

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 2.2 基础教程

这是第一篇  
[Django 简介 »](#)

django

Django 是用Python开发的一个免费开源的Web框架, 可以用于快速搭建高性能, 优雅的网站!

你一定可以学会, Django 很简单! 本教程一直在维护, 更新, 提供免费答疑, 免费邮件咨询。

从2015年01月18日写第一篇Django教程, 一直到现在, 每一篇教程由于新版 Django 一些改变, 或用户的反馈, 随时可能会进行更新完善, 可以在网站首页看到最近更新的情况。

截止2017年02月16日, 此Django教程已经有4400多条评论, 两年多以来, 几乎每天回复评论, 回复邮件。教程也根据评论进行了大量的改进, 包含大量的工程经验, 致力打造出中国最实用的 Django 教程。

我在2015年初时, 阅读了全部的 Django 英文的官方文档, 觉得国内比较好的Django学习资源不多, 所以决定写自己的教程。

Django 2.2 已经发布, 变动比较大, 不再兼容Python 2 版本, 本教程采用 **Python 3.7 + Django 2.2**。

如果想学习 Django 1.4 - Django 1.11 系列的 Django 1.x 的版本, 请[点击此处](#)。

本教程作者: 涂伟忠(未经同意,禁止转载!)

《Django开发从入门到实践》书籍出版啦! 基于 Python3 + Django 2.x

image.png

书籍购买: [淘宝](#)、[京东](#)、[当当](#)

这是一本写给初学者的书籍, 简介可以关注右侧的公众号了解。

遇到问题请直接回复对应的教程, 如果有截图, 附件等不方便可以发邮件到

[自强学堂答疑邮箱 tuweizhong#163.com](mailto:tuweizhong#163.com) (请把#换成@)

书籍读者交流 QQ群: 1020663804

### 除了本教程外的其它教程

自强学堂 学习分享 的文章: [Python 学习资源](#) 和 [Django 学习资源](#), 如果有更好的教程也可以在文章下推荐哦!

Django 是基于 Python, 所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

# 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包，所以你得知道一些 [Python](#) 基础知识。

2. 其次你最好有一些做网站的经验，懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入

学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

### 优雅的网址

用正则匹配网址，传递到对应函数，随意定义，如你所想！

### 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

### 缓存系统

与Memcache, Redis等缓存系统联用，更出色的表现，更快的加载速度。

### 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

[小额赞助，支持作者！](#)

这是第一篇

[Django 简介](#) »



零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包, 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 2.2 基础教程

这是第一篇

[Django 简介 »](#)

django

Django 是用[Python](#)开发的一个免费开源的Web框架，可以用于快速搭建高性能，优雅的网站！

你一定可以学会，Django 很简单！**本教程一直在维护，更新，提供免费答疑，免费邮件咨询。**

从2015年01月18日写第一篇Django教程，一直到现在，每一篇教程由于新版 Django 一些改变，或用户的反馈，随时可能会进行更新完善，可以在网站首页看到最近更新的情况。

截止2017年02月16日，此Django教程已经有4400多条评论，两年多以来，几乎每天回复评论，回复邮件。教程也根据评论进行了大量的改进，包含大量的工程经验，致力打造出中国最实用的 Django 教程。

我在2015年初时，阅读了全部的 Django 英文的官方文档，觉得国内比较好的Django学习资源不多，所以决定写自己的教程。

Django 2.2 已经发布，变动比较大，不再兼容Python 2 版本，本教程采用 **Python 3.7 + Django 2.2**。

如果想学习 **Django 1.4 - Django 1.11** 系列的 **Django 1.x** 的版本，[请点击此处](#)。

本教程作者: **涂伟忠(未经同意,禁止转载!)**

《Django开发从入门到实践》书籍出版啦！基于 Python3 + Django 2.x

image.png

书籍购买：[淘宝](#)、[京东](#)、[当当](#)

这是一本写给初学者的书籍，简介可以关注右侧的公众号了解。

遇到问题请直接回复对应的教程，如果有截图，附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong#163.com](mailto:tuweizhong#163.com) (请把#换成@)

书籍读者交流 QQ群：1020663804

## 除了本教程外的其它教程

自强学堂 学习分享 的文章：[Python 学习资源](#) 和 [Django 学习资源](#)，如果有更好的教程也可以在文章下推荐哦！

Django 是基于 Python，所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

## 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包，所以你得知道一些 [Python](#) 基础知识。
2. 其次你最好有一些做网站的经验，懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入

学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

## 优雅的网站

用正则匹配网址，传递到对应函数，随意定义，如你所想！

## 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

## 缓存系统

与Memcache, Redis等缓存系统联用，更出色的表现，更快的加载速度。

## 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

[小额赞助，支持作者！](#)

这是第一篇

[Django 简介 »](#)

 CODING  
CLOUD DEVELOPMENT

**零成本开启敏捷开发 五人以下团队免费**

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 基础教程

这是第一篇  
[Django 简介 »](#)

django

Django 是用Python开发的一个免费开源的Web框架, 可以用于快速搭建高性能, 优雅的网站!

你一定可以学会, Django 很简单! 本教程一直在维护, 更新, 提供免费答疑, 免费邮件咨询。

从2015年01月18日写第一篇Django教程, 一直到现在, 每一篇教程由于新版 Django 一些改变, 或用户的反馈, 随时可能会进行更新完善, 可以在网站首页看到最近更新的情况。截止2017年02月16日, 此Django教程已经有4400多条评论, 两年多以来, 几乎每天回复评论, 回复邮件。教程也根据评论进行了大量的改进, 包含大量的工程经验, 致力打造出中国最实用的 Django 教程。

我在2015年初时, 阅读了全部的 Django 英文的官方文档, 觉得国内比较好的Django学习资源不多, 所以决定写自己的教程。本教程开始写的时候是 Django 的版本是 1.6, Django的更新很快, 自强学堂的教程也随着更新了, 兼顾了后来的新版本, 从 Django 1.4 到最新的 Django 1.11 应该都没有问题。

自强学堂 就是用 Django 搭建的站点! 提供书籍下载, 可以在 kindle, ipad 等上阅读。

本教程作者: 涂伟忠(未经同意,禁止转载!)

遇到问题请直接回复对应的教程, 如果有截图, 附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong#163.com](mailto:tuweizhong#163.com) (请把#换成@)

目前不提供QQ群, 我觉得邮件可以集中处理, 更高效些。

本教程系列文章电子书下载: 链接: <https://pan.baidu.com/s/1rap4t20> 密码: uta2 (不含附件)

## 除了本教程外的其它教程

自强学堂 学习分享 的文章: [Python 学习资源](#) 和 [Django 学习资源](#), 如果有更好的教程也可以在文章下推荐哦!

