Internet Week 2011 Document Session 2

リリース *1.0*

IW2011 Document Session Team

2011 年 11 月 21 日

目 次

運用ドキュメントと言っても、その種類は多くあります。また、種類毎に書かれている内容やフォーマットは大抵異なっていると思います。しかし、ドキュメントの個々のの表現はともかく、見出しはこのように記述する、箇条書きは ‥ といった記述様式は共通している方が良いでしょう。

例えば見出しや箇条書きは以下のようにしているかもしれません。

```
= IW 案件議事録 2011/11/20 =

- 参加者: 波田野、関根、清水川
- 場所: 渋谷のルノアール

== 議題 ==

- 議題 1
- 議題 2
```

このセッションでは、ドキュメントの利用者、種類、記述についていくつかの例を挙げ考察したいと思います。

また、効率の良い記述様式により多くのフォーマットでの出力が可能なオープンソースのドキュメントツール『Sphinx』を例に、情報の構造化や再利用を意識した運用ドキュメントの作成について解説します。

Who/Whom: 誰のためのドキュメント？

1 Who: 誰視点で書けばいいのかわからないので、つい主観が強くなってしまう

ドキュメントには読者がいます。当たり前のことですが、仕事でドキュメントを書いているとこの点についてあまり考えられていないように思います。

誰視点で書けばいいのかわからない、または、そのこと自体を意識せずにドキュメントを書いてしまうと、主観が強くなってしまいます。

その結果、「どのような情報を載せるか」「どこまで書くか」といった書き手の思いを中心にドキュメントが書かれていきます。

気がつくと、関連する情報をどんどん書き足してしまったり、大前提となる情報が書かれていなかったりと、必要十分ではないドキュメントができあがってしまいます。

このようなドキュメントは最終的には自分自身も読みづらいものになってしまいます。

2 Whom: 誰相手に書くのかわからない

「読者は誰」で「どう書いたらその読者にとって読みやすいか」といったことを考えることがドキュメントを書く上で重要です。読者の例としては以下があります。

- リーダー・マネージャ
- 設計者
- 開発者
- インフラ担当
- 運用者

この読者の分類は「用語の違い」で分けています。よく、同じ物や概念を指しているのに異なる用語を使ってしまい話が通じなかったり、ある用語の指す範囲が異なっていて話がかみ合わなかったりということがあります。このような「用語のセット」＝「語彙」を統一して読み手が普段使っている範囲に合わせることによって、読みやすさ、理解しやすさを向上させることが出来ます。

また、ドキュメントの読み手が仕事を行う上で何が必要かを把握しておきましょう。運用時に「この機能はどんな目的でこんな作りになったのか」を把握出来ると、実装者が意図した使い方が出来る様になりますし、要件が変わってしまったときに潔く捨てることも出来る様になります。

Which: 何から書き始めよう？(構造化テキストの導入)

3 何から書き始めるか

何も無いところからドキュメントを書く場合、どのように書き始めれば良いのか分からない場合があります。そういった場合、とにかく文章を書き始めるというのも一つの方法ですが、まずは書きたいものの構成を整理するためにもタイトルを書き出していきます。

```
* タイトル 1

  * サブタイトル 1
  * サブタイトル 2

* タイトル 2
* タイトル 3
```

そうして、思いついた箇所にどんどん肉付けをしていきましょう。

```
* タイトル 1

  * サブタイトル 1
  * サブタイトル 2
  * サブタイトル 3

    サブタイトル 3 の内容をここにどんどん書き足していきます。
    サブタイトル 3 以外に入れるべき内容を思いついたら、他の箇所にも
    ぶら下げて書き足していきます。

* タイトル 2

  * サブタイトル 1
  * サブタイトル 2

* タイトル 3

  * サブタイトル 1
  * サブタイトル 2
```

このように書かれたテキストは、タイトルとその内容とで親子関係を持っている箇条書きと空白文字でインデントされた小さな構造化テキストです。形の見えないドキュメントを書く際には、この「タイトル」と「内容を表す要素」(この例ではサブタイトル) を書き出していく作業を繰り返していきます。

構造化テキストで書いていくと、書き進める上で様々なメリットが得られます。

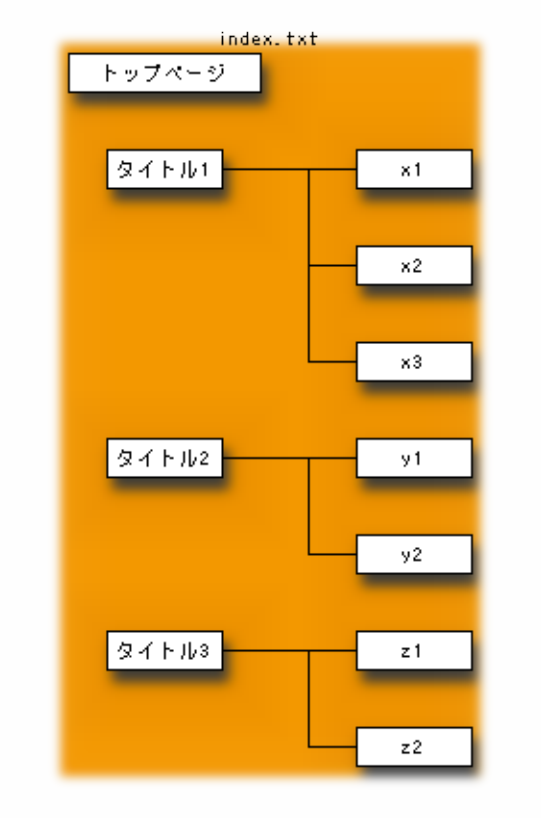
それでは、構造化テキストの利点を紹介します。

4 構造化テキストを書こう

4.1 1つのファイル内での構造化

テキスト文章を構造化することで、見た目にも分かりやすくなりますし、そのままで自然に読むことができます。

構造化されていると、各情報が同列なのか親子なのかが明確になり、組み替えや並び替えがしやすいという利点があります。



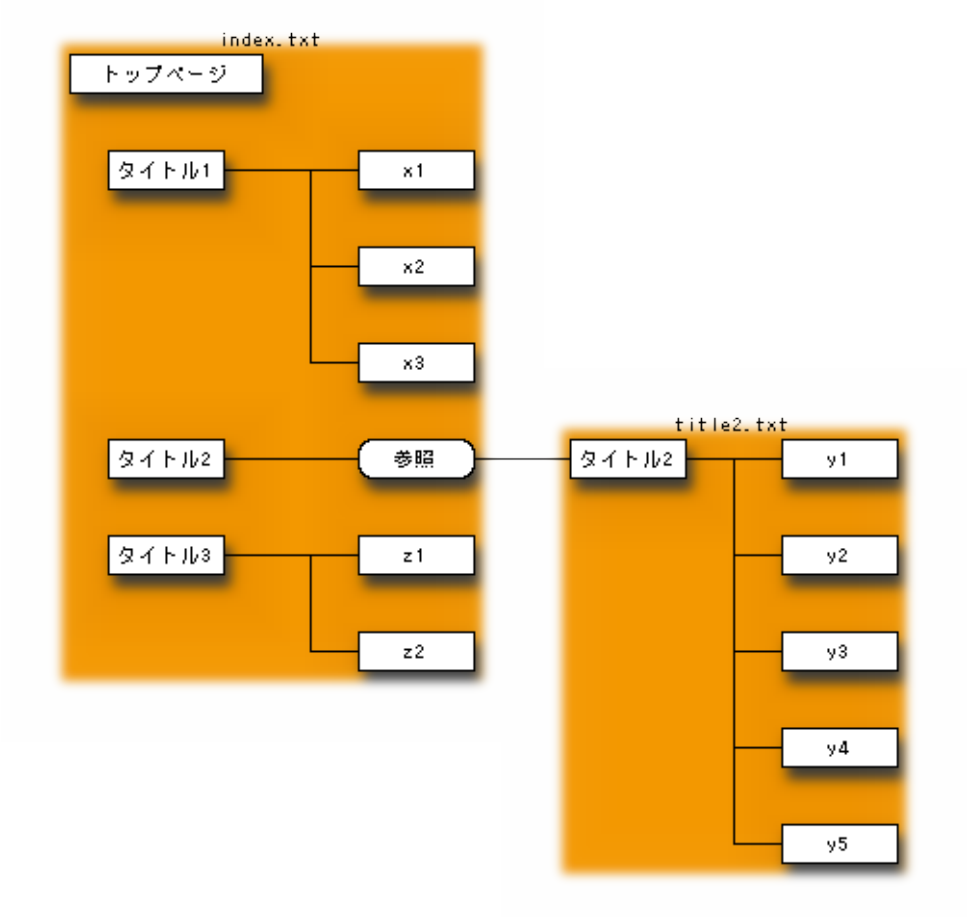
こういった構造化の記述ルールで書かれたテキストは、ツリー構造を持つことになります。

4.2 ファイルとディレクトリの構造化

書き進めていくとぶら下げた内容が詳しくなりすぎてしまう場合がありますし、1つのファイルだけで書き足していくと見通しが悪くなっていきます。

ある程度の規模になって、内容を分類分割したくなった場合は、ファイルとディレクトリで文章を分割して構造化します。分割元から分割先へは親子の関連付けをかならず1つもたせます。

