第10章 プロジェクトのドキュメント作成

ドキュメント作成は、開発者、ときにはマネージャもさぼってしまいがちな作業です。開発サイクルの終了に向かうにつれて時間がなくなったり、自分は文章を書くのが苦手だと考えている人がいたりするのが、さぼってしまう原因です。確かに文章を書くのが苦手な人もいますが、大多数の開発者とマネージャは、すばらしいドキュメントを書くことができます。

急いで書かれた文書が集められた結果、内容が整理されていないドキュメントが作られてしまうこともあります。開発者はたいてい、この手の仕事をするのが好きではありません。既存のドキュメントをアップデートしなければならない場合には、さらに事態は悪化します。マネージャがそのような事態にどう対処すればいいのかわからない場合、多くのプロジェクトでは貧相で内容の古いドキュメントしか提供できないということになります。

しかし、プロジェクトの開始時にドキュメント作成プロセスを整備し、ドキュメントをソフトウェアのコードのモジュールと同じように扱うようにすると、ドキュメントを書くのがもっと簡単になります。いくつかのルールに従うと、書くのが楽しくなる場合すらあります。

本章では、次のような内容を通じて、プロジェクトのドキュメントを書き始めるための Tips を紹介していきます。

- 技術的な文章を書くためのベストプラクティスをまとめた、7つのルール
- ほとんどの Python プロジェクトで使用されている、reStructuredText というプレーンテキストのテキストマークアップ文法の入門
- 良いドキュメントを構築するためのガイド

10.1 技術文書を書くための7つのルール

さまざまな面から見て、良いドキュメントを書くことは良いコードを書くことよりも簡単です。多くの開発者は、ドキュメントを書くのはとても難しいことであると考えていますが、いくつかのシンプルなルールに従うだけで本当に簡単になります。

ここでお話しするのは、ポエムの本を書くための方法ではなく、プログラムの設計や API、コードベースを作り上げる上で必要となるものを理解するための包括的なテキストを書くための方法になります。

すべての開発者は、そのようなテキストを書くことができます。本節では、あらゆる場面で適用できる 7 つのルールを提供します。

- 2 つのステップで書く：まずはアイデアにフォーカスし、その後レビューを行ってテキストの形を整えます。
- 読者のターゲットを明確にする：それを読むのはだれですか？
- シンプルなスタイルを使用する：わかりやすくシンプルに保ちます。正しい文法を使用しましょう。
- 情報のスコープを絞る：一度に 1 つの概念だけを導入します。
- 実在するようなコードのサンプルを使用する：`Foo`、`Bar` はもうやめましょう。
- なるべく少なく、かつ十分なドキュメント：あなたが書いているのは書籍ではありません！
- テンプレートの使用：読み手がどこに何が書いてあるかを把握しやすくなります。

これらのルールは、Andreas Rüping の著作 *Agile Documenting* にインスパイアされています。その本の中で書かれていることにも適合します。この本は、ソフトウェアプロジェクトの中で最高のドキュメントを生成する方法に焦点を当てています。

10.1.1 2つのステップで書く

Peter Elbow は著書 *Writing with Power* の中で、どんな人も、一回で完璧な文章を書くことはほぼ不可能であるということを説明しています。多くの開発者がドキュメントを書くときに、直接完全なテキストを書き上げようとするところに問題があります。これを改善するプラクティスの 1 つは、毎回 2 行だけ文章を書き、読み返して修正するというものです。これをすると、文章の作成と、テキストのスタイルの両方にフォーカスすることになります。

この方法には、脳に負担がかかりすぎるのと、結果が最良のものになるとは限らないという問題があります。内容を十分に考える前に、多くの時間とエネルギーをテキストのスタイルと形を整えるのに費やしてしまうからです。

別の方法は、まずはスタイルと構成を考えないで、中身にフォーカスするというものです。どのように書くかは別にして、すべての考えをまず紙の上に書き出します。開発者は、流れるように次々と書いていきます。文法まちがいを理由に手を止めてはいけません。文法は内容に関わることではないからです。考えていることを紙の上に書き出し続けている限りは、文の意味がほとんど理解できなくても問題はありません。まずは言いたいことを、大ざっぱに分類しながら書き出すことだけに集中します。

そうしていくと、開発者は自分が言いたかった内容にのみ意識を集中させることができるため、最初に考えていたよりも多くの内容が頭の中から次々に出てくるようになります。

この方法には、直接は関係ないトピックに関するアイデアも容易に出てきやすくなるという、別の効果もあります。そのようなアイデアが出てきた場合には、別の紙かファイルに書き出すようにしましょう。そのように残しておけばその内容が失われることはありませんし、今取りかかっているメインのドキュメントに、すぐに戻ることができます。

二番目のステップは、今書いた全文を読み返して、多くの人にわかりやすいように磨き上げる作業になります。テキストを磨くというのは、スタイルを良くして、まちがいを直し、構成を直し、重複した情報を整理するということです。

ドキュメントを書くために使用できる時間が限られているときは、この 2 つのステップ（一番目のステップで内容を書き出し、二番目のステップでテキストを掃除してまとめる）の時間のうち、二番目の時間を削って一番目と同じ程度にすると良いでしょう。

まずは、中身にフォーカスしましょう。スタイルときれいさは二の次です。

10.1.2 読者のターゲットを明確にする

テキストを書き始めるときに、ドキュメント作成者が考えておくべきシンプルな質問が 1 つだけあります。それは「誰がそのドキュメントを読むのか？」です。

これは、いつも明確になっているわけではありません。ソフトウェアを構成するパーツがどのように動作しているかを説明しているドキュメントであれば、そのコードを入手して使用するすべての人に対して書かれる場合がほとんどでしょう。読者は、問題に対して適切な技術的解決策を探しているマネージャかもしれませんし、そのコードに新しい機能を追加する必要のある開発者かもしれません。設計者は、アーキテクチャの視点から、そのパッケージが自分のニーズに合っているかどうかを知るために読むでしょう。

ドキュメントを書く場合には、シンプルなルールを適用しましょう。それぞれのテキストには、1 種類の読者のみを設定すべきです。

この哲学を持ってドキュメント作成に取りかかると、作業が簡単になります。ドキュメント作成者は、その読者が何をしているのかを正確に知ることができるため、簡潔で正確なドキュメントを提供できるようになります。幅広い読者に合わせようとして、漠然としたドキュメントになることがなくなります。

それぞれのドキュメントが何のために書かれているのかを紹介する文を、それぞれ 1 行だけつける

のは良いプラクティスです。このコンパクトな説明によって、読者を適切なドキュメントに誘導することができます。

```
AtomisatorはRSSフィードを取得して、フィルタリングを行ってから
データベースに格納する製品です。

あなたが開発者なら、APIの説明(api.txt)をお読みください。

あなたがマネージャなら、機能リストとFAQ(features.txt)をお読みください。

設計者であれば、アーキテクチャとインフラストラクチャに関するメモ
(arch.txt)をお読みください。
```