Django 是基于 Python, 所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

## 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包, 所以你得知道一些 [Python](#) 基础知识。
2. 其次你最好有一些做网站的经验, 懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

### 优雅的网址

用正则匹配网址，传递到对应函数，随意定义，如你所想！

### 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

### 缓存系统

与Memcached, Redis等缓存系统联用，更出色的表现，更快的加载速度。

### 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

这是第一篇

[Django 简介 »](#)

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)  
[打印页面](#)



[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 基础教程

这是第一篇  
[Django 简介 »](#)

django

Django 是用[Python](#)开发的一个免费开源的Web框架，可以用于快速搭建高性能，优雅的网站！

你一定可以学会，Django 很简单！本教程一直在维护，更新，提供免费答疑，免费邮件咨询。

从2015年01月18日写第一篇Django教程，一直到现在，每一篇教程由于新版 Django 一些改变，或用户的反馈，随时可能会进行更新完善，可以在网站首页看到最近更新的情况。截止2017年02月16日，此Django教程已经有4400多条评论，两年多以来，几乎每天回复评论，回复邮件。教程也根据评论进行了大量的改进，包含大量的工程经验，致力打造出中国最实用的 Django 教程。

我在2015年初时，阅读了全部的 Django 英文的官方文档，觉得国内比较好的Django学习资源不多，所以决定写自己的教程。本教程开始写的时候是 Django 的版本是 1.6，Django的更新很快，自强学堂的教程也随着更新了，兼顾了后来的新版本，从 Django 1.4 到最新的 Django 1.11 应该都没有问题。

自强学堂 就是用 Django 搭建的站点！提供书籍下载，可以在 kindle, ipad 等上阅读。

本教程作者: 涂伟忠(未经同意,禁止转载!)

遇到问题请直接回复对应的教程，如果有截图，附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong#163.com](mailto:tuweizhong#163.com) (请把#换成@)

目前不提供QQ群，我觉得邮件可以集中处理，更高效些。

本教程系列文章电子书下载：链接: <https://pan.baidu.com/s/1rap4t20> 密码: uta2 (不含附件)



除了本教程外的其它教程

自强学堂 学习分享 的文章：[Python 学习资源](#) 和 [Django 学习资源](#)，如果有更好的教程也可以在文章下推荐哦！

Django 是基于 Python，所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

## 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包，所以你得知道一些 [Python](#) 基础知识。

2. 其次你最好有一些做网站的经验，懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入

学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

### 优雅的网站

用正则匹配网址，传递到对应函数，随意定义，如你所想！

### 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

### 缓存系统

与Memcached, Redis等缓存系统联用，更出色的表现，更快的加载速度。

### 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

这是第一篇

[Django 简介 »](#)

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 2.2 基础教程

这是第一篇  
[Django 简介 »](#)

django

Django 是用Python开发的一个免费开源的Web框架, 可以用于快速搭建高性能, 优雅的网站!

你一定可以学会, Django 很简单! 本教程一直在维护, 更新, 提供免费答疑, 免费邮件咨询。

从2015年01月18日写第一篇Django教程, 一直到现在, 每一篇教程由于新版 Django 一些改变, 或用户的反馈, 随时可能会进行更新完善, 可以在网站首页看到最近更新的情况。

截止2017年02月16日, 此Django教程已经有4400多条评论, 两年多以来, 几乎每天回复评论, 回复邮件。教程也根据评论进行了大量的改进, 包含大量的工程经验, 致力打造出中国最实用的 Django 教程。

我在2015年初时, 阅读了全部的 Django 英文的官方文档, 觉得国内比较好的Django学习资源不多, 所以决定写自己的教程。

Django 2.2 已经发布, 变动比较大, 不再兼容Python 2 版本, 本教程采用 **Python 3.7 + Django 2.2**。

如果想学习 Django 1.4 - Django 1.11 系列的 Django 1.x 的版本, 请点击[此处](#)。

本教程作者: 涂伟忠(未经同意,禁止转载!)

《Django开发从入门到实践》书籍出版啦! 基于 Python3 + Django 2.x

image.png

书籍购买: [淘宝](#)、[京东](#)、[当当](#)

这是一本写给初学者的书籍, 简介可以关注右侧的公众号了解。

遇到问题请直接回复对应的教程, 如果有截图, 附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong#163.com](mailto:tuweizhong#163.com) (请把#换成@)

书籍读者交流 QQ群: 1020663804

### 除了本教程外的其它教程

自强学堂 学习分享 的文章: [Python 学习资源](#) 和 [Django 学习资源](#), 如果有更好的教程也可以在文章下推荐哦!

Django 是基于 Python, 所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

# 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包，所以你得知道一些 [Python](#) 基础知识。

2. 其次你最好有一些做网站的经验，懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入

学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

### 优雅的网址

用正则匹配网址，传递到对应函数，随意定义，如你所想！

### 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

### 缓存系统

与Memcache, Redis等缓存系统联用，更出色的表现，更快的加载速度。

### 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

[小额赞助，支持作者！](#)

这是第一篇

[Django 简介](#) »

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包, 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 2.2 基础教程

这是第一篇

[Django 简介 »](#)

django

Django 是用[Python](#)开发的一个免费开源的Web框架，可以用于快速搭建高性能，优雅的网站！

你一定可以学会，Django 很简单！**本教程一直在维护，更新，提供免费答疑，免费邮件咨询。**

从2015年01月18日写第一篇Django教程，一直到现在，每一篇教程由于新版 Django 一些改变，或用户的反馈，随时可能会进行更新完善，可以在网站首页看到最近更新的情况。

截止2017年02月16日，此Django教程已经有4400多条评论，两年多以来，几乎每天回复评论，回复邮件。教程也根据评论进行了大量的改进，包含大量的工程经验，致力打造出中国最实用的 Django 教程。

我在2015年初时，阅读了全部的 Django 英文的官方文档，觉得国内比较好的Django学习资源不多，所以决定写自己的教程。

Django 2.2 已经发布，变动比较大，不再兼容Python 2 版本，本教程采用 **Python 3.7 + Django 2.2**。

如果想学习 **Django 1.4 - Django 1.11** 系列的 **Django 1.x** 的版本，[请点击此处](#)。

本教程作者: **涂伟忠(未经同意,禁止转载!)**

《Django开发从入门到实践》书籍出版啦！基于 Python3 + Django 2.x

image.png

书籍购买：[淘宝](#)、[京东](#)、[当当](#)

这是一本写给初学者的书籍，简介可以关注右侧的公众号了解。

遇到问题请直接回复对应的教程，如果有截图，附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong@163.com](mailto:tuweizhong@163.com) (请把#换成@)

书籍读者交流 QQ群：1020663804

## 除了本教程外的其它教程

自强学堂 学习分享 的文章：[Python 学习资源](#) 和 [Django 学习资源](#)，如果有更好的教程也可以在文章下推荐哦！

Django 是基于 Python，所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

## 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包，所以你得知道一些 [Python](#) 基础知识。
2. 其次你最好有一些做网站的经验，懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入

学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

## 优雅的网站

用正则匹配网址，传递到对应函数，随意定义，如你所想！

## 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

## 缓存系统

与Memcache, Redis等缓存系统联用，更出色的表现，更快的加载速度。

## 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

[小额赞助，支持作者！](#)

这是第一篇

[Django 简介 »](#)

 CODING  
CLOUD DEVELOPMENT

**零成本开启敏捷开发 五人以下团队免费**

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 简介

[« Django 2.2 基础教程](#)  
[Django 环境搭建 »](#)

自强学堂的django教程将节省你大量的时间, 并且使你的web开发充满乐趣。通过Django, 你可以建立一个高性能的web应用而只花费最少的时间和精力。

Django 中提供了开发网站经常用到的模块, 常见的代码都为你写好了, 通过减少重复的代码, Django 使你能够专注于web 应用上有趣的关键性的东西。为了达到这个目标, Django 提供了通用Web开发模式的高度抽象, 提供了频繁进行的编程作业的快速解决方法, 以及为“如何解决问题”提供了清晰明了的约定。Django的理念是DRY(Don't Repeat Yourself)来鼓励快速开发!

