**软件目录结构规范**

@(学习)[python]

声明：此文是根据[Alex](http://www.cnblogs.com/alex3714)教学中整理而来。

* [软件目录结构规范](http://blog.csdn.net/ygqygq2/article/details/53838896#%E8%BD%AF%E4%BB%B6%E7%9B%AE%E5%BD%95%E7%BB%93%E6%9E%84%E8%A7%84%E8%8C%83)
* [为什么要设计好目录结构](http://blog.csdn.net/ygqygq2/article/details/53838896#1%E4%B8%BA%E4%BB%80%E4%B9%88%E8%A6%81%E8%AE%BE%E8%AE%A1%E5%A5%BD%E7%9B%AE%E5%BD%95%E7%BB%93%E6%9E%84)
  + [设计项目目录结构](http://blog.csdn.net/ygqygq2/article/details/53838896#1%E8%AE%BE%E8%AE%A1%E9%A1%B9%E7%9B%AE%E7%9B%AE%E5%BD%95%E7%BB%93%E6%9E%84)
  + [目录组织方式](http://blog.csdn.net/ygqygq2/article/details/53838896#2%E7%9B%AE%E5%BD%95%E7%BB%84%E7%BB%87%E6%96%B9%E5%BC%8F)

**1.为什么要设计好目录结构?**

**1.设计项目目录结构**

“项目目录结构”是属于”可读性和可维护性”的范畴，我们设计一个层次清晰的目录结构，就是为了达到以下两点:

* **可读性高**: 不熟悉这个项目的代码的人，一眼就能看懂目录结构，知道程序启动脚本是哪个，测试目录在哪儿，配置文件在哪儿等等。从而非常快速的了解这个项目。就像我们所使用的Linux系统一样，固定约定熟成的目录代表不同的功能等。
* **可维护性高**: 定义好组织规则后，维护者就能很明确地知道，新增的哪个文件和代码应该放在什么目录之下。这个好处是，随着时间的推移，代码/配置的规模增加，项目结构不会混乱，仍然能够组织良好。

所以，保持一个层次清晰的目录结构是有必要的。更何况组织一个良好的工程目录，其实是一件很简单的事儿。

**2.目录组织方式**

关于如何组织一个较好的Python工程目录结构，已经有一些得到了共识的目录结构。在Stackoverflow的这个问题上，能看到大家对Python目录结构的讨论。   
这里面说的已经很好了。   
假设你的项目名为foo, 比较建议的最方便快捷目录结构这样就足够了:

Foo/

|-- bin/

| |-- foo

|

|-- foo/

| |-- tests/

| | |-- \_\_init\_\_.py

| | |-- test\_main.py

| |

| |-- \_\_init\_\_.py

| |-- main.py

|

|-- conf/

| | --foo.conf

|-- docs/

| |-- abc.rst

|

|-- setup.py

|-- requirements.txt

|-- README

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

简要解释一下:

bin/: 存放项目的一些可执行文件，当然你可以起名script/之类的也行。   
foo/: 存放项目的所有源代码。(1) 源代码中的所有模块、包都应该放在此目录。不要置于顶层目录。(2) 其子目录tests/存放单元测试代码； (3) 程序的入口最好命名为main.py。   
conf/: 存放配置文件。   
docs/: 存放一些文档。   
setup.py: 安装、部署、打包的脚本。   
requirements.txt: 存放软件依赖的外部Python包列表。   
README: 项目说明文件。   
除此之外，有一些方案给出了更加多的内容。比如LICENSE.txt,ChangeLog.txt文件等。

**关于README的内容**   
这个应该是每个项目都应该有的一个文件，目的是能简要描述该项目的信息，让读者快速了解这个项目。   
它需要说明以下几个事项:

软件定位，软件的基本功能。   
运行代码的方法: 安装环境、启动命令等。   
简要的使用说明。   
代码目录结构说明，更详细点可以说明软件的基本原理。   
常见问题说明。

在软件开发初期，由于开发过程中以上内容可能不明确或者发生变化，并不是一定要在一开始就将所有信息都补全。但是在项目完结的时候，是需要撰写这样的一个文档的。

可以参考Redis源码中Readme的写法，这里面简洁但是清晰的描述了Redis功能和源码结构。

**关于requirements.txt和setup.py**   
*setup.py*

一般来说，用setup.py来管理代码的打包、安装、部署问题。业界标准的写法是用Python流行的打包工具setuptools来管理这些事情。这种方式普遍应用于开源项目中。不过这里的核心思想不是用标准化的工具来解决这些问题，而是说，一个项目一定要有一个安装部署工具，能快速便捷的在一台新机器上将环境装好、代码部署好和将程序运行起来。

setup.py可以将这些事情自动化起来，提高效率、减少出错的概率。”复杂的东西自动化，能自动化的东西一定要自动化。”是一个非常好的习惯。   
setuptools的文档比较庞大，刚接触的话，可能不太好找到切入点。学习技术的方式就是看他人是怎么用的，可以参考一下Python的一个Web框架，flask是如何写的: setup.py   
当然，简单点自己写个安装脚本（deploy.sh）替代setup.py也未尝不可。

*requirements.txt*

这个文件存在的目的是:

方便开发者维护软件的包依赖。将开发过程中新增的包添加进这个列表中，避免在setup.py安装依赖时漏掉软件包。

方便读者明确项目使用了哪些Python包。

这个文件的格式是每一行包含一个包依赖的说明，通常是flask>=0.10这种格式，要求是这个格式能被pip识别，这样就可以简单的通过 pip install -r requirements.txt来把所有Python包依赖都装好了。

*关于配置文件的使用方法*

注意，在上面的目录结构中，没有将conf.py放在源码目录下，而是放在conf/目录下。

很多项目对配置文件的使用做法是:

配置文件写在一个或多个python文件中，比如此处的conf.py。

项目中哪个模块用到这个配置文件就直接通过import conf这种形式来在代码中使用配置。

这种做法我不太赞同:

这让单元测试变得困难（因为模块内部依赖了外部配置）

另一方面配置文件作为用户控制程序的接口，应当可以由用户自由指定该文件的路径。

程序组件可复用性太差，因为这种贯穿所有模块的代码硬编码方式，使得大部分模块都依赖conf.py这个文件。

所以，更好的方式是，

模块的配置都是可以灵活配置的，不受外部配置文件的影响。

程序的配置也是可以灵活控制的。

能够佐证这个思想的是，用过nginx和mysql的同学都知道，nginx、mysql这些程序都可以自由的指定用户配置。

所以，不应当在代码中直接import conf来使用配置文件。上面目录结构中的conf.py，是给出的一个配置样例，不