このように、読者が考えていることに注意することで、良いドキュメントが作成できるようになります。

書き始める前に、読者がだれかを知っておきましょう。

10.1.3 シンプルなスタイルを使用する

Seth Godin は、マーケティングの分野におけるベストセラー作家の一人です。あなたが興味を持つと思われる *Unleashing the Ideavirus*（邦題『バイラルマーケティング――アイディアバイルスを解き放て！』大橋禅太郎訳、翔泳社刊）はインターネット上で無料で読むことができます（http://en.wikipedia.org/wiki/Unleashing_the_Ideavirus）。

彼は最近、自分の本が売れた理由を理解しようと、ブログ上で分析を行いました。彼はマーケティングの分野のすべてのベストセラーのリストを作成し、それぞれの本の一文当たりの平均ワード数を比較しました。

彼が発見したのは、彼の本の文は平均 13 ワードで、他の本と比べて一番短かったということでした。このシンプルな事実から、読者は長くてスタイリッシュな文よりも短くて簡単な文を好むことが証明できた、と彼は説明しています。

文章を短く保つと、読者が内容を読み取って、処理をして理解するために必要な脳のエネルギー消費が少なくなります。技術的なドキュメントの目的は、読者に対してソフトウェアのガイドを提供することです。フィクションの物語を書いているわけではないため、Stephen King の小説よりも電子レンジの説明書に近い文章にすべきです。

シンプルにするために覚えておくべき Tips は次の通りです。

- シンプルな文を使用しましょう。2 行以上にわたる文であってはなりません。
- 各段落は、最長でも 3 つか 4 つの行で構成され、主要な 1 つの考えのみを説明します。ウサギ（=その考え）が呼吸できるように。[*1]
- 何度も繰り返さないようにしましょう。読者に理解させようとする場所で、何度も何度も考えを繰り返すような、ジャーナリスト的なスタイルは避けましょう。
- 複雑な時制は不要です。現在形で十分です。
- 本当に優れたドキュメント作成者でなければ、ジョークを入れてはいけません。技術文書をユーモラスにするのは困難です。それをマスターしている作成者は少ししかいません。どうしてもジョークを入れたい場合には、コードサンプルに限定してください。そうすれば満足がいくでしょう。

あなたが書いているのはフィクションではないので、できるだけシンプルなスタイルにしましょう。

10.1.4 情報のスコープを絞る

ソフトウェアの世界では、悪いドキュメントかどうかを簡単に見分けるサインがあります。あなたは今、とあるドキュメントの中から情報を探そうとしているとします。どこかに存在することはわかっているのですが、それを見つけることができません。目次をじっと解読しても、grep コマンドを使っていくつかの単語の組み合わせを試してファイルの中を検索しても、あなたが探しているものを見つけることができません。

このような事態は、ドキュメント作成者が話題ごとにきちんとテキストをまとめていないときに発生します。その作成者は大量の情報を提供しているかもしれませんが、整理せずにただ 1 つに集めたにすぎません。たとえば、読者がアプリケーションの概要を知りたがっているときには、API のドキュメントを読む必要はありません。API ドキュメントには、細かい低レベルの説明しかないからです。

関連のあるセクションにパラグラフをきちんと収め、意味のあるタイトルをつけると、このような事態を避けることができます。ドキュメント自体のタイトルは、コンテンツを統合した短いフレーズにします。

良いタイトルがついていれば、すべてのセクションのタイトルを集めるだけで目次ができあがります。

タイトルを考えるときには、「私はこのセクションを見つけるために、どのような文をタイプして Google で検索するだろうか？」と自分に質問するという、簡単な習慣を身につけましょう。

1 ウサギ（=考え）たちを小さな箱に詰め込みすぎると息ができなくなるので、箱に入れるウサギは 1 匹にしましょう。

10.1.5 実在するようなコードのサンプルを使用する

foo、bar を利用するのは悪い習慣です。読者がコードサンプルを読んでそのコード片を理解しようとしたときに、現実的なサンプルでなければ理解はしにくくなります。

なぜ、現実世界のサンプルを使用しないのでしょうか？　コードのサンプルを実際のプログラムにカット・アンド・ペーストできるようにするのは一般的なプラクティスです。

悪い使用例のサンプルです。

```
パースのための関数が用意されています::

   >>> from atomisator.parser import parse

次のようにしてパースの関数を利用することができます::

   >>> stuff = parse('some-feed.xml')
   >>> stuff.next()
   {'title': 'foo', 'content': 'blabla'}
```

次の例は良いサンプルです。トップレベルの関数として提供されているパーサ関数が、どのようなフィード内容を返すかを知ることができるでしょう。

```
パースのための関数が用意されています::

   >>> from atomisator.parser import parse

次のようにしてパースの関数を利用することができます::

   >>> my_feed = parse('http://tarekziade.wordpress.com/feed')
   >>> my_feed.next()
   {'title': 'eight tips to start with python',
    'content': 'The first tip is..., ...'}
```

わずかな違いを強調しすぎだと思われるかもしれませんが、この差はドキュメントの便利さにとっての大きな差となります。読者はこのサンプルのコードをコピーして、簡単にインタラクティブシェル上で動かしてみることができます。その結果、このパーサがパラメータに URL を指定して動作するということを理解し、ブログのエントリを含むイテレータを返すということが理解できるでしょう。

サンプルコードは、実際のプログラムとして直接再利用できるようにすべきです。

10.1.6 なるべく少なく、かつ十分なドキュメント

ほとんどのアジャイルな方法論においては、ドキュメントは一番大切なものではありません。詳細に書かれたドキュメントを書くよりも、実際に動作するソフトウェアを作ることのほうが大切です。Scott Ambler によって書かれた『アジャイルモデリング――XP と統一プロセスを補完するプラクティス（原題 *Agile Modeling*）』（株式会社オージス総研訳、翔泳社刊）には、次のような良いプラクティスが書かれています。

> エクストリーム・プログラミングと統一プロセスの両方にとって効果的なプラクティスとは、徹底的にドキュメントを作成していくのではなく、本当に必要なドキュメントのニーズを明確にすることである。

たとえば、Atomisator がどのように動作しているのかをシステム管理者に説明するには、ドキュメントが 1 つあれば十分です。どのようにそのツールを設定して実行すればいいのかという方法以外の情報は、まったく必要ありません。このドキュメントのスコープは、次の質問に答えることができる範囲に限定されます。

私はどのようにすれば、Atomisator を自分のサーバ上で実行できるのか？

スコープと読者層のほかに、それぞれのセクションのページ数を制限するというのも良いアイデアです。ほとんどの場合、考えがうまくまとまれば、それぞれのセクションは 4 ページ以内に収まるでしょう。もしもそれ以上に必要であれば、そのソフトウェアは複雑すぎて、説明したり使用することが困難であるということを意味します。

