルから [ライブラリからアイテム挿入.jsx] を実行します。このとき使用するライブラリは開いておく必要があります。

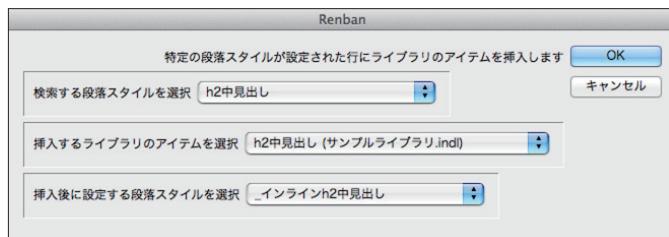


ダイアログボックスが表示されるので、検索する段落スタイル（現在セクションタイトルに設定されている段落スタイル）、挿入するパート、パートを挿入した親行に設定するインライン用の段落スタイルを選択します。



スタイルやパートを選択する

[OK] をクリックするとセクションタイトル用のパートが挿入されます。同様に今度は中見出しのパートを挿入します。



中見出しのパートを挿入

ドキュメントを見ると、パートが挿入されています。残念ながら現段階ではテキストを入れることはできないので、手作業でカット＆ペーストしていきます。

The image shows two columns of InDesign text frames. The left column contains the heading 'h1 インラインとグリッドを理解する'. The right column contains the heading 'h2 一段組みなら楽勝！というフォーマット作りを目指す'. A green callout points to the first part, and a blue callout points to the second part.

パートが挿入された

## 2-5 セクションタイトルや中見出しのパートを自動配置する

これでほぼ完成状態です。コラムなどは、必要に応じて手作業または自動で処理してください。また、h1、h2、CAPなどの作業用の記号には「alert」という文字スタイルが設定されているので、書式付きの検索置換でまとめて削除できます。

**1-1 インラインとグリッドを理解する**

**一段組みなら楽勝！というフォーマット作りを目指す**

中級者向けの「IT書」でもっとも多いデザインは、「一段組みで文書の途中に図版やソースコードを挿入する末尾またはセクション末尾でリニアに続かないタイプ」です。このタイプがより速く組めるようになります。その分を校正期間に回してクオリティアップを図れるようになります。

図版と本文の組み方

このタイプのデザイン指摘は、たいていの場合、「各ページのサイズ」と「ページ間隔」の指定になりますが、それをそのままフォーマットにしてはいけません。

そのままでページがバラバラの状態で組んでしまうと、後でテキストの追加や削除が発生したときに、それ以降の手作業でずらすことになってしまいます。

テキストをカット&ペーストしてほぼ完成状態に

## コラム：後送原稿が届いたときは？

後から一部の原稿が後送になったり、数ページ分まとめて差し替えになることがあります。本書で説明する方法での流し込み方法でも普通に追加流し込みすればいいのですが、注意点が3つあります。

- XMLツリーが複数あると読み込みがうまく行かないので、[構造]パネルが空の状態になっていることを確認する（P.77 参照）
- グリッドフレームにカット&ペーストする際にフレームの書式でオーバーライドされるので、[グリッドフォーマットを適用せずにペースト]を使用する（P.17 参照）
- 「ライブラリからアイテム挿入」スクリプトは、同じストーリー（連結されたフレーム）内のみを処理するので、別ストーリーになっているうちに実行する

XML用のスクリプトはXMLタグが割り当てられているオブジェクトにしか使えないで、XMLタグを削除した後であれば、画像サイズなどが変わることはあります。

## 2-6

# 表とソースコードを挿入する

IT書にはつまもの表とソースコードも自動化が可能です。表は見出し行の設定、ソースコードはキーワードの色分けを自動で行うことができます。

## Markdownでの表とソースコードの記述

先に説明したようにソースコードは表組みとして実装します。つまり、通常の表もソースコードも、どちらも表組みなので作り方はかなり似てきます。

まず、Markdownでの書き方について説明しておきましょう。表は「|」(パイプ)と「-」(ハイフン)を組み合わせて書きます。本来は半角スペースなどを使ってMarkdown原稿上でも見やすくなるように揃えますが、省略可能です。またセルの終わりを表す「|」も省略できるので、表として認識される必要最低限に絞ると次のようになります。