## 让我们一览 Django 全貌

### [urls.py](#)

网址入口, 关联到对应的views.py中的一个函数(或者generic类), 访问网址就对应一个函数。

### [views.py](#)

处理用户发出的请求, 从urls.py中对应过来, 通过渲染templates中的网页可以将显示内容, 比如登陆后的用户名, 用户请求的数据, 输出到网页。

### [models.py](#)

与数据库操作相关, 存入或读取数据时用到这个, 当然用不到数据库的时候 你可以不使用。

### [forms.py](#)

表单, 用户在浏览器上输入数据提交, 对数据的验证工作以及输入框的生成等工作, 当然你也可以不使用。

### **templates 文件夹**

views.py 中的函数渲染templates中的Html模板, 得到动态内容的网页, 当然可以用缓存来提高速度。

### [admin.py](#)

后台, 可以用很少量的代码就拥有一个强大的后台。

### [settings.py](#)

Django 的设置, 配置文件, 比如 DEBUG 的开关, 静态文件的位置等。

[小额赞助, 支持作者!](#)  
[« Django 2.2 基础教程](#)  
[Django 环境搭建 »](#)



零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



零成本开启敏捷开发 五人以下团队免费

免费体验

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的,自强学堂网站源代码免费下载](#)

## Django 简介

[« Django 2.2 基础教程](#)

[Django 环境搭建 »](#)

自强学堂的django教程将节省你大量的时间，并且使你的web开发充满乐趣。通过Django，你可以建立一个高性能的web应用而只花费最少的时间和精力。

Django 中提供了开发网站经常用到的模块，常见的代码都为你写好了，通过减少重复的代码，Django 使你能够专注于web 应用上有趣的关键性的东西。为了达到这个目标，Django 提供了通用Web开发模式的高度抽象，提供了频繁进行的编程作业的快速解决方法，以及为“如何解决问题”提供了清晰明了的约定。Django的理念是DRY(Don't Repeat Yourself)来鼓励快速开发！

## 让我们一览 Django 全貌

[urls.py](#)

网址入口，关联到对应的views.py中的一个函数（或者generic类），访问网址就对应一个函数。

[views.py](#)

处理用户发出的请求，从`urls.py`中对应过来，通过渲染`templates`中的网页可以将显示内容，比如登陆后的用户名，用户请求的数据，输出到网页。

## [models.py](#)

与数据库操作相关，存入或读取数据时用到这个，当然用不到数据库的时候 你可以不使用。

## [forms.py](#)

表单，用户在浏览器上输入数据提交，对数据的验证工作以及输入框的生成等工作，当然你也可以不使用。

## `templates` 文件夹

`views.py` 中的函数渲染`templates`中的`Html`模板，得到动态内容的网页，当然可以用缓存来提高速度。

## [admin.py](#)

后台，可以用很少量的代码就拥有一个强大的后台。

## [settings.py](#)

Django 的设置，配置文件，比如 `DEBUG` 的开关，静态文件的位置等。

[小额赞助，支持作者！](#)

[« Django 2.2 基础教程](#)

[Django 环境搭建 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的,自强学堂网站源代码免费下载](#)

## Django 环境搭建

[« Django 简介](#)  
[Django 基本命令 »](#)

本文最后面讲了如何使用 `virtualenv` 实现多个互不干扰的开发环境。

### 一. 版本选择

Django 2.0.x 支持 Python 3.4, 3.5 和 3.6 (最后一个支持 Python 3.4 的版本)

Django 2.1.x 支持 Python 3.5, 3.6 和 3.7

**Django 2.2.x 支持 Python 3.5, 3.6 和 3.7 (LTS 长期支持版本)**

一般来说,选择长期支持版本比较好。

另外说一句 Django 3.0 测试版本也发布了,但主要是支持 ASGI,不影响学习,所以不要纠结,选择一个版本好好学下去就行了。

### 二. 安装 Django

推荐: 在 [Cloud Studio](#) 中进行 Django 开发

新手目前还有一个免费云主机,体验在线开发和部署的乐趣。

- step1: 访问 [Cloud Studio](#), 注册/登录账户。
- step2: 在右侧的运行环境菜单选择: "PHP + Python + Java 三种语言环境"
- step3: 在终端上安装 Django, 启动项目, 如图

```
Cloud Studio Beta 工作空间 文件 编辑 版本 工具 帮助
Home zqxt zqxt settings.py
zqxt
  _init_.py
  settings.py
  settings.pyc
  urls.py
  wsgi.py
  db.sqlite3
  manage.py
  Coding.jpg
..ace/zqxt/zqxt ..ace/zqxt/zqxt cd x +
settings.py x24
25 # SECURITY WARNING: don't run with debug turned on in
26 DEBUG = True
27
28 ALLOWED_HOSTS = ['*']
29
30 # Application definition
31
32 INSTALLED_APPS = [
33     'django.contrib.admin',
34
workspace git:(master) X sudo -H pip install "Django<2"
Looking in indexes: http://mirrors.aliyun.com/pypi/simple
Requirement already satisfied: Django<2 in /usr/local/lib/python2.7/dist-packages (1.11.13)
Requirement already satisfied: pytz in /usr/local/lib/python2.7/dist-packages (from Django<2) (2018.4)
workspace git:(master) X django-admin.py startproject zqxt
workspace git:(master) X cd zqxt && python manage.py runserver 0.0.0.0:8000
Performing system checks...
System check identified no issues (0 silenced).
You have 13 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
June 27, 2018 - 00:10:04
Django version 1.11.13, using settings 'zqxt.settings'
Starting development server at http://0.0.0.0:8000/
终端 项目网络
skuoetz 行:30 列:2 UTF-8 python P master
```

注意：以下方法中任何一种方法安装都可，不用每个都试一次。

另外 建议自行安装 **bpython**，这样在用起来会爽很多。进入终端的时候输入 **bpython** 可以有提示。当然也可以选择用 **ipython**

## 2.1. 用 **pip** 来安装

### 2.1.1 需要先安装 **pip**

(1). ubuntu:

```
sudo apt-get install python-pip
```

(2). Fedora:

```
yum install python-pip
```

(3). Linux, Mac OSX, Windows 下都可用 **get-pip.py** 来安装 **pip**: <https://pip.pypa.io/en/latest/installing.html>

或者直接下载: [get-pip.py](#) 然后运行在终端运行 **python get-pip.py** 就可以安装 **pip**。

Note: 也可以下载 **pip** 源码包，运行 **python setup.py install** 进行安装

### 2.1.2 利用 **pip** 安装 Django

如果安装太慢，可以配置镜像，参考 <https://github.com/twz915/configfiles/blob/master/pip.conf>

```
(sudo) pip3 install Django  
或者 (sudo) pip3 install Django==2.2.7
```

