A Brief Tutorial On Making Beauty Documents

Release 0.1

Zhi Liu

Apr 06, 2019

Contents

- 邮箱（**email**）：zhiliu.mind@gmail.com
- 博客（**blog**）：http://blog.csdn.net/enjoyyl
- 主页（**homepage**）：http://iridescent.ink




Hint: PDF 版文档可以点这里查看

写于 **2015** 年 **6** 月 **24** 日

CHAPTER 1

引言

1.1 手册自述

1.1.1 目标

本手册为 Sphinx 和 reStructuredText 简明教程, 利用这些工具，你可以很容易的做出漂亮的文档, 教程力求简明，包含但不限于如下内容：

- Sphinx 简明教程
- reStructuredText 简明教程
- 版本控制系统 Git 教程

$$\alpha \neq \beta \tag{1.1}$$

$\alpha \neq \beta$

$\alpha \neq \beta$

$\alpha \neq \beta$

1.1.2 撰写与发布

该手册采用 reStructuredText 标记语言撰写, 使用 Sphinx 进行发布.

属性	值
开发语言	reStructuredText
发布工具	Sphinx

- **邮箱（email）**：zhiliu.mind@gmail.com

- 博客（**blog**）：http://blog.csdn.net/enjoyyl
- 主页（**homepage**）：www.iridescent.ink

CHAPTER 2

基础教程

2.1 文档制作简介

该手册采用 reStructuredText 标记语言（*Markup Language*）撰写, 使用 Sphinx 进行发布.

2.1.1 标记语言简介

你可能觉得标记语言是个新名词, 然而你大错特错, 如果我说 HTML、Latex 也是标记语言, 前者用于创建网页, 后者用于排版文档书籍, 你可能就知道一点了.

简而言之, 标记语言就是特定的标记符号集合, 每个特定标记符可以实现特定的功能（自己总结的）, 如倾斜、加粗、连接、引用等等, 如在 reStructureText 或者 Markdown 中, 使用星号括住文本, 可将文本渲染成斜体或粗体, 即 `* 我变斜了 *` 被渲染成 *我变斜了*, `** 我变粗了 **` 被渲染成 **我变粗了**.

2.1.1.1 Markdown 标记语言

Markdown 的 **设计哲学是易读（easy-to-read）易写（easy-to-write）, 是一种轻量级标记语言**, 之所以称为轻量级是因为其标记符号集合较小, 容易记住和使用. Markdown 标记语言, 如今被越来越多的写作爱好者和撰稿者广泛使用, 也被大多数现代网站所采用, 如 GitHub、StackOverflow、简书、CSDN、有道云笔记、Gitblog 等等. 此外, 很多开源项目的自述文件 `README.md` 就是采用 Markdown 语言编写的.

Markdown 的语法手册可以参见：

- Markdown——入门指南
- Markdown 中文语法指南
- 创始人 John Gruber 的 Markdown 语法说明

由于 Markdown 的语法简单, 使得其共功能有限, 因而有了基于 Markdown 的各种语法扩展, 这里就不再细说, 有兴趣地话自行了解.

2.1.1.2 reStructuredText 标记语言

reStructuredText 是比 Markdown 功能更为强大但语法也更为复杂的一种标记语言. 配合 Sphinx 使用, 可以撰写渲染排版出优美的文档, 且输出格式丰富. 两者的介绍分别见：*reStructuredText 简明教程* (page 36) 和*Sphinx 简明教程* (page 6) .

2.1.1.3 标记语言的优点

标记语言的撰写很简单, 不需要很高大上的 IDE, 只需要一个能够编辑文字的文本编辑器即可, 完全可以只用 Windows 系统的“记事本”或者 Linux 上的“vi”来完成撰写. 这样的类似代码的语言, 很容易使用诸如 Git 这样的版本控制系统进行管理！可以总结出以下优点：

- 专注于文本内容而不是排版样式
- 兼容所有文本编辑器与字处理软件
- 渲染导出格式丰富, 如 HTML、PDF, 借助一些工具还可以导出 Latex、epub 等文件
- 可以使用 Git 等版本控制系统管理文章版本
- 可读、直观、易学

2.1.2 使用标记语言需要会什么?

- 哈哈, 首先得知道写什么
- 嘿嘿, 你得会打字
- 正经的, 你必须学习一门标记语言的语法（如 Markdown、reStructureText）
- 工具 1：你需要一个文本编辑器（notepad、vi、**Sublime Text**）
- 工具 2：你需要一个该标记语言的解释渲染器（太多了, 不同平台不一样, 如 Sphinx）

让我们开始吧!!!

2.2 Sphinx 简明教程

2.2.1 What is Sphinx

> Sphinx 是种令人可以轻松撰写出优美文档的工具, 由 Georg Brandl 在 BSD 许可证下创造, 它允许开发人员以纯文本格式编写文档, 以便采用满足不同需求的格式轻松生成输出. 这在使用 Version Control System 追踪变更时非常有用. 纯文本文档对不同系统之间的协作者也非常有用. 纯文本是当前可以采用的最便捷的格式之一.

> 虽然 Sphinx 是用 Python 编写的, 并且最初是为 Python 语言文档而创建, 但它并不一定是以语言为中心, 在某些情况下, 甚至不是以程序员为中心. Sphinx 有许多用处, 比如可以用它来发布你的项目文档, 或编写整本书!

Sphinx 官网：http://www.sphinx-doc.org/en/stable/, 就是采用 reStructuredText 标记语言撰写, Sphinx 发布的.

2.2.2 Why Sphinx

Sphinx 具有以下亮点：

- **丰富的输出格式**：HTML（包括 Windows HTML Help）, LaTex（用于可打印 PDF）, epub, Texinfo, manual pages, plain text
- **完备的交叉引用**：语义化的标签, 并自动链接函式、类、引文、术语以及类似片段信息
- **明晰的层级结构**：轻松定义文档树, 并自动化链接同级/父级/下级文章
- **美观的自动索引**：自动生成索引以及特定语言模块的索引
- **精确的语法高亮**: 基于 Pygments 自动语法高亮
- **开方的扩展**: 支持代码片段的自动测试, 从 Python 模块的文档字符串包含（API 文档）, 参考 more
- **用户贡献的扩展**: 用户提供的 大约 50 个扩展, 大多可以通过 PyPI 安装
- **强大简洁的书写语言**: 使用 新结构化文本 (reStructuredText) 作为标记语言.

2.2.3 安装 Sphinx

Sphinx 的安装很简单, 只需通过 `pip` 或 `easy_install` 安装, 且支持 Windows, Linux, Mac 等系统. 下面详细介绍 Windows, Linux 系统下 Sphinx 的安装.

2.2.3.1 依赖

Sphinx 需要以下依赖工具:

- Python 2.4 及以上版本
- docutils 0.5 及以上版本库
- jinjia2 库
- Pygments 库

不用担心, 这些依赖库会自动安装的, 如果没有请自行安装.

2.2.3.2 Linux 系统

由于 Linux 系统中都会自带有 Python, 故可以直接在终端 (`Ctrl + Alt + T` 快捷键打开终端) 通过 `pip` 命令安装:

```
$ pip install Sphinx
```

Attention: 安装过程需要连接网络, 请确保网络畅通!

2.2.3.3 Windows 系统

Windows 系统中一般不会自带 Python, 需要用户自行安装, 首先安装 Python 和 pip, 然后通过 pip 安装 Sphinx 即可.

1. 安装 Python 和 pip

目前 Python 最新版本为 Python 3.5.1 , 可到 Python 官网 下载安装即可, 该版本已经包含 pip, 无需再安装 pip.

2. 安装 Sphinx

打开命令提示符 (运行 --> cmd 或 所有程序 --> 附件 --> 命令提示符), 输入如下命令

```
pip install Sphinx
```

Attention: 安装过程需要连接网络, 请确保网络畅通!

2.2.4 使用 Sphinx

2.2.4.1 手册文档

- Sphinx 官网英文手册
- Sphinx 1.3.1 中文手册 (推荐查看)
- 使用 sphinx 记笔记 (中文)
- 用 Sphinx 写书

Hint: 这些文档大部分都是用 Sphinx 发布的, 可以下载 HTML, PDF, epub 查看.

你可以查看 Sphinx 初尝 来学习, 也可以跟着我一起动起来!

2.2.4.2 一起动起来吧

由于 Sphinx 是跨平台的, 其在 Windows 上和 Linux 上的用法相同, 故以 Linux 系统下 Sphinx 为例.

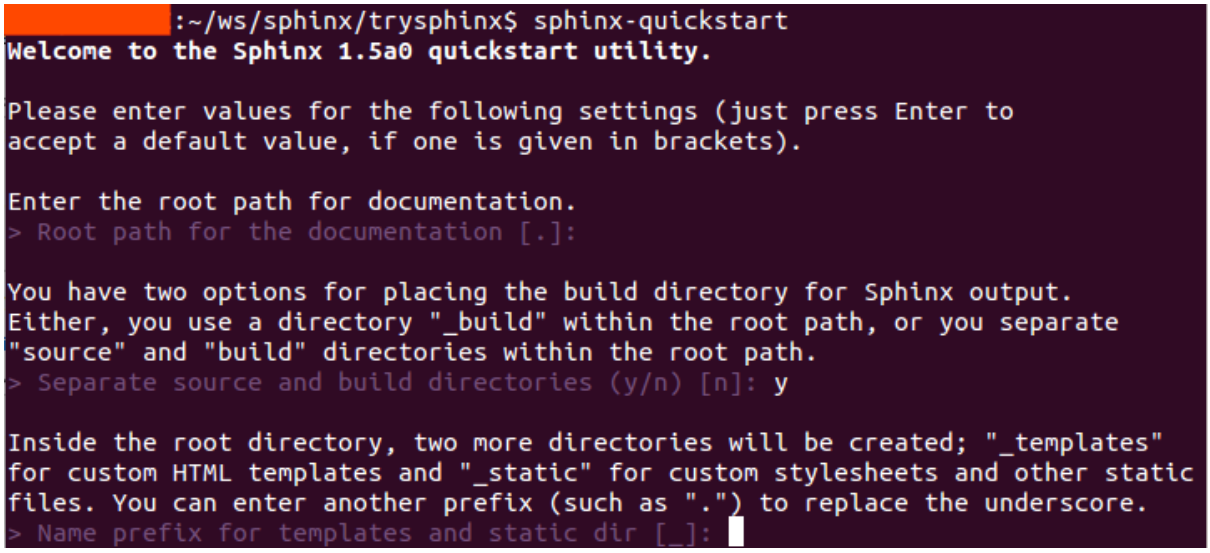
本节旨在让大家了解如何快速使用 Sphinx, 故较为精简, 详细的问题请自行查阅手册.

配置文档资源

在你的磁盘里新建一个工作目录, 比如我在用户主目录下的 ws 文件夹中建立了个 sphinx 目录, 又在其中建立了一个 trysphinx 目录用于本次演示, 即 ws/sphinx/trysphinx , 然后切换到此目录并启动 sphinx 以创建并配置工程, 整个过程命令表示如下 (注意: 仅输入终端命令提示符 $ 后面的代码)

```
$ mkdir ws/sphinx/trysphinx
$ cd ws/sphinx/trysphinx
$ sphinx-quickstart
```

之后进入文档资源配置对话框, 比如询问的第一项是 文档的根目录, 由于现在处于根目录, 所以使用 ** 默认的当前目录 [.]**, 即直接回车即可; 询问的第二项是 * 你想把构建的最终文档目录放在哪 *, 是放在”source”里创建成”_build”, 即 trysphinx/source/_build , 还是放在根目录里创建成”build”, 即 trysphinx/build , 我选择后者, 所以输入 y. 如下图所示:



询问的第四项为工程名字, 比如就叫: “Try Sphinx”, 那么就输入它, 接下来输入作者名字, 我就填 iridescent 了哈; 紧接着是工程版本, 以及发布版本; 然后是选择文档语言, 打开提供的链接 (http://sphinx-doc.org/config.html#confval-language) 查看找到”zh_CN – Simplified Chinese”, 所以输入 zh_CN. 接着还会有很多配置询问, 你可以采用默认, 但建议仔细读一下, 有些是可以通过修改”conf.py”文件再修改, 有些就不可以.

Note: 如果你想修改 conf.py 文件, 参考官方文档吧: http://www.sphinx-doc.org/en/stable/config.html

我采用了下面的配置

```
$ sphinx-quickstart
Welcome to the Sphinx 1.5a0 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Enter the root path for documentation.
```

(continues on next page)

(continued from previous page)

```
> Root path for the documentation [.]:

You have two options for placing the build directory for Sphinx output.
Either, you use a directory "_build" within the root path, or you separate
"source" and "build" directories within the root path.
> Separate source and build directories (y/n) [n]: y

Inside the root directory, two more directories will be created; "_templates"
for custom HTML templates and "_static" for custom stylesheets and other static
files. You can enter another prefix (such as ".") to replace the underscore.
> Name prefix for templates and static dir [_]:

The project name will occur in several places in the built documentation.
> Project name: Try Sphinx
> Author name(s): iridescent

Sphinx has the notion of a "version" and a "release" for the
software. Each version can have multiple releases. For example, for
Python the version is something like 2.5 or 3.0, while the release is
something like 2.5.1 or 3.0a1.  If you don't need this dual structure,
just set both to the same value.
> Project version: 0.1
> Project release [0.1]:

If the documents are to be written in a language other than English,
you can select a language here by its language code. Sphinx will then
translate text that it generates into that language.

For a list of supported codes, see
http://sphinx-doc.org/config.html#confval-language.
> Project language [en]: zh_CN

The file name suffix for source files. Commonly, this is either ".txt"
or ".rst".  Only files with this suffix are considered documents.
> Source file suffix [.rst]:

One document is special in that it is considered the top node of the
"contents tree", that is, it is the root of the hierarchical structure
of the documents. Normally, this is "index", but if your "index"
document is a custom template, you can also set this to another filename.
> Name of your master document (without suffix) [index]:

Sphinx can also add configuration for epub output:
> Do you want to use the epub builder (y/n) [n]: y

Please indicate if you want to use one of the following Sphinx extensions:
> autodoc: automatically insert docstrings from modules (y/n) [n]: y
```

(continues on next page)

(continued from previous page)