見出し   見出し
-   -
セル1   セル2
セル3   セル4

見出し	見出し
セル1	セル2
セル3	セル4

ブラウザプレビューでの結果

なお、Markdownの記法では「:」(コロン)を使って右寄せ・左寄せの指定もできますが、これは反映されないため、流し込んだ後での調整が必要です。

ソースコードは「```」で前後を囲み、始まりのほうにはソースコードの種類を表すキーワードを入れます(P.64参照)。字下げのタブは半角スペースに変換しておくことをおすすめします。タブのままにした場合、HTML変換時に適当な数の半角スペースに置換されるため、幅がバラバラになることがあります。

```
```javascript
var Highlight = require('./highlight');
var hljs      = new Highlight();
```
```

```
var Highlight = require('./highlight');
var hljs      = new Highlight();
```

ブラウザプレビューでの結果

ソースコードは変換後の XML が HTML とかなり違うので見せておきましょう。ソースコード全体が <codelist> という独自タグで囲まれ、1 行ずつ <pre> と <code> で囲まれます。色分け範囲は span タグで囲まれ、「hljs-」で始まるクラスが設定されます。それが最終的に <span\_hljs-●●●> というタグに置換されます。

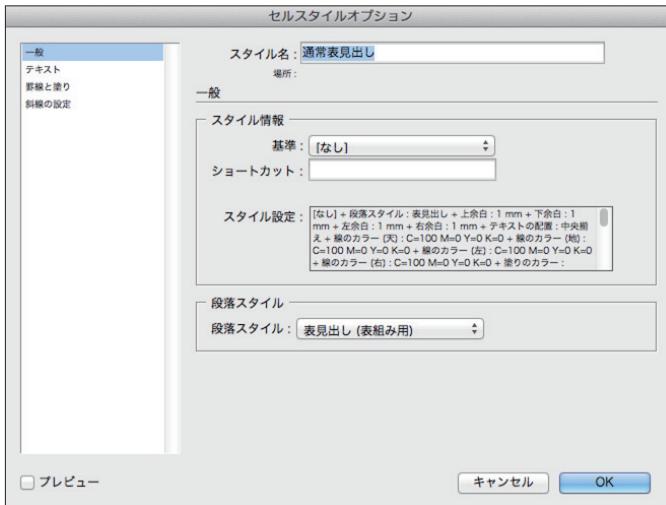
```
<codelist>
<pre><code><span_hljs-keyword>var</span_hljs-keyword> Highlight = <span_hljs-
built_in>require</span_hljs-built_in>(<span_hljs-string>'./highlight'</span_hljs-
string>);</code></pre>
<pre><code><span_hljs-keyword>var</span_hljs-keyword> hljs      = <span_hljs-
keyword>new</span_hljs-keyword> Highlight();</code></pre>
</codelist>
```