包括的なドキュメントよりも動くソフトウェア

―― アジャイル・マニフェスト[*2]

10.1.7 テンプレートの使用

Wikipedia のページは、すべて似たような構成をしています。左側にデータのサマリーが書かれたボックスがあります。ドキュメントの最初にはリンクの張られた目次があり、各テキストに飛ぶことができます。ページの末尾には、参考情報のセクションがあります。

ユーザーはその構成に見慣れています。実際、ユーザーは目次をすばやく見ることで、自分の探している情報が載っているかどうかがわかるということを知っています。もしも見つからなければ、参考情報のセクションを探し、そのトピックを扱っている他の Web サイトを探しに行きます。このやり

方は、Wikipedia のどのページでも使用することができます。そして、Wikipedia においては、この方法を取るほうが効率がよいということを学ぶでしょう。

テンプレートを使用して、ドキュメントを共通パターンに強制することで、読者がドキュメントを読む効率が良くなります。構造に慣れるとすばやく読む方法がわかります。

ドキュメントの種類ごとにテンプレートを用意すると、ドキュメントをすばやく書くことも可能になります。

本章では、ソフトウェア開発に必要なさまざまな種類のドキュメントについて見ていきます。次に、Paster を利用したドキュメントのひな形の作成についても紹介します。しかし、まず最初に、Python のドキュメント作成で使用すべき、マークアップの文法の説明から行っていきます。

10.2 reStructuredText入門

reStructuredText は reST とも呼ばれます（http://docutils.sourceforge.net/rst.html を参照してください）。reST は、プレーンテキストを使ったマークアップ言語で、パッケージのドキュメントを作成する際に、Python のコミュニティ内で広く使用されています。reST の優れている点は、ソースのテキストの状態でもそのまま読むことができるという点です。LaTeX のように、マークアップの文法のせいで読みにくくなるということはありません。

以下のテキストは、reST 形式のドキュメントのサンプルです。

```
========
タイトル
========

セクション 1
============

この **単語** は強調されます 。

セクション 2
============

サブセクション
::::::::::::::

テキスト 。
```

reST は docutils パッケージに含まれています。docutils には、reST のファイルを HTML、LaTeX、XML などのさまざまなフォーマットに変換するスクリプトが含まれます。S5 という、Eric Meyer のスライドショーのシステム（http://meyerweb.com/eric/tools/s5/を参照）のためのフォーマットにも変換することができます。

ドキュメント作成者は内容にフォーカスし、必要に応じて、その後にどのようにレンダリングされるのかを決めることができます。Python 自身のドキュメントも reST で書かれています。このファイルは後で HTML に変換され、http://docs.python.org の Web サイトが構築されています。PDF などの他のフォーマットにも変換されています。

reST のテキストを書き始めるにあたって、まず知るべき最小の項目は次の通りです。

- セクション構造
- リスト
- インラインマークアップ
- リテラルブロック
- リンク

このセクションでは、文法を駆け足で説明しています。http://docutils.sourceforge.net/docs/user/rst/quickref.html にあるクイックリファレンスを見ると、より多くの情報を得ることができます。この Web サイトは、reST を仕事で使い始める際にはとても参考になるでしょう。

reStructuredText をインストールするために、docutils をインストールします。

```
$ easy_install docutils
```

インストールが完了すると、rst2 から始まるスクリプトがインストールされます。これらのスクリプトを使用すると、reST をさまざまな形式にレンダリングすることができます。

たとえば、rst2html スクリプトを使うと、与えられた reST ファイルから HTML ファイルを出力することができます。

```
$ more text.txt
タイトル
========

内容 。

$ rst2html.py text.txt > text.html
$ more text.html
<?xml version="1.0" encoding="utf-8" ?>
...
<html ...>
<head>
...
</head>
<body>
<div class="document" id="title">
<h1 class="title">タイトル</h1>
```

```
<p>内容 。</p>
</div>
</body>
</html>
```

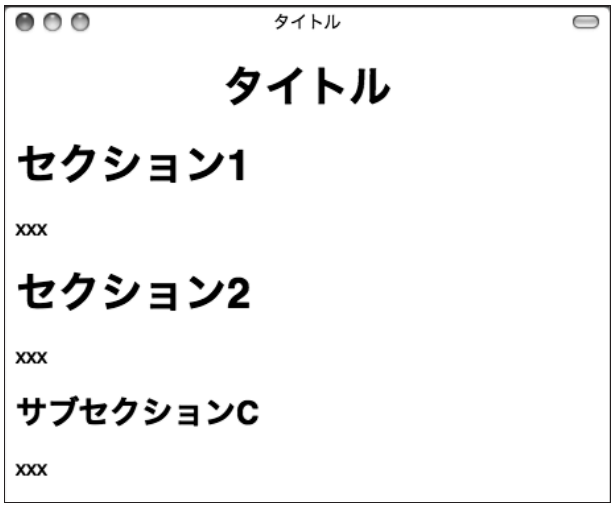
10.2.1 セクション構造

ドキュメントのタイトルとセクションは、アルファベットでも数字でもない文字を使ってアンダーラインにすることで表現されます。オーバーラインとアンダーラインの両方を使用することもできます。タイトルを表現するときには上下につけ、セクション名に対してはアンダーラインのみを利用するのが一般的に使用されるプラクティスです。

セクションタイトルのアンダーラインに頻繁に使用される記号としては、=、-、_、:、#、+、^、があります。

セクションタイトルに使用する記号の種類を変えていくことで、セクションの深さのレベルが変わっていきます。使用する記号の種類はドキュメント内で一貫している必要があります。

```
========
タイトル
========

セクション 1
============

xxx

セクション 2
============

xxx

サブセクション C
----------------

xxx
```



このファイルを HTML 出力すると、上記の図で示したような見た目になります。

10.2.2 リスト

reST では、箇条書き、列挙型、定義リストを使用することができます。自動採番機能も使用できます。

```
箇条書きリスト：

- 一番
- 二番
- 三番

列挙型リスト：

1. 一番
2. 二番
#. #を前につけると、自動で番号が割り振られます

定義リスト

一
  一は数字です。
二
  二も数字です。
```

10.2.3 インラインマークアップ

次のようなインラインマークアップを使用すると、テキストのスタイルを変えることができます。

- *強調*：イタリック
- **強い強調**：太字
- ``インラインリテラル``：インラインフォーマット済みテキスト
- `リンクつきのテキスト`_：同じ名前のものがドキュメント内にあれば、ハイパーリンクに置き換えられます（詳しくはリンクのセクションで説明します）

10.2.4 リテラルブロック

コードサンプルを表現したい場合には、リテラルブロックを使用することができます。コロンを 2 つ「::」書くとブロックを表す記号になります。インデントされたパラグラフがブロックとして扱われます。

```
コードサンプルです

::

   >>> 1 + 1
   2

ここからテキストに戻ります 。
```