こうすることで、ファイルを分割してもツリー構造が維持されます。

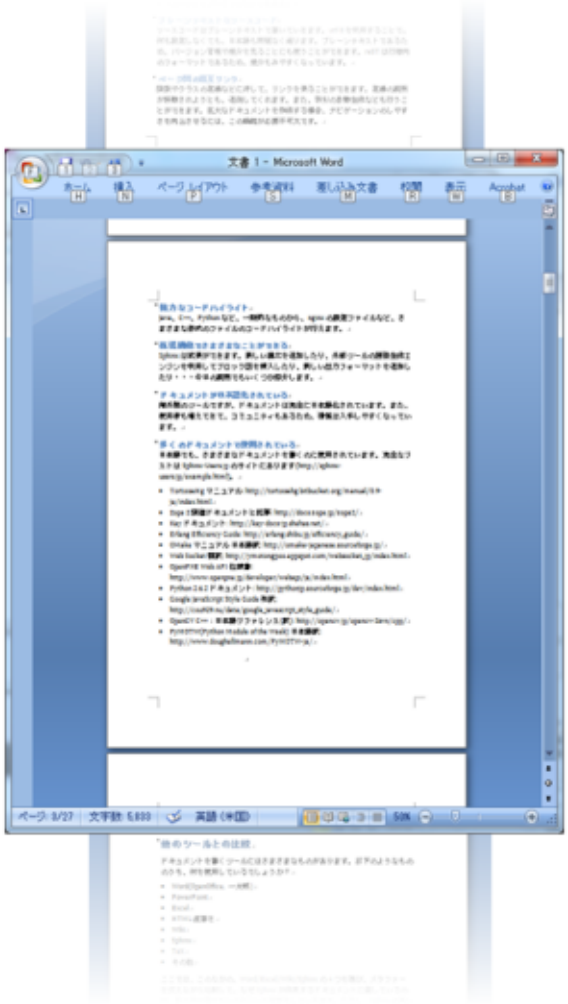


ファイルが多くなってきたらファイルをグルーピングしてディレクトリで階層化します。このようにファイルとディレクトリを活用して構造化を進めていきます。

ドキュメントの章や節をディレクトリやファイルで分割しておくことで、後から節の付け替えや章の並び替えなどが簡単にできるようになります。また、ファイルが分割されているため、同時に複数の箇所を 1 人または複数人で分担して編集することが容易になります。

同時に複数のことを思いついてもファイルを複数開いてメモを取ることが出来るようになり、単一ファイルで編集しているときに比べて情報の置き場所が明確になったり、作業を中断してまたもとの場所にもどるのが楽になります。

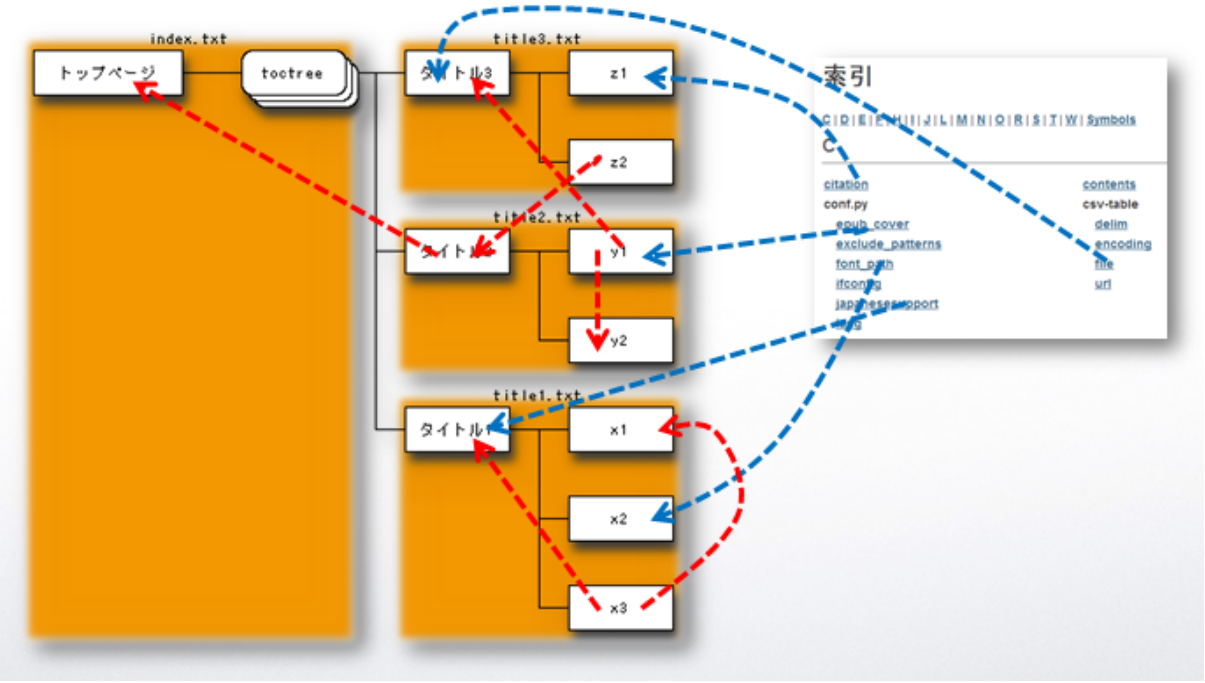
ノート: 某 Office 製品のように 1 つのファイルで縦方向に伸びていった場合、文章を書いている途中で別の所を修正したくなった時などに、一旦別の場所にスクロール移動してから、元の場所に戻ってくる必要がありますが、元の場所を探すのはなかなか面倒なものです。



4.3 ネットワーク構造

ツリー構造を横断するリンク

ドキュメントに記載するキーワードや別の章へのポインタを一定のルールで持たせておくことで、Wiki のような柔軟なネットワーク構造を持たせます。脚注、クロスリファレンス、用語集、索引などがこれに相当します。

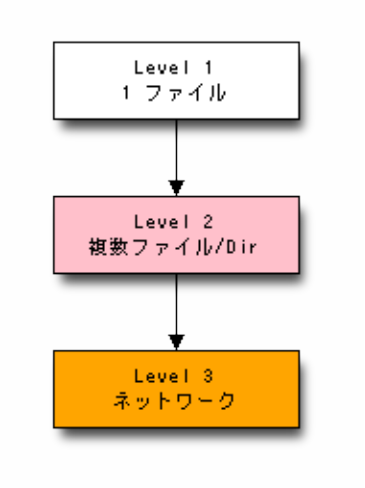


このリンクによって必要な情報にたどり着きやすいドキュメントを作ることが出来ます。用語集や API リファレンスなどを分かりやすく提供し、この機能を活用して用語の統一などを進める事が出来ます。

4.4 構造化の 3 レベル

ここまで紹介したように、構造化には 3 つのレベルがあります。

- 1 ファイル内での構造化
- 複数ファイル/ディレクトリに分割して階層化
- 直接の親子関係の無い構造間のネットワーク接続



このように、ファイル内外で文章を階層化・グルーピングしておくことでドキュメントが自然に整理され、書き進めやすくなります。

また、Wiki のような ネットワークのみのセミラティス構造 とは異なり、ツリー構造を基本としているため、見通しが良くなります。ツリー構造を主としてドキュメント全体に 1 本の背骨を通して、構造間を接続する神経ネットワークを追加してドキュメントの読み手に柔軟性を提供することが出来ます。

Sphinx によるドキュメント作成 ここまで紹介してきたような 構造化テキストやファイル・ディレクトリ構造を扱うことの出来るドキュメントツール として、様々なオープンソースや製品のドキュメント作成に活用されているドキュメンテーションツール `Sphinx` があります。

Sphinx ではどんなことが出来るのか、まずはツールの機能を紹介します。

5 Sphinx の機能

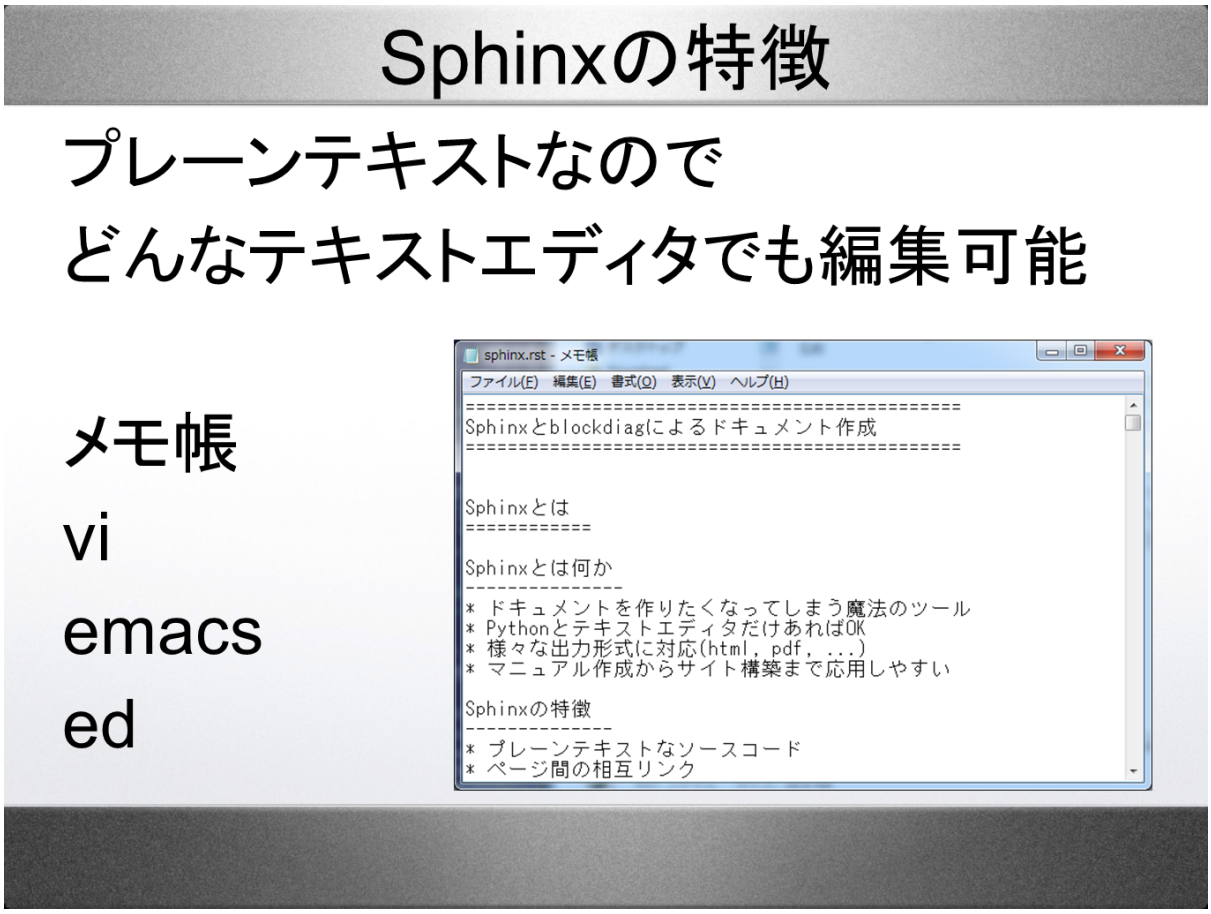


図 1: テキストフォーマットなので好みのエディタで編集することができます。ファイルを分割してあれば、複数同時に編集したり、プレーンテキストなので差分を比較したりといったことが簡単にできます。