## ▶ 表スタイルを用意する

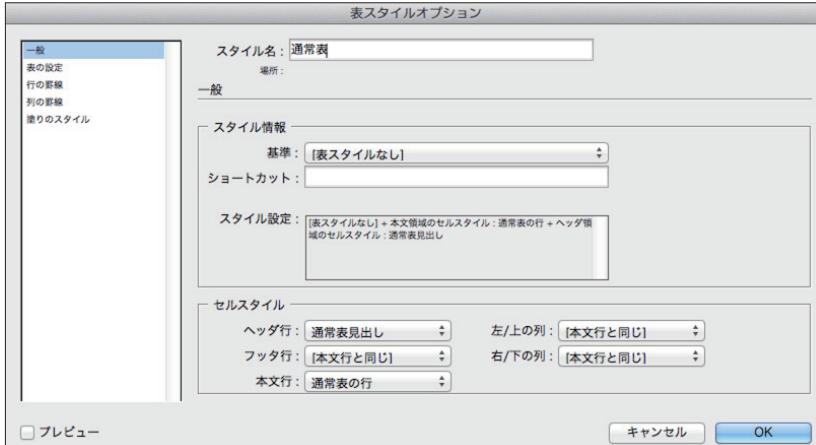
通常表とソースコード用に表スタイルも作成しておきます。ソースコード用のスタイルは第1章で説明しましたが（P.28 参照）、通常の表も最低限「見出し」と「表本文」用のセルスタイルを持つものを作成します。



表の見出しと本文用の段落スタイルを作成



それをもとにセルスタイルを作成



さらにそれをもとにした表スタイルを作成

表スタイルを作成するときは、[ヘッダ行] に表見出し用のセルスタイルを設定しておきます。InDesign 本来のヘッダ行の使い方とは異なるのですが、スクリプトでヘッダ行・フッタ行の設定を確認して、セルスタイルが登録されていれば表に自動設定しているようにしているからです。

表スタイルを作成する時は、表の内容が複数行になることも想定して行送りを設定しておきましょう。

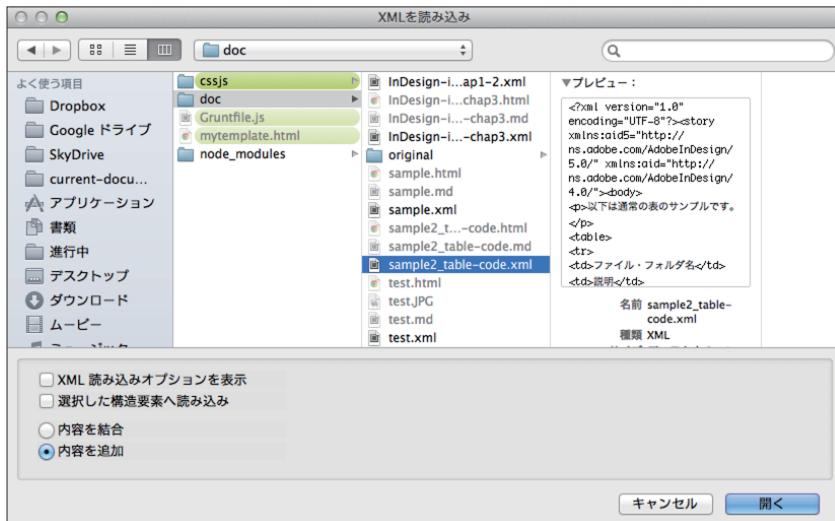
表見出し	表見出し
セル	セル□□□□■□□□□□ ■□□□□□■□□□□□ □□□□□■
セル	セル□□□□■□□□□□ ■□□□□□■