::の後とブロックの後には、空行を入れるのを忘れないようにしてください。入れ忘れるとレンダリングされなくなります。

::はテキスト行にも書くことが可能です。テキスト行の末尾に書くと、「:」1 つにレンダリングされます。

```
コードのサンプル::

   >>> 1 + 1
   2

テキストの続き 。
```

コロンを残したくない場合には、行末の「サンプル」の文字と::の記号の間にスペースを入れます。こうすると、::はブロックの始まりとして解釈されますが、レンダリングされません。

10.2.5 リンク

もしも、ソースのテキスト内にドット2つから始まる特殊な外部参照リンク情報の行があると、インラインマークアップのリンクつきのテキストは、外部参照リンクに置き換えられます。

```
`Plone CMS`_ を試してみてください。これはすばらしいですよ！ Zope_ 上に作られています。

.. _`Plone CMS`: http://plone.org
.. _Zope: http://zope.org
```

一般的には、外部参照リンクのグループはドキュメントの末尾にまとめて置かれます。リンクされるテキストにスペースが含まれたり、日本語を使用する場合には、`（バッククオート）文字で囲むようにしてください。

内部リンクは、テキストの中にマーカーを追加することでも実現することができます。

```
これはコードの例です。

.. _example:

::

   >>> 1 + 1
   2

テキストの続き。コード例に戻る場合はこちら example_
```

セクションはターゲットとして使用することもできます。日本語やスペースを含む場合には`でくくります。

```
========
タイトル
========

セクション 1
============

xxx

サブセクション A
----------------

xxx
```

```
サブセクション B
-----------------

-> `サブセクション A`_ に戻る

セクション 2
============

xxx
```

10.3 ドキュメントの構築

ドキュメントのヘルパーとガイドラインを提供すると、読者と作成者の両方の仕事を簡単に改善することができます。そのために、前節で学んできたことを活用していきます。

ドキュメント作成者の視点から考えると、ガイドと一緒に再利用可能なテンプレート集が提供され、それをプロジェクトの中でいつどのように使用するのかという説明があると、この目標を達成できます。これをドキュメントポートフォリオと呼びます。

読者の視点から考えると、どのようなドキュメントが提供されるのかという展望が与えられると、楽にドキュメントを読むことができるようになりますし、必要なドキュメントを効率良く探して利用できるようになります。

10.3.1 ポートフォリオの構築

ソフトウェア開発プロジェクトの中で作られるドキュメントには、ソースコードを直接参照するような低レベルなドキュメントから、アプリケーションの概要を伝える設計書まで、多くの種類があります。

実際に、Scott Ambler は、彼の著作の『アジャイルモデリング——XP と統一プロセスを補完するプラクティス』(詳しくは http://www.agilemodeling.com/essays/agileArchitecture.htm を参照)の中で、ドキュメントの種類に関する長いリストを定義しました。彼は早期の仕様書から、操作ドキュメントまでのポートフォリオを構築しています。プロジェクト管理のドキュメントもカバーされているので、ドキュメントに関するニーズのすべてが、標準化されたテンプレートのセットで構築されています。

実際に完璧なポートフォリオを考えようとすると、ソフトウェア開発に使用している方法論の影響を強く受けるため、本章では多くの人に共通なサブセットに限定して説明します。このサブセットを元にして、特定のニーズを満たすポートフォリオを作成することもできます。時間をかければ、あなたの仕事の習慣を網羅するような、効果的なポートフォリオを構築することもできるでしょう。

ソフトウェアのプロジェクトにおけるドキュメントの共通セットは、次の3つのカテゴリに分類されます。

- 設計：アーキテクチャの情報や、クラス図やデータベースのダイアグラムのような低レベルの設計情報を提供する、すべてのドキュメント。
- 使用方法：ソフトウェアをどのように使用するのかが書かれたドキュメント。このドキュメントは、クックブック、チュートリアル、モジュールレベルのヘルプなどの形態で提供される。
- 運用：デプロイやアップグレード、ソフトウェアの運用に関するガイドラインを提供する。

設計

設計ドキュメントの目的は、ソフトウェアがどのように動作するのか、どのようにコードが組み立てられているのかを説明することです。開発者はシステムを理解するためにこのドキュメントを利用しますが、その他のアプリケーションがどのように動作しているのかを理解しようとしている人にとっても、良い入り口となります。

ソフトウェア開発には、次のような、さまざまな種類の設計ドキュメントがあります。

- アーキテクチャ概要
- データベースのモデル
- 依存関係、継承関係を含むクラス図
- ユーザーインタフェースのスケッチ
- インフラストラクチャの説明

これらのドキュメントのほとんどは、いくつかのダイアグラムと、最小量のテキストで構成されます。ダイアグラムの描き方は、チームやプロジェクトごとに定義されます。すばらしいドキュメントにするために、一貫したスタイルを使用しましょう。

UMLは、ソフトウェア設計のほとんどの切り口をサポートする13種類の図を提供しています。おそらく、この中で一番使用されているのはクラス図ですが、UMLはソフトウェアのあらゆる面の説明に使用することができます。

`http://en.wikipedia.org/wiki/Unified_Modeling_Language#Diagrams_overview` を参照してください。[*4]

4 日本語のWikipediaにもUMLに関するページがあります。「`http://ja.wikipedia.org/wiki/統一モデリング言語`」を参照してください。

UML のような特定の言語を、上から下まで完全に使って記述していくということは滅多にありません。多くのチームは、それぞれの経験から独自の方法を作り上げています。そのようなチームでは、UML や他のモデリング言語の中から良いと思うものをピックアップした独自のレシピを持っています。

アーキテクチャの概要を表す図を描く場合、単に四角い箱と矢印をホワイトボード上に描いて済ませてしまう設計者も数多くいます。決まったルールに従わないこともありますし、記録も取らないで消してしまうこともあります。また、Dia（`http://www.gnome.org/projects/dia`）や、Microsoft Visio（オープンソースでもフリーでもありません）などのシンプルなドロープログラムを使い、設計するのに必要な程度の最小の図を描く人もいます。本書の 6 章では、すべてのアーキテクチャの図を、OmniGraffle（`http://www.omnigroup.com/products/omnigraffle/`）を使って作成しています。

データベースのダイアグラムは、使用するデータベースの種類に依存します。データモデリングソフトの中には、作図機能に加えて、テーブルとその関連を自動的に生成してくれるものもあります。しかし、Python では、こういったツールは大げさでしょう。SQLAlchemy などの OR マッパーを使用している場合、実装前にデータベースのモデルを説明するためには、6 章でお見せしたように、フィールドリストが書かれたシンプルな箱をならべ、テーブル間の関連をつけ足すだけで十分です。

クラス図には、UML のクラス図をシンプルにしたものを使うことが多いでしょう。たとえば、Python ではクラスに protected メンバーを定義することはありません。そのため、アーキテクチャの概要を描くようなツールが定義されていれば、それで十分なことがほとんどです。