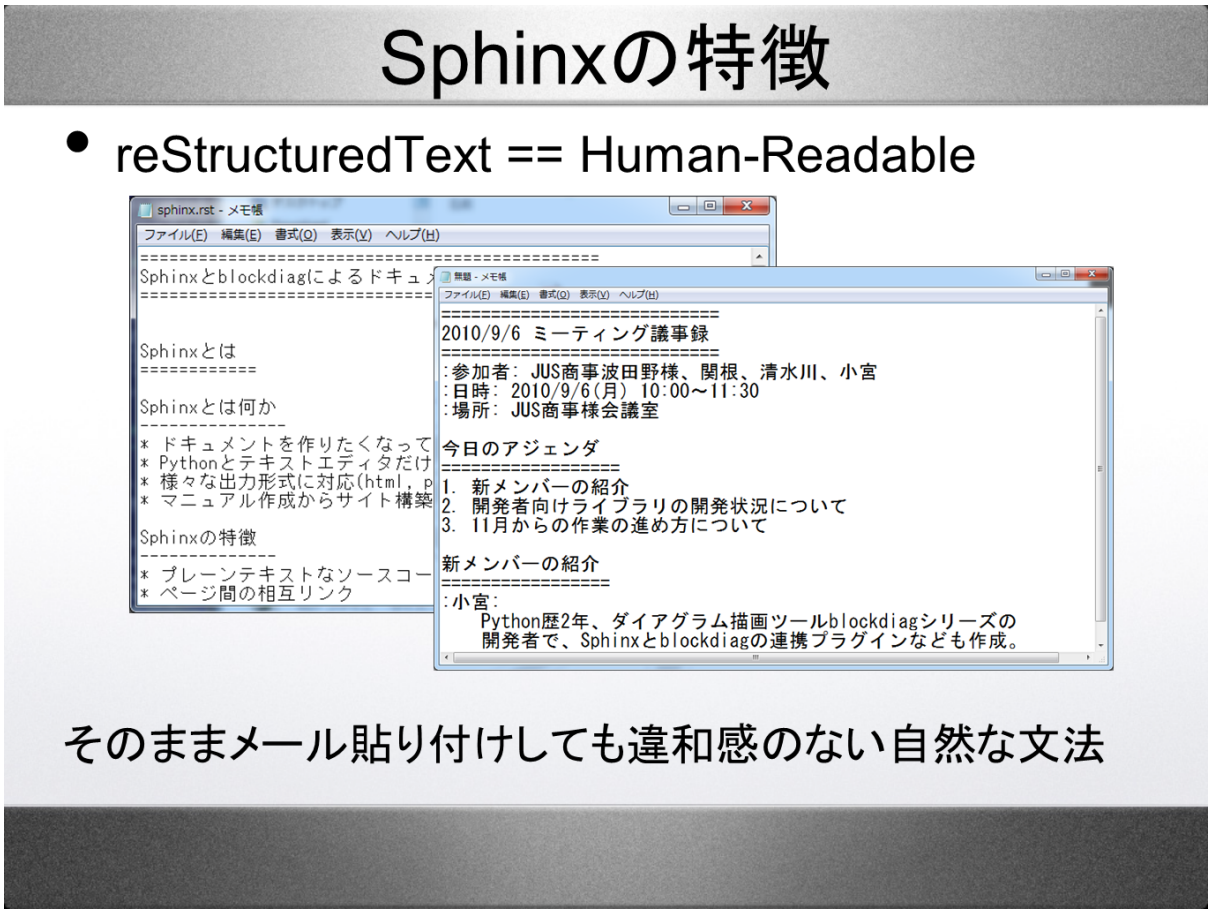


図 2: 文面は reStructuredText という構造化テキストで書きます。
人間が自然によめるフォーマットなので、議事録を reStructuredText で書いてそのままメールで送っても問題ありません。

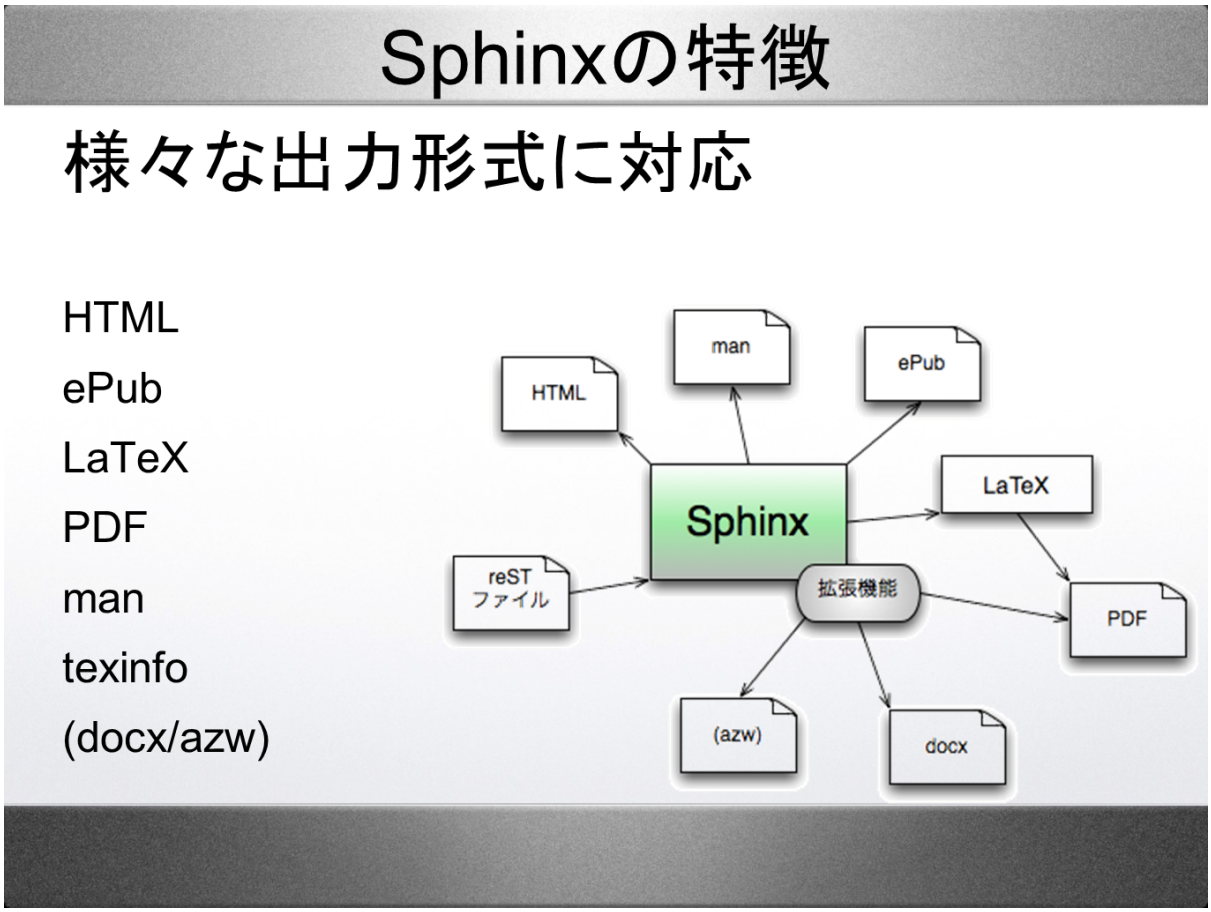


図 3: Sphinx は 1 つのソースから多数のフォーマットにアウトプット出来ます

reStructuredText で書かれたファイル・ディレクトリをビルドして、様々なフォーマットで出力することが出来ます。最も多く使われるのは HTML と PDF。他にも、ePub, LaTeX, man, texinfo, 等があります。

出力形式はプラグインで追加することも出来ます。プラグインは誰でも作る事が出来ます。docx や azw 出力のプラグインが開発途中です。はてな記法での出力を作った人もいるようです。