設定した表スタイル。複数行になったときの見栄えも考慮する

## 表とソースコードの流し込み

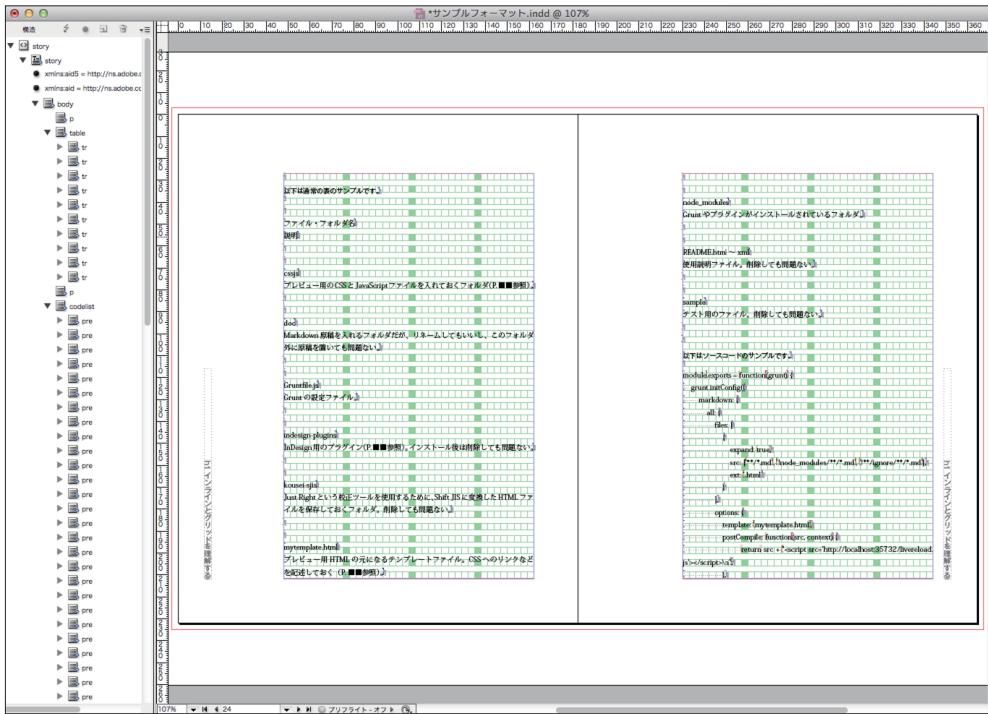
表とソースコードを組むには、先に説明したのと同様に XML を読み込んでから、スクリプトを使って表に変換します。この変換は **XML タグがある状態で行う** 必要があります。「フレームのタグを取り除く」を実行した後（P.77 参照）は変換できないので注意してください。また、スクリプトは複数の表・ソースコードをまとめて変換しますが、取り消しの履歴に 1 つずつ残ってしまうため、変換後にアンドウで元に戻すのは困難です。先にドキュメントを保存して、いつでも復帰できる状態にしておくことをおすすめします。

表とソースコードを記述した XML を読み込みましょう。



XML を読み込む

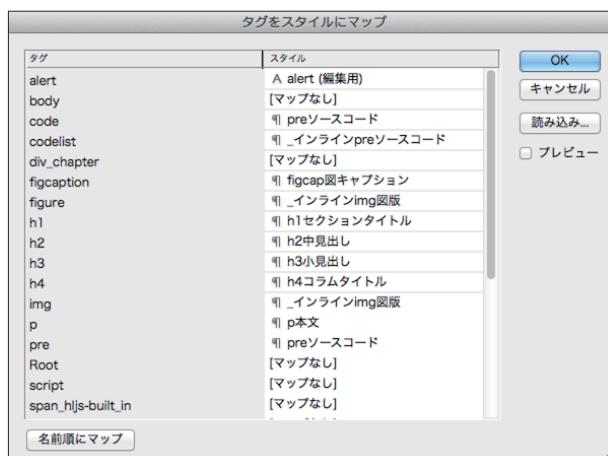
ドラッグして配置した直後の状態では、通常の文章のように見えます。しかし [構造] パネルを見ると、table や tr、pre などの独特的なタグでマークアップされていることがわかります。



配置した直後の状態

この時点では隣のページに続きを流し込んで、オーバーフローを解決しておくようにしてください。表の幅は親のフレームの幅一杯となるため、**オーバーフローした状態のテキストを変換するとセルが最小の幅になってしまい、後から広げる手間が増えます。**