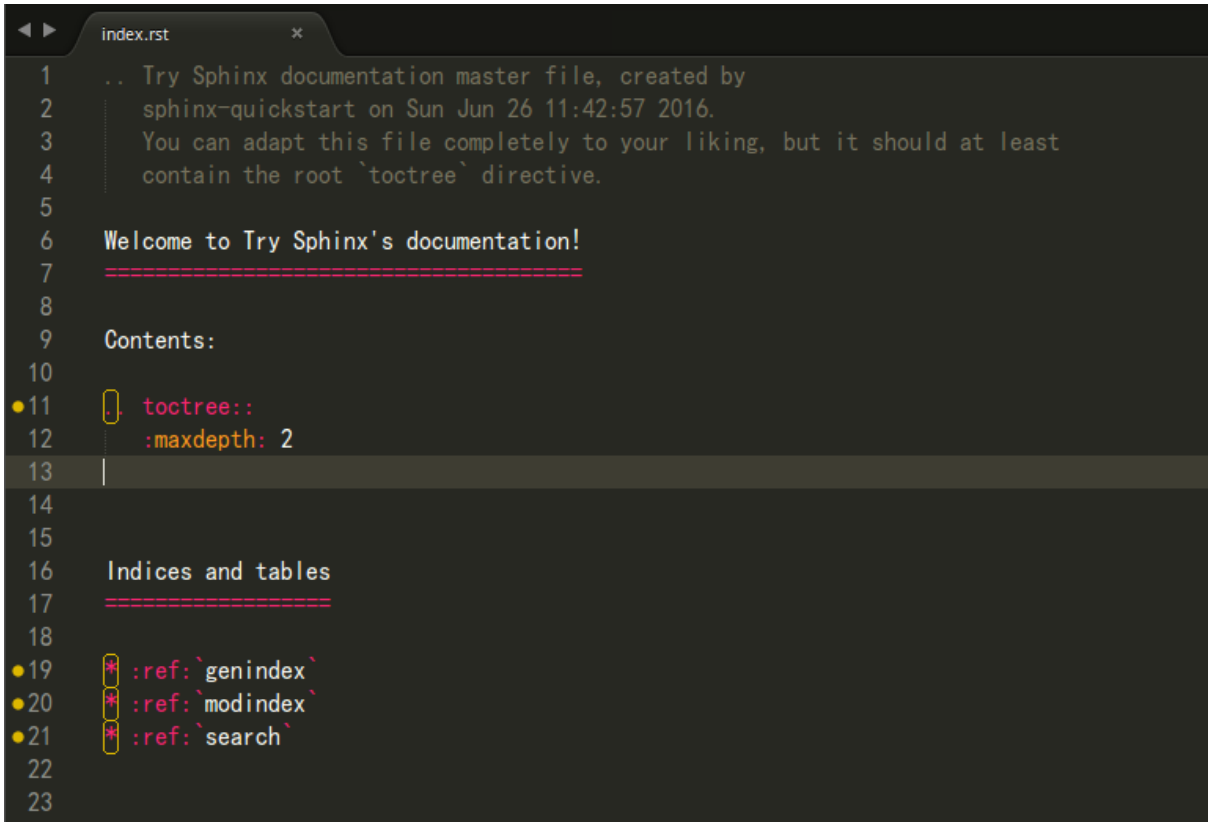
```
> doctest: automatically test code snippets in doctest blocks (y/n) [n]: y
> intersphinx: link between Sphinx documentation of different projects (y/n) [n]: y
> todo: write "todo" entries that can be shown or hidden on build (y/n) [n]: y
> coverage: checks for documentation coverage (y/n) [n]: y
> imgmath: include math, rendered as PNG or SVG images (y/n) [n]: n
> mathjax: include math, rendered in the browser by MathJax (y/n) [n]: y
> ifconfig: conditional inclusion of content based on config values (y/n) [n]: y
> viewcode: include links to the source code of documented Python objects (y/n)␣
↪[n]: y
> githubpages: create .nojekyll file to publish the document on GitHub pages (y/n)␣
↪[n]: y

A Makefile and a Windows command file can be generated for you so that you
only have to run e.g. `make html' instead of invoking sphinx-build
directly.
> Create Makefile? (y/n) [y]: y
> Create Windows command file? (y/n) [y]: y

Creating file ./source/conf.py.
Creating file ./source/index.rst.
Creating file ./Makefile.
Creating file ./make.bat.

Finished: An initial directory structure has been created.

You should now populate your master file ./source/index.rst and create other␣
↪documentation
source files. Use the Makefile to build the docs, like so:
   make builder
where "builder" is one of the supported builders, e.g. html, latex or linkcheck.
```

Note: 其中, 最后两项询问是否创建”Makefile”文件和”Windows command file”, 选择是 (`y`), 因为这样, 只需在终端简单执行 `make target` 来编译生成目标格式文档, 如 `make html` 生成静态网页文件, 你现在就可以这样做了! 不过还是等等吧, 先看看 `sphinx-quickstart` 都干了啥吧!

打开你的根目录, 你会看到:

我知道你想看看 `build` 目录, 可现在里面确实啥都没有, 看看 `source` 目录吧:

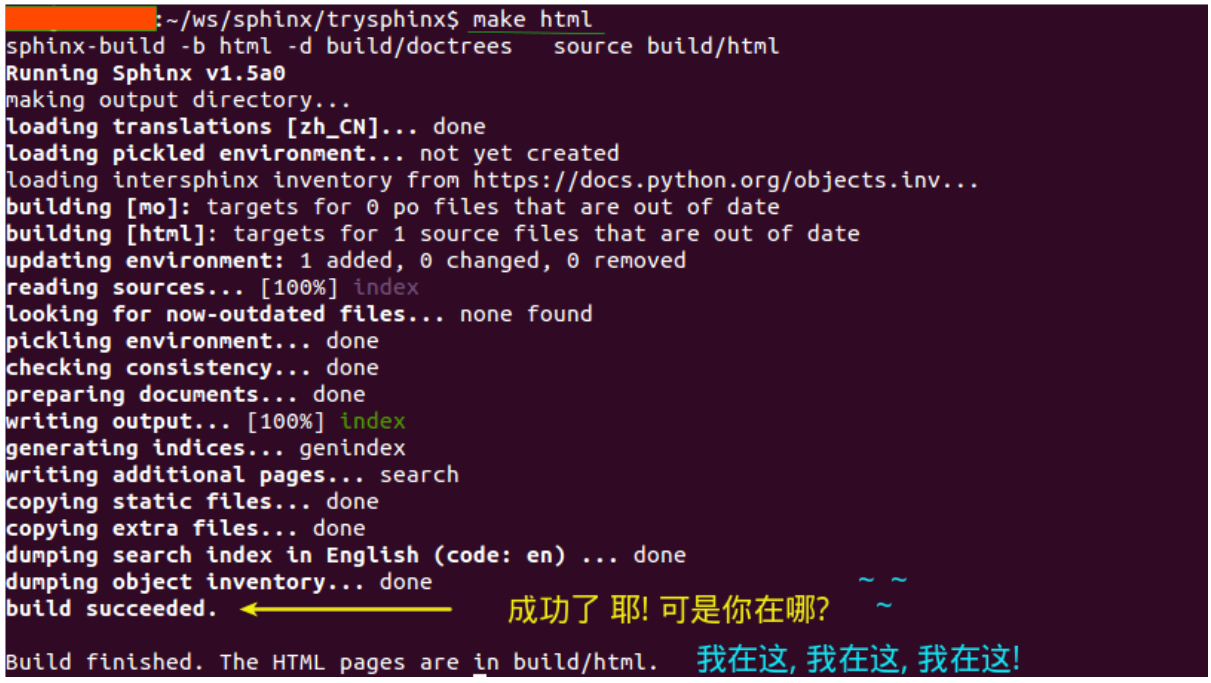
是不是想看看 `index.rst` 里藏着写什么, 那么你看吧, 下面就是:



好配置已经结束了, 我想你一定迫不及待地想看看生成的文档效果吧, 那就一起来构建文档吧!

构建文档

啊!!! 不是吧, 就这么点东西能干啥, 太小了吧, 是啊, 难道不好吗, 我知道你现在一定想知道这能不能生成静态网站, 是的, 那就在终端[1] 输入 `make html` , 你看到了这些:



[1] 由于创建工程时, 选择了生成 *make.bat* 文件, 所以 Windows, Linux 系统上命令操作一样: `make html` .

我想, 你知道去哪找了, 是的, `build` 目录, 快去打开看看吧 (`build/html/index.html`) !

我知道现在你在想: **我都能 make 出来什么呢?** 是的, 我也好奇, 终端输入 `make` 看看吧

```
Please use `make <target>' where <target> is one of

html       to make standalone HTML files
dirhtml    to make HTML files named index.html in directories
singlehtml to make a single large HTML file
pickle     to make pickle files
json       to make JSON files
htmlhelp   to make HTML files and a HTML help project
qthelp     to make HTML files and a qthelp project
applehelp  to make an Apple Help Book
devhelp    to make HTML files and a Devhelp project
epub       to make an epub
epub3      to make an epub3
latex      to make LaTeX files, you can set PAPER=a4 or PAPER=letter
latexpdf   to make LaTeX files and run them through pdflatex
latexpdfja to make LaTeX files and run them through platex/dvipdfmx
text       to make text files
man        to make manual pages
texinfo    to make Texinfo files
info       to make Texinfo files and run them through makeinfo
gettext    to make PO message catalogs
changes    to make an overview of all changed/added/deprecated items
xml        to make Docutils-native XML files
pseudoxml  to make pseudoxml-XML files for display purposes
linkcheck  to check all external links for integrity
doctest    to run all doctests embedded in the documentation (if enabled)
coverage   to run coverage check of the documentation (if enabled)
```

是不是突然发现了好多, 多到你都不知道选哪个了, 没事, 常用的就 `HTML`, `PDF`, `epub` 等等那几个, 不过

你也可以把这些都给大家提供, 没人会阻拦你, 而且你可能还会遇到点小麻烦, 不过我相信你也不太想这样做!

See also:

关于如何发布一些格式的文档, 参见生成不同格式的文档 (page 17), 如果你觉得生成的文档太丑, 请继续往下看使用主题 (page 16).

自动构建并预览文件

如果你觉得如果能自动打开编译好的文件预览就更好了, 这个是可以的, 真的吗? 是的, 真的!

以 **编译预览 HTML 文件为例**, 假设我们用 Google Chrome 预览.

- 对于 Windows 系统:

  1) 首先找到你的浏览器可执行主文件目录, 把它添加到系统环境变量 `PATH` 中去

  `计算机 --> 系统 --> 高级系统设置 --> 高级 --> 环境变量 --> 系统变量`里找到`path`双击打开, 在最前面添加: `C:\Program Files (x86)\Google\Chrome\Application;` (注意你的浏览器路径)

  2) 打开 *make.bat* 文件, 在末尾 `:end` 后追加代码 (`Ctrl + End` 快捷键可以跳到文件末尾), 即:

```
:end

REM -----------------------------------------------
REM Added by Zhi Liu - Auto open build file
REM -----------------------------------------------

if "%1" == "html" (
chrome build/html/index.html
)
```

- 对于 Linux 系统:

  打开 *Makefile* 文件, 按如下代码所示, 添加对应代码, 注意自己的浏览器名称:

```
.PHONY: html
html:
  $(SPHINXBUILD) -b html $(ALLSPHINXOPTS) $(BUILDDIR)/html
  @echo
  @echo "Build finished. The HTML pages are in $(BUILDDIR)/html."
  # ------------------------------------------------------------
  # Added by Zhi Liu - auto open html
  google-chrome buil/html/index.html
  # ------------------------------------------------------------
```

撰写文档

我知道, 你发现你的文档没啥内容啊! 你突然感觉要写点什么了, 要不然空空的给谁看谁会看呢? 那就开始学习 **reStructuredText** 吧, 请移步*reStructuredText 简明教程* (page 36) .

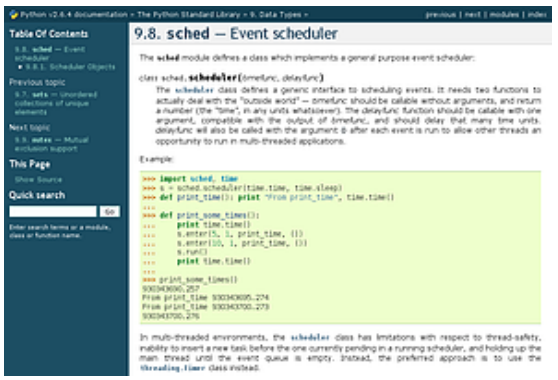
2.2.5 使用主题

如果你觉得生成的 HTML 不符合你的审美观, 请参考手册 HTML theme support 部分, 里面会详细介绍, 这里只简单介绍主题的更换.

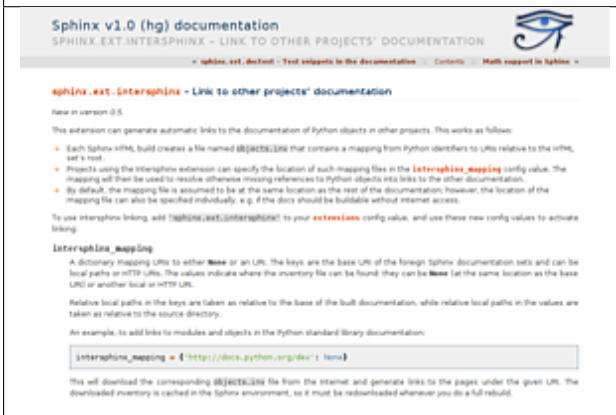
2.2.5.1 Sphinx 自带主题

可供选的内置主题有: **basic, alabaster, sphinx_rtd_theme, classic, sphinxdoc, scrolls, agogo, traditional, nature, haiku, pyramid, bizstyle, epub**. 下面给出几个主题的效果, 更多参见: HTML theming support .

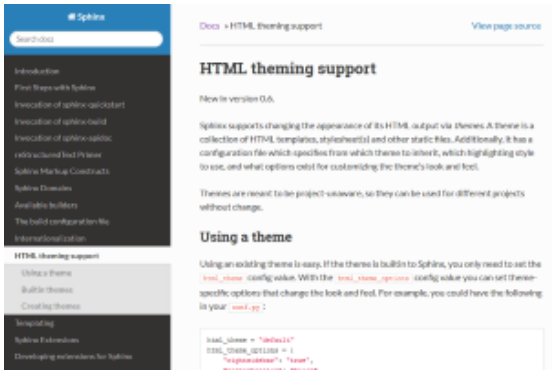
alabaster



classic



haiku



sphinx_rtd_theme

更改内置主题

修改 `conf.py` 文件, 将 `html_theme` 修改成你想用的主题名字

```
html_theme = 'alabaster'
```

然后重新编译 `make html` 即可.

使用 readthedocs 主题

Read the Docs 提供项目文档托管服务, 不过这里只是使用其提供的主题.