ユーザーインタフェースの図に関しては、作成しようとしているアプリケーションが、デスクトップアプリケーションか、それとも Web アプリケーションかによって異なります。Web アプリケーションの場合には、ヘッダやフッタ、左右のパネルは全画面で共通なことが多いため、スクリーンの真ん中の部分だけを説明することが多いでしょう。人によっては、HTML でプロトタイプを作ったり、スクリーンショットを作成します。デスクトップアプリケーションの場合には、プロトタイプの画面のスクリーンショットを撮ったり、Gimp や Photoshop といった画像ソフトを使用して、注釈つきのモックアップを作成するのが一般的です。

インフラストラクチャの概要を描いた図は、アーキテクチャの図に似ていますが、メールサーバやデータベース、その他のデータストリームなどの外部のシステムと、どのようにインタラクションを取っているのかという点に焦点を合わせて描いていきます。

共通テンプレート

このようなドキュメントを作成するときの重要なポイントは、ドキュメントのターゲットとなる読者を完全に把握し、内容のスコープを絞ることです。ドキュメント作成者への簡単なアドバイスもついた、シンプルな構造を持つ汎用テンプレートとして提供することで、これらを実現することができます。この構造には、次のものが含まれます。

- タイトル
- 作成者
- タグ（キーワード）
- 説明（概要）
- 対象（だれがこれを読むべきか？）
- 内容（図入り）
- 他のドキュメントへの参照

内容は、平均的な画面サイズを 1024 × 768 と想定した場合に、最大でも 3〜4 画面に収まる量にすべきです。もしもこれよりも多くなってしまったら、いくつかに分割するか、サマリーとしてまとめましょう。

テンプレートには、作成者の名前と、ドキュメントの評価を管理したり、分類を簡単にするためのタグも含まれています。これについては、本章の後半で説明します。

ドキュメントのテンプレートを提供するために使用するツールは Paster です。`pbp.skels` というファイルは、これまで説明してきた設計のテンプレートを実装しています。対象となるフォルダを指定し、いくつかの質問に答えるだけで、コードジェネレータと同じように使用できます。

```
$ paster create -t pbp_design_doc design
Selected and implied templates:
  pbp.skels#pbp_design_doc  A Design document
Variables:
  egg:      design
  package:  design
  project:  design
Enter title ['Title']: Database specifications for atomisator.db
Enter short_name ['recipe']: mappers
Enter author (Author name) ['John Doe']: Tarek
Enter keywords ['tag1 tag2']: database mapping sql
Creating template pbp_design_doc
Creating directory ./design
  Copying +short_name+.txt_tmpl to ./design/mappers.txt
```

実行が完了すると、次のようなファイルが生成されます。

```
========================================
Database specifications for atomisator.db
========================================

:Author: Tarek
```

```
:Tags: database mapping sql
:abstract:
 Write here a small abstract about your design document.

.. contents ::

Who should read this ?
::::::::::::::::::::::

Explain here who is the target readership.

Content
:::::::

Write your document here. Do not hesitate to split it in several
sections.

References
::::::::::

Put here references, and links to other documents.
```

訳注：pbp_design_doc テンプレートからは、上記のように英語のドキュメントが生成されます。しかし、reST ドキュメントを日本語（utf-8）で書いても問題ありません。上記のドキュメントを日本語で書き直した場合は次のようになります。

```
================================
atomisator.db のデータベース仕様
================================

:著者: Tarek
:タグ: database mapping sql
:概要:
ここにこのドキュメントの概要を小さくまとめて書いてください。

.. contents::

このドキュメントの対象者
::::::::::::::::::::::::

ここで、このドキュメントが想定する読者を説明してください。

本文
::::
```

```
ここにドキュメントを書いてください。また、長い内容はためらわずに
複数の節に分けて書くようにしてください。

参考文献
::::::::

ここに参考文献のリンク先やドキュメント名を記載してください。
```

使用方法

使用方法のドキュメントでは、ソフトウェアの一部分がどのように動作するのかを説明します。このドキュメントは、特定の機能がどのように動作するのかという低レベルな説明から、プログラム呼び出し時に使用するコマンドライン引数のような高レベルな説明まで含まれます。想定読者がコードを再利用しようとしている開発者なので、アプリケーションのフレームワークのドキュメントは、あらゆるドキュメントの中でもっとも重要なドキュメントになります。

このドキュメントは、主に3種類あります。

- レシピ：何かを行うための方法を説明した短いドキュメント。この種類のドキュメントは、特定の読者に絞ったものか、あるトピックに限定して焦点を当てたものになります。
- チュートリアル：ソフトウェアの機能を使用するための方法を一歩一歩説明したドキュメント。このドキュメントはレシピを参照します。また、それぞれの項目は、特定の読者のために書かれます。
- モジュールヘルパー：モジュールに含まれる要素を説明する低レベルなドキュメント。このドキュメントは、モジュール越しに組み込みのヘルプを呼び出す場合に読まれます。

レシピ

レシピは、特定の問題に対する解答や解決策を提供するドキュメントです。

たとえば、ActiveStateのWebサイトでは、Pythonクックブックオンラインが提供されています。

クックブックとはレシピを集めたもので、Pythonで何か特定のことを行う方法について、開発者が自分で記述することができます（`http://aspn.activestate.com/ASPN/PythonCookbook`を参照）。これらのレシピは、次のような簡潔な構造を持ちます。

- タイトル
- 送信者
- 最終更新日

- バージョン
- カテゴリ
- 説明
- ソースコード
- 議論（コードを説明するためのテキスト）
- コメント（Web から）

ほとんどの場合、これらは 1 画面に収まる長さで、詳しい詳細説明は書かれません。この構造は、ソフトウェアに対する要望を記述する場合にもそのまま使用できますし、"対象読者"を追加して、"カテゴリ"を"タグ"に置き換えると、汎用的な構造として適用することができます。

- タイトル（短い説明）
- 著者
- タグ（キーワード）
- これを読むべき人はだれか？
- 事前条件（たとえば、先に読んでおくべきドキュメントなど）
- 問題（短い説明）
- 解決策（メインのテキスト。1〜2 画面に収まる量）
- 参照（他のドキュメントへのリンク）

更新日とバージョンは、入れてもそれほど便利ではありません。プロジェクト内でソースコードのようにドキュメントも管理されていれば、後で見ることができるからです。

設計のテンプレートと同様に pbp.skels ファイルを使用すると、このような構造をした pbp_recipe_doc テンプレートが使えるようになります。

```
$ paster create -t pbp_recipe_doc recipes
Selected and implied templates:
  pbp.skels#pbp_recipe_doc  A recipe
Variables:
  egg:      recipes
  package:  recipes
  project:  recipes
Enter title (use a short question): How to use atomisator.db
Enter short_name ['recipe'] : atomisator-db
Enter author (Author name) ['John Doe']: Tarek
Enter keywords ['tag1 tag2']: atomisator db
Creating template pbp_recipe_doc
Creating directory ./recipes
  Copying +short_name+.txt_tmpl to ./recipes/atomisator-db.txt
```