如果想升级 **pip** 可以用:

```
(sudo) pip3 install --upgrade pip
```

Windows 用户不要加 **sudo**，如果提示 '**python**'不是内部或外部命令，也不是可运行的程序或批处理文件。

那说明你的 **Python** 没有安装好，或者环境变量没有配置正确，安装新版本的 **Python** (3.7 以上)，里面集成了 **pip**，安装时要勾选上环境变量这一个

**windows 安装 Python 图片**



还可以参见: [Python 环境搭建](#)

## 2.2. 下载源码安装

<https://www.djangoproject.com/download/>

如果是源码包, 比如 `django-2.2.7.tar.gz`

### 2.2.1 Linux 或 Mac 下

```
tar -xvf django-2.2.7.tar.gz
cd django-2.2.7
(sudo) python setup.py install
```

### 2.2.2 Windows 下

直接用解压软件解压, 然后到命令行 (XP/Win7 点击开始, 在下面的那个输入框中输入 `cmd`, Win8 在开始那里点右键, 选择命令行)

比如在 `D:\django-2.2.7\` 这个文件夹下

```
cd D:
cd django-2.2.7
python setup.py install
```

什么? 提示 `'python'` 不是内部或外部命令, 也不是可运行的程序或批处理文件。

那说明你的 `Python` 没有安装好, 或者路径没有配置正确, 参见: [Python 环境搭建](#)

### 2.3. Linux 用自带源进行安装 (不推荐)

#### 2.3.1 ubuntu 下安装 Django

```
sudo apt-get install python-django -y
```

#### 2.3.2 Fedora 下安装用 yum

```
yum install python-django
```

注意: 自带源安装的 Django 一般版本比较旧, 而用 `pip` 可以安装最新的版本。

## 三. 检查是否安装成功

终端上输入 `python`, 点击 `Enter`, 进行 `python` 环境

```
>>> import django
>>> django.VERSION
(2, 2, 7, 'final', 0)
>>>
>>> django.get_version()
'2.2.7'
```

如果运行后看到版本号, 就证明安装成功了, 有问题请评论!

## 四. 搭建多个互不干扰的开发环境 (可选)

我们有的时候会发现, 一个电脑上有多个项目, 一个依赖 Django 2.2, 另一个比较旧的项目又要用 Django 1.11, 这时候怎么办呢?

我们需要一个依赖包管理的工具来处理不同的环境。

如果不想搭建这个环境, 只想用某一个版本的 `Django` 也可以, 但是推荐学习此内容!

### 4.1 虚拟环境依赖安装

开发会用 `virtualenv` 来管理多个开发环境

#### Linux/MacOS 下

`virtualenvwrapper` 使得 `virtualenv` 变得更好用, 所以我们一起安装了

```
# 安装:
(sudo) pip install virtualenv virtualenvwrapper
```

修改 `~/.bash_profile` 或其它环境变量相关文件 (如 `.bashrc` 或用 ZSH 之后的 `.zshrc`), 添加以下语句

```
export WORKON_HOME=$HOME/.virtualenvs
```

```
export PROJECT_HOME=$HOME/workspace
source /usr/local/bin/virtualenvwrapper.sh
```

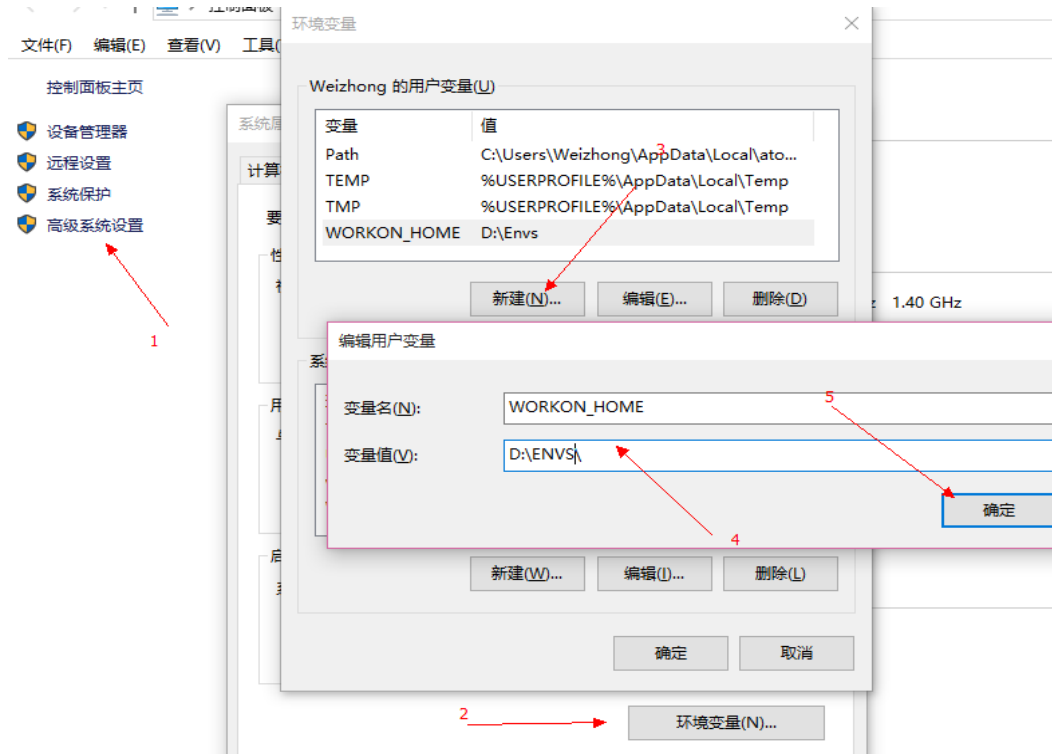
修改后使之立即生效(也可以重启终端使之生效):

```
source ~/.bash_profile
```

## Windows 下:

```
pip install virtualenv virtualenvwrapper-win
```

【可选】Windows下默认虚拟环境是放在用户名下面的Envs中的，与桌面，我的文档，下载等文件夹在一块的。更改方法：计算机，属性，高级系统设置，环境变量，添加WORKON\_HOME，如图（windows 10 环境变量设置截图）：



## 4.2 虚拟环境使用方法:

**mkvirtualenv zqxt**: 创建运行环境zqxt

**workon zqxt**: 工作在 zqxt 环境 或 从其它环境切换到 zqxt 环境

**deactivate**: 退出终端环境

其它的:

**rmvirtualenv ENV**: 删除运行环境ENV

**mkproject mic**: 创建mic项目和运行环境mic

**mktmpenv**: 创建临时运行环境

**lsvirtualenv**: 列出可用的运行环境

**lssitepackages**: 列出当前环境安装了包

创建的环境是独立的，互不干扰，无需sudo权限即可使用 pip 来进行包的管理。



[小额赞助, 支持作者!](#)  
[« Django 简介](#)  
[Django 基本命令 »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 环境搭建

[« Django 简介](#)  
[Django 基本命令 »](#)

本文最后面讲了如何 使用 `virtualenv` 实现多个互不干扰的开发环境。

### 一. 版本选择

Django 2.0.x 支持 Python 3.4, 3.5 和 3.6 (最后一个支持 Python 3.4 的版本)