1. 终端输入 `pip install sphinx_rtd_theme` 安装 readthedocs 主题
2. 然后修改 `conf.py` 文件:

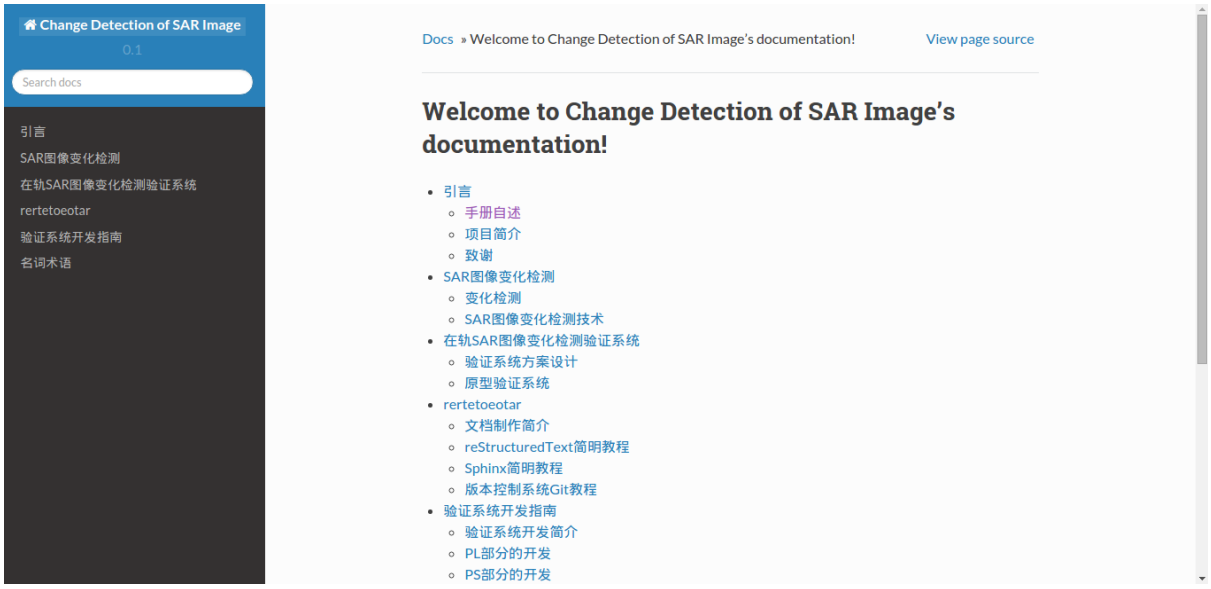
首先在文件开头导入主题包

```
# for using Read the Docs theme
import sphinx_rtd_theme
```

然后修改 `html_theme` 和 `html_theme_path` 的值

```
# html_theme = 'sphinxdoc'
html_theme = 'sphinx_rtd_theme'

#html_theme_path = []
html_theme_path = [sphinx_rtd_theme.get_html_theme_path()]
```

3. 最后终端执行 `make html` 即可, 效果图如下：



2.2.6 生成不同格式的文档

2.2.6.1 生成 html 静态网站文件

进入 Sphinx 工程根目录, 直接终端运行：`make html` 即可在 `build/html` 目录中看到生成的静态网站文件, 双击首页 html 文件即可打开浏览.

See also:

如果文档含有中文, 且需要中文搜索功能, 请参考中文分词问题 (page 31) .

2.2.6.2 生成 epub 文档

进入 Sphinx 工程根目录, 直接终端运行：`make epub` 即可在 `build/epub` 目录中看到生成的 epub 文件.

See also:

如果文档含有中文, 且需要中文搜索功能, 请参考中文分词问题 (page 31)

2.2.6.3 生成 LaTex 文档

要生成 LaTex 文档, 需要安装 LaTex 编译环境, 这里建议使用 TeXLive , 安装包约 2GB, 虽然安装后约占 4GB 空间, 但最为完整且支持 Windows、Linux、Mac 系统.

安装完成后, 进入 Sphinx 工程根目录, 在终端输入：`make latex` 即可生成对应的 LaTex 文件, 在目录 `build/latex` 里.

See also:

如果文档含有中文, 请参考中文文档问题 (page 30)

这里还可以自定义 `Latex` 输出样式, 具体参见 LaTeX customization . 主要修改 *conf.py* 文件, 本人采用的配置如下:

Hint: 上述代码中包含的颜色信息, 可以从 svgnames Colors 找到.

2.2.6.4 生成 PDF 文档

由于 Sphinx 通过 Latex 间接生成 PDF 文档, 所以需要安装 LaTex 编译环境. 安装完成后, 进入 Sphinx 工程根目录, 直接终端运行：`make latexpdf` 即可在 `build/latex` 目录中看到生成的 PDF 文件.

其实，我们还可以先通过 `make latex` 生成 *tex* 文件，然后就可以像处理普通 tex 文件那样自由编辑编译生成 PDF 文件。

See also:

如果文档含有中文, 请参考中文文档问题 (page 30) , 如果你觉得生成的 PDF 不好看, 可以参考生成 *LaTex* 文档 (page 18) 自定义.

自定义生成的 PDF 文件部分环境预览如下

2.2.7 Sphinx 重要扩展介绍

2.2.7.1 Sphinx 扩展

可以通过以下链接找到 Sphinx 的扩展:

- sphinx extensions doc : 官方文档介绍, 含 **Built-in extensions** 和 **Third-party extensions**

(continued from previous page)

```
.. important:: This is an important

.. seealso:: This is seealso
```

被渲染成:

Tip: This is a tip

Note: This is a note

Hint: This is a hint

Danger: This is a danger

Error: This is an error

Warning: This is a warning

Caution: This is a caution

Attention: This is an attention

Fig. 1: 自定义生成的 PDF 文件部分环境预览结果

自定义生成的 PDF 文件部分环境预览结果

- Awesome Sphinx : A curated list of awesome tools for Sphinx Python Documentation Generator.
- sphinx-contrib : a collection of Sphinx extensions maintained by their respective authors. It is not an official part of Sphinx.
- Survey of Sphinx extensions : This is list of Sphinx extensions at October, 2014.

2.2.7.2 内置扩展简介

目录树 (toctree)

由于 reST 没有处理多个文档, 或将文档分割成多个输出文件的机制, Sphinx 使用一个自定义指令来添加组成整篇文档的单个文件间的关系, 以及目录. 这个指令的核心就是 `toctree` .

Tip: 简单包含某个文件, 可以使用 include 指令.

代码与语法着色

更多功能, 参考 Showing code examples

```
.. code-block:: python
    :lineno-start: 10
    :emphasize-lines: 9
    :linenos:
    :caption: demo_python.m
    :name: bind-id

    import pytool
    import numpy as np
    import matplotlib.pyplot as plt

    # ====================generate Ellipse====================
    a = 6  # major axis
    b = 2  # minor axis
    x0 = 10  # center x0
    y0 = 10  # center y0
    N = 1000  # number of points

    # angle for rotating ellipse data
    theta = np.pi * 30 / 180

    x, y = pytool.ellipse_surface(a, b, x0, y0, N, 'rand')

    x = x - np.mean(x)
    y = y - np.mean(y)
```

将被渲染成

Listing 1: demo_python.m

```
import pytool
import numpy as np
import matplotlib.pyplot as plt

# ====================generate Ellipse====================
a = 6  # major axis
b = 2  # minor axis
x0 = 10  # center x0
y0 = 10  # center y0
N = 1000  # number of points

# angle for rotating ellipse data
theta = np.pi * 30 / 180

x, y = pytool.ellipse_surface(a, b, x0, y0, N, 'rand')

x = x - np.mean(x)
y = y - np.mean(y)
```

2.2.7.3 第三方扩展

如下扩展可以通过类似 `pip install extensions_name` 的命令安装, 在 *conf.py* 文件中的 `extensions` 中加入该扩展, 以下不在赘述.

sphinxcontrib-proof

sphinxcontrib-proof 提供定理, 定义, 证明等支持. 在 *conf.py* 文件中的 `extensions` 中加入该扩展 (`sphinxcontrib.proof`).

然后在 *_static* 目录下新建 *proof.css* 和 *proof.js* 两个文件, 加入如下内容, 你可以自己定义其它的样式.

proof.css

```
.proof {
  margin-top: 1em;
  margin-bottom: 1em;
}

/* Titles */
.proof .proof-title {
  background-color: #0000EE;
  border: 1px solid #86989b;
  color: white;
  font-size: 120%;
```

(continues on next page)

(continued from previous page)

```
  }

/* Content */
.proof-content {
  border: 1px solid #9fb1b4;
  background-color: #F0F8FF;
  padding: 0.5em 1em;
}


/* Toggle proof */
.proof-type-proof > .proof-title {
    display: block;
    clear: both;
}

.proof-type-proof > .proof-title:after {
    content: " ▼";
}

.proof-type-proof > .proof-title.open:after {
    content: " ▲";
}
```

proof.js

```
$(document).ready(function() {
    $(".proof-type-proof > *").hide();
    $(".proof-type-proof .proof-title").show();
    $(".proof-type-proof .proof-title").click(function() {
        $(this).parent().children().not(".proof-title").toggle(400);
        $(this).parent().children(".proof-title").toggleClass("open");
    })
});
```

使用举例:

```
.. _righttriangle:

.. proof:definition:: Right triangle

   A *right triangle* is a triangle in which one angle is a right angle.

.. _pythagorean:

.. proof:theorem:: Pythagorean theorem
```

(continues on next page)

(continued from previous page)

```
   In a :ref:`righttriangle`, the square of the hypotenuse is equal to the sum of␣
↪the squares of the other two sides.

.. _proof:

.. proof:proof::

   The proof is left to the reader.

You can label and reference definition and theorems (e.g. :numref:`theorem {number}
↪ <pythagorean>`). You can also reference proofs (see the :ref:`proof of the␣
↪Pythagorean theorem <proof>`).
```

代码将被渲染为

Definition 1 (Right triangle) *A* right triangle *is a triangle in which one angle is a right angle.*

Theorem 2 (Pythagorean theorem) *In a* Right triangle *(page 23), the square of the hypotenuse is equal to the sum of the squares of the other two sides.*

Proof 3 *The proof is left to the reader.*

You can label and reference definition and theorems (e.g. `theorem {number}`). You can also reference proofs (see the *proof of the Pythagorean theorem* (page 23)).

图表编号

借用 jterrace 的论文模版 sphinxtr 中的 `numfig` 可以实现. 从 这里 下载源码, 将其中的 *sphinxtr* 放到你的文档源码根目录下, 然后 *conf.py* 添加

Listing 2: Code Blocks can have captions.

```
sys.path.insert(0, os.path.join(os.path.abspath(os.path.dirname(__file__)),
↪'extensions'))

extensions = [
  'sphinx.ext.autodoc',
  'sphinx.ext.doctest',
  'sphinx.ext.intersphinx',
  'sphinx.ext.todo',
  'sphinx.ext.coverage',
  # 'sphinx.ext.imgmath',
  # 'sphinx.ext.mathjax',
  'sphinxcontrib.katex',
  'sphinxcontrib.proof',  # https://framagit.org/spalax/sphinxcontrib-proof/
  'sphinxcontrib.bibtex',  # https://sphinxcontrib-bibtex.readthedocs.io/en/
↪latest/
  'sphinxcontrib.seqdiag',  # http://blockdiag.com/en/
```

(continues on next page)

(continued from previous page)

```
    'sphinx.ext.ifconfig',
    # 'sphinx.ext.viewcode',
    # 'sphinx.ext.githubpages',
    # 'rst2pdf.pdfbuilder',
    # 'sphinx.ext.napoleon',
    'numequ',  # https://github.com/jterrace/sphinxtr/tree/master/extensions
    'numfig',  # https://github.com/jterrace/sphinxtr/tree/master/extensions
    'subfig',  # https://github.com/jterrace/sphinxtr/tree/master/extensions
]

math_numfig = True
number_figures = True
figure_caption_prefix = 'Figure'
```

比如, 这里通过如下代码插入图片:

```
.. _fig-testFigureNumber:

.. figure:: ../_static/figs/logo.*
    :alt: Test Figure Number
    :width: 30%
    :align: center

    Test Figure Number
```

代码将被渲染为

在其它地方可以通过 `:num:`fig-testFigureNumber`` 引用, fig-testFigureNumber .

sphinxcontrib-bibtex

在 Sphinx 中可以使用 BibTex , 通过 `pip install sphinxcontrib-bibtex` 安装扩展, 并在 *conf.py* 中添加该扩展 `sphinxcontrib.bibtex` , 官方文档在 这里 .

然后, 新建 *reference.rst* , 加入如下代码:

Listing 3: reference.rst.

```
.. bibliography:: ./refs.bib
    :list: enumerated
    :start: 1
```

假如 *refs.bib* 文件中的内容如下:

```
  @Proceedings{1993:PatiOMP,
  author={Y. C. {Pati} and R. {Rezaiifar} and P. S. {Krishnaprasad}},
  booktitle={Proceedings of 27th Asilomar Conference on Signals, Systems and␣
↪Computers},
```

(continues on next page)

Fig. 2: Test Figure Number

(continued from previous page)

```
  title={Orthogonal matching pursuit: recursive function approximation with␣
↪applications to wavelet decomposition},
  year={1993},
  volume={},
  number={},
  pages={40-44 vol.1},
  doi={10.1109/ACSSC.1993.342465},
  ISSN={1058-6393},
  month={Nov},
}


@article{2003JChPh.118.6720W,
   author = {{Wu}, Y. and {Batista}, V.~S.},
    title = "{Matching-pursuit for simulations of quantum processes}",
  journal = {\jcp},
 keywords = {Tunneling traversal time quantum Zeno dynamics, Foundations of␣
↪quantum mechanics, measurement theory, Fourier analysis, Integral transforms},
     year = 2003,
    month = apr,
   volume = 118,
    pages = {6720-6724},
      doi = {10.1063/1.1560636},
   adsurl = {http://adsabs.harvard.edu/abs/2003JChPh.118.6720W},
  adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

可以通过 `:cite:`1993:PatiOMP`` , `:cite:`2003JChPh.118.6720W`` 来引用, 即 [1] , [2] . 如果一次性引用多个文献, 可以用逗号分开, 但不要有空格, 比如这样 `:cite:`1993:PatiOMP,2003JChPh.118.6720W`` 得到 [1][2] .

Hint: 如果你想自定义参考文献引用格式, 可以通过 `pip install pybtex` 安装 pybtex , 然后参考 这里 或者下面的讲述配置使用.

Note: `pybtex` 提示: 安装好 pybtex 后, 若想在你的文档工程中使用, 需要在 *conf.py* 文件中添加该扩展, 即 `extensions = ['pybtex']` , 然后你就可以使用了, 在 `.. bibliography:: ./refs.bib` 里添加 `:style: unsrt` 即可以更改文献引用格式.

Listing 4: reference.rst.