実行が完了したら、ドキュメント作成者が内容を埋めていきます。

```
========================
How to use atomisator.db
========================

:Author: Tarek
:Tags: atomisator db

.. contents ::

Who should read this ?
::::::::::::::::::::::

Explain here who is the target readership.

Prerequisites
:::::::::::::

Put here the prerequisites for people to follow this recipe.

Problem
:::::::

Explain here the problem resolved in a few sentences.

Solution
::::::::

Put here the solution.

References
::::::::::

Put here references, and links to other recipes.
```

チュートリアル

チュートリアルはレシピと似ていますが、目的が異なります。チュートリアルは、個別の問題を解決するのではなく、アプリケーションの機能の使用方法をステップ・バイ・ステップで説明したものです。このドキュメントは、レシピよりもアプリケーションの多くの部分に触れることになるため、文章も長くなるでしょう。たとえば、Djangoはチュートリアルを Web サイト上で提供しています。初めて Django のアプリケーションを作成する方法について書かれたページ（http://www.djangoproject.com/documentation/tutorial01 を参照[*5]）は、10 ページ分の長さがあります。

ドキュメントの構造は次のようになります。

5 日本語訳は http://djangoproject.jp/doc/ja/1.0/intro/tutorial01.html を参照してください。

- タイトル（短い文章）
- 著者
- タグ（ことば）
- 説明（概要）
- これを読むべき人はだれか？
- 事前条件（たとえば、先に読んでおくべきドキュメントなど）
- チュートリアル（メインのテキスト）
- 参照（他のドキュメントへのリンク）

ここで説明したような構造を持った pbp_tutorial_doc テンプレートも、pbp.skels で提供されています。これも、設計のテンプレートと同じように使用することができます。

モジュールヘルパー

私たちのコレクションに最後に追加するテンプレートは、モジュールヘルパーのテンプレートです。モジュールヘルパーは、1 つのモジュールについて書かれていて、そのモジュールに含まれる要素の説明と使用方法のサンプルを提供します。

docstring を収集してモジュールのヘルプを作成する pydoc や Epydoc（http://epydoc.sourceforge.net を参照）といったツールを使うと、このようなドキュメントを自動的に構築することができます。API の自動探索をベースにして、網羅的にドキュメントを作成することができます。この種類のドキュメントを提供している Python のフレームワークは、かなりの数に上ります。実際に Plone が http://api.plone.org で提供している情報は、モジュールヘルパーを収集することで最新のドキュメントとして維持されています。

このアプローチの主な問題は次の 2 点です。

- ドキュメント全体に対して処理するときに、本当にドキュメントにしたい部分をスマートに選択する機能がない。
- ドキュメントが大量に含まれるため、コードがわかりにくくなる可能性がある。

その上、モジュールの構成部品のいくつかの部分にまたがっていて、関数やクラスの docstring に分解しにくいサンプルがモジュールのドキュメントとして提供されることがあります。モジュールの先頭にテキストを書くと、そのモジュールの docstring になるので、この目的を達成することができます。しかし、この方法ではソースファイルがテキストのブロックとコードのブロックに分かれた構成になってしまいます。コードの行数が全体の 50%を切ると、さらにわかりにくくなります。あなたがモジュールの作者であるとすれば、まったく問題はありません。しかし、ドキュメントではなくコードを読もうと思っている人は、docstring 部分を飛ばさないといけなくなります。

他の方法として、テキストを別のファイルに切り離すことができます。手動で選択すると、どのPython モジュールがモジュールヘルパーファイルを持っているのかを決定することができます。ドキュメントはコードベースと切り離されるため、後で説明するように、ドキュメントの寿命を延ばすことができます。これが、Python のドキュメントの書き方になります。

ドキュメントとコードを分離することが docstring よりも良いかどうかという点に関しては、多くの開発者の間で意見が分かれるところです。このアプローチを取るには、ドキュメント作成のプロセスを完全に開発サイクルに統合しなければなりません。そうしなければ、ドキュメントはすぐに古新聞になってしまいます。docstring は、コードと使用例を近づけることによってこの問題に対処していますが、よりわかりやすいドキュメントの一部として使用できるテキストを提供するという、高いレベルの解決策を得ることはできません。

モジュールヘルパーのテンプレートは極めてシンプルで、内容に加えて少量のメタデータが含まれる程度になっています。読者はそのモジュールを使用する開発者に限定されるため、だれが読むべきかという定義はここには含まれません。

- タイトル（モジュール名）
- 著者
- タグ（単語）
- 内容

次章では、doctest とモジュールヘルパーを利用したテスト駆動開発について言及します。

運用

運用に関するドキュメントは、ソフトウェアをどのように操作・運用すればいいのかを説明するのに使用されます。たとえば、

- インストールとデプロイに関するドキュメント
- 管理ドキュメント
- ユーザーが問題に直面したときに手助けを行うための「よくある質問と答え（FAQ）」
- ヘルプへの問い合わせ方法や、フィードバックを提供する方法を説明したドキュメント

これらのドキュメントは、それぞれが特異なものですが、テンプレートとしては前の部分で紹介したチュートリアルのテンプレートを使用することができます。

10.4 自分自身のポートフォリオを構築する

これまでの説明で紹介してきたテンプレートは、あなたのソフトウェアのドキュメント作成に使用することができる基盤となります。ここからは、Paster を使用して、自分自身のドキュメントのポートフォリオを構築するために、それ以外のテンプレートを追加するための方法を紹介していきます。

プロジェクト中に作成するドキュメントの量に関しては、必要最低限で十分というのを心がけてください。追加されるそれぞれのドキュメントについても、対象となる読者を明確に定義し、実際のニーズを満たすようにします。実際に価値の増加に寄与しないドキュメントは書くべきではありません。

10.4.1 ドキュメントのランドスケープの構築

前節で説明をしてきたドキュメントポートフォリオによって、複数の種類のドキュメントが提供されますが、読者の手に渡った後でそのドキュメントを再グループ化したり、再構成したりする方法は提供されません。読者は、ドキュメントにざっと目を通すときに、心の中でドキュメントのインデックスページのようなものを作成してそれを参照します。Andreas Rüping は、これをドキュメントランドスケープと呼んでいます。彼は、ドキュメントを組織化するための最適な手法は、ロジカルツリーを構築する方法であると結論づけています。

田舎に行って景色を眺めると、木々や平地、家々などを見ることができますが、このようなさまざまな要素が含まれる景色をランドスケープと呼びます。ドキュメントランドスケープもこれと同じく、レシピ、チュートリアルなどの多くのドキュメントからなります。ランドスケープは、目次、イメージマップなどのさまざまな形式で表現することができます。読者や作成者ごとのランドスケープを用意することもできます。