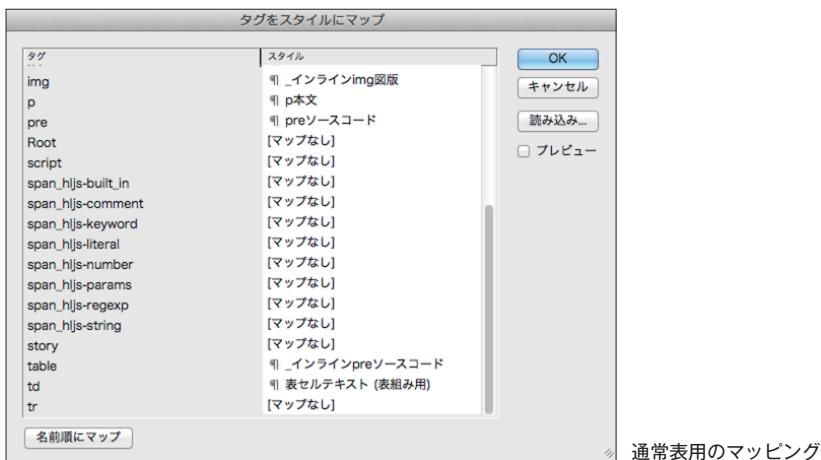
[タグにスタイルをマップ] ダイアログボックスを表示し、いくつか下処理のマッピングをしておきます。ソースコードのために、pre と code をソースコード用の段落スタイルに、codelist をINLINE親行用の段落スタイルにマッピングします。



ソースコード用のマッピング

通常の表のために、td を表の本文用の段落スタイルに、table をINLINE親行用の段落スタイルに

マッピングします。



表組みへの変換を行います。[スクリプト] パネルから「XML 通常表の変換」を実行します。



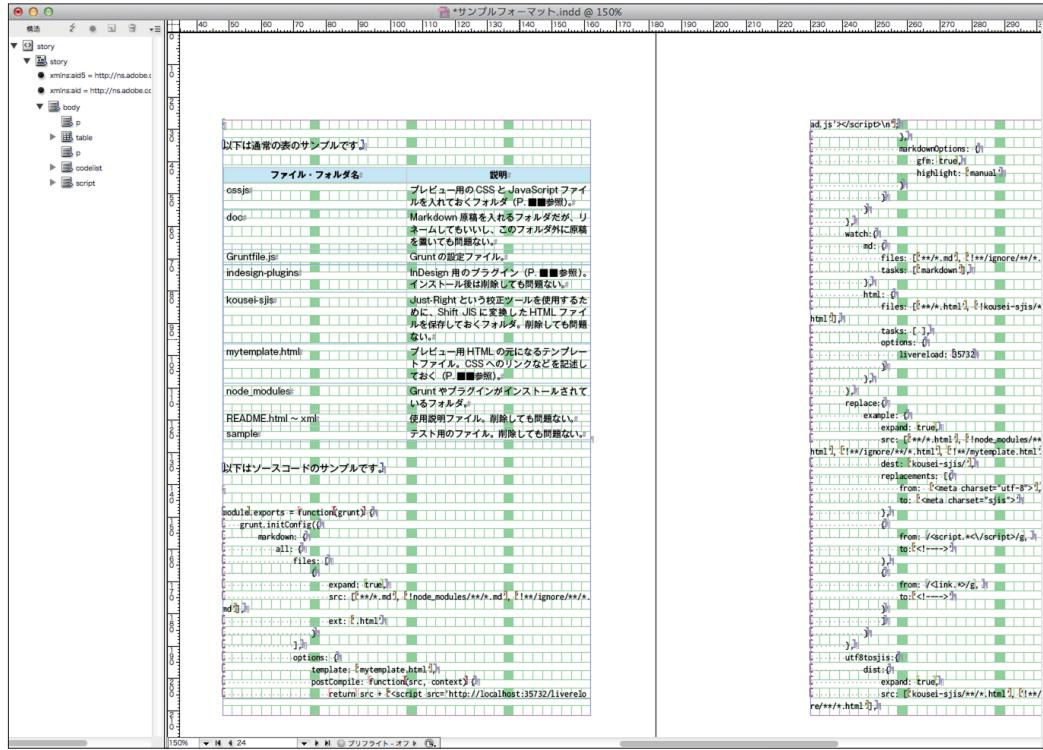
スクリプトを実行する

通常表用の表スタイルを選択して [OK] をクリックします。



通常表の表スタイルを選択して実行する

少しして「コンバート終了」というメッセージが表示されるので [OK] をクリックします。この段階でセルスタイル・段落スタイルともに設定済みなので、後は幅を調整していくば通常の表が完成します。