```
.. bibliography:: ./refs.bib
   :style: unsrt
```

注意, 如果添加 `list` 或 `start` 等域, 不能正常渲染, 不能跳转!

sphinxcontrib-xxxdiag

xxxdiag 包含以下几种类型:

- blockdiag : blockdiag
- seqdiag : seqdiag
- actdiag : actdiag
- nwdiag : nwdiag

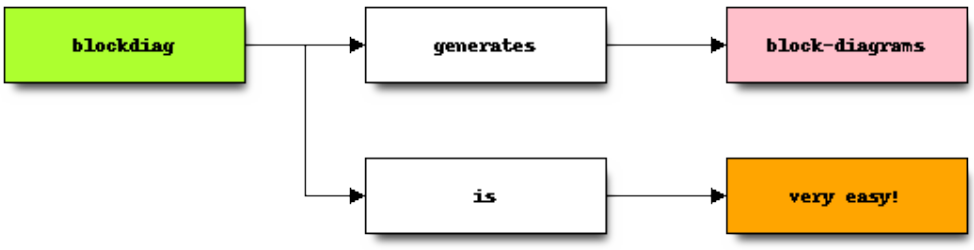
通过 pip install sphinxcontrib-xxxdiag 安装扩展, 并在 *conf.py* 中添加该扩展 sphinxcontrib.xxxdiag , 官方文档在 这里 .

举例: 如

原始代码

```
.. blockdiag::

 blockdiag {
   blockdiag -> generates -> "block-diagrams";
   blockdiag -> is -> "very easy!";

   blockdiag [color = "greenyellow"];
   "block-diagrams" [color = "pink"];
   "very easy!" [color = "orange"];
 }
```

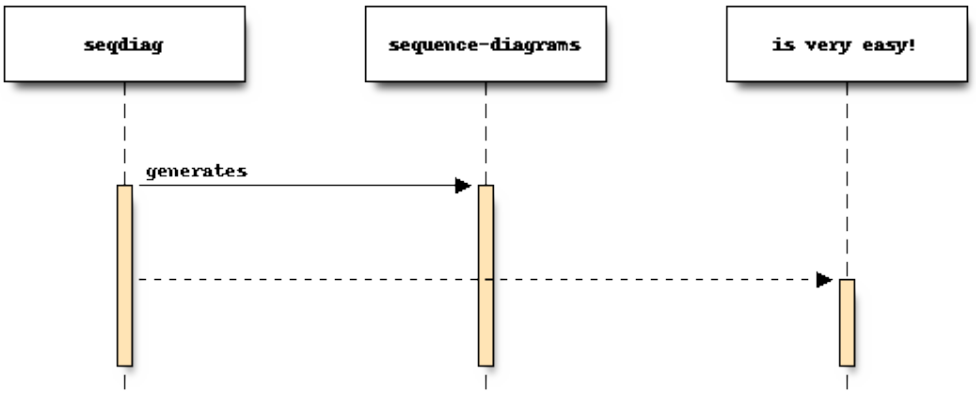
渲染结果



原始代码

```
.. seqdiag::

 seqdiag {
   seqdiag -> "sequence-diagrams" [label = "generates"];
   seqdiag --> "is very easy!";
 }
```

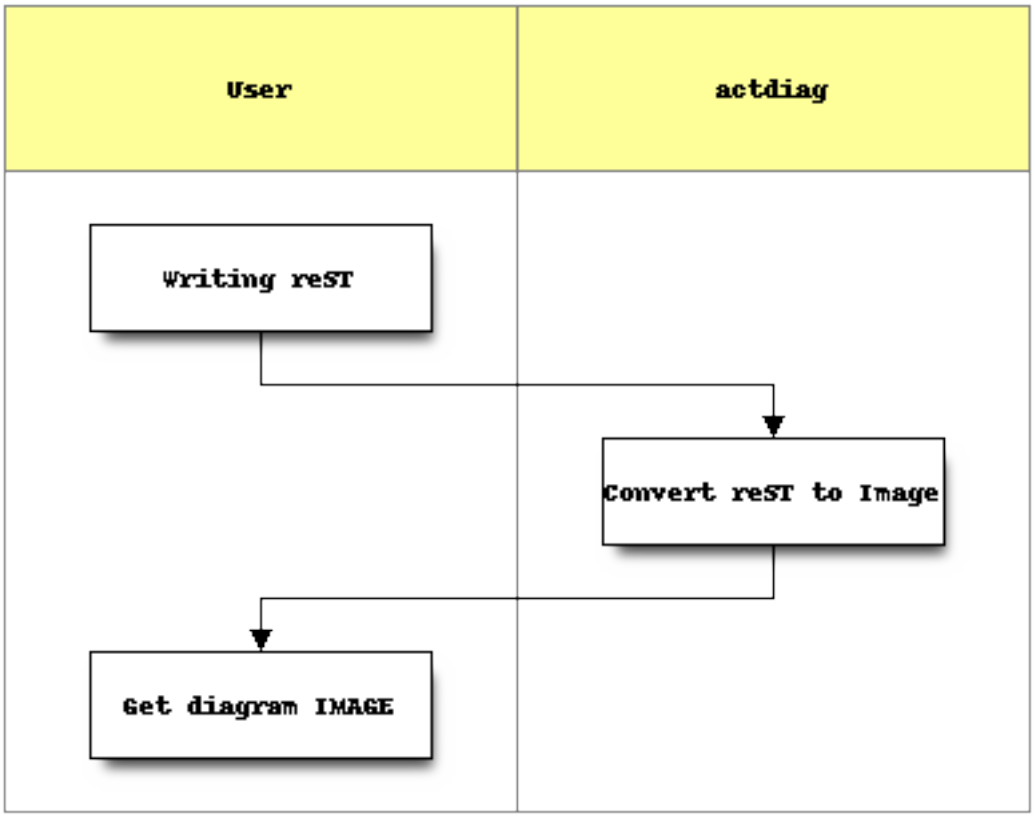
渲染结果



原始代码

```
.. actdiag::

   actdiag {
     write -> convert -> image

     lane user {
        label = "User"
        write [label = "Writing reST"];
        image [label = "Get diagram IMAGE"];
     }
     lane actdiag {
        convert [label = "Convert reST to Image"];
     }
   }
```

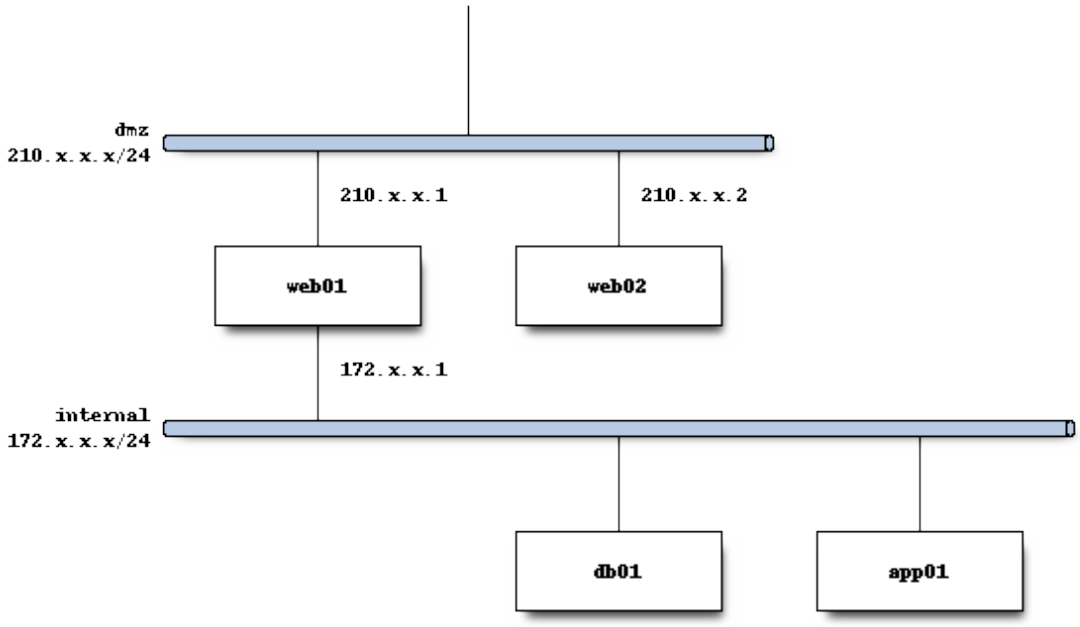
渲染结果



原始代码

```
.. nwdiag::

   nwdiag {
     network dmz {
         address = "210.x.x.x/24"

         web01 [address = "210.x.x.1"];
         web02 [address = "210.x.x.2"];
     }
     network internal {
         address = "172.x.x.x/24";

         web01 [address = "172.x.x.1"];
         db01;
         app01;
     }
   }
```

渲染结果



2.2.8 问题集锦

2.2.8.1 源编码

在 reST 使用 Unicode 字符可以容易的包含特殊字符如破折号, 版权标志. Sphinx 默认源文件使用 UTF-8 编码; 你可以通过修改 *conf.py* 文件的 `source_encoding` 的配置值改变编码.

2.2.8.2 中文文档问题

在使用 Sphinx 发布 ** 中文文档 ** 生成 LaTex 或 PDF 文件时, 会发现报出如下类似错误

```
! Package inputenc Error: Unicode char \u8: 引 not set up for use with LaTeX.
```

这是因为, Sphinx 默认使用 `pdflatex` 编译文档, 而要让 `latex` 支持中文, 需要包含一些中文包. 最好的解决方案是使用 `xelatex` 编译引擎并包含 `CTEX` 包, 只需要修改两个文件：

1. 修改 `conf.py` 文件（约 220 行）

添加如下代码, 使得程序自动在 latex 文件的导言区加入包引用

```
# -- Options for LaTeX output --------------------------------------------

latex_elements = {
```

(continues on next page)

(continued from previous page)

```
# The paper size ('letterpaper' or 'a4paper').
#'papersize': 'letterpaper',

# The font size ('10pt', '11pt' or '12pt').
#'pointsize': '10pt',

# Additional stuff for the LaTeX preamble.
#'preamble': '',

# Latex figure (float) alignment
#'figure_align': 'htbp',

# Using Package for ZH
'preamble' : r'''
\usepackage{ctex}
''',
}
```

添加的代码为 `# Using Package for ZH` 下面的三行, 然后保存.

2. 修改 `Makefile` 文件（约 142 行）

添加如下代码中的第 5 行代码, 即可更改编译引擎为 xelatex

```
.PHONY: latexpdf
latexpdf:
    $(SPHINXBUILD) -b latex $(ALLSPHINXOPTS) $(BUILDDIR)/latex
    @echo "Running LaTeX files through pdflatex..."
    sed -i s/pdflatex/xelatex/ $(BUILDDIR)/latex/Makefile
    $(MAKE) -C $(BUILDDIR)/latex all-pdf
    @echo "pdflatex finished; the PDF files are in $(BUILDDIR)/latex."
```

修改后保存, 然后终端运行 `make latexpdf` 即可看到 PDF 成功生成的提示, 在 `build/latex` 文件夹中可以找到.

2.2.8.3 中文分词问题

Sphinx 原生只支持英文搜索, 参考 让 sphinx 支持简体中文搜索的插件 , 来实现中文搜索的支持, 下载插件并安装, 具体过程如下：

1. 安装结巴分词, Linux：`sudo pip install jieba` , Windows：`pip install jieba`
2. 复制文件 `zh_CN.py` 到 sphinx 的 search 目录下, Linux：`/usr/lib/python2.7/dist-packages/sphinx/search`, Windows:`C:\Python27\Lib\site-packages\sphinx\search`
3. 打开 `search` 目录下的 `__init__.py` 文件, 找到代码, 添加对中文的支持：

```
from sphinx.search import en, ja
languages = {
```

(continues on next page)

(continued from previous page)

```
    'en': en.SearchEnglish,
    'ja': ja.SearchJapanese,
}

修改成:

from sphinx.search import en, ja, zh_CN
languages = {
    'en': en.SearchEnglish,
    'ja': ja.SearchJapanese,
    'zh_CN': zh_CN.SearchChinese
}
```

4. 修改 *conf.py* 文件, 将语言设置成 `language = 'zh_CN'`, 保存然后重新编译 `make html` 即可.

关于 CJK 支持请参考: CJK 支持 .

2.2.8.4 中文乱码问题

问题描述: 如果你使用 Sphinx 生成含有中文字符的静态网页时, 当你你视图点击 `View page source` 时可能发现中文乱码的问题, 如下图:

这时要确保编码的匹配一致性: **源码文件编码**, **源码编码设置值**, **浏览器编码**要一致, Sphinx 采用 `UTF-8` 来支持不同特殊的字符. 所以, 请检查以上三项: 第二项是要确保 *conf.py* 文件中的编码为 `UTF-8`, 即 `#source_encoding = 'utf-8-sig'`, 这也是默认值; 第三项, 可参见下图更改浏览器编码:

2.2.8.5 Sphinx 数学支持

有关 Sphinx 中对数学公式的支持, 参考手册 Math support in Sphinx 部分, 下面简单介绍使用.

还记得在创建工程时, 询问如何处理数学公式的吗:

```
> imgmath: include math, rendered as PNG or SVG images (y/n) [n]: n
> mathjax: include math, rendered in the browser by MathJax (y/n) [n]: y
```

当时选择的是使用 MathJax (两者只能选一个, 不信你试试) , 是的, 所以”conf.py”里的 `extensions` 项里只有 `'sphinx.ext.mathjax'`, 如下:

```
extensions = [
  'sphinx.ext.autodoc',
  'sphinx.ext.doctest',
  'sphinx.ext.intersphinx',
  'sphinx.ext.todo',
  'sphinx.ext.coverage',
  'sphinx.ext.mathjax',
  'sphinx.ext.ifconfig',
  'sphinx.ext.viewcode',
  'sphinx.ext.githubpages',
]
```

如果你想更改, 可以, 注释掉 `'sphinx.ext.mathjax',` , 并添加 `'sphinx.ext.imgmath',` 即可.

Note: Sphinx 提供了两种数学公式的渲染方式 (**二选一**):

- `'sphinx.ext.imgmath'` : 通过 **LaTex** 渲染成图片 (`png` 或 `svg`).
- `'sphinx.ext.mathjax'` : 通过 **MathJax** 渲染.

使用 LaTex 渲染公式

如上所述, 在 `extensions` 列表变量里添加项 `'sphinx.ext.imgmath',` 即可. 数学公式通过 LaTex 引擎渲染成图片, 然后在 HTML 中显示. 但要求 PC 机安装有 LaTex.

如果你不使用默认配置, 请参考 Math support in Sphinx .

使用 KaTex 渲染公式

> KaTex 具有比 MathJax 更快的解析速度. 使用 `pip install sphinxcontrib.katex` 安装 KaTex 扩展. 如上所述, 在 `extensions` 列表变量里添加项 `'sphinxcontrib.katex',` 即可. 数学公式通过 KaTex 引擎渲染公式在 HTML 中显示.

使用 MathJax 渲染公式

如上所述, 使用 MathJax (准确说是 Render math via JavaScript) 渲染公式, 需要在 `extensions` 列表变量里添加项 `'sphinx.ext.mathjax',` .

此外, 你还需要配置 MathJax 的路径 `mathjax_path` , Sphinx 默认使用 MathJax CDN 提供的 JS 文件. Sphinx 默认使用的 MathJax 的路径值为 `https://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML` 但你可能发现不管用, 可能是 `https` 安全协议的问题, 改成 `http` 即可, 即设置配置文件 *conf.py* 中的 `mathjax_path = 'https://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML'` .

当然你也可以使用本地的 MathJax, 参考 Installing Your Own Copy of MathJax , 到 这里 下载 MathJax 包 (约 34.1MB) 并解压放到“_static”目录, 然后设置路径, 即在 *conf.py* 文件中添加代码: `mathjax_path =`

`MathJax/MathJax.js` . 重新 `make html` 即可.

更多细节问题请参考 Math support in Sphinx .

2.2.9 发布静态网站

如果你想让别人看到你的项目文档, 你有两个选择:

- 放到 Read the Docs 进行托管;
- 建立你自己的网站.

这里介绍第二种方案.

2.2.9.1 使用 Apache 建立自己的网站服务器

> *Apache <http://httpd.apache.org/>* 是世界使用排名第一的 Web 服务器软件. 它可以运行在几乎所有广泛使用的计算机平台上, 由于其跨平台和安全性被广泛使用, 是最流行的 Web 服务器端软件之一. 它快速、可靠并且可通过简单的 API 扩充, 将 Perl/Python 等解释器编译到服务器中. 同时 Apache 音译为阿帕奇, 是北美印第安人的一个部落, 叫阿帕奇族, 在美国的西南部. 也是一个基金会的名称、一种武装直升机等等.

以 **Ubuntu** 系统为例:

安装 apache2

使用命令 `sudo apt-get install apache2` , 安装完成后浏览器输入 `http://127.0.0.1` 测试显示 *It works!* 表明安装成功.

配置 apache2

修改主配置文件 `/etc/apache2/apache2.conf` 和端口监听设置文件 `/etc/apache2/port.conf` 文件:

使用 `sudo gedit /etc/apache2/apache2.conf` 打开文件, 在文件末尾添加:

```
ServerName localhost
DirectoryIndex index.html index.htm index.php
```

使用 `sudo gedit /etc/apache2/port.conf` 打开文件, 在文件末尾添加:

```
- ** 重启 apache2**: ``sudo /etc/init.d/apache2 restart`` .
```

现在

2.2.10 使用 Git 版本控制系统

请参考版本控制系统 *Git* 教程 (page 56) .

2.3 reStructuredText 简明教程

简要介绍 **reStructuredText** 的概念, 语法等.

2.3.1 What is reStructuredText

reStructuredText 官网说 **reStructuredText** 是 **Docutils** (Documentation Utilities) 标记语法和解析器组件 (Markup Syntax and Parser Component of Docutils).

> Docutils 项目的主要是为了创造一套将纯文本转换为一些常用格式的工具, 这些常用格式包括: HTML、XML 和 LaTeX.

reStructuredText 是一个易读, 易写, 所见即所得的明文标记语法和解析系统, 对于“in-line”程序文档 (如 Python docstrings) 非常有用, 用于快速创建简单的网页和本地文档. reStructuredText 是专为特定应用领域的扩展而设计, 其解析器是 Docutils 的一个组件. reStructuredText 是轻量级标记系统 StructuredText 和 Setext 的修订和重解释版本.

reStructuredText 的主要目的是定义和实现一个用于 Python 文档和其他文档域的 **标记语法**, 要求是 **可读的** 和 **简单的**, 同时 **足够胜任非凡的任务**. 标记的目的是能够将 reStructuredText 文档转换成有用的结构化数据格式.

上面是对下面这段话的翻译:

> reStructuredText is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system. It is useful for in-line program documentation (such as Python docstrings), for quickly creating simple web pages, and for standalone documents. reStructuredText is designed for extensibility for specific application domains. The reStructuredText parser is a component of Docutils. reStructuredText is a revision and reinterpretation of the StructuredText and Setext lightweight markup systems.
>
> The primary goal of reStructuredText is to define and implement a markup syntax for use in Python docstrings and other documentation domains, that is readable and simple, yet powerful enough for non-trivial use. The intended purpose of the markup is the conversion of reStructuredText documents into useful structured data formats.

就这些吧! 如果你现在就想尝试, 可以在线试一试: http://rst.ninjs.org/.

> **Warning:** “reStructuredText” is **ONE** word, *not two!*, 以下可能会简写成 `reST`.

2.3.2 Why reStructuredText

我不想再多说了, 简要列举一下吧:

- 专注于文本内容而不是排版样式
- 兼容所有文本编辑器与字处理软件
- 渲染导出格式丰富, 如 HTML、PDF, 借助一些工具还可以导出 Latex、epub 等文件
- 可以使用 Git 等版本控制系统管理文章版本

- 可读、直观、易学

2.3.3 reST 环境搭建

正如前面所说, 你可以在线试一试: http://rst.ninjs.org/, 也可以去下载一些相关工具, 可以参考 这里 , 如 Docutils , Sphinx , 当然你也可以参照我的博客: 超级文本编辑器 Sublime Text3 来安装 Sublime Text 并配置 reStructuredText 环境, 不过我下面会讲到.

如果你想使用 `Sphinx` 请移步*Sphinx 简明教程* (page 6) , 下面介绍 **Sublime Text** 下环境的配置.

2.3.3.1 Sublime Text 及其安装

用过 Notepad++ 的童鞋都可能会对其爱不释手, 但如果你基本掌握了 Sublime Text (以下会简称为”ST”), 你应该会有一种相见恨晚的感觉!

> Sublime Text is a sophisticated text editor for code, markup and prose. You’ll love the slick user interface, extraordinary features and amazing performance.

如作者自己对 ST 的定位, ST 是一个高雅且精良的 **文本编辑器**, 由程序员 Jon Skinner 于 2008 年 1 月份所开发出来, 可以访问 ‘Sublime Text 主页 <http://www.sublimetext.com/>‘_ 查看效果展示.

> Sublime Text 具有漂亮的用户界面和强大的功能, 例如代码缩略图, Python 的插件, 代码段等. 还可自定义键绑定, 菜单和工具栏. Sublime Text 的主要功能包括：拼写检查, 书签, 完整的 Python API , Goto 功能, 即时项目切换, 多选择, 多窗口等等.

Sublime Text 支持三大主流操作系统: Windows, Linux, OS X. 几乎你需要的功能都有, 一切可修改（快捷键、插件包 etc.）, 界面优美, 可惜的是不开源, 不过即使不注册也可以使用, 功能不受限. Lime Text 是其开源版的一种实现, 我还没打算用这个.

See also:

如果你在使用 ST 的过程中遇到了一些问题, 可以先访问本人博客 超级文本编辑器 Sublime Text3, 也许有你要的答案.

2.3.3.2 Sublime Text 的安装

对于 Windows 系统, 其实是不需要安装的, 是的, 你没听错, 请访问 ST 下载页面 , 并选择 **便携版 (portable version)**, 然后解压后双击 sublime_text.exe 当然, 你也可以选择安装版, 不过我会选择便携版, 因为这样复制到其它机器上就照常用了.

对于 Linux 系统, 可以访问 ST 下载页面 下载安装, 对于 Ubuntu 系统, 直接双击下载的 `deb` 包, 然后到软件中心单击 安装. 当然, 可以通过命令安装: `sudo dpkg -i <package.deb>` , 安装完成后, 终端输入: `subl` 即可启动 ST3.

2.3.3.3 ST 中插件包的安装

在 ST 中可以配置各种代码编辑器环境, 一般通过安装 ST 的插件包来实现, **Package Control** 是 ST 的插件包管理器, 访问 Package Control 查看插件包, 插件包的安装方式一般有如下两种.

下载并解压安装包

到 Package Control 或 GitHub 搜索并下载插件包, 然后解压到 ST 根目录即可.

- 对于 Windows, 请解压至 `Sublime Text3 x64/Data/Packages` 目录下
- 对于 Linux 系统, 解压至 `~/.config/sublime-text-3/Packages` 下

然后, 重启 ST 即可.

通过 Package Control 安装管理包

1. 首先需要给 ST 安装 `Package Control` 包管理器插件.

安装方法很简单, 可以参见官网：https://packagecontrol.io/installation#st3, 也可以看下面的介绍.

通过 `View --> Show Console` 打开打开控制台, 粘贴如下代码并回车安装

For ST3:

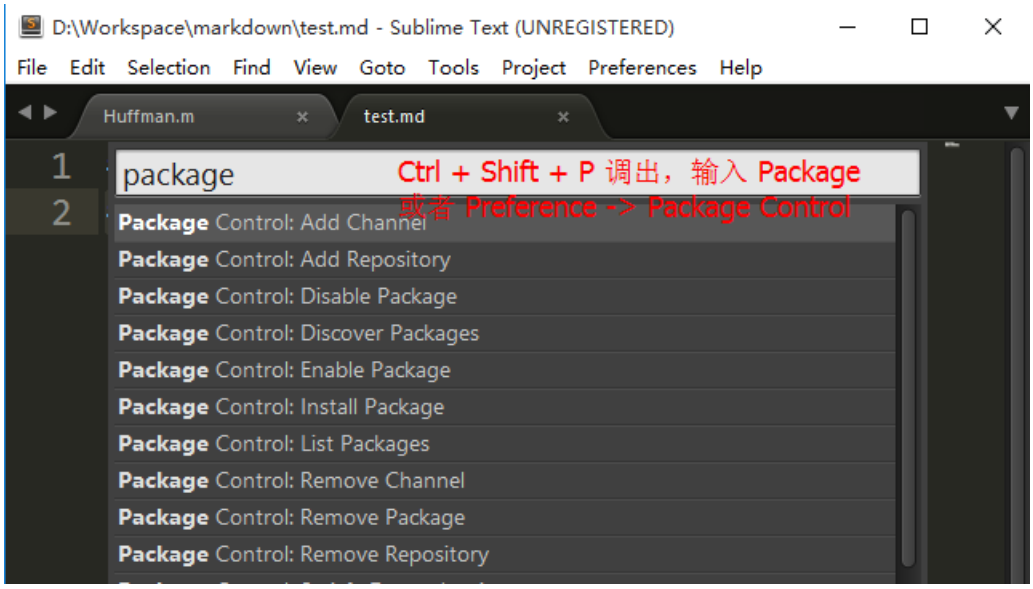
```
import urllib.request,os,hashlib; h = '2915d1851351e5ee549c20394736b442' +
↪'8bc59f460fa1548d1514676163dafc88'; pf = 'Package Control.sublime-package'; ipp␣
↪= sublime.installed_packages_path(); urllib.request.install_opener( urllib.
↪request.build_opener( urllib.request.ProxyHandler()) ); by = urllib.request.
↪urlopen( 'http://packagecontrol.io/' + pf.replace(' ', '%20')).read(); dh =␣
↪hashlib.sha256(by).hexdigest(); print('Error validating download (got %s instead␣
↪of %s), please try manual install' % (dh, h)) if dh != h else open(os.path.join(␣
↪ipp, pf), 'wb' ).write(by)
```

For ST2:

```
import urllib2,os,hashlib; h = '2915d1851351e5ee549c20394736b442' +
↪'8bc59f460fa1548d1514676163dafc88'; pf = 'Package Control.sublime-package'; ipp␣
↪= sublime.installed_packages_path(); os.makedirs( ipp ) if not os.path.
↪exists(ipp) else None; urllib2.install_opener( urllib2.build_opener( urllib2.
↪ProxyHandler()) ); by = urllib2.urlopen( 'http://packagecontrol.io/' + pf.
↪replace(' ', '%20')).read(); dh = hashlib.sha256(by).hexdigest(); open( os.path.
↪join( ipp, pf), 'wb' ).write(by) if dh == h else None; print('Error validating␣
↪download (got %s instead of %s), please try manual install' % (dh, h) if dh != h␣
↪else 'Please restart Sublime Text to finish installation')
```

2. 通过 `Package Control` 安装插件包.

使用 `Ctrl + Shift + P` 打开 PackageControl, 如果没有输入 `package` 就出现了, 选择 `Install Package` , 如下图



接着在弹出的窗口中输入 **插件包的名字**选择安装即可.

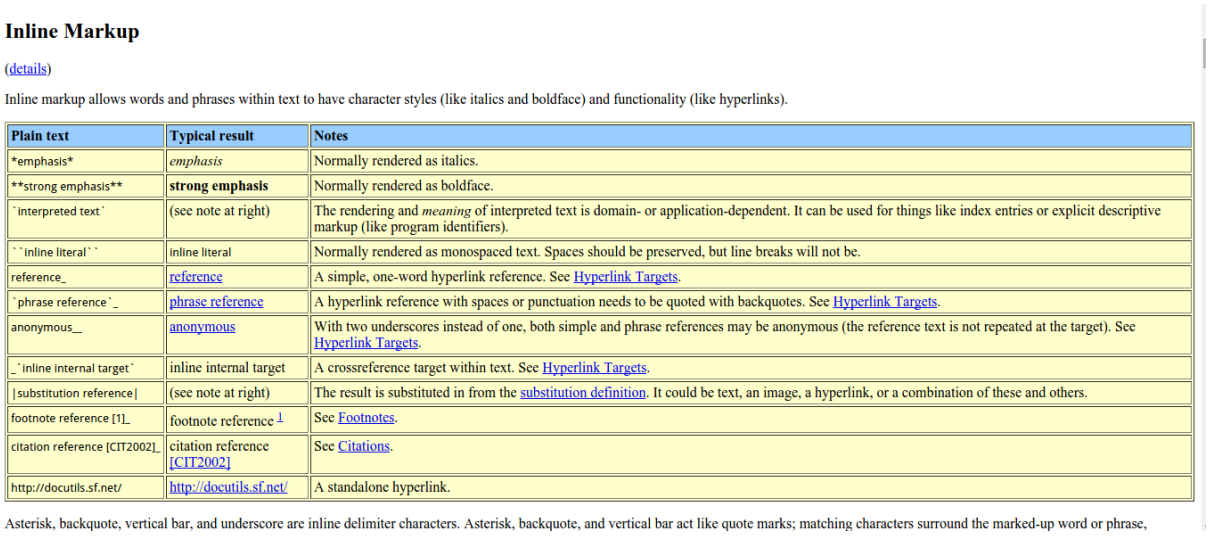
2.3.3.4 配置 reST 环境

给 ST 安装以下插件包:

- OmniMarkupPreviewer , 也可以到 这里 下载, (用于解析渲染 reST 等多种标记语法, **必选**), 安装好后, 按快捷键 `Ctrl + Alt + O` 预览.
- reStructuredText Improved (用于语法着色, **可选**, 建议安装)
- sublime-rst-completion ,(用于自动补全, **可选**, 但推荐安装), 在制表时很有用, 请点击链接查看.

2.3.4 reST 语法简介

请访问 Quick reStructuredText 查看快速参考, 页面给出了示例代码及效果的对比显示, 非常易学, 见下图:



Inline Markup

(details)

Inline markup allows words and phrases within text to have character styles (like italics and boldface) and functionality (like hyperlinks).

Plain text	Typical result	Notes
emphasis	*emphasis*	Normally rendered as italics.
strong emphasis	**strong emphasis**	Normally rendered as boldface.
`interpreted text`	(see note at right)	The rendering and *meaning* of interpreted text is domain- or application-dependent. It can be used for things like index entries or explicit descriptive markup (like program identifiers).
``inline literal``	inline literal	Normally rendered as monospaced text. Spaces should be preserved, but line breaks will not be.
reference_	reference	A simple, one-word hyperlink reference. See Hyperlink Targets.
`phrase reference`_	phrase reference	A hyperlink reference with spaces or punctuation needs to be quoted with backquotes. See Hyperlink Targets.
anonymous__	anonymous	With two underscores instead of one, both simple and phrase references may be anonymous (the reference text is not repeated at the target). See Hyperlink Targets.
_`inline internal target`	inline internal target	A crossreference target within text. See Hyperlink Targets.
|substitution reference|	(see note at right)	The result is substituted in from the substitution definition. It could be text, an image, a hyperlink, or a combination of these and others.
footnote reference [1]_	footnote reference [1]	See Footnotes.
citation reference [CIT2002]_	citation reference [CIT2002]	See Citations.
http://docutils.sf.net/	http://docutils.sf.net/	A standalone hyperlink.

Asterisk, backquote, vertical bar, and underscore are inline delimiter characters. Asterisk, backquote, and vertical bar act like quote marks; matching characters surround the marked-up word or phrase,

也可以参考以下手册:

- Sphinx-reStructuredText Primer
- Restructured Text (reST) and Sphinx CheatSheet
- reStructuredText 入门
- reStructuredText Markup Specification
- `sphinx_rtd_theme` 的文档中使用了 `reStructuredText` 的大量语法, 可以从 这里 下载其文档源码来学习相关指令.

下面进行基本语法介绍.

Note: *OmniMarkupPreviewer* 有时预览效果不好, 可以使用它做个大致预览, 然后可以使用其它工具如 Sphinx. 此外, Sphinx 对 reST 进行了一些扩展, 请参考 Sphinx 手册.

2.3.4.1 章节 (Section Structure)

章节头部 (参考) 由下线 (也可有上线) 和包含标点的标题组合创建, 其中下线要至少等于标准文本的长度, 如:

```
======
Title
======

Subtitle1
---------

SubSubtitle
++++++++++

Subtitle2
----------
```

通常没有专门的符号表示标题的等级, 但是对于 Python 文档, 可以使用如下约定:

- # 下划线及上划线表示部分
- * 下划线及上划线表示章节
- = 下划线表示小章节
- - 下划线表示子章节
- ^ 下划线表示子章节的子章节
- " 下划线表示段落

在 Markdown 中, 用 # , * 的多少来代表标题等级, 如:

```
# Title
## Subtitle1
```

(continues on next page)

(continued from previous page)

```
### SubSubtitle
## Subtitle2
```

2.3.4.2 段落 (Paragraphs)

段落由空白行分割, 且应左对齐, 与 Markdown 相同; 在 reST 中, 缩进的段落意味着引用, 这在 Markdown 中是通过标记符号 > 实现的.

如:

```
这是第一段

这是第二段
这个还是第二段
```

被渲染成:

这是第一段

这是第二段这个还是第二段

2.3.4.3 行内标记 (Inline Markup)

reST 文本	解析渲染结果	注解
`*emphasis*`	*emphasis*	通常渲染成斜体, 与 Markdown 相同
`**emphasis**`	**emphasis**	通常渲染成粗体, 与 Markdown 相同
`` `interpreted text` ``	*interpreted text*	强调解释.
`` ``inline literal`` ```	`inline literal`	常用于行内代码, 与 Markdown 相同
A `:sub:`xxx``	A $_{xxx}$	下标 (subscript)
A `:sup:`xxx``	A ^{xxx}	上标 (superscript)
`:guilabel:`Action``	*Action*	GUI labels
`:kbd:`Ctrl+Shift``	`Ctrl+Shift`	Key-bindings
`:menuselection:`A-->B-->C``	*A→B→C*	菜单选择

Warning: 对于行内标记, 标记前后要留有至少一个空格. 如 `你好 * 我没变斜 * 你好` –> 你好 * 我没变斜 * 你好, 正确为: `你好 * 我变斜了 * 你好` –> 你好 我变斜了你好, 或 `你好\ * 我变斜了 *\你好` –> 你好我变斜了你好.

2.3.4.4 列表 (Lists)

```
无序列表使用 ``-`` , ``*`` , ``+`` 来标记:
```

(continues on next page)

(continued from previous page)

```
- 无序列表第一项
- 无序列表第二项

有序列表使用 ``num.`` 来标记:

1. 有序列表第一项
2. 有序列表第二项

自动编号列表必须使用 ``#.`` 来标记:

#. 自动编号的列表第一项
#. 自动编号的列表第二项

这是一个定义列表:

term
    术语定义必须缩进

    可以包含多个段落

next term
    术语描述

下面是一个嵌套列表, 每一级别向右缩进一次, 同级别缩进应相同:

1. 有序列表第一项
    * 无序列表第一项
    * 无序列表第二项
#. 有序列表第二项
    + 无序列表第一项
    + 无序列表第二项
```

将被渲染成:

无序列表使用 -, *, + 来标记:

- 无序列表第一项
- 无序列表第二项

有序列表使用 num. 来标记:

1. 有序列表第一项
2. 有序列表第二项

自动编号列表必须使用 #. 来标记:

1. 自动编号的列表第一项
2. 自动编号的列表第二项

这是一个定义列表:

term 术语定义必须缩进

可以包含多个段落

next term 术语描述

下面是一个嵌套列表, 每一级别向右缩进一次, 同级别缩进应相同:

1. **有序列表第一项**
    - 无序列表第一项
    - 无序列表第二项
2. **自动编号列表第二项**
    - 无序列表第一项
    - 无序列表第二项

2.3.4.5 源代码 (Source Code)

标记符号 `::` 紧接一空白行, 然后紧跟代码, 整个代码文本块必须缩进 (同所有的段落一样, 使用空白行和周围文本完成分隔), 如:

```
::

    some codes
    some codes
    some codes

没有缩进, 这里不是代码, 是正常段落!
```

将被渲染成:

```
some codes
some codes
some codes
```

没有缩进, 这里不是代码, 是正常段落!

此外, 高级的代码高亮功能可是使用 `.. code-block::` , 举例:

```
.. code-block:: python
   :caption: Code Blocks can have captions.
   :linenos:
   :emphasize-lines: 3,5

   def some_function():
       interesting = False
       print 'This line is highlighted.'
       print 'This one is not...'
       print '...but this one is.'
```

被渲染成:

Listing 5: Code Blocks can have captions.

```
def some_function():
    interesting = False
    print 'This line is highlighted.'
    print 'This one is not...'
    print '...but this one is.'
```

2.3.4.6 侧边栏 (Sidebar)

```
.. sidebar:: 这是一个侧边栏

    这是一个侧边栏，可以放入代码，也可以放入图像代码等等，它下面可以是文字，图像，代码等等，如本例中下面是一段文字.

冬日，在暖暖的午后，泡上一杯茶，随便拿起一本书，凑到阳光跟前，是何等的惬意与享受……

风虽然不大，但走在路上，鼻子冷的刺骨的疼；而阳光却那么地温热，温热地忍不住想和她亲吻。

我泡上一杯碧螺春，从书架上随便拿起一本书，走向靠窗的位置，凑到阳光面前，任由她吻着我的脸，就像吻着自己的情人，这感觉美好的让你忘却了所有的烦恼。

也许是身边暖气的缘故，空气的影子，映衬到桌子上、书纸上。影影绰绰如月下之花影，飘飘忽忽如山间之云气，生生腾腾如村落之炊烟，荡荡漾漾如湖面之微波，似乎在这图书馆的这一隅便可看尽天地间的朴素与祥和。
```

被渲染为:

这是一个侧边栏

这是一个侧边栏, 可以放入代码, 也可以放入图像代码等等, 它下面可以是文字, 图像, 代码等等, 如本例中下面是一段文字.

冬日，在暖暖的午后，泡上一杯茶，随便拿起一本书，凑到阳光跟前，是何等的惬意与享受……

风虽然不大，但走在路上，鼻子冷的刺骨的疼；而阳光却那么地温热，温热地忍不住想和她亲吻。

我泡上一杯碧螺春，从书架上随便拿起一本书，走向靠窗的位置，凑到阳光面前，任由她吻着我的脸，就像吻着自己的情人，这感觉美好的让你忘却了所有的烦恼。

也许是身边暖气的缘故，空气的影子，映衬到桌子上、书纸上。影影绰绰如月下之花影，飘飘忽忽如山间之云气，生生腾腾如村落之炊烟，荡荡漾漾如湖面之微波，似乎在这图书馆的这一隅便可看尽天地间的朴素与祥和。

2.3.4.7 表格 (Tables)

支持两种表格. 一种是 网格表格, 可以自定义表格的边框. 如下

```
+------------------------+----------+----------+----------+
| Header row, column 1   | Header 2 | Header 3 | Header 4 |
| (header rows optional) |          |          |          |
+========================+==========+==========+==========+
| body row 1, column 1   | column 2 | column 3 | column 4 |
+------------------------+----------+----------+----------+
| body row 2             | ...      | ...      |          |
+------------------------+----------+----------+----------+
```

将被渲染成:

Header row, column 1 (header rows optional)	Header 2	Header 3	Header 4
body row 1, column 1	column 2	column 3	column 4
body row 2	…	…	

简单表格书写简单, 但有一些限制: 需要有多行, 且第一列元素不能分行显示, 如下:

```
=====  =====  =======
A      B      A and B
=====  =====  =======
False  False  False
True   False  False
False  True   False
True   True   True
=====  =====  =======
```

将被渲染成:

A	B	A and B
False	False	False
True	False	False
False	True	False
True	True	True

Tip: 如果你使用”*Restructured Text (RST) Snippets*”, 即 `sublime-rst-completion`, 那么表格的制作将变得极为简单, 如下, 更多内容参考 sublime-rst-completion :

There is a particular *magic* expansion for tables. Here is how it works:

1. 创建表格提纲, 用一个或多个空格分割列:

```
This is paragraph text *before* the table.

Column 1  Column 2
Foo  Put two (or more) spaces as a field separator.
```

(continues on next page)

(continued from previous page)

```
Bar  Even very very long lines like these are fine, as long as you do not put␣
↪in line endings here.

This is paragraph text *after* the table.
```

2. 把光标放在要转化成表格的内容里

3. 按下 `Ctrl + T, Enter` (Linux or Windows) or `Super + Shift + T, Enter` (Mac). 将会自动格式化成表格:

```
This is paragraph text *before* the table.

+----------+-------------------------------------------------------+
| Column 1 | Column 2                                              |
+==========+=======================================================+
| Foo      | Put two (or more) spaces as a field separator.        |
+----------+-------------------------------------------------------+
| Bar      | Even very very long lines like these are fine, as long  |
|          | as you do not put in line endings here.               |
+----------+-------------------------------------------------------+

This is paragraph text *after* the table.
```

现在假设你想增加某一单元格内容:

```
+----------+-------------------------------------------------------+
| Column 1 | Column 2                                              |
+==========+=======================================================+
| Foo is longer now     | Put two (or more) spaces as a field separator.          |
+----------+-------------------------------------------------------+
| Bar      | Even very very long lines like these are fine, as long  |
|          | as you do not put in line endings here.               |
+----------+-------------------------------------------------------+
```

按下同样的快捷键, 表格结构会自动调整:

```
+------------------+-------------------------------------------------------+
| Column 1         | Column 2                                              |
+==================+=======================================================+
| Foo is longer now | Put two (or more) spaces as a field separator.        |
+------------------+-------------------------------------------------------+
| Bar              | Even very very long lines like these are fine, as long |
|                  | as you do not put in line endings here.               |
+------------------+-------------------------------------------------------+
```

2.3.4.8 直接标记 (Explicit Markup)

直接标记用于处理 reST 中的特殊内容, 如脚注, 高亮段落, 注释以及通用指令.

直接标记的标记符号是: 顶格的两个句点紧跟一个空格 `..` , 其后会紧跟直接标记对象, 如 `.. NOTE:: some notes` 将对应一个注解, `.. image:: imagepath` 对应一幅图像, 会被渲染成 `imagepath` 所表示的图像.

如果紧跟的是普通文本, 相当于代码注释或者叫评论, 如 `.. some comments` 则不会渲染显示, 又如:

```
.. 这是一个注释, 你只能在源码中看到我, 我不会被渲染出来.
```

渲染结果:

可以通过缩进产生多行注释:

```
..
   这整个缩进块都是
   一个注释.
   你只能在源码中看到我们, 我们不会被渲染出来

   仍是一个评论.

你可以看到我, 我不是注释.
```

渲染结果:

你可以看到我, 我不是注释.

Note: 下面涉及的很多语法, 都和直接标记".. " 有关, 如脚注, 引文, 超链接, 指令, 注释等等.

2.3.4.9 指令 (Directives)

官方文档的描述是:

> Directives are indicated by an explicit markup start (".. ") followed by the directive type, two colons, and whitespace (together called the "directive marker"). Directive types are case-insensitive single words (alphanumerics plus isolated internal hyphens, underscores, plus signs, colons, and periods; no whitespace). Two colons are used after the directive type for these reasons:

即指令由 直接标记 + 指令类型 + 两个冒号 + 空格 (多余 0 个) 组成指令标记 ("directive marker"), 指令类型是大小写不敏感的单个单词 (字母数字加内连字符、下划线、加号, 冒号, 句点；不含空格). 关于为什么使用两个冒号, 可以参考官方文档 Directives .

指令是 reST 的又一个扩展机制, Sphinx 大量使用了指令, 支持的指令很多, 这里仅介绍常用的几个, 请自行查阅手册:

See also:

Sphinx 中的 reST 教程中的 指令部分: http://www.sphinx-doc.org/en/stable/rest.html#directives .

数学

数学公式指令 `math`

支持 LaTex 数学语法, 以及公式引用 (通过 :eq:`x x x` 引用), 如我是下面的带标签公式 (2.2) , 示例:

```
行内公式 :math:`\alpha > \beta` :

Display 公式:

.. math::

    n_{\mathrm{offset}} = \sum_{k=0}^{N-1} s_k n_k

带标签公式:

.. math::
   :label: This is a label

    n_{\mathrm{offset}} = \sum_{k=0}^{N-1} s_k n_k

多行公式:

.. math::

   (a + b)^2 = a^2 + 2ab + b^2

   (a - b)^2 = a^2 - 2ab + b^2

对齐多行公式:

.. math::

   (a + b)^2  &=  (a + b)(a + b) \\
              &=  a^2 + 2ab + b^2
```

将被渲染成:

行内公式 $\alpha > \beta$:

Display 公式:

$$n_{\mathrm{offset}} = \sum_{k=0}^{N-1} s_k n_k \tag{2.1}$$

带标签公式:

$$n_{\mathrm{offset}} = \sum_{k=0}^{N-1} s_k n_k \tag{2.2}$$

多行公式:

$$\begin{aligned}(a + b)^2 &= a^2 + 2ab + b^2 \\ (a - b)^2 &= a^2 - 2ab + b^2\end{aligned} \tag{2.3}$$

对齐多行公式:

$$\begin{aligned}(a+b)^2 &= (a+b)(a+b) \\ &= a^2+2ab+b^2\end{aligned} \tag{2.4}$$

Hint:

- 当使用 MathJax 对数学公式渲染时, 可能不能达到预期效果, 请参考*Sphinx 数学支持* (page 33) ;
- 关于 Sphinx 中的如何使用 Katex 渲染公式, 参见*Sphinx 数学支持* (page 33) .
- 关于 Sphinx 中的如何使用定义定理等环境, 参见*sphinxcontrib-proof* (page 21) ;

See also:

更多内容参考 reST 手册, 这里 . 关于 Sphinx 中对数学公式的支持部分, 请参考 Math support in Sphinx . 或*Sphinx 数学支持* (page 33) .

图像

- 图像指令 `image`

即通过 `.. image:: imagepath` 实现插入图像:

```
.. image:: picture.jpeg
   :height: 100px
   :width: 200 px
   :scale: 50 %
   :alt: 对于不能显示图片的时候, 显示这些文字
   :align: right
```

下面通过 `.. image:: ../_static/figs/mkdocs/insertimage.png` 插入一幅图像:

- 图像指令 `figure`, 包含图例和标题, 可以像下面这样使用:

```
.. figure:: picture.png
   :scale: 50 %
   :alt: map to buried treasure

   This is the caption of the figure (a simple paragraph).