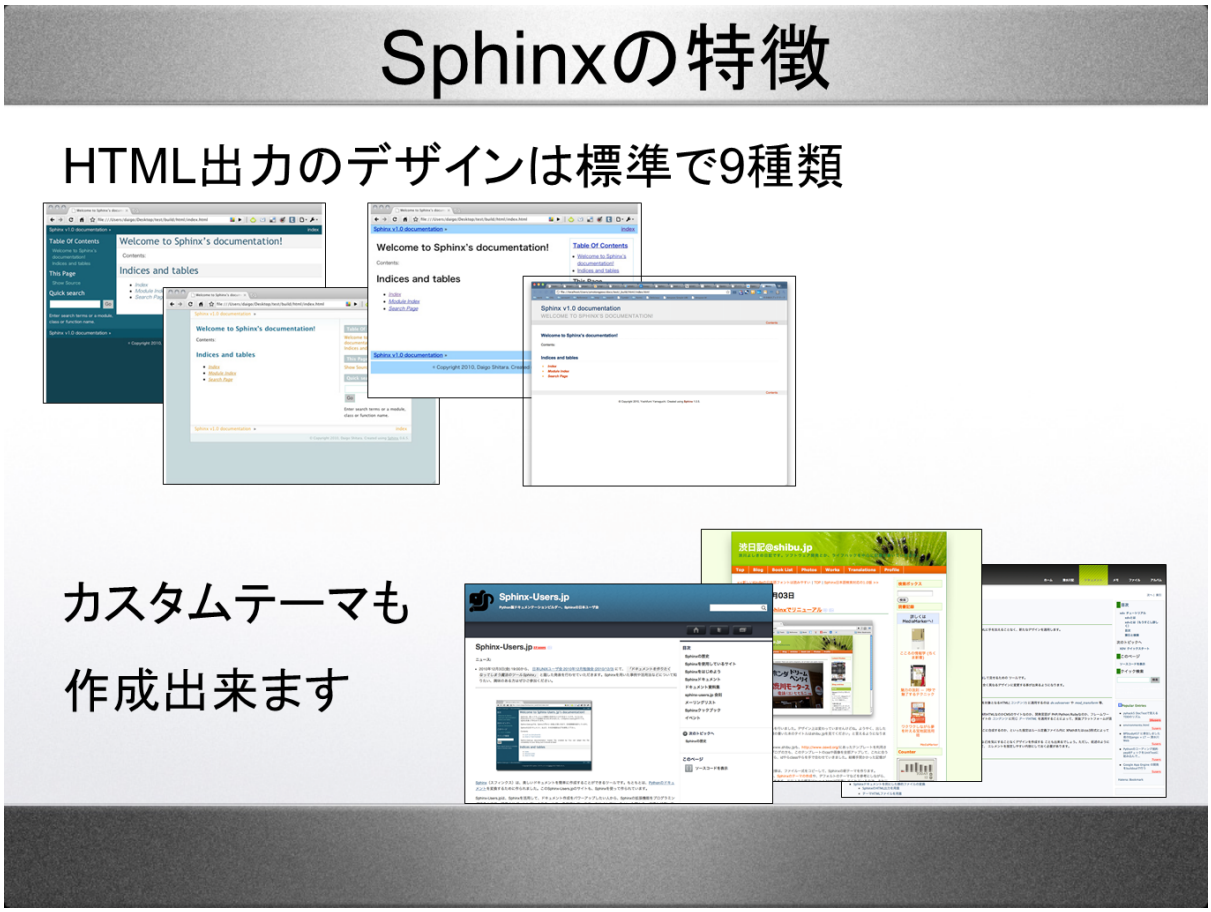


図 4: HTML 出力ならデザイン theme 切り替えにより見た目を変更可能

- テーマは標準で 9 種類
- ツリー構造なので HTML 出力ならパンくず表示もあります
- テーマを自作することも、他の人が作ったものを使うことも可能
- テーマの中には HTML でプレゼンテーション表示する機能も -> slide, s6

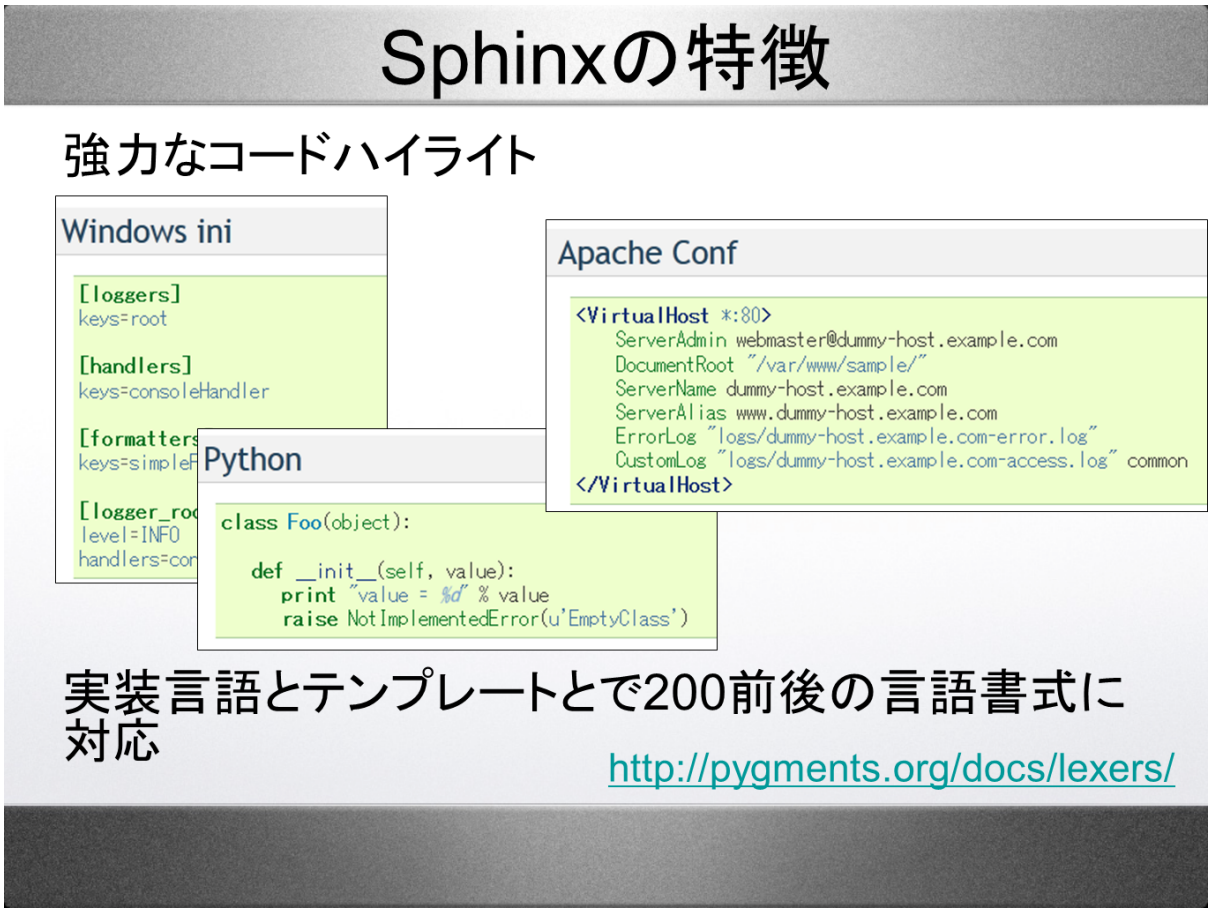


図 5: 強力なコードハイライト機能を搭載、対応フォーマット数は 200 前後

コードハイライト機能は同梱されている Pygments モジュールが提供しています。対応しているフォーマット数は、プログラム言語、設定ファイル、HTML テンプレートなどで合計 200 前後あり、随時対応数は増えています。

http://pygments.org/docs/lexers/

以下は ABC 順での対応フォーマット一覧です:

ABAP, ActionScript, ActionScript 3, Ada, ANTLR, ANTLR With ActionScript Target, ANTLR With C# Target, ANTLR With CPP Target, ANTLR With Java Target, ANTLR WithObjectiveC Target, ANTLR With Perl Target, ANTLR With Python Target, ANTLR WithRuby Target, ApacheConf, AppleScript, aspx-cs, aspx-vb, Asymptote, autohotkey, Bash, Bash Session, Batchfile, BBCode, Befunge, BlitzMax, Boo, Brainfuck, C, C#, C++, c-objdump, cfstatement, Cheetah, Clojure, CMake, CoffeeScript, Coldfusion HTML, Common Lisp, cpp-objdump, CSS, CSS+Django/Jinja, CSS+Genshi Text, CSS+Mako, CSS+Myghty, CSS+PHP, CSS+Ruby, CSS+Smarty, Cython, D, d-objdump, Darcs Patch, Debian Control file, Debian Sourcelist, Delphi, Diff, Django/Jinja, Duel, Dylan, Embedded Ragel, ERB, Erlang, Erlang erl session, Evoque, Factor, Felix, Fortran, GAS, Genshi, Genshi Text, Gettext Catalog, Gherkin, GLSL, Gnuplot, Go, GoodData-CL, Groff, Haml, Haskell, haXe, HTML, HTML+Cheetah, HTML+Django/Jinja, HTML+Evoque, HTML+Genshi, HTML+Mako, HTML+Myghty, HTML+PHP, HTML+Smarty, HTML+Velocity, Hybris, INI, Io, Ioke, IRC logs, Jade, Java, Java Server Page, JavaScript, JavaScript+Cheetah, JavaScript+Django/Jinja, JavaScript+Genshi Text, JavaScript+Mako, JavaScript+Myghty, JavaScript+PHP, JavaScript+Ruby, JavaScript+Smarty, Lighttpd configuration file, Literate Haskell, LLVM, Logtalk, Lua, Makefile, Makefile, Mako, MAQL, Mason, Matlab, Matlab session, MiniD, Modelica, Modula-2, MoinMoin/Trac Wiki markup, MOOCode, MuPAD, MXML, Myghty, MySQL, NASM, Newspeak, Nginx configuration file, NumPy, objdump, Objective-C, Objective-J, OCaml, Ooc, Perl, PHP, PostScript, POVRay, Prolog, Properties, Protocol Buffer, Python, Python 3, Python 3.0 Traceback, Python console session, Python Traceback, Ragel, Ragel in C Host, Ragel in CPP Host, Ragel in D Host, Ragel in Java Host, Ragel in Objective C Host, Ragel in Ruby Host, Raw to-