通常表が変換された

同じように「XML ソースコード表変換」を実行します。2つのスクリプトはどちらを先に実行しても構いません。



ソースコード用の表スタイルを選択して実行する

ソースコードの変換が完了しました。前に先頭行のセルスタイルをコードタイトル用に設定していたため少しおかしなことになっていますが（P.38 参照）、必要に応じて調整してください。

The screenshot shows a Microsoft Word document titled "サンプルフォーマット.indd" at 150% zoom. The document contains several tables and code snippets. On the left, there's a navigation pane with sections like "story", "body", "table", "codeblock", and "script". The main content area has two large tables. The first table has a green header row and contains text in Japanese. The second table has a green header row and contains code snippets, including a Gruntfile.js and a template.html file. To the right of these tables is a vertical sidebar with various configuration options for the "load.js" script, such as "template", "postCompile", "return src", "load src", "markdownOptions", "watch", "md", "files", "tasks", "html", "options", and "liveReload".

ソースコードの変換が完了した

一通りの作業が完了したら、「フレームのタグを取り除く」で XML のタグを削除します (P.77 参照)。

## ソースコードを色分けする

最後にソースコードを色分けをする方法について説明しましょう。grunt-markdown に含まれる Highlight.js によってクラス付きの span タグが挿入されるので、XML 変換の中で「span\_hljs-comment」「span\_hljs-keyword」という独自タグに置き換えていきます。後は「タグをスタイルにマップ」機能を利用して、色だけが設定された文字スタイルにマッピングしていくれば、自動で色分けが完了します。

このように理屈は簡単なのですが、いくつか注意が必要な点があります。

### ・どのタグをどの色にするかを決める必要がある

Highlight.js が書き出すクラスの種類が非常に多く、それらが何を表しているのかがわかりにくいので、判断するのに手間がかかります。すべてのタグを色分けする必要はないので、ブラウザプレビューや IDE での表示などを参考にしながら取捨選択して、スウォッチと文字スタイルを作っていきます。

### ・入れ子のタグに対応できない

たとえば HTML のタグを Highlight.js で色分けすると、次のように「hljs-tag」の中に「hljs-title」

「hljs-attribute」「hljs-value」が入る形になります。

```
<span class="hljs-tag"><&gt;<span class="hljs-title">script</span> <span class="hljs-attribute">type</span>=<span class="hljs-value">"text/javascript"</span>&gt;</span>
```

InDesign の文字スタイルは入れ子で設定できないため、XML 変換時に内側のタグだけを残す仕様にしています。その結果、「<」と「>」に色が付かなくなります（つまり黒字になります）。

```
&lt;<span_hljs-title>script</span_hljs-title> <span_hljs-attribute>type</span_hljs-attribute>=<span_hljs-value>"text/javascript"</span_hljs-value>&gt;
```

そのまま通すのが一番楽ですが、どうしても「<」と「>」を付けることを求められた場合は、InDesign の検索置換機能でスタイル設定を行うなどの対応が必要です。

- **複数行にまたがるタグに対応できない**

XML 変換機能は開始タグと終了タグが同じ行にないと正しく変換できません。ソースコード中に開始タグや終了タグが単独である場合は自動的に削除します。そのため、複数行コメントなどが正しく色分けできないことがあります。