   The legend consists of all elements after the caption.  In this
   case, the legend consists of this paragraph
```

下面通过如下命令插入一幅图像:

```
.. figure:: ../_static/figs/mkdocs/insertfigure.png
   :width: 1024
   :scale: 50%
   :align: center

   大唐芙蓉园 - This is the caption of the figure (a simple paragraph).

    The legend consists of all elements after the caption.  In this
    case, the legend consists of this paragraph.
```

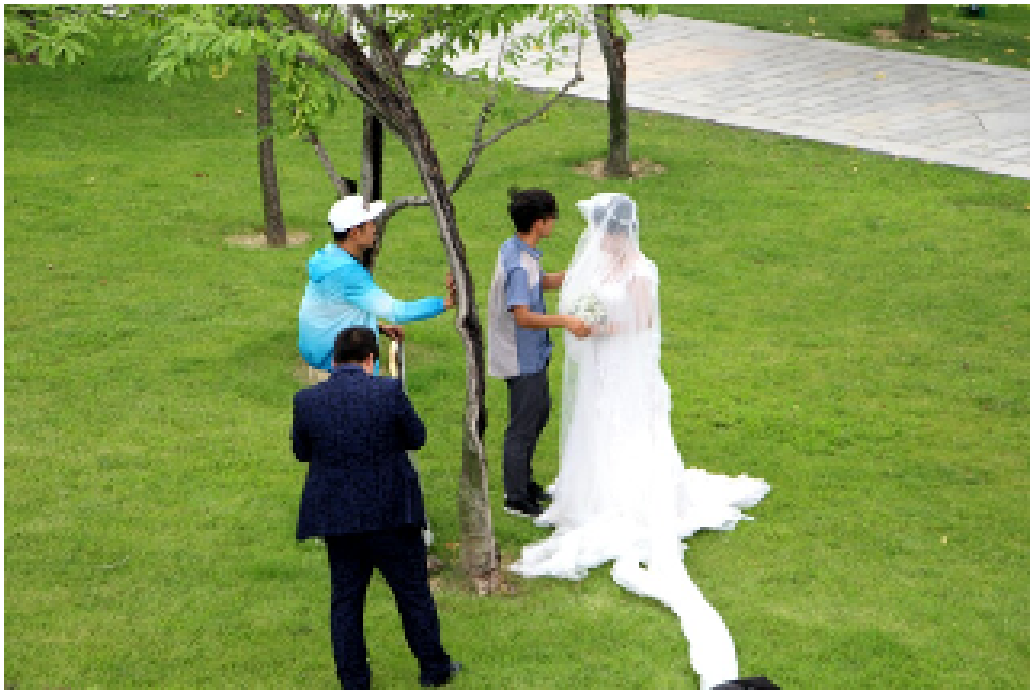


Fig. 3: 大唐芙蓉园 - This is the caption of the figure (a simple paragraph).

The legend consists of all elements after the caption. In this case, the legend consists of this paragraph.

Tip: 在 Markdown 中, 插入图片很简单, `![caption](imagepath)` 即可.

提示警告类

有很多: `tip`, `note`, `hint`, `danger`, `error`, `warning`, `caution`, `attention` , `important`

reST 标记代码:

```
.. tip:: This is a tip

.. note:: This is a note

.. hint:: This is a hint

.. danger:: This is a danger

.. error:: This is an error

.. warning:: This is a warning

.. caution:: This is a caution

.. attention:: This is an attention
```

(continues on next page)

(continued from previous page)

```
.. important:: This is an important

.. seealso:: This is seealso
```

被渲染成:

Tip: This is a tip

Note: This is a note

Hint: This is a hint

Danger: This is a danger

Error: This is an error

Warning: This is a warning

Caution: This is a caution

Attention: This is an attention

Important: This is an important

See also:

This is seealso

提示警告类

额外的主体元素

- `contents <table-of-contents>` (本地, 仅是当前文件的内容表格)
- `container` (自定义容器, 用来生成 HTML 的 `<div>`)
- `rubric` (和文档章节无关的标题)
- `topic` (高亮显示的主体元素)
- `parsed-literal` (支持内联标记的斜体模块)
- `epigraph` (可选属性行的摘要模块)
- `highlights` (有自己的类属性的摘要模块)
- `compound` (复合段落)

专用表格

- `table` (有标题的表格)
- `csv-table` (CSV 自动生成表格)
- `list-table` (列表生成的表格)

专用指令

- `raw` (包含原始格式的标记)
- **`include` (包含 reStructuredText 标记的文件)** – 在 Sphinx 中, 如果包含绝对文件路径, 指令会以源目录地址做为参照
- `class` (将类属性指派给下一个元素)

HTML 特性

- `meta` (生成 HTML `<meta>` 标签)
- `title` (覆盖文档标题)
- 影响标记:
- `default-role` (设置新的默认角色)
- `role` (创建新的角色)

Sphinx 新增指令可查阅 Sphinx Markup Constructs .

2.3.4.10 超链接 (Hyperlinks)

外部链接

使用 `Link text <http://example.com/>`_ 来表示超链接, 将被渲染成 Link text 如果文字本身就是链接, 那不用作任何标记, 解析器可以自动将链接和邮箱地址转换为超链接.

也可以单独定义链接目标用引用:

```
This is a paragraph that contains `a link`_.

.. _a link: http://example.com/
```

渲染成:

This is a paragraph that contains a link.

Tip: 在 Markdown 中, 插入链接很简单, `[Link Text](http://example.com/)` 即可.

内部链接

1. 首先需要在标题, 图像, 表等对象前放置一个标签 `.. _label:`, 比如我在上幅图上方放置了一个标签 `figure-datangfurongyuan` , 注意空白行:

   ```
   .. _figure-datangfurongyuan:

   .. figure:: ../_static/figs/mkdocs/insertfigure.png
   ...
   ```

2. 引用. 通过 `:ref:`label`` , 接上例, 使用 `:ref:`figure-datangfurongyuan`` 引用, 那么渲染结果为: 大唐芙蓉园 - *This is the caption of the figure (a simple paragraph).* (page 51), 点击会跳到图像位置.

其它的交叉引用, 请参考手册 交叉引用部分,

2.3.4.11 脚注 (Footnotes)

包含两步:

- 在文档底部放置脚注主体, 以 `rubric` 指令标示:

  ```
  .. rubric:: Footnotes

  .. [#name] 这里是脚注内容
  ```

- 在需要插入脚注的地方插入脚注名 `[#name]`

其中, 使用 `[#name]_` 可以实现自动编号, 当然你也可以使用数字来指示确定的脚注编号 `[1]_`.

举例:

```
我后面插入了一个自编号的脚注 [#f1]_ , 后面又跟了一个手动编号的脚注 [2]_ , 后面还跟着一个自动编号
的 [#fn]_ .

.. rubric:: Footnotes

.. [#f1] 我是自编号脚注 1
.. [2] 我是手动编号脚注 2
.. [#fn] 我是自编号脚注 3
```

我后面插入了一个自编号的脚注[1] , 后面又跟了一个手动编号的脚注[2] , 后面还跟着一个自动编号的[3] .

2.3.4.12 引文 (Citations)

Sphinx 支持标准的 reST 引文, 此外, 在 Sphinx 里, 所有的引文都是全局的, 所有文件都能引用任意的文献, 像下面这样使用引文:

```
Lorem ipsum [Ref]_ dolor sit amet.

.. [Ref] Book or article reference, URL or whatever.
```

Lorem ipsum [Ref] dolor sit amet.

引文的用法与脚注用法相似, 但标签不是数字, 也不是以 # 开头.

Hint: 关于 Sphinx 中如何使用 bibtex 参考文献, 请参考*sphinxcontrib-bibtex* (page 24) 小结.

2.3.4.13 替换 (Substitutions)

reST 支持替换, 你可以像下面这样使用替换:

首先定义替换操作:

```
.. |name| replace:: 替换文本
```

或者:

```
.. |caution| image:: warning.png
              :alt: Warning!
```

然后在需要替换的地方使用 |name| , 或者 |caution|

示例 1:

reST 源码:

[1] 我是自编号脚注 1
[2] 我是手动编号脚注 2
[3] 我是自编号脚注 3

```
你看到了吗? 第二个单词 word |word| !

.. |word| replace:: 替换成我了
```

被渲染成:

你看到了吗? 第二个单词 word 替换成我了!

示例 2

```
" 大唐芙蓉园-婚纱照" 本来是个短语, 使用 | 大唐芙蓉园-婚纱照 | 会被替换成图像!

.. | 大唐芙蓉园-婚纱照 | image:: ../_static/figs/mkdocs/insertfigure.png
                          :alt: 大唐芙蓉园-婚纱照!
```

被渲染成:

“大唐芙蓉园-婚纱照”本来是个短语, 使用会被替换成图像!

你可以进行任意的替换!

2.3.4.14 Sphinx 扩展指令

有关 Sphinx 的扩展指令, 如 **目录树**, **术语**, **特定语法着色**等, 请移步本手册*Sphinx 重要扩展介绍* (page 18) 指南, 或参考 Sphinx 官方手册 The TOC tree 部分.

2.4 版本控制系统 Git 教程

什么是版本控制系统 (Version Control System)? 版本这一词大家都不陌生, 那么想必也能猜出来版本控制系统的含义: 能够自动管理文件版本的系统. 这句话似乎是废话, 举个例子来说, 平时在写文章或代码时, 经常会复制多个文件以保存不同版本的修改, 而且你还的重命名文件, 以告诉自己这一版本在哪里修改了, 是不是很繁琐, 不仅如此, 你还要存储不同的文件版本, 那么版本控制系统就是帮你完成这件事, 比如, 它可能会像下面这样自动记录每个版本的变动:

版本	用户	说明	日期
1	张三	删除了软件服务条款 5	7/12 10:38
2	张三	增加了 License 人数限制	7/12 18:09
3	李四	财务部门调整了合同金额	7/13 9:51
4	王五	延长了免费升级周期	7/14 15:17

你可能还希望恢复不同的版本状态时的文件, 可以的, 就是说你有后悔药可以吃的. 怎么样? 是不是突然觉得自己要解放了, 不过你得先学会使用一款版本控制系统软件才行. 下面开始吧!!!

此教程部分内容参考了 Git 教程 - 廖雪峰的官方网站 号称最浅显易懂的教程, 是吗? 是的, 至少我觉得是.

2.4.1 What is Git?

> Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git (读音/gɪt/) 是一个免费开源的分布式版本系统, 无论项目大小, 都能游刃有余地管理项目的版本. 我们还希望它是易学的, 是的, 它也确实如此! 右图为 Git 的 logo.

Git 与 GitHub: 我们知道 Git 是一个免费开源的分布式版本系统, 跟 svn、cvs 是同级的概念, 那么 GitHub 是什么呢? 是一个免费的远程仓库, 顾名思义就是用来存储你的文件的, 给用户提供 git 服务. 这样你就不用自己部署 git 系统, 直接用注册个账号, 用他们提供的 git 服务就可以. 右图为其吉祥物 Octocat .

Fig. 4: Logo of Git

2.4.2 Why Git?

- 免费开源
- 分布式
- 高效高速
- 简单易学

2.4.3 How to

下面给出一些学习参考资料:

- Git 教程 - 廖雪峰的官方网站 自称最浅显易懂的 Git 教程.
- Pro Git : 由 Scott Chacon and Ben Straub 写的一本书, 可以在线浏览, 也可以下载 PDF, epub, mobi, html 版本查看, 被翻译成多种语言, 简体中文版在 这里 查看与下载.

Fig. 5: Github 吉祥物 Octocat

2.4.3.1 安装 Git

Windows

可以从 Git 官网 下载 Windows 版 Git 安装包 https://git-scm.com/download/win , 根据提示安装即可.

此外, *Git for Windows* 是一个轻量级且具有 Git SCM 全特性的 Git 软件, 可以从 https://git-for-windows.github.io 下载安装.

> Git for Windows focuses on offering a lightweight, native set of tools
> that bring the full feature set of the Git SCM to Windows while providing appropriate user interfaces
> for experienced Git users and novices alike.

Linux

Git 在 ubuntu 上的安装很简单, 首先终端 (快捷键 `Ctrl + Alt + T`) 输入 `git` , 查看是否已安装, 如果没有, 对于 Ubuntu 系统通过如下命令安装即可:

```
sudo apt-get install git
```

其它系统可以根据 Git 官网 提示下载安装.

2.4.3.2 创建本地仓库

2.4.3.3 版本库管理

2.4.3.4 远程仓库

2.4.3.5 分支管理

2.4.3.6 标签话管理

2.4.3.7 自定义 Git

2.5 参考文献

CHAPTER 3

进阶指南

3.1 简介

该部分为进阶教程，通过该部分教程学习，你学会从代码注释 API 生成函数手册！

待完成……敬请期待……

3.2 API 手册制作指南

通过该部分教程学习，你学会从代码注释 API 生成函数手册！

待完成……敬请期待……

3.2.1 一个例子

3.2.1.1 撰写代码和文档

以本人写的一个 Python 工具代码为例, 在 这里 下载源码.

3.2.1.2 初始化 Sphinx 工程

0. 在源码根目录, 新建 *docs* 文件夹, 进入该文件夹
1. 终端输入: `sphinx-quickstart` 启动 Sphinx
2. 设置项目名称为 *PyTool* , 作者名, 工程发布版本 (Project release) 等
3. 设置项目文档语言等, 类似如下

```
PS D:\ws\github\pytool\docs> sphinx-quickstart.exe
Welcome to the Sphinx 1.8.1 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Selected root path: .