言い換えると、ポートフォリオを構成するドキュメントごとに、ディレクトリツリー内に配置するべき場所を見つける必要があるということです。この場所は、ドキュメント作成者がドキュメントを作成するとき、もしくはドキュメントの読者がそれを探すときの、どちらの場合においてもわかりやすい必要があります。

ドキュメントを探索する上で手助けとなるのは、作成者と読者の両方が読むことができるように、それぞれのレベルで目次のページを作ることです。

ドキュメントのランドスケープの構築は、次の 2 つのステップで行うことができます。

- 制作者向けのツリーの構築
- 顧客のためのツリーを構築し、制作者向けのツリーの上位に配置する

制作者と顧客は異なった場所で、異なった形式のドキュメントにアクセスするので、制作者と顧客を明確に区別することが大切です。

制作者のレイアウト

制作者の視点から見た場合、それぞれのドキュメントは、Python モジュールと同じように処理されます。ドキュメントはバージョン管理システムに格納され、コードと同じように扱われます。

作成者は、自分が作成している文章が最終的にどのような外見になるのか、ということを気にしません。彼らが確実に行いたいのは、適切な場所に目的のドキュメントを書くことです。

フォルダのツリーに保存された reStructuredText のファイルは、バージョン管理システムを使用して、ソフトウェアのコードと一緒に管理することができます。この方法は、制作者向けのドキュメントのランドスケープを構築するための便利な方法です。

6 章で説明した Atomisator の中で紹介したフォルダ構造を振り返ってみると、このツリーのルートにある `docs` フォルダを使用できることがわかります。

このツリーを組織化するためのもっともシンプルな方法は、特性に従ってドキュメントをグループにすることです。

```
$ cd atomisator
$ find docs
docs
docs/source
docs/source/design
docs/source/operations
docs/source/usage
docs/source/usage/cookbook
docs/source/usage/modules
docs/source/usage/tutorial
```

この `docs` フォルダでは、次節で使用するツールで利用できるようにするため、ドキュメントを `source` フォルダの下に格納している点に注意してください。

ルート以外のそれぞれのレベルには、`index.txt` というファイルを追加することができます。これは、どのような種類のドキュメントがそのフォルダに含まれているのかを説明したり、それぞれのサブフォルダに何が含まれているのかというサマリーを書くことができます。たとえば、運用フォルダの場合には、読むことができる運用ドキュメントのリストを含めます。

```
====
運用
====

このセクションには運用に関するドキュメントが含まれます:
```

```
- Atomisatorのインストールと実行方法
- Atomisatorで使用する 、PostgreSQLのデータベースのインストールと管理
```

作成した人が更新するのを忘れないように、リストを自動生成することもできるでしょう。

顧客のレイアウト

顧客の視点から考えると、目次を見て概要をつかんだり、ドキュメント全体がきちんとフォーマットされていて、読みやすく、なおかつ見た目が良いということが重要になります。

フォーマットとしては、Web ページがベストな選択肢です。reStructuredText のファイルから生成するのも簡単です。

Sphinx（http://sphinx.pocoo.org）を使用すると、テキストのツリーから構造化された HTML を生成することができます。Sphinx は、いくつかのスクリプトと docutils 拡張で作られています。このツールは Python 本体のドキュメントも含め、多くのプロジェクトのドキュメント作成のツールとして使用されています。組み込みの機能にはさまざまなものがあり、本当にすばらしいナビゲーションや、JavaScript で実装されクライアントサイドで動作する、軽量だが十分な機能を持つ検索エンジンと一緒にコンテンツを生成します。また、コードのサンプルをレンダリングするのに Pygments を使用しているため、見た目のすばらしいシンタックスハイライトも行われます。

本節の前半で定義したドキュメントのランドスケープの設定も、Sphinx を使うと容易に行えます。

Sphinx のインストールには、easy_install を使用します。

```
$ sudo easy_install Sphinx
Searching for Sphinx
Reading http://cheeseshop.python.org/pypi/Sphinx/
...
Finished processing dependencies for Sphinx
```

インストールを行うと、sphinx-quickstart などのいくつかのスクリプトがインストールされます。このスクリプトを実行すると、Web のドキュメントを生成するために使用するスクリプトや、Makefile が生成されます。docs フォルダの中でこのスクリプトを実行して、次のように質問に答えていきましょう。

```
$ sphinx-quickstart
Welcome to the Sphinx quickstart utility.

Enter the root path for documentation.
> Root path for the documentation [.]:
> Separate source and build directories (y/n) [n]: y
> Name prefix for templates and static dir [.]:
> Project name: Atomisator
> Author name(s): Tarek Ziade
```

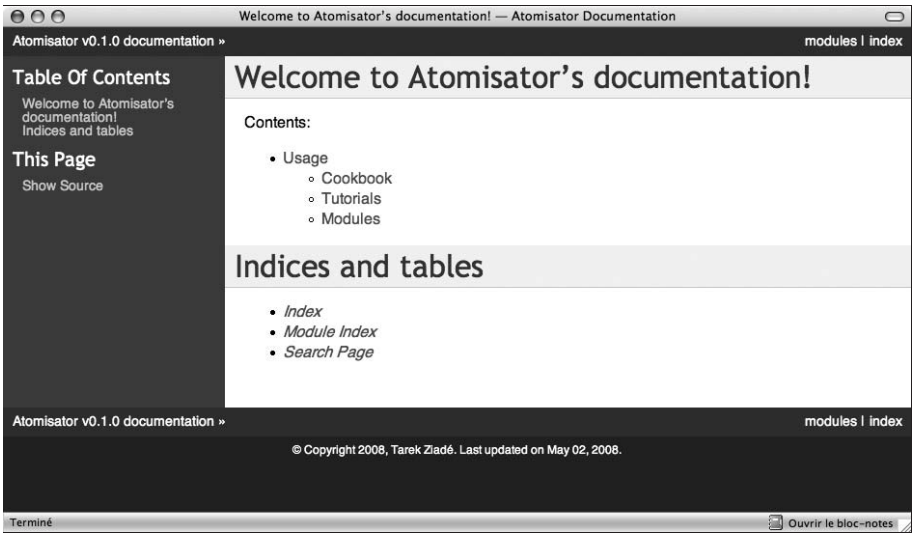
```
> Project version: 0.1.0
> Project release [0.1.0]:
> Source file suffix [.rst]: .txt
> Name of your master document (without suffix) [index]:
> Create Makefile? (y/n) [y]: y

Finished: An initial directory structure has been created.

You should now populate your master file ./source/index.txt and create
other documentation
source files. Use the sphinx-build.py script to build the docs, like so:

   make <builder>
```