- **異なる言語で同じタグが使われることがある**

前の入れ子の問題ともからむ話ですが、Highlight.js では「hljs-comment」や「hljs-keyword」などのクラスが複数の言語で使われています。親クラスの「hljs-tag」などとの組み合わせで区別する仕様ですが、XML 変換時に親クラスの部分を削除してしまうため、InDesign に読み込んだ時点では区別できなくなってしまいます。要するに **「HTML のコメントは赤で、CSS のコメントは青」といった対応はできない**ので、クライアントやデザイナーと事前にすり合わせるようにしてください。

- **たまに間違ったタグが付く**

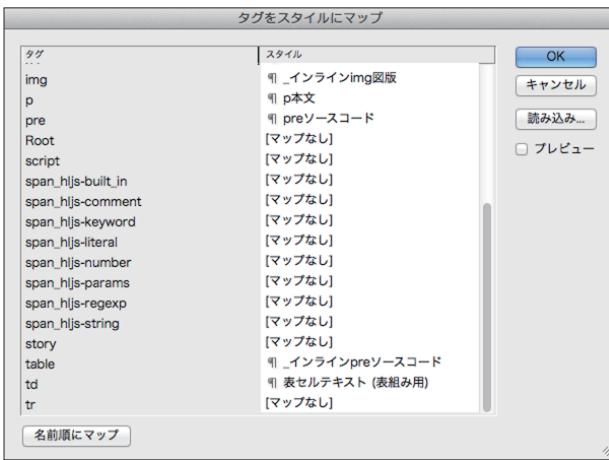
Highlight.js が色分けを間違えることがあります。よくあるのが「#」や「/」などの記号をコメントと誤認識するパターンです。また、bashなどのコマンドラインもおかしな色分けになることがあります。

- **関数名・変数名などが色分けされない**

関数名や変数名のような開発者が独自で決めた名前は、Highlight.js が判別する手がかりがないので色分けされません。Highlight.js の言語用設定を書き換えるという方法で対処したことがあるのですが、かなり難易度が高めです。

## ▶ 色分けの実践

それでは実際に色分けのための設定をやってみましょう。最初に [タグをスタイルにマップ] ダイアログボックスなどで、色分け用にどんなタグが存在するのかを確認します。複数言語を扱う書籍の場合は、すべての言語を含むサンプル原稿を作成し、それを読み込んでおくことをおすすめします。



[タグをスタイルにマップ] ダイアログボックスで確認する

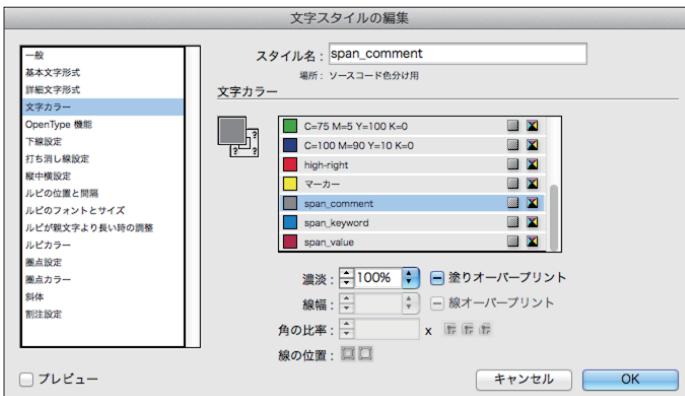
この例では8つのタグがありますが、「built\_in」と「keyword」は同じ色、「literal」「number」「regexp」「string」も同じ色を設定することにします。「params」は色分けしません。