You have two options for placing the build directory for Sphinx output.
Either, you use a directory "_build" within the root path, or you separate
"source" and "build" directories within the root path.
> Separate source and build directories (y/n) [n]: y

Inside the root directory, two more directories will be created; "_templates"
for custom HTML templates and "_static" for custom stylesheets and other static
files. You can enter another prefix (such as ".") to replace the underscore.
> Name prefix for templates and static dir [_]:

The project name will occur in several places in the built documentation.
> Project name: PyTool
> Author name(s): Zhi Liu
> Project release []: 1.0

If the documents are to be written in a language other than English,
you can select a language here by its language code. Sphinx will then
translate text that it generates into that language.

For a list of supported codes, see
http://sphinx-doc.org/config.html#confval-language.
> Project language [en]:

The file name suffix for source files. Commonly, this is either ".txt"
or ".rst".  Only files with this suffix are considered documents.
> Source file suffix [.rst]:

One document is special in that it is considered the top node of the
"contents tree", that is, it is the root of the hierarchical structure
of the documents. Normally, this is "index", but if your "index"
document is a custom template, you can also set this to another filename.
> Name of your master document (without suffix) [index]:
Indicate which of the following Sphinx extensions should be enabled:
> autodoc: automatically insert docstrings from modules (y/n) [n]: y
> doctest: automatically test code snippets in doctest blocks (y/n) [n]: y
> intersphinx: link between Sphinx documentation of different projects (y/n) [n]: y
> todo: write "todo" entries that can be shown or hidden on build (y/n) [n]: y
> coverage: checks for documentation coverage (y/n) [n]: y
> imgmath: include math, rendered as PNG or SVG images (y/n) [n]: y
> mathjax: include math, rendered in the browser by MathJax (y/n) [n]: y
```

(continues on next page)

(continued from previous page)

```
> ifconfig: conditional inclusion of content based on config values (y/n) [n]: y
> viewcode: include links to the source code of documented Python objects (y/n)␣
↪[n]: n
> githubpages: create .nojekyll file to publish the document on GitHub pages (y/n)␣
↪[n]: y
Note: imgmath and mathjax cannot be enabled at the same time. imgmath has been␣
↪deselected.

A Makefile and a Windows command file can be generated for you so that you
only have to run e.g. `make html' instead of invoking sphinx-build
directly.
> Create Makefile? (y/n) [y]: y
> Create Windows command file? (y/n) [y]: y

Creating file .\source\conf.py.
Creating file .\source\index.rst.
Creating file .\Makefile.
Creating file .\make.bat.

Finished: An initial directory structure has been created.

You should now populate your master file .\source\index.rst and create other␣
↪documentation
source files. Use the Makefile to build the docs, like so:
   make builder
where "builder" is one of the supported builders, e.g. html, latex or linkcheck.
```

3.2.1.3 配置 Sphinx 工程

1. 添加库路径.

打开 *source/conf.py* 文件, 可以看到:

> # If extensions (or modules to document with autodoc) are in another directory, # add these directories to sys.path here. If the directory is relative to the # documentation root, use os.path.abspath to make it absolute, like shown here. #sys.path.insert(0, os.path.abspath('.'))

因而在 *conf.py* 中添加 `sys.path.insert(0, os.path.abspath('../../'))`

Hint: 如果你生成的 html 文档中没有注释, 只有包名, 那么很有可能是没找到包, 可参见问题解决部分.

3.2.1.4 生成 API 注释文档

本部分使用 `sphinx-apidoc` 命令自动从代码中提取注释并生成 *rst* 文件.

回退到源码工程目录, 终端执行命令: `sphinx-apidoc -o ./docs/source/ ./` 将在 *source* 文件夹创建各模块的注释文档 (".rst"格式).

3.2.1.5 编译生成 API 手册

接着打开 *index.rst* 文件, 在其中添加 *module.rst* 文件, 如下:

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

.. toctree::
   :maxdepth: 2
   :caption: Contents:

   modules

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

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

可以随意打开一个模块的 *rst* 文件, 如下:

```
pytool.file package
===================

Submodules
----------

pytool.file.binfile module
--------------------------

.. automodule:: pytool.file.binfile
    :members:
    :undoc-members:
    :show-inheritance:

pytool.file.copy module
-----------------------

.. automodule:: pytool.file.copy
    :members:
    :undoc-members:
    :show-inheritance:
```

可以看到文件中并没有注释, 那么怎么生成文档呢? 原来是在 `build` 时, Sphinx 才提取文档注释, 生成文档.

Hint: 生成的模块的 *rst* 文件中的 `.. automodule::` 用于自动抽取文档注释.

然后像构建普通文档一样编译即可. 当然你还可以修改 *conf.py* 文件, 以修改文档主题等等.

生成的文档示例如下图所示:

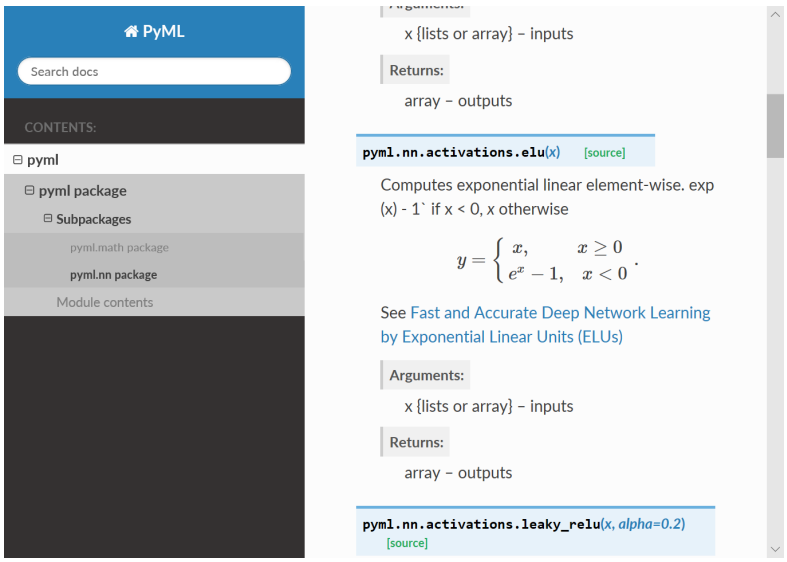


Fig. 1: 自动生成的文档示例

使用 Sphinx 自动从 PyML 包中的注释, 提取并生成文档, 注释可以使用 reStructedText 语言撰写.

3.2.2 文档注释风格支持

有两种:

- google 风格
- numpy 风格

如果代码中包含两种风格的注释, 可以在 *conf.py* 中的 *extensions* 处添加扩展 `sphinx.ext.napoleon` 即可.

3.2.3 问题解决

3.2.3.1 生成的文档无注释

如果你生成的 html 文档中没有注释, 只有包名, 那么很有可能是没找到包, 或者 **代码中用到的库没有装**. 在 *conf.py* 中添加 `sys.path.insert(0, os.path.abspath('../../'))` 设置好路径, 并安装缺失的相应模块即可.

3.2.3.2 注释中不显示公式

在注释起始符前加 `r` , 如:

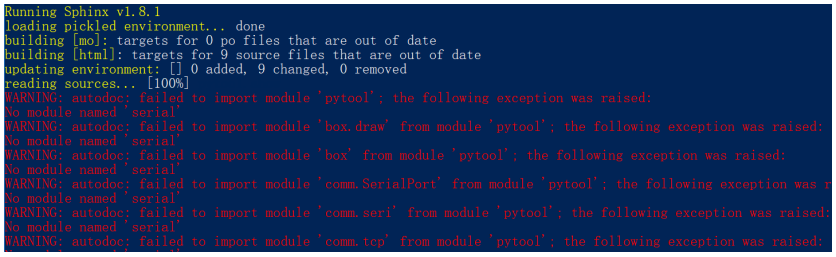


Fig. 2: no module error

no module error reported by Sphinx

```
r"""Computes tanh of `x` element-wise.

 Specifically, :math:`y = {\rm tanh}(x) = {{e^{2x} - 1} \over {e^{2x} + 1}}`.

 Arguments:
     x {lists or array} -- inputs

 Returns:
     array -- outputs

 """
```

3.3 项目文档制作指南

待完成……敬请期待……

CHAPTER 4

名词术语

Architecture 体系结构

Board Support Package 板级支持包 (Board Support Package, BSP) 是介于主板硬件和操作系统中驱动层程序之间的一层, 一般认为它属于操作系统一部分, 主要是实现对操作系统的支持, 为上层的驱动程序提供访问硬件设备寄存器的函数包, 使之能够更好的运行于硬件主板. 在嵌入式系统软件的组成中, 就有 BSP. BSP 是相对于操作系统而言的, 不同的操作系统对应于不同定义形式的 BSP, 例如 VxWorks 的 BSP 和 Linux 的 BSP 相对于某一 CPU 来说尽管实现的功能一样, 可是写法和接口定义是完全不同的, 所以写 BSP 一定要按照该系统 BSP 的定义形式来写（BSP 的编程过程大多数是在某一个成型的 BSP 模板上进行修改）. 这样才能与上层 OS 保持正确的接口, 良好的支持上层 OS.

Cross Compiling 简单地说, 就是在一个平台上生成另一个平台上的可执行代码. 包含两个概念: 体系结构 (Architecture) 和操作系统 (Operating System). 同一体系结构可以运行不同的操作系统；同样, 同一操作系统也可以在不同的体系结构上运行. 举例来说, 我们常说的 x86 Linux 平台实际上是 Intel x86 体系结构和 Linux for x86 操作系统的统称；而 x86 WinNT 平台实际上是 Intel x86 体系结构和 Windows NT for x86 操作系统的简称. 对于嵌入式 Linux 开发中的交叉编译, 一般是指, 在 Linux PC 机上, 利用交叉编译器 *arm-linux-gcc* , 编译生成 Linux ARM 上的可执行程序.

DMIPS 即 Dhrystone Million Instructions executed Per Second, Dhrystone 是一种整数运算测试程序, DMIPS 表示了在 Dhrystone 这样一种测试方法下的 MIPS. 举例来说, 如果某处理器性能为: 2DMIPS/MHZ, 那么对于主频为 1000MHz 即 1GHz 的处理器, 其计算能力为 2000DMIPS.

CPU 性能评估采用综合测试程序, 较流行的有 Whetstone 和 Dhrystone 两种. Dhrystone 主要用于测整数计算能力, 计算单位就是 DMIPS. 采用 Whetstone 主要用于测浮点计算能力, 计算单位就是 MFLOPS. DMIPS 只适宜于评估标量机, 不能用于评估向量机. 而 MFLOPS 则比较适用于衡量向量机的性能.

Embedded Operating System 嵌入式操作系统（Embedded Operating System, 简称：EOS）是指用于嵌入式硬件系统的操作系统. 嵌入式操作系统是一种用途广泛的系统软件, 通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等. 嵌入式操作系统负责

嵌入式系统的全部软、硬件资源的分配、任务调度, 控制、协调并发活动. 它必须体现其所在系统的特征, 能够通过装卸某些模块来达到系统所要求的功能. 目前在嵌入式领域广泛使用的操作系统有：嵌入式实时操作系统 μC/OS-II、嵌入式 Linux、Windows Embedded、VxWorks 等, 以及应用在智能手机和平板电脑的 Android、iOS 等.

Embedded Real-time Operation System 嵌入式实时操作系统 (Embedded Real-time Operation System, EROS) , 具备实时性的嵌入式操作系统.

GPU 图形处理单元 (Graphic Processing Unit, GPU)

GUI 图形用户界面（Graphical User Interface, 简称 GUI, 又称图形用户接口）是指采用图形方式显示的计算机操作用户界面.

Markup Language 标记语言（也称置标语言、标记语言、标志语言、标识语言）是一种将文本（Text）以及文本相关的其他信息结合起来, 展现出关于文档结构和数据处理细节的计算机文字编码. 与文本相关的其他信息（包括例如文本的结构和表示信息等）与原来的文本结合在一起, 但是使用标记（markup）进行标识. 当今广泛使用的标记语言是超文本标记语言（HyperText Markup Language, HTML）和可扩展标记语言（eXtensible Markup Language, XML）. 标记语言广泛应用于网页和网络应用程序. 标记最早用于出版业, 是作者、编辑以及出版商之间用于描述出版作品的排版格式所使用的.

MIPS 每秒百万条指令（Million Instructions executed Per Second，MIPS），用来计算同一秒内系统的处理能力，即每秒执行了多少百万条指令.

OpenCL 开放计算语言 (Open Computing Language, OpenCL), 是一个为异构平台编写程序的框架, 此异构平台可由 CPU、GPU、DSP、FPGA 或其他类型的处理器与硬件加速器所组成. OpenCL 由一门用于编写 kernels（在 OpenCL 设备上运行的函数）的语言（基于 C99）和一组用于定义并控制平台的 API 组成. OpenCL 提供了基于任务分区和数据分区的并行计算机制.

OpenCL 类似于另外两个开放的工业标准 OpenGL 和 OpenAL, 这两个标准分别用于三维图形和计算机音频方面. OpenCL 扩充了 GPU 图形生成之外的能力. OpenCL 由非盈利性技术组织 Khronos Group 掌管.

Processing System 处理系统 (Processing System, PS), 在 Xilinx Zynq 系列设备中, SOC 被分成 PS (ARM 部分) 与 PL (FPGA 部分).

Programmable Logic 可编程逻辑 (Programmable Logic, PL), 在 Xilinx Zynq 系列设备中, SOC 被分成 PS (ARM 部分) 与 PL (FPGA 部分).

Real Time multi-tasking Operation System 实时多任务操作系统 (Real Time multi-tasking Operation System, RTOS) . 在嵌入式应用中通常注重其实时性.

Stream Multiprocessor 流处理器 (Stream Multiprocessor, SM)

Stream Processor 流处理器 (Stream Processor, SP)

Thread Processor Cluster 线程处理器簇 (Thread Processor Cluster, TPC)

XADC 赛灵思模拟到数字转换器 (Xilinx Analogue to Digital Converter, XADC)

XDC 赛灵思设计约束 (Xilinx Design Constraints, XDC)

Xilinx Software Development Kit 赛灵思软件开发套件 (Xilinx Software Development Kit, SDK)

CHAPTER 5

Indices and tables

- genindex
- search

Bibliography

[Ref] Book or article reference, URL or whatever.

[1] *Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition*, volume, Nov 1993. doi:10.1109/ACSSC.1993.342465.

[2] Y. Wu and V. S. Batista. Matching-pursuit for simulations of quantum processes. *jcp* , 118:6720–6724, April 2003. doi:10.1063/1.1560636.