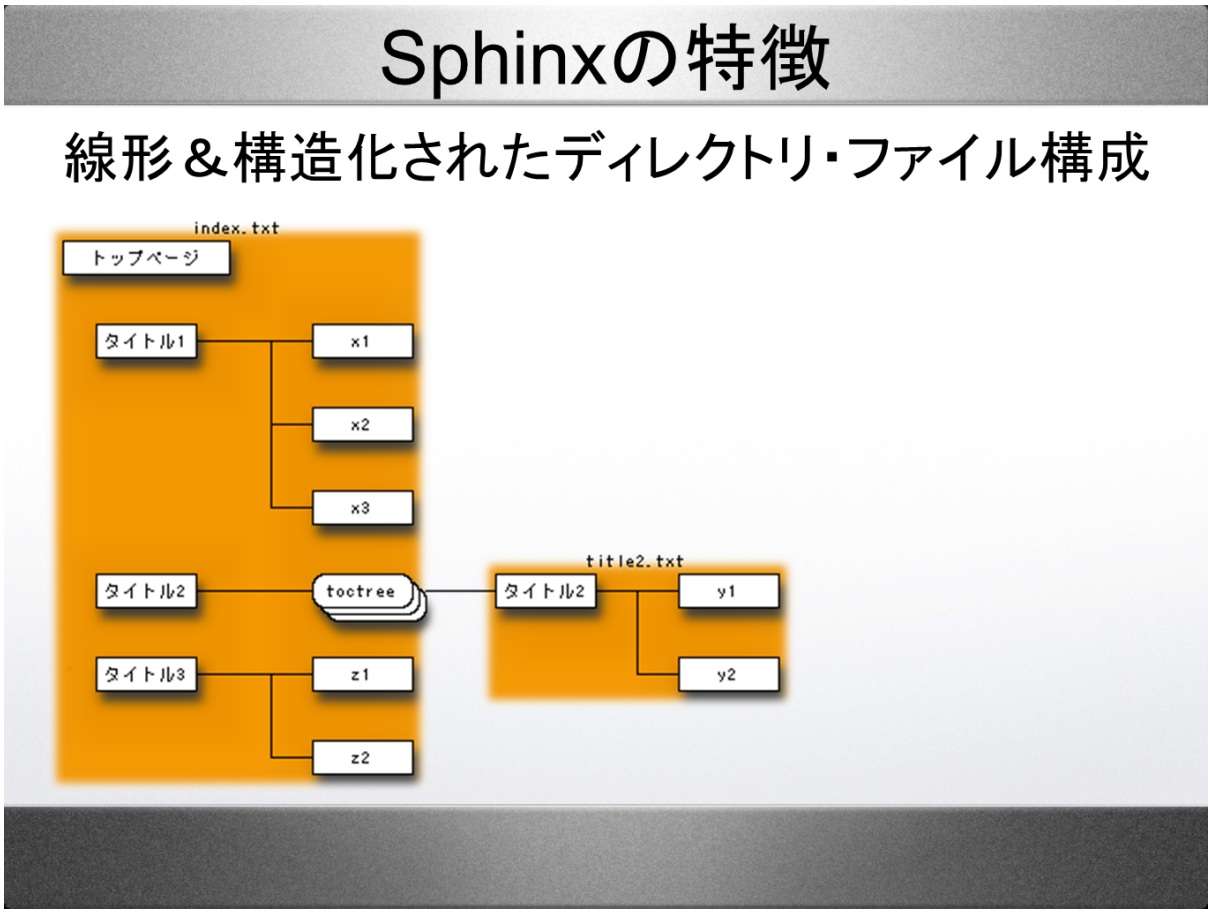


図 6: reStructuredText と Sphinx による概念の構造化

情報の階層化とグルーピングをファイルやディレクトリで行うことにより、情報の所属を明確にしたり、付け替えを容易にします。

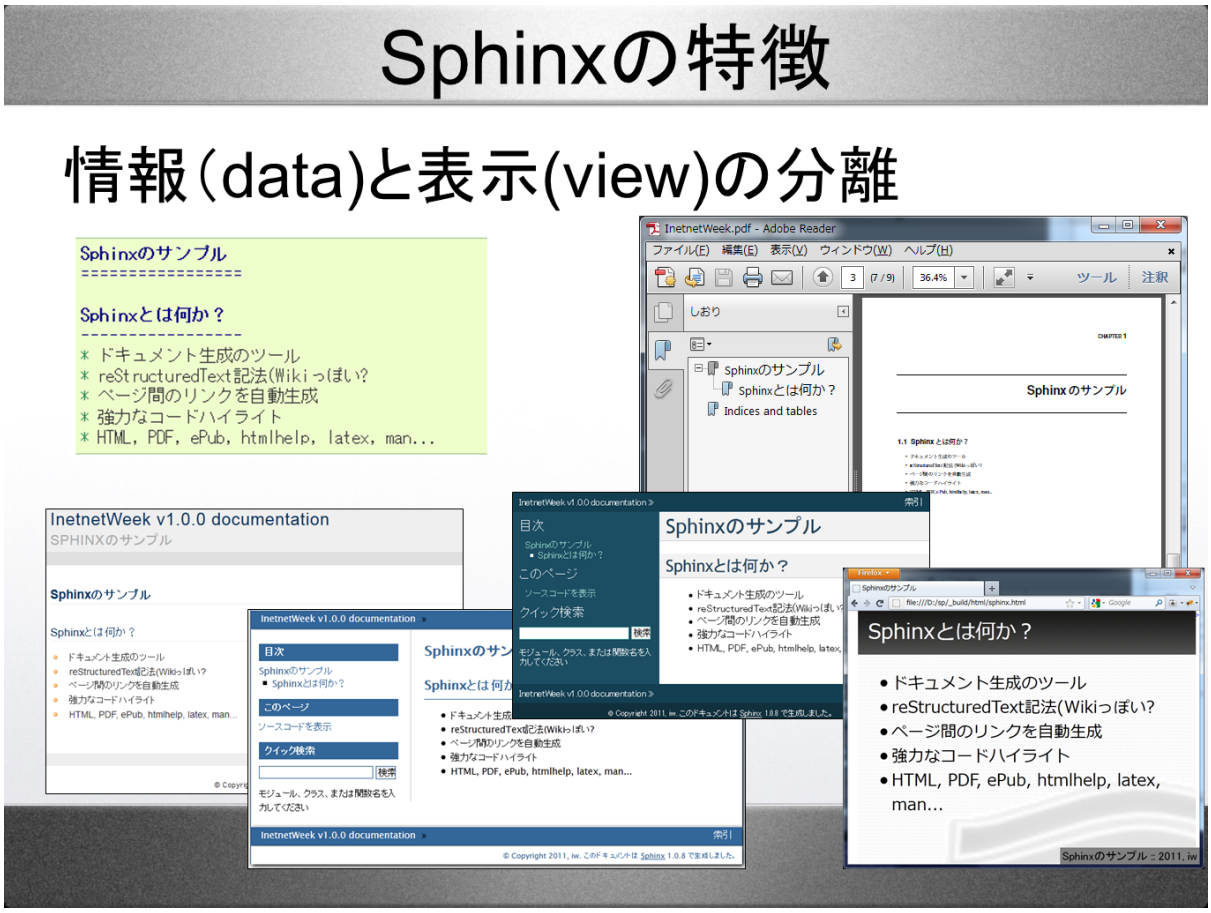


図 7: 情報 (data) と表示 (view) の分離により記述に集中できます

- 構造化テキストで書いている時は最終的なアウトプットは見ない
- 論理構造化された本文の「記述に集中できる」
- 最終的なデザインは Sphinx 側 (もしくはテーマを作るデザイナーさん) が調整
- 他のツールでは Word などのアウトライン機能に相当

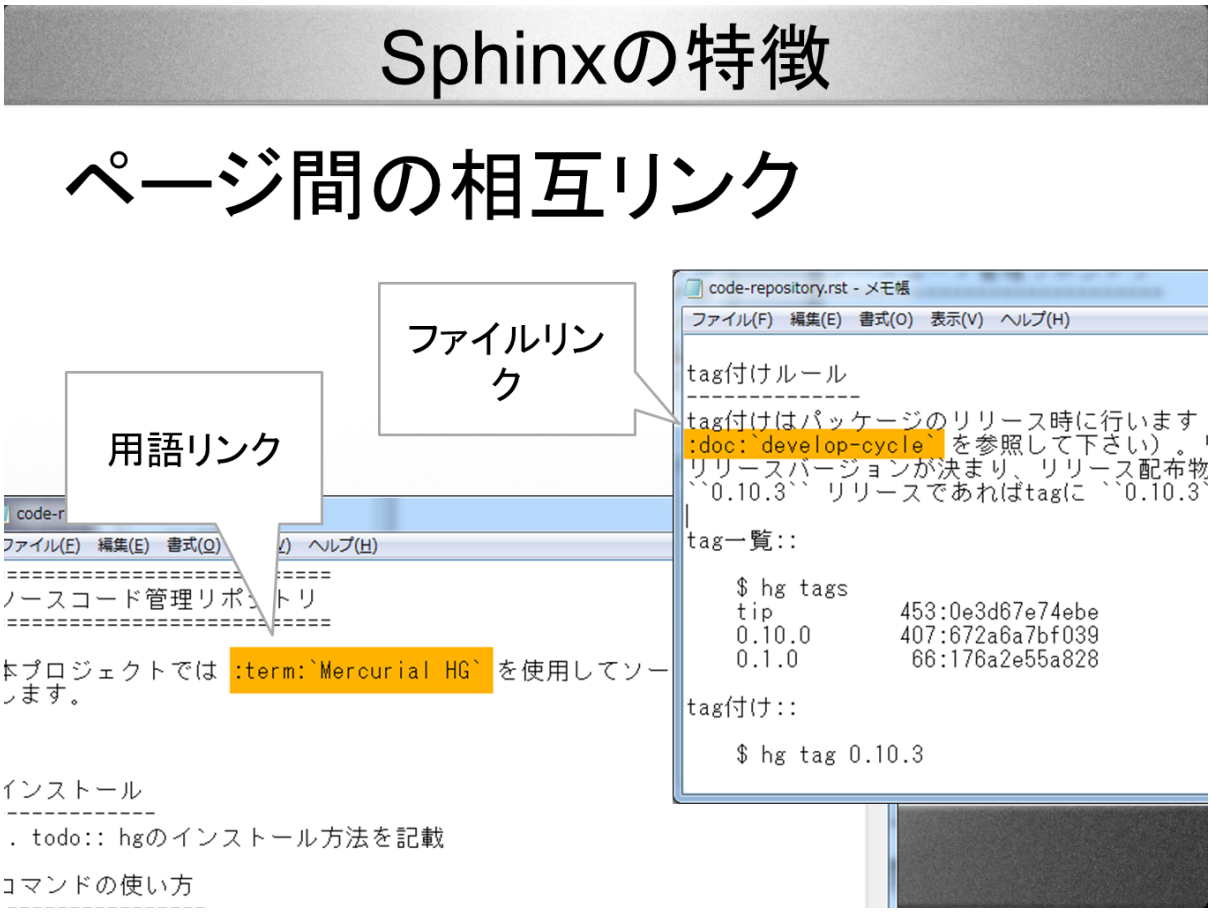


図 8: 脚注、クロスリファレンス、用語集、索引など、横の情報を接続する

Sphinx はツリー構造ですが、横の構造を柔軟に追加することが出来ます。文章を書いている途中で統一するべき用語を見つけたらその段階では**:term:‘用語‘** のように用語を書き込んでおけば、Sphinx が make 時に対応する用語説明が無いことを検出してくれるので、文章を書く手を止めずに後から用語説明を追加するといった使い方も出来ます。

他にも **:doc:‘../sub/index‘** のように相対パスで参照ページを指定すれば、そのページのタイトルとリンクを make 時に自動的にその部分に埋め込んでくれます。