Django 2.1.x 支持 Python 3.5, 3.6 和 3.7

**Django 2.2.x 支持 Python 3.5, 3.6 和 3.7 (LTS 长期支持版本)**

一般来说, 选择长期支持版本比较好。

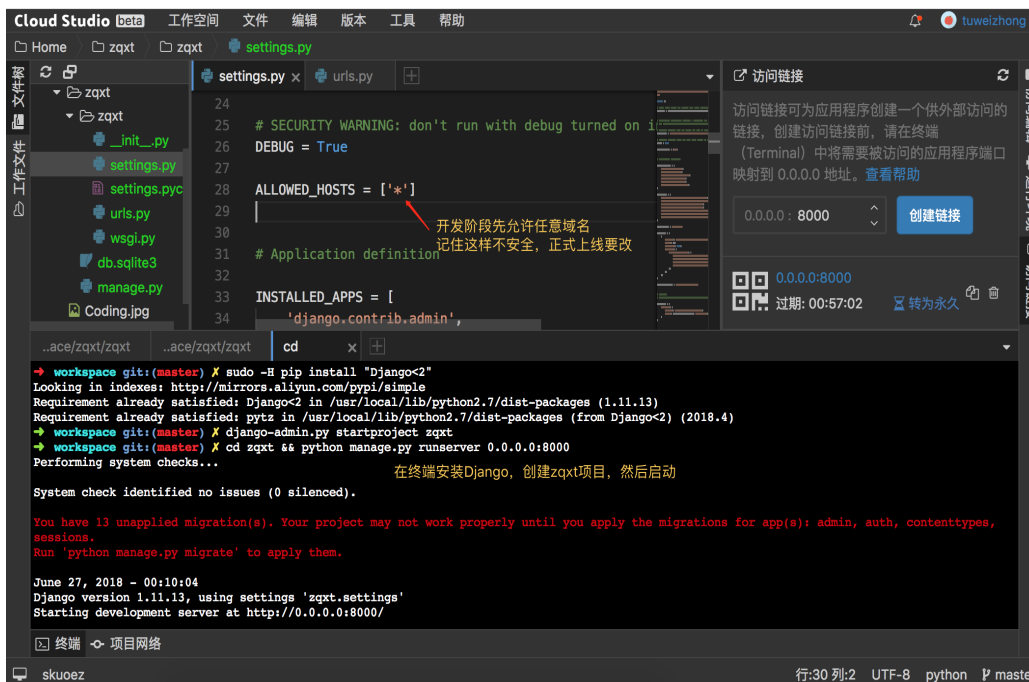
另外说一句 Django 3.0 测试版本也发布了, 但主要是支持 ASGI, 不影响学习, 所以不要纠结, 选择一个版本好好学下去就行了。

### 二. 安装 Django

## 推荐：在Cloud Studio中进行Django开发

新手目前还有一个免费云主机，体验在线开发和部署的乐趣。

- step1: 访问[Cloud Studio](#)，注册/登录账户。
- step2: 在右侧的运行环境菜单选择：**"PHP + Python + Java 三种语言环境"**
- step3: 在终端上安装Django，启动项目，如图



注意：以下方法中任何一种方法安装都可，不用每个都试一次。

另外 建议自行安装 **bpython**，这样在用起来会爽很多。进入终端的时候输入 **bpython** 可以有提示。当然也可以选择用 **ipython**

### 2.1. 用 pip 来安装

#### 2.1.1 需要先安装 pip

(1). ubuntu:

```
sudo apt-get install python-pip
```

(2). Fedora:

```
yum install python-pip
```

(3). Linux, Mac OSX, Windows 下都可用 **get-pip.py** 来安装 pip: <https://pip.pypa.io/en/latest/installing.html>

或者直接下载: [get-pip.py](#) 然后运行在终端运行 **python get-pip.py** 就可以安装 **pip**。

Note: 也可以下载 **pip** 源码包，运行 **python setup.py install** 进行安装

#### 2.1.2 利用 pip 安装 Django

如果安装太慢，可以配置镜像，参考 <https://github.com/tw915/configfiles/blob/master/pip.conf>

```
(sudo) pip3 install Django  
或者 (sudo) pip3 install Django==2.2.7
```

如果想升级 **pip** 可以用:

```
(sudo) pip3 install --upgrade pip
```

Windows 用户不要加 `sudo`，如果提示 ‘python’不是内部或外部命令，也不是可运行的程序或批处理文件。

那说明你的 **Python** 没有安装好，或者环境变量没有配置正确，安装新版本的 **Python**（3.7 以上），里面集成了 **pip**，安装时要勾选上环境变量这一个

windows 安装 Python 图片



还可以参见：[Python 环境搭建](#)

## 2.2. 下载源码安装

<https://www.djangoproject.com/download/>

如果是源码包, 比如 `django-2.2.7.tar.gz`

### 2.2.1 Linux 或 Mac 下

```
tar -xvf django-2.2.7.tar.gz
cd django-2.2.7
(sudo) python setup.py install
```

### 2.2.2 Windows 下

直接用解压软件解压，然后到命令行（XP/Win7点击开始，在下面的那个输入框中输入 `cmd`, Win8在开始那里点右键，选择命令行）

比如在 **D:\django-2.2.7** 这个文件夹下

```
cd D:
cd django-2.2.7
python setup.py install
```

什么？提示 ‘python’不是内部或外部命令，也不是可运行的程序或批处理文件。

那说明你的 **Python** 没有安装好，或者路径没有配置正确，参见：[Python 环境搭建](#)

## 2.3. Linux用自带源进行安装（不推荐）

### 2.3.1 ubuntu 下安装 Django

```
sudo apt-get install python-django-y
```

### 2.3.2 Fedora 下安装用 yum

```
yum install python-django
```

注意：自带源安装的 Django 一般版本比较旧，而用 pip 可以安装最新的版本。

### 三. 检查是否安装成功

终端上输入 `python`, 点击 `Enter`, 进行 `python` 环境

```
>>> import django
>>> django.VERSION
(2, 2, 7, 'final', 0)
>>>
>>> django.get_version()
'2.2.7'
```

如果运行后看到版本号，就证明安装成功了，有问题请评论！

### 四. 搭建多个互不干扰的开发环境（可选）

我们有的时候会发现，一个电脑上有多个项目，一个依赖 Django 2.2，另一个比较旧的项目又要用 Django 1.11，这时候怎么办呢？

我们需要一个依赖包管理的工具来处理不同的环境。

如果不想搭建这个环境，只想用某一个版本的 Django 也可以，但是推荐学习此内容！

#### 4.1 虚拟环境依赖安装

开发会用 `virtualenv` 来管理多个开发环境

##### Linux/MacOS 下

`virtualenvwrapper` 使得 `virtualenv` 变得更好用，所以我们一起安装了

```
# 安装:
(sudo) pip install virtualenv virtualenvwrapper
```