このスクリプトを実行すると、conf.py というファイルが source フォルダの中に追加されます。このファイルには、質問に回答して定義された設定が含まれます。それ以外には、スクリプトを実行したフォルダのルートに index.txt と Makefile も生成されます。

make html を実行すると、build ディレクトリの中にツリーが生成されます。

```
$ make html
mkdir -p build/html build/doctrees
sphinx-build.py -b html -d build/doctrees -D latex_paper_size=  source
build/html
Sphinx v0.1.61611, building html
trying to load pickled env... done
building [html]: targets for 0 source files that are out of date
updating environment: 0 added, 0 changed, 0 removed
creating index...
writing output... index
finishing...
writing additional files...
copying static files...
dumping search index...
build succeeded.
Build finished. The HTML pages are in build/html.
```

build/html ディレクトリの中にドキュメントが生成されます。index.html をブラウザで開いて読むことができます。



ソースコードから生成された HTML 版のドキュメント以外にも、モジュールリストや索引などのページが自動的に生成されます。Sphinx では、これらの機能のために `docutils` を拡張しています。主に次の項目が、拡張された箇所です。

- 目次を構築するためのディレクティブ
- ドキュメントをモジュールヘルパーとして登録するためのマーカー
- 索引に項目を追加するためのマーカー

インデックスのページでの作業

Sphinx では、`toctree` ディレクティブが用意されています。これは、ドキュメント内に目次を挿入して、他のドキュメントへのリンクを張るために使用します。ディレクティブ内のそれぞれの行は、現在のドキュメントからの相対パスで書かれたファイル名でなければなりません。`glob` スタイルの名前を使用して、式にマッチする複数ファイルをまとめて登録する機能も提供されています。

たとえば、制作者のランドスケープで定義した、`cookbook` フォルダの `index` ファイルは次のような内容になるでしょう。

```
============
クックブック
============

クックブックへようこそ！

現在利用可能なレシピ:
```

```
.. toctree::
  :glob:

  *
```

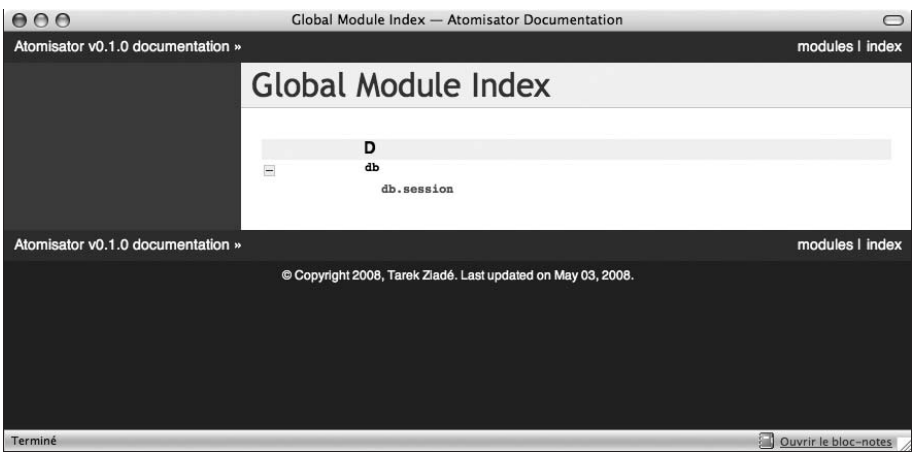
このようにソースに書いてビルドすると、cookbook フォルダ内にある、利用可能なすべての reStructuredText のリストが表示されるようになります。このディレクティブは、ブラウズ可能なドキュメントにビルドされる、すべての index ファイル内で利用することができます。

モジュールヘルパーの登録

モジュールヘルパーのドキュメントで使用できる機能として、モジュールのマーカーを追加することができます。これを書くと、モジュールの索引ページから参照することができるようになります。

```
=======
session
=======

.. module:: db.session

このsessionモジュールでは...
```

モジュール名の衝突を防ぐために使用される、db という名前空間がここで設定されていることに注意してください。Sphinx は、これをモジュールのカテゴリとして使用します。図では 1 つしかありませんが、db. という名前で始まるすべてのモジュールがグループにまとめられて表示されます。



Atomisator の場合には、db、feed、main、parser というパッケージが、モジュールのグループ化に利用されるでしょう。

あなたのドキュメントでも、多くのモジュールが含まれる場合に、この機能を使用することができます。

最初のほうで作成したモジュールヘルパーのテンプレート（pbp_module_doc）を変更して、デフォルトで、この module ディレクティブを追加するようにできます。

索引のマーカーの追加

モジュール以外にも、索引ページにドキュメント内の要素へのリンクを追加するためのオプションを使用することができます。

```
.. index::
   single: データベースアクセス
   single: セッション

=======
session
=======

.. module:: db.session

このsessionモジュールでは...
```

"データベースアクセス"と"セッション"の、2 つの新しい項目が索引ページに追加されます。

クロスリファレンス

最後になりますが、Sphinx ではクロスリファレンスの設定を行うためのインラインマークアップも提供されています。たとえば、次のように書くと、指定されたモジュールへのリンクを作成することができます。

```
:mod:`db.session`
```

:mod:は、モジュールを表すマーカーです。db.session が、リンクを張ろうと思っているモジュールの名前になります。このリンク先のドキュメントは、どこかで登録しておく必要があります。:mod:もそうですが、これまで説明してきた項目は Sphinx によって reStructuredText に追加された特別なディレクティブです。

Sphinx はもっと多くの機能を提供しています。それらの情報は Web サイトで見ることができます。たとえば、autodoc 機能は、ドキュメントを構築する際に doctest を自動収集することができる、すばらしいオプションになります。詳しくは、http://sphinx.pocoo.org を参照してください。[*7]

10.5 まとめ

本章では、次のようなことをどのように行うのかについて具体的に説明してきました。

- 効果的に文章を書くためのいくつかのルールの使用方法
- Pythonista の LaTeX とも言うべき reStructuredText の使用方法
- ドキュメントのポートフォリオとランドスケープの構築
- すばらしい Web のドキュメントを生成することができる Sphinx の使用方法

プロジェクトのドキュメントを作成する上で一番難しいことは、ドキュメントを正確かつ最新に保つことです。コードリポジトリの中にドキュメント保存用の領域を作ることで、それがずっと簡単になります。

そのような置き場を設置したら、開発者がモジュールに変更を加えるときには、いつも関連するドキュメントにも変更を加えるようにします。

この作業は、大型のプロジェクトでは極めて困難になることもあるでしょう。その場合、モジュールのヘッダに関連するドキュメントのリストを追加すると簡単になります。

ドキュメントが常に正しいということを保証するための補助的なやり方としては、doctest を通じて、ドキュメントとテストを一体化してしまう方法があります。

これらに関しては、テスト駆動開発やドキュメント駆動開発の原則について説明を行う次章で触れていきます。

7 日本語訳は http://sphinx-users.jp/doc.html を参照してください。