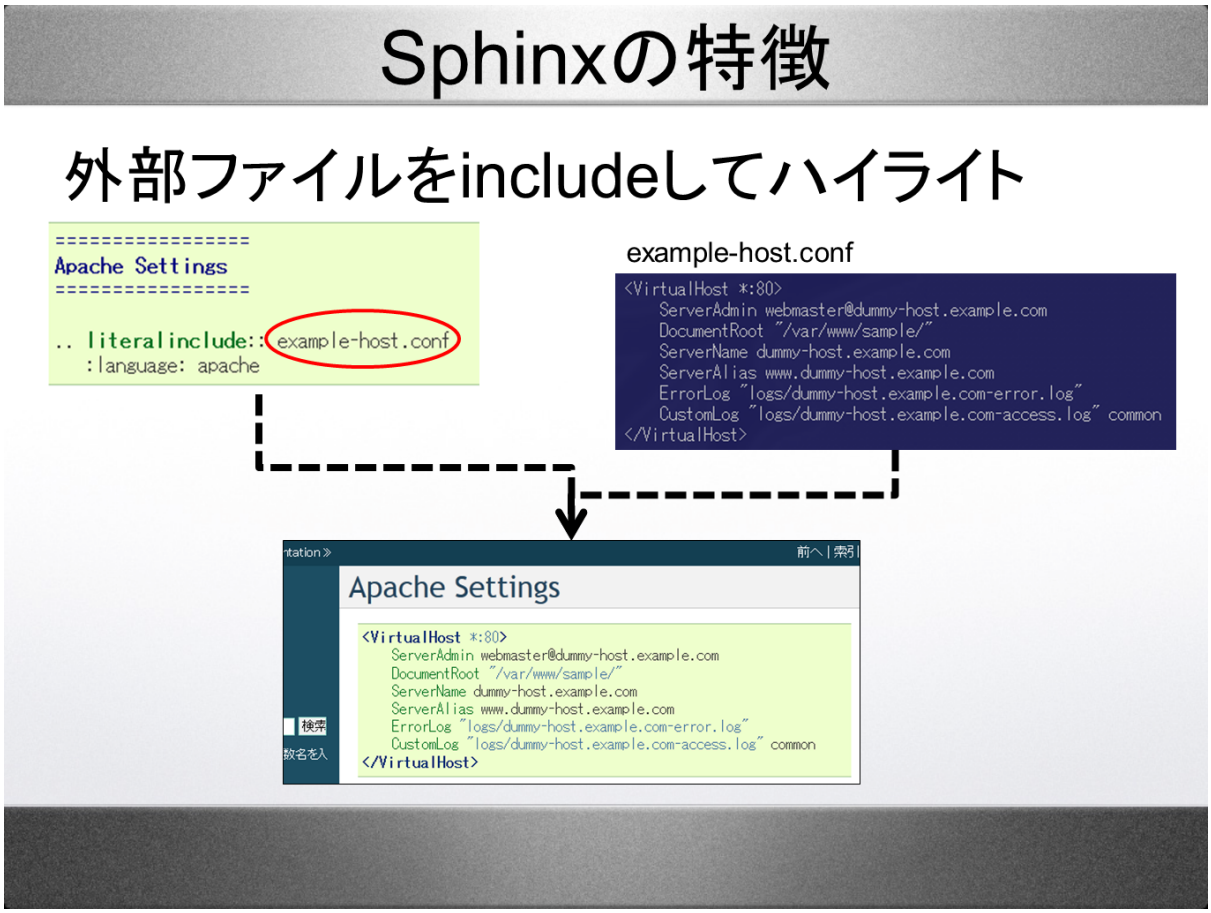


図 9: 外部ファイルの include とハイライト

外部ファイルを相対パスで読み込んで、コードハイライトした状態で出力します。literalinclude は読み込んだ内容を構造化テキストとして認識 **しません** 。

この機能を使用すると、設定ファイルをドキュメントに転記するといった手間を無くし、実際に動作している設定ファイルとドキュメントに書かれている内容が異なるという問題を防ぐことが出来ます。

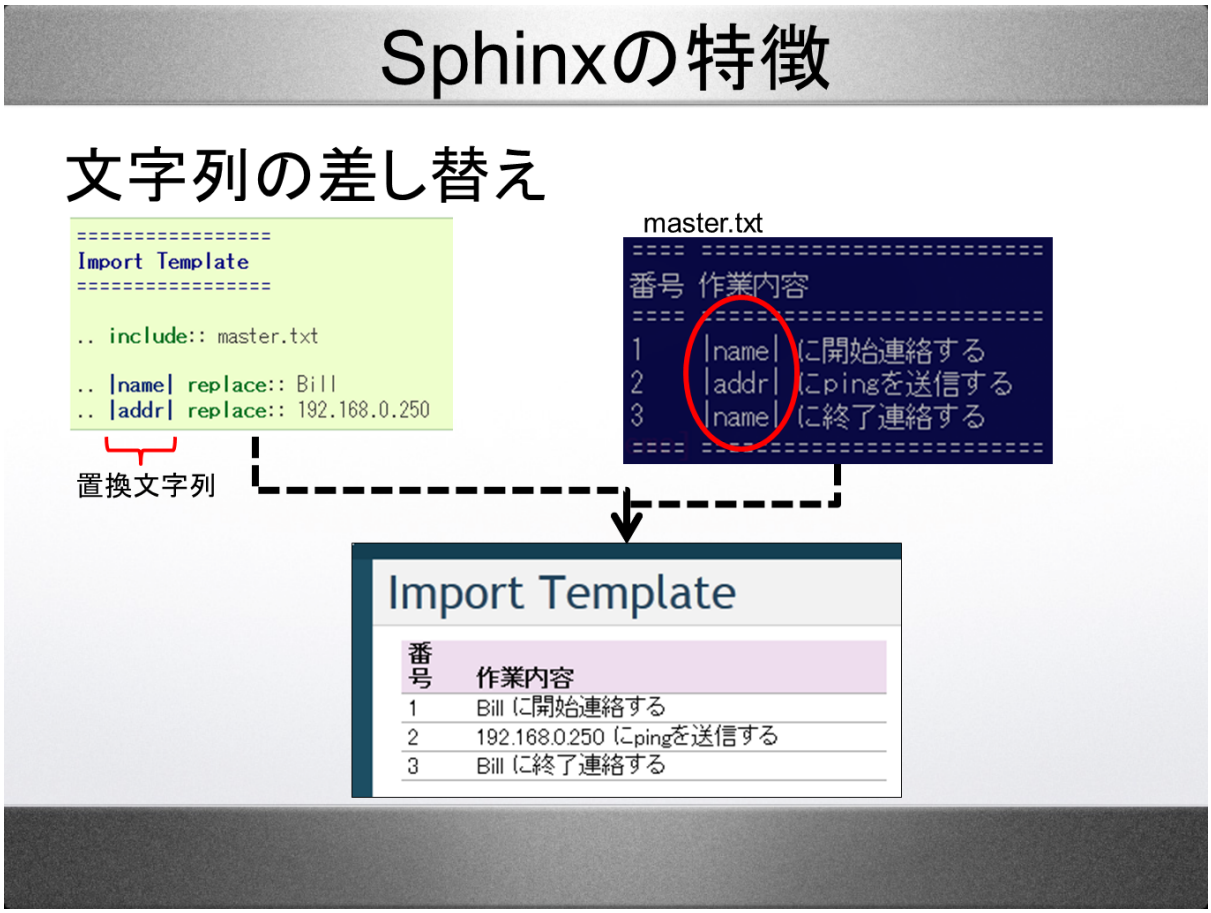


図 10: 外部ファイル読み込みとキーワード置換の組み合わせでテンプレート化

外部ファイルを読み込む include は、読み込んだ内容を構造化テキストとして認識するので、複数のページに同じ内容を埋め込みたい場合や、Sphinx のディレクトリ外にある reStructuredText フォーマットのファイルを埋め込む場合に使用します。

include とキーワード置換を組み合わせると、テンプレートのような使い方ができます。これを活用して、似たような手順で IP アドレス等が異なる場合に共通部分を別ファイルに分割しておくことで再利用することができます。