修改 `~/.bash_profile` 或其它环境变量相关文件(如 `.bashrc` 或用 ZSH 之后的 `.zshrc`)，添加以下语句

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/workspace
source /usr/local/bin/virtualenvwrapper.sh
```

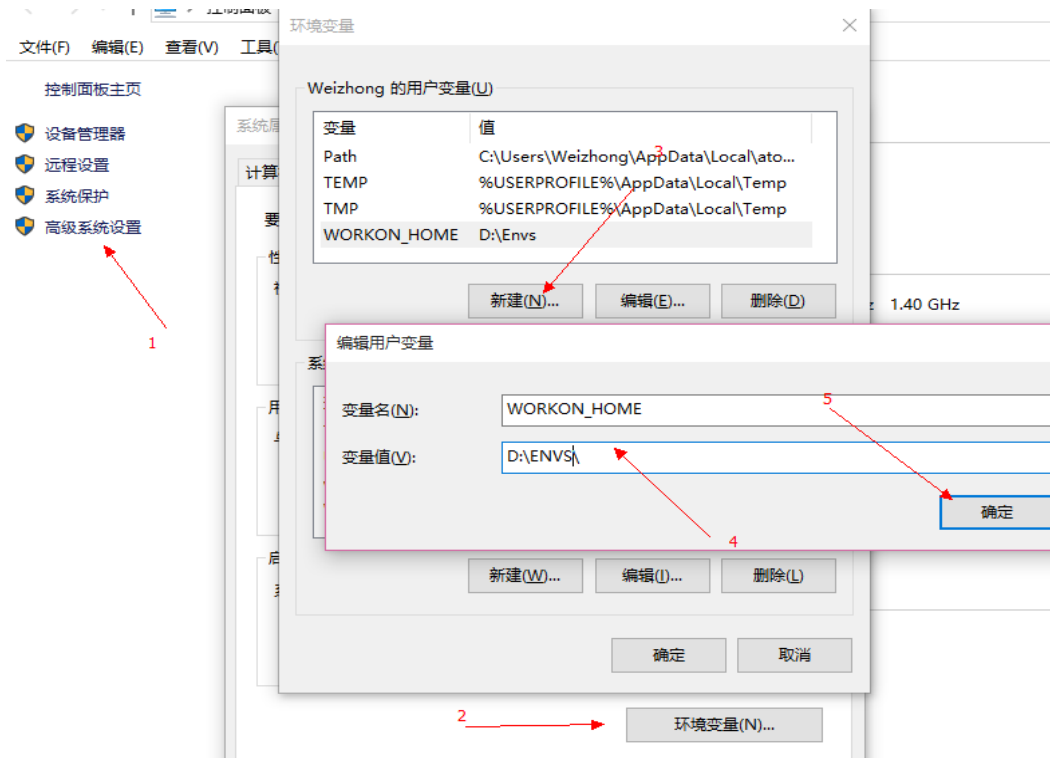
修改后使之立即生效(也可以重启终端使之生效):

```
source ~/.bash_profile
```

##### Windows 下:

```
pip install virtualenv virtualenvwrapper-win
```

【可选】Windows 下默认虚拟环境是放在用户名下面的 `Envs` 中的，与桌面，我的文档，下载等文件夹在一块的。更改方法：计算机，属性，高级系统设置，环境变量，添加 `WORKON_HOME`，如图（windows 10 环境变量设置截图）：



#### 4.2 虚拟环境使用方法:

**mkvirtualenv zqxt:** 创建运行环境zqxt

**workon zqxt:** 工作在 zqxt 环境 或 从其它环境切换到 zqxt 环境

**deactivate:** 退出终端环境

其它的:

**rmvirtualenv ENV:** 删除运行环境ENV

**mkproject mic:** 创建mic项目和运行环境mic

**mktmpenv:** 创建临时运行环境

**lsvirtualenv:** 列出可用的运行环境

**lssitepackages:** 列出当前环境安装了包

创建的环境是独立的, 互不干扰, 无需sudo权限即可使用 pip 来进行包的管理。

[小额赞助, 支持作者!](#)

[« Django 简介](#)

[Django 基本命令 »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 基本命令

[« Django 环境搭建](#)  
[Django 视图与网址 »](#)

本节主要是为了让您了解一些django最基本的命令, **请尝试着记住它们, 并且多多练习下, 特别是标记为红色的那些**

打开 **Linux** 或 **MacOS** 的 **Terminal** (终端) 直接在 终端中输入这些命令 (不是 **python** 的 **shell**中)

如果是 **windows** 用 **cmd** (开始 搜索 **cmd** 或者 快捷键 **win + R**, 输入 **cmd**) 直接在 **cmd** 上操作。

## 1. 新建一个 django project

```
django-admin.py startproject project_name
```

特别是在 windows 上, 如果报错, 尝试用 **django-admin** 代替 **django-admin.py** 试试

注意 **project\_name** 是自己的项目名称, 需要为合法的 Python 包名, 如不能为 **1a** 或 **a-b**。

## 2. 新建 app

要先进入项目目录下, **cd project\_name** 然后执行下面的命令 (下同, 已经在项目目录下则不需要 **cd project\_name**)

```
python manage.py startapp app_name  
或 django-admin.py startapp app_name
```

一般一个项目有多个app, 当然通用的app也可以在多个项目中使用。

与项目名类似 **app name** 也需要为合法的 Python 包名, 如 **blog**, **news**, **aboutus** 等都是合法的 app 名称。

## 3. 创建数据库表 或 更改数据库表或字段

```
# 1. 创建更改的文件  
python manage.py makemigrations
```

```
# 2. 将生成的py文件应用到数据库  
python manage.py migrate
```

这种方法可以在SQL等数据库中创建与**models.py**代码对应的表, 不需要自己手动执行SQL。

备注: 对已有的 **models** 进行修改, Django 1.7之前的版本的Django都是无法自动更改表结构的, 不过有第三方工具 **south**, 详见 [Django 数据库迁移](#) 一节。

## 4. 使用开发服务器

开发服务器, 即开发时使用, 一般修改代码后会自动重启, 方便调试和开发, 但是由于性能问题, 建议只用来测试, 不要用在生产环境。

```
python manage.py runserver
```

```
# 当提示端口被占用的时候, 可以用其它端口:  
python manage.py runserver 8001  
python manage.py runserver 9999  
(当然也可以kill掉占用端口的进程, 具体后面有讲, 此处想知道的同学可查下 lsof 命令用法)
```

```
# 监听机器所有可用 ip (电脑可能有多个内网ip或多个外网ip)
python manage.py runserver 0.0.0.0:8000

# 如果是外网或者局域网电脑上可以用其它电脑查看开发服务器
# 访问对应的 ip加端口, 比如 http://172.16.20.2:8000
```

## 5. 清空数据库

```
python manage.py flush
```

此命令会询问是 yes 还是 no, 选择 yes 会把数据全部清空掉, 只留下空表。

## 6. 创建超级管理员

```
python manage.py createsuperuser
```

# 按照提示输入用户名和对应的密码就好了邮箱可以留空, 用户名和密码必填

# 修改 用户密码可以用:

```
python manage.py changepassword username
```

## 7. 导出数据 导入数据

```
python manage.py dumpdata appname > appname.json
python manage.py loaddata appname.json
```

关于数据操作 详见: [数据导入数据迁移](#), 现在了解有这个用法就可以了。

## 8. Django 项目环境终端