その方針でスウォッチと文字スタイルを作成します。



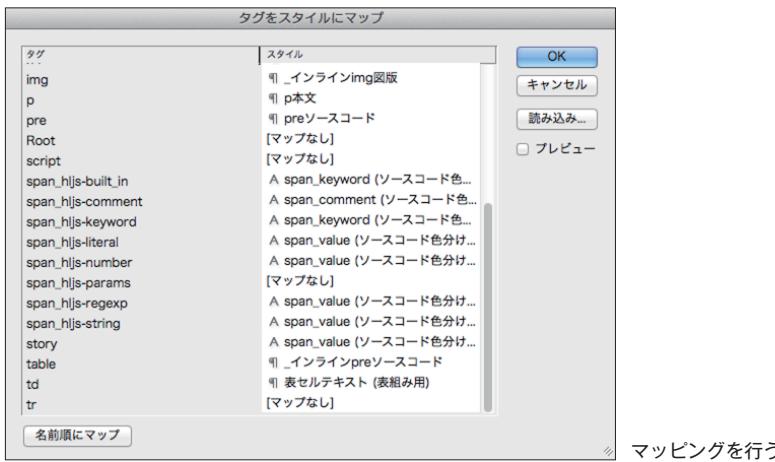
色分け用のスウォッチを作成 3つの文字スタイルを作成

文字スタイルを作成する際は、テキストを選択しないよう注意してください。選択した状態だとそのテキストの書式を拾ったスタイルになってしまいます。



文字スタイルでは文字色のみを設定

文字スタイルの作成が終わったら、マッピングを行います。



ソースコードが色分けされました。このサンプルコードは独自のプロパティ名が並ぶオブジェクト定義なので今ひとつ黒の部分が多い印象を受けますが、これは Highlight.js の限界なのでやむをえないところです。

```
|  
module.exports = function(grunt) {  
  grunt.initConfig({  
    markdown: {  
      all: {  
        files: [  
          {  
            expand: true,  
            src: ['**/*.md', '!node_modules/**/*.md', '!**/igno  
re/**/*.md'],  
            ext: '.html'  
          }  
        ],  
        options: {  
          template: 'mytemplate.html',  
          postCompile: function(src, context) {  
            return src + "<script src='http://localhost:35732/liver  
eload.js'></script>\n";  
          },  
          markdownOptions: {  
            gfm: true,  
            highlight: 'manual'  
          }  
        }  
      }  
    }  
  }  
};
```

ソースコードが色分けされた

ソースコード色分けは4色刷りで使うものというイメージもありますが、スウォッチの作り方次第で2色刷りでも効果を発揮します。また、1色刷りの本でもコメントだけをグレーにして読みやすくすることができます。

## コラム：スクリプトで画像に枠をまとめて設定する

画面ショットに 0.1mm 程度の細罫を付けることがよくあります。それを一気に設定するスクリプトを紹介しましょう。

### PNG 画像にオブジェクトスタイルまとめて設定.jsx

```
main();
function main(){
    if (app.documents.length != 0){
        var myDocument = app.activeDocument;
        var images = myDocument.allGraphics;
        var objstyle = myDocument.objectStyles.itemByName("0.1 黒ケイ");

        for(var i=0; i<images.length; i++){
            var img = images[i];
            if(img.imageTypeName == 'PNG'){
                img.parent.appliedObjectStyle = objstyle;
            }
        }
        alert ("コンバート完了");
    } else {
        alert ("ドキュメントを開いてください");
    }
}
```

まず、InDesign の Document オブジェクトの allGraphics プロパティで、グラフィックの配列を取得します。後は for ループで 1 つずつチェックし、imageTypeName プロパティが PNG だったら枠罫用のオブジェクトスタイルを設定します。

オブジェクトの階層を理解するには、公式のスクリプティングガイドや API リファレンスが参考になります。

#### ・スクリプティングガイド

[http://www.adobe.com/content/dam/Adobe/en/devnet/inDesign/cs5\\_docs/inDesign\\_scripting/guide-jp/InDesignCS5\\_ScriptingGuide\\_JS.pdf](http://www.adobe.com/content/dam/Adobe/en/devnet/inDesign/cs5_docs/inDesign_scripting/guide-jp/InDesignCS5_ScriptingGuide_JS.pdf)

#### ・API リファレンス

<http://jongware.mit.edu/idcs4js/>