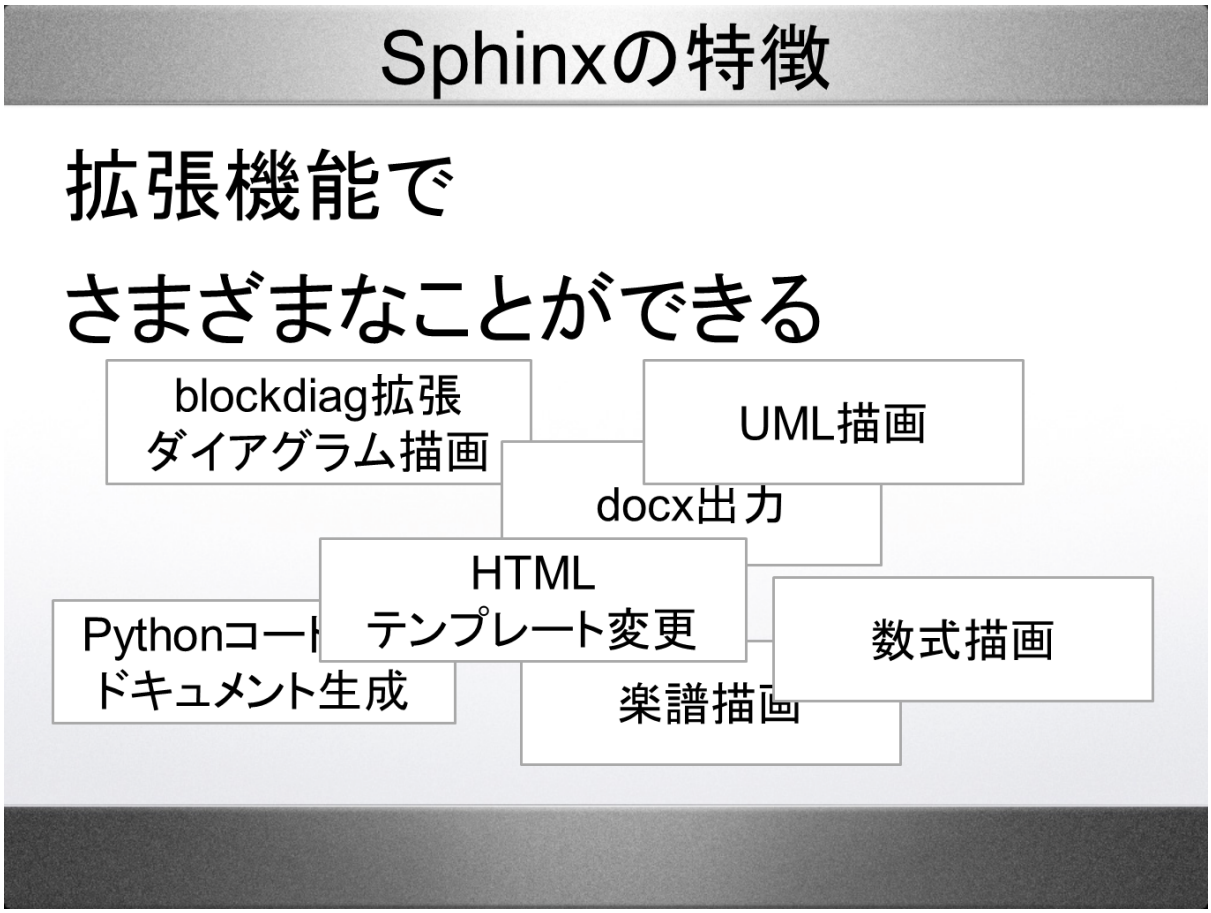


図 11: 拡張プラグインを追加して機能だけでなく文法の追加も可能

Sphinx は拡張プラグインの仕組みを持っているので、手軽に様々な拡張機能を追加することが出来ます。プラグインには、HTML の追加テーマや数式・図形描画、出力フォーマット追加などさまざまなものがあります。

本セッションの資料でも sphinxcontrib-blockdiag という拡張を使用しており、この拡張を用いる事で Graphviz のような拡張文法で図形を書けるようになります。

以下の記述から自動的に画像が生成されます:

```
.. blockdiag::

   {
      foo -> bar [label = "click", textcolor="red"];
      bar -> baz [label = "click"];
      baz -> foo;
   }
```

blockdiag について詳しくは http://blockdiag.com/ja/ を参照してください。

図 12: 包括的で英語よりも充実している日本語マニュアル・ドキュメント

オープンソースソフトウェアはドキュメントが英語だったりあまり書かれていなかったりするのでは‥と思われる事が多いのですが、Sphinx はドキュメンテーションのためのツールということもあり、包括的なドキュメントがあります。

ドキュメントは Sphinx-users.jp の活動によって、全て日本語に翻訳されています。また、日本独自のドキュメントとして導入チュートリアルや逆引き辞典など多くのドキュメントが http://sphinx-users.jp で公開されています。

図 13: Sphinx はドキュメントを書きたくなる機能で構成されています

- simple (読みやすいソース, WiKi に近い記法)
- structurable (構造を意識して書きたくなる)
- extensionable (柔軟な拡張性, blockdiag, etc)
- normal (ちょっとしたメモ書き風 -> 議事録, メール)

6 Sphinxを使ったドキュメント作成の流れ

この InternetWeek 向けプレゼン資料も Sphinx を使って草稿から最終成果物までを作りました。その流れを紹介します。

まず各セクションタイトルを箇条書きで書き出していきます。IW2011 本セッションの草稿はこんな感じでした。

index.rst:

```
==================================================================
InternetWeek 2011 document session
==================================================================

* 16:00 - 16:30 運用ドキュメントのあり方と課題 (30分 波田野 裕一 / 日本UNIXユーザ会)
* 16:30 - 16:50 運用ドキュメントのバージョン管理 〜 Mercurialを例にして(20分 清水川 貴
* 16:50 - 17:35 運用ドキュメントの構造化 〜 Sphinx による実例 (45分 清水川 貴之)
* 17:35 - 17:50 休憩

* 17:50 - 18:15 運用ドキュメントの公開と変更管理 (25分 波田野 裕一 / 日本UNIXユーザ会)
* 18:15 - 18:30 質疑応答 (15分)
```

これに内容として書きたい物をぶら下げて行きます。

index.rst:

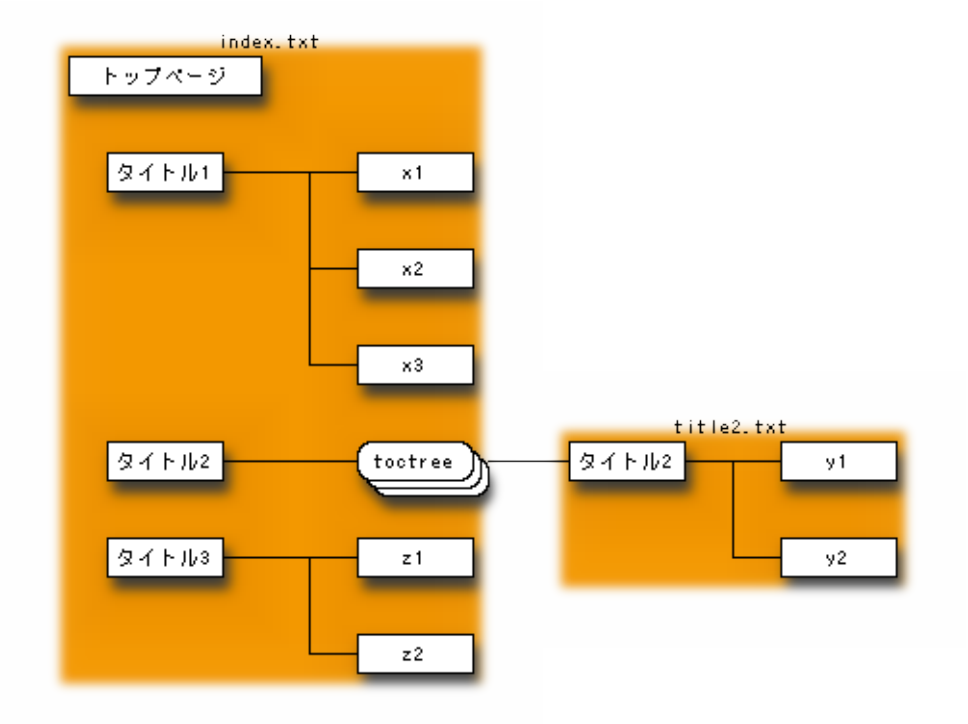
```
==================================================================
InternetWeek 2011 document session
==================================================================

* 16:00 - 16:30 運用ドキュメントのあり方と課題 (30分 波田野 裕一 / 日本UNIXユーザ会)

  * 運用ドキュメントの価値論
  * 運用ドキュメント管理の現状と課題
  * 運用ドキュメント新潮流

* 16:30 - 16:50 運用ドキュメントのバージョン管理 〜 Mercurialを例にして(20分 清水川 貴

  * 分散バージョン管理の時代が来た
  * なにが嬉しい (ネットワークレスコミット、ブランチ、バックアップ不要)
  * なにが違う
  * Mercurialを例に

* 16:50 - 17:35 運用ドキュメントの構造化 〜 Sphinx による実例 (45分 清水川 貴之)

  * 構造化記法による恩恵 (include / replace / クロスリファレンス)
  * ドキュメントプロセッサ/構造化記法の台頭
  * シングルソース、マルチアウトプット
  * Sphinx/reSTを例に
  * reSTの嬉しいところ
  * 業務事例
```

* 17:35 - 17:50 休憩
* 17:50 - 18:15 運用ドキュメントの公開と変更管理 (25 分 波田野 裕一 / 日本 UNIX ユーザ会)
  * 修正の自動反映 (hook を使った事例)
  * チケット駆動ドキュメント管理 (Redmine と連携)
  * まとめ
* 18:15 - 18:30 質疑応答 (15 分)

このように各セッションでどのような内容を紹介するか、目次に相当するものを書いていきます。

ここまでは 1 つのファイルでどんどん書き足していきますが、あまり詳細をぶら下げすぎると構造が見えにくくなってきます。そこで、詳細を書きたい部分についてはファイルを分割して書くことにします。

以下のように分割するイメージです。



それでは、3 つめのブロックについて、別のファイルに以下のように移動しましょう。セクション名から document-structuring.rst というファイル名にして内容を移動します。

document-structuring.rst:

* 16:50 - 17:35 運用ドキュメントの構造化 ～ Sphinx による実例 (45 分 清水川 貴之)
  * 構造化記法による恩恵 (include / replace / クロスリファレンス)
  * ドキュメントプロセッサ/構造化記法の台頭

```
* シングルソース、マルチアウトプット
* Sphinx/reST を例に
* reST の嬉しいところ
* 業務事例
```

reST の文法にあわせてタイトルを設定してインデントを調整します。箇条書きの 1 項目に 6 つの箇条書き項目がぶら下がっていましたが、以下のようにタイトル 1 つとインデント無しの箇条書き 6 項目に書き換えました。

document-structuring.rst:

```
==========================================================================
16:50 - 17:35 運用ドキュメントの構造化 ～ Sphinx による実例 (45 分 清水川 貴之)
==========================================================================

* 構造化記法による恩恵 (include / replace / クロスリファレンス)
* ドキュメントプロセッサ/構造化記法の台頭
* シングルソース、マルチアウトプット
* Sphinx/reST を例に
* reST の嬉しいところ
* 業務事例
```

次に、箇条書きの各項目はこのファイル内でサブセクションとなるので、以下のように書き換えます。

document-structuring.rst:

```
==========================================================================
16:50 - 17:35 運用ドキュメントの構造化 ～ Sphinx による実例 (45 分 清水川 貴之)
==========================================================================


構造化記法による恩恵 (include / replace / クロスリファレンス)
=============================================================

(ここに書く予定)

ドキュメントプロセッサ/構造化記法の台頭
=======================================

(ここに書く予定)

シングルソース、マルチアウトプット
==================================

(ここに書く予定)

Sphinx/reST を例に
==================

(ここに書く予定)
```

```
reST の嬉しいところ
====================

(ここに書く予定)

業務事例
=========

(ここに書く予定)
```

最後に、分割した document-structuring.rst をツリー構造に組み込むために元のファイルに toctree を設定します。

index.rst:

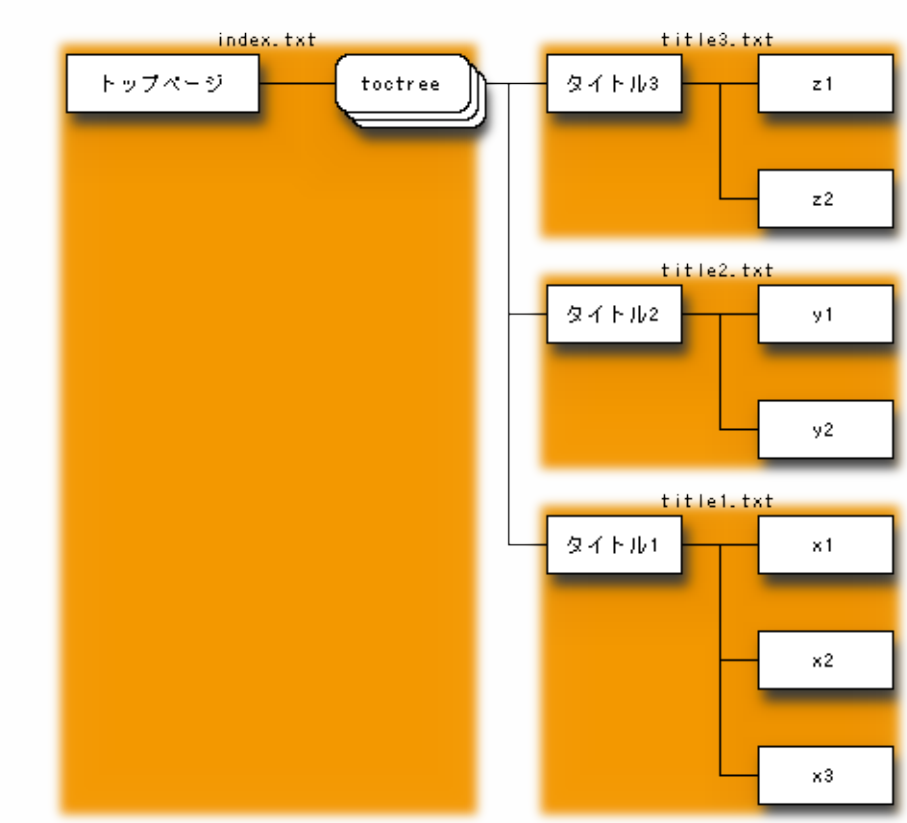
```
=================================================================
InternetWeek 2011 document session
=================================================================

* 16:00 - 16:30 運用ドキュメントのあり方と課題 (30 分 波田野 裕一 / 日本 UNIX ユーザ会)

  * 運用ドキュメントの価値論
  * 運用ドキュメント管理の現状と課題
  * 運用ドキュメント新潮流

* 16:30 - 16:50 運用ドキュメントのバージョン管理 〜 Mercurial を例にして (20 分 清水川 貴

  * 分散バージョン管理の時代が来た
  * なにが嬉しい (ネットワークレスコミット、ブランチ、バックアップ不要)
  * なにが違う
  * Mercurial を例に

.. toctree::

   document-structuring

* 17:35 - 17:50 休憩

* 17:50 - 18:15 運用ドキュメントの公開と変更管理 (25 分 波田野 裕一 / 日本 UNIX ユーザ会)

  * 修正の自動反映 (hook を使った事例)
  * チケット駆動ドキュメント管理 (Redmine と連携)
  * まとめ

* 18:15 - 18:30 質疑応答 (15 分)
```

これを繰り返して各セクションを別ファイルに分割していった結果、index.rst は以下のようになりました。

index.rst:

```
==================================================================
InternetWeek 2011 document session
==================================================================

.. toctree::

   overview
   version-control
   document-structuring
   share-modify
```

構造は以下のように分割されました。



Which: 実際の現場での Sphinx 活用例

7 利用シーン

ドキュメントを書くシーンとして以下のような部署の担当者が、どんな時にどんなドキュメントを書く、または必要としますよね？

- 担当 A: プロジェクトの要件をまとめる
- 担当 B: プロジェクトのレギュレーションを決める

- 担当 C: プロジェクトを進行して議事録を作る, また設計と実装を行う
- 担当 D: 納品物としてドキュメントを作成する
- 担当 E: 運用する

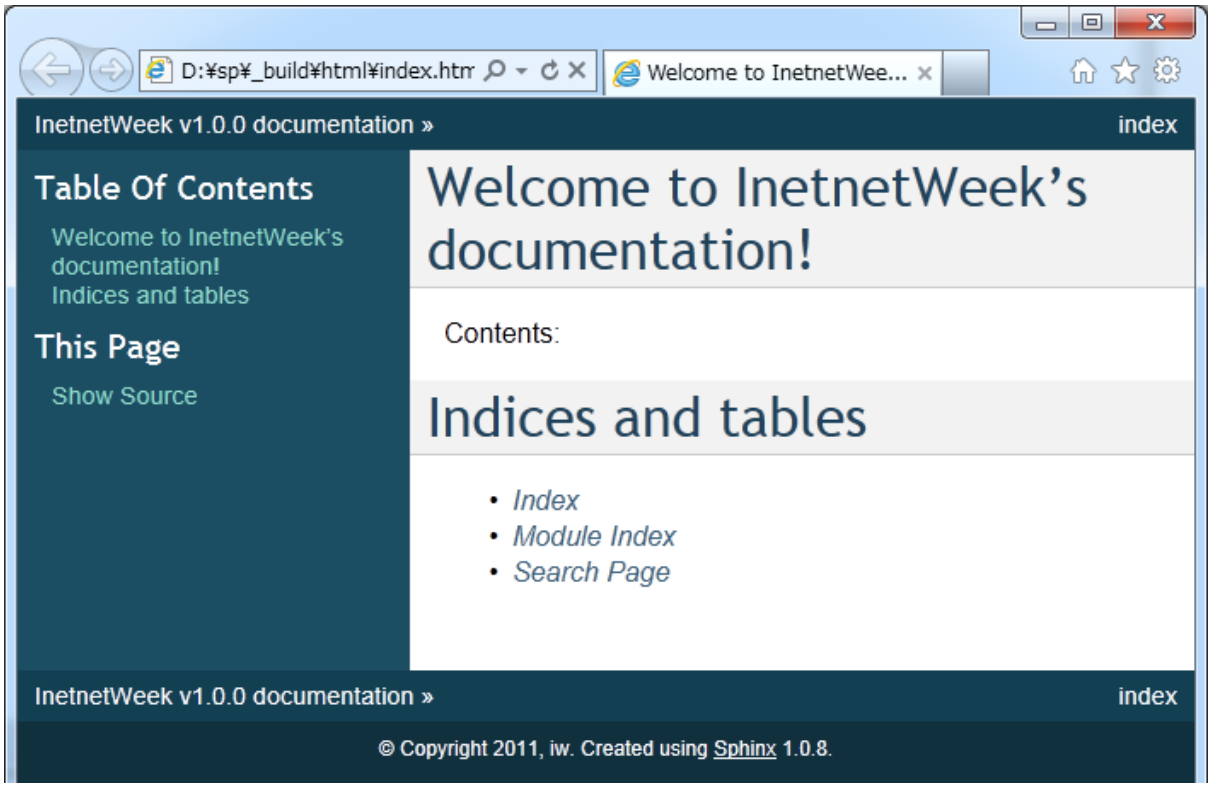
8 ドキュメントテンプレート、一巡り

Sphinx には `sphinx-quickstart` というコマンドがこのコマンドを実行すると Sphinx として make html などを実行するために必要な基本的なファイルが用意されます。

ここで出力されるのは、設定が書かれた conf.py と index.rst です。index.rst の内容を見てみましょう。

```
Welcome to InternetWeek's documentation!
========================================

Contents:

.. toctree::
   :maxdepth: 2

Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

これを `make html` すると以下の HTML が出力されます。



しかしこれはドキュメントを書き始めるためのテンプレートというにはあまりにも弱すぎます。そこで、先ほどの利用シーンを元に、テンプレートが必要になりそうなドキュメントの種類をいくつか考えてみました。

- ポリシー/ルール系
- 運用要求/要件
- 議事録 (要求を設計につなぐ内容)
- 設計
- 運用手順書

8.1 reST サンプル: 議事録

議事録は配布形態や情報の塊として見たときに、1 つのファイルに収まる構成になっています。打ち合わせのちょっとしたメモ書きを後から議事録に清書するというパターンも多いでしょう。

このように、ちょっとした単独のメモを後から再利用したい場合にも構造化をを意識して以下のようなメモを残すようにします。

```
==============================
2010/6/3 ミーティング議事録
==============================
:参加者: IW 商事波田野様、SPHINX 渋川、清水川、山口
```

:日時: 2010/6/3(木) 10:00〜11:30
:場所: IW 商事様会議室

スケジュール

* 6/7(月)〜8/13(金) 10 週 10 イテレーションで開発 (SPHINX)
* 8/16(月)〜検証 (IW 商事/SPHINX)
* 9/上〜 他環境と結合していく (IW 商事)

役割分担

* 全体の S/C の関係を描く【TODO:IW 商事】
* 設計【TODO:SPHINX】
* 実装【TODO:SPHINX】
* 結合【TODO:IW 商事】
* 運用【TODO:IW 商事】

ターゲットスコープ

* 認証ライブラリ群
* 認証の仕組みをアプリから連携可能とする範囲

定例 MTG

* 月曜と木曜の午前

今後のアクション

SPHINX

* システム構成とそれぞれで使用するライブラリを検討/確認し、資料にまとめる
* 公開鍵 (1024bit 以上) とアクセス元 IP をお渡しする SPHINX -> IW 商事
* 課題管理掲示板の開設（波田野様、渋川、清水川、山口）

IW 商事

* 既存の全体の構成と今回のシステムの位置づけが分かる資料を作成
* リファレンス環境となる VM を作成

次回ミーティング

* 日時： 2010/6/7(月) 10:00〜11:30
* 場所： IW 商事様会議室

この時点から reStructuredText(reST) 文法で書かれていますが、それを知らなくてもこの内容は意識せずに読むことが出来ます。