```
python manage.py shell
```

如果你安装了 bpython 或 ipython 会自动用它们的界面, 推荐安装 bpython。

这个命令和 直接运行 python 或 bpython 进入 shell 的区别是: 你可以在这个 shell 里面调用当前项目的 models.py 中的 API, 对于操作数据, 还有一些小测试非常方便。

## 9. 数据库命令行

```
python manage.py dbshell
```

Django 会自动进入在 settings.py 中设置的数据库, 如果是 MySQL 或 postgresSQL, 会要求输入数据库用户密码。

在这个终端可以执行数据库的 SQL 语句。如果您对 SQL 比较熟悉, 可能喜欢这种方式。

## 10. 更多命令

终端上输入 python manage.py 可以看到详细的列表, 在忘记子名称的时候特别有用。

更详细的介绍, 点击对应版本去官网查看: [2.0](#) [2.1](#) [2.2](#) [dev](#)

[小额赞助, 支持作者!](#)

[«Django 环境搭建](#)

[Django 视图与网址»](#)



↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。



关注自强学堂官方微信, 随时查教程 提升自己!   
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。  
自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于[Django](#)开发的, [自强学堂网站源代码免费下载](#)

## Django 基本命令

[« Django 环境搭建](#)  
[Django 视图与网址 »](#)

本节主要是为了让您了解一些django最基本的命令, **请尝试着记住它们, 并且多多练习下, 特别是标记为红色的那些**

打开 **Linux** 或 **MacOS** 的 **Terminal** (终端) 直接在 终端中输入这些命令 (不是 **python** 的 **shell**中)

如果是 **windows** 用 **cmd** (开始 搜索 **cmd** 或者 快捷键 **win + R**, 输入 **cmd**) 直接在 **cmd** 上操作。

### 1. 新建一个 django project

```
django-admin.py startproject project_name
```

特别是在 windows 上, 如果报错, 尝试用 **django-admin** 代替 **django-admin.py** 试试

注意 **project\_name** 是自己的项目名称, 需要为合法的 Python 包名, 如不能为 **1a** 或 **a-b**。

### 2. 新建 app

要先进入项目目录下, **cd project\_name** 然后执行下面的命令 (下同, 已经在项目目录下则不需要 **cd project\_name**)

```
python manage.py startapp app_name  
或 django-admin.py startapp app_name
```

一般一个项目有多个app, 当然通用的app也可以在多个项目中使用。

与项目名类似 `app name` 也需要为合法的 Python 包名，如 `blog`, `news`, `aboutus` 等都是合法的 `app` 名称。

### 3. 创建数据库表 或 更改数据库表或字段

```
# 1. 创建更改的文件
python manage.py makemigrations
```

```
# 2. 将生成的py文件应用到数据库
python manage.py migrate
```

这种方法可以在SQL等数据库中创建与`models.py`代码对应的表，不需要自己手动执行SQL。

备注：对已有的 `models` 进行修改，Django 1.7之前的版本的Django都是无法自动更改表结构的，不过有第三方工具 `south`, 详见 [Django 数据库迁移](#) 一节。

### 4. 使用开发服务器

开发服务器，即开发时使用，一般修改代码后会自动重启，方便调试和开发，但是由于性能问题，建议只用来测试，不要用在生产环境。

```
python manage.py runserver
```

```
# 当提示端口被占用的时候，可以用其它端口：
```

```
python manage.py runserver 8001
```

```
python manage.py runserver 9999
```

（当然也可以kill掉占用端口的进程，具体后面有讲，此处想知道的同学可查下 `lsof` 命令用法）

```
# 监听机器所有可用 ip （电脑可能有多个内网ip或多个外网ip）
```

```
python manage.py runserver 0.0.0.0:8000
```

```
# 如果是外网或者局域网电脑上可以用其它电脑查看开发服务器
```

```
# 访问对应的 ip加端口，比如 http://172.16.20.2:8000
```

### 5. 清空数据库

```
python manage.py flush
```

此命令会询问是 `yes` 还是 `no`, 选择 `yes` 会把数据全部清空掉，只留下空表。

### 6. 创建超级管理员

```
python manage.py createsuperuser
```

```
# 按照提示输入用户名和对应的密码就好了邮箱可以留空，用户名和密码必填
```

```
# 修改 用户密码可以用：
```

```
python manage.py changepassword username
```

### 7. 导出数据 导入数据

```
python manage.py dumpdata appname > appname.json
```

```
python manage.py loaddata appname.json
```

关于数据操作 详见：[数据导入数据迁移](#)，现在了解有这个用法就可以了。

### 8. Django 项目环境终端

```
python manage.py shell
```

如果你安装了 `bpython` 或 `ipython` 会自动用它们的界面，推荐安装 `bpython`。

这个命令和 直接运行 `python` 或 `bpython` 进入 `shell` 的区别是：你可以在这个 `shell` 里面调用当前项目的 `models.py` 中的

API, 对于操作数据, 还有一些小测试非常方便。

## 9. 数据库命令行

```
python manage.py dbshell
```

Django 会自动进入在`settings.py`中设置的数据库, 如果是 MySQL 或 postgresSQL, 会要求输入数据库用户密码。

在这个终端可以执行数据库的SQL语句。如果您对SQL比较熟悉, 可能喜欢这种方式。

## 10. 更多命令

终端上输入 `python manage.py` 可以看到详细的列表, 在忘记子名称的时候特别有用。

更详细的介绍, 点击对应版本去官网查看: [2.0](#) [2.1](#) [2.2](#) [dev](#)

[小额赞助, 支持作者!](#)

[«Django 环境搭建](#)

[Django 视图与网址»](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 视图与网址

[« Django 基本命令](#)  
[Django 视图与网址进阶 »](#)

Django中网址是写在 `urls.py` 文件中, 用正则表达式对应 `views.py` 中的一个函数(或者 `generic`类),我们用个项目来演示。

下载本节所有源代码:

学习编程最好的办法就是动手敲代码, 请按照教程做, 本节很简单, 不提供源代码了, 动手开始吧!

## 一, 首先, 新建一个项目(project), 名称为 `mysite`

```
django-admin startproject mysite
```

备注:

1. 如果 `django-admin` 不行, 请用 `django-admin.py`
2. 如果是在Linux是用源码安装的, 或者用 `pip` 安装的, 也是用 `django-admin.py` 命令

django-admin 还是 django-admin.py?

运行后,如果成功的话, 我们会看到如下的目录样式 (没有成功的请参见[环境搭建](#)一节):

```
mysite
├── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

我们会发现执行命令后, 新建了一个 `mysite` 目录, 其中还有一个 `mysite` 目录, 这个子目录 `mysite` 中是一些项目的设置 `settings.py` 文件, 总的 `urls`配置文件 `urls.py` 以及部署服务器时用到的 `wsgi.py` 文件, `__init__.py` 是python包的目录结构必须的, 与调用有关。

我们到外层那个 `mysite` 目录下(不是 `mysite` 中的 `mysite` 目录)

## 二, 新建一个应用(app), 名称叫 `learn`

```
python3 manage.py startapp learn # learn 是一个app的名称
```

我们可以看到 `mysite` 中多个一个 `learn` 文件夹, 其中有以下文件。

```
learn
├── __init__.py
├── admin.py      后台相关的设置
├── apps.py       app相关的设置文件
├── migrations    数据库变更相关
│   └── __init__.py
├── models.py     数据库模型相关
├── tests.py      单元测试
└── views.py      视图逻辑
```

把我们新定义的 `app` 加到 `settings.py` 中的 `INSTALL_APPS` 中

修改 `mysite/mysite/settings.py`

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'learn',    # 注意添加了这一行
]

```

备注,这一步是干什么呢? 新建的 **app** 如果不加到 **INSTALLED\_APPS** 中的话, **django** 就不能自动找到app中的模板文件(app-name/templates/下的文件)和静态文件(app-name/static/中的文件), 后面你会学习到它们分别用来干什么。

## 定义视图函数（访问页面时的内容）

我们在**learn**这个目录中,把**views.py**打开,修改其中的源代码,改成下面的

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("欢迎光临 自强学堂!")

```

第一行引入**HttpResponse**, 它是用来向网页返回内容的, 就像Python中的 **print** 一样, 只不过 **HttpResponse** 是把内容显示到网页上。

我们定义了一个**index()**函数, 第一个参数必须是 **request**, 与网页发来的请求有关, **request** 变量里面包含**get**或**post**的内容, 用户浏览器, 系统等信息在里面（后面会讲, 先了解一下就可以）。

函数返回了一个 **HttpResponse** 对象, 可以经过一些处理, 最终显示几个字到网页上。

那问题来了, 我们访问什么网址才能看到刚才写的这个函数呢? 怎么让网址和函数关联起来呢?

## 定义视图函数相关的URL(网址)（即规定 访问什么网址对应什么内容）

我们打开 **mysite/mysite/urls.py** 这个文件, 修改其中的代码:

```
from django.contrib import admin
from django.urls import path
from learn import views as learn_views    # new

urlpatterns = [
    path("", learn_views.index),    # new
    path('admin/', admin.site.urls),
]

```

以上都修改并保存后,我们来看一下效果!

在终端上运行 **python3 manage.py runserver** 我们会看到类似下面的信息:

```
tu@mac ~/test/mysite $ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

November 10, 2019 - 09:19:50
Django version 2.2.7, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

```

提示: 上面有一些数据库相关的提示, 但目前我们没有用到, 暂时先忽略他们。

我们打开浏览器,访问 <http://127.0.0.1:8000/>

不出意料的话会看到:

**注意:** 如果是在另一台电脑上访问要用 **python manage.py ip:port** 的形式, 比如监听所有ip:

```
python manage.py runserver 0.0.0.0:8000

监听机器上所有ip 8000端口, 访问时用电脑的ip代替 127.0.0.1

```

参考链接:

(a). <https://docs.djangoproject.com/en/2.2/topics/http/urls/>

[小额赞助, 支持作者!](#)

[« Django 基本命令](#)

[Django 视图与网址进阶 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。  
自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的,自强学堂网站源代码免费下载](#)

# Django 视图与网址

[« Django 基本命令](#)  
[Django 视图与网址进阶 »](#)

Django中网址是写在 `urls.py` 文件中，用正则表达式对应 `views.py` 中的一个函数(或者generic类),我们用一个项目来演示。

下载本节所有源代码:

学习编程最好的办法就是动手敲代码，请按照教程做，本节很简单，不提供源代码了，动手开始吧！

## 一，首先，新建一个项目(project), 名称为 `mysite`

```
django-admin startproject mysite
```

备注：

1. 如果 `django-admin` 不行，请用 `django-admin.py`
2. 如果是在Linux是用源码安装的，或者用 `pip` 安装的，也是用 `django-admin.py` 命令

django-admin 还是 django-admin.py?

运行后,如果成功的话,我们会看到如下的目录样式 (没有成功的请参见[环境搭建](#)一节):

mysite

```
├── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

我们会发现执行命令后, 新建了一个 **mysite** 目录, 其中还有一个 **mysite** 目录, 这个子目录 **mysite** 中是一些项目的设置 **settings.py** 文件, 总的urls配置文件 **urls.py** 以及部署服务器时用到的 **wsgi.py** 文件, **\_\_init\_\_.py** 是python包的目录结构必须的, 与调用有关。

我们到外层那个 **mysite** 目录下(不是mysite中的mysite目录)

## 二, 新建一个应用(app), 名称叫 learn

`python3 manage.py startapp learn` # learn 是一个app的名称

我们可以看到mysite中多个一个 **learn** 文件夹, 其中有以下文件。

```
learn
├── __init__.py
├── admin.py      后台相关的设置
├── apps.py       app相关的设置文件
├── migrations    数据库变更相关
├── __init__.py
├── models.py     数据库模型相关
├── tests.py      单元测试
└── views.py      视图逻辑
```

把我们新定义的app加到settings.py中的INSTALL\_APPS中

修改 `mysite/mysite/settings.py`

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'learn',    # 注意添加了这一行
]
```

备注,这一步是干什么呢? 新建的 **app** 如果不加到 **INSTALL\_APPS** 中的话, **django** 就不能自动找到app中的模板文件(app-name/templates/下的文件)和静态文件(app-name/static/中的文件), 后面你会学习到它们分别用来干什么。

### 定义视图函数（访问页面时的内容）

我们在**learn**这个目录中,把**views.py**打开,修改其中的源代码,改成下面的

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("欢迎光临 自强学堂!")
```

第一行引入**HttpResponse**, 它是用来向网页返回内容的, 就像Python中的 **print** 一样, 只不过 **HttpResponse** 是把内容显示到网页上。

我们定义了一个**index()**函数, 第一个参数必须是 **request**, 与网页发来的请求有关, **request** 变量里面包含**get**或**post**的内容, 用户浏览器, 系统等信息在里面（后面会讲, 先了解一下就可以）。

函数返回了一个 **HttpResponse** 对象, 可以经过一些处理, 最终显示几个字到网页上。

那问题来了, 我们访问什么网址才能看到刚才写的这个函数呢? 怎么让网址和函数关联起来呢?

### 定义视图函数相关的URL(网址)（即规定 访问什么网址对应什么内容）

我们打开 `mysite/mysite/urls.py` 这个文件, 修改其中的代码:

```
from django.contrib import admin
from django.urls import path
from learn import views as learn_views # new

urlpatterns = [
    path("", learn_views.index), # new
    path('admin/', admin.site.urls),
]
```

以上都修改并保存后,我们来看一下效果!

在终端上运行 `python3 manage.py runserver` 我们会看到类似下面的信息:

```
tu@mac ~/test/mysite $ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
```



```
You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
```

```
November 10, 2019 - 09:19:50
Django version 2.2.7, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

提示：上面有一些数据库相关的提示，但目前我们没有用到，暂时先忽略他们。

我们打开浏览器,访问 <http://127.0.0.1:8000/>

不出意料的话会看到:



**注意：**如果是在另一台电脑上访问要用 `python manage.py ip:port` 的形式，比如监听所有ip:

```
python manage.py runserver 0.0.0.0:8000
```

监听机器上所有ip 8000端口，访问时用电脑的ip代替 127.0.0.1

参考链接：

(a). <https://docs.djangoproject.com/en/2.2/topics/http/urls/>

[小额赞助，支持作者！](#)

[« Django 基本命令](#)

[Django 视图与网址进阶 »](#)

 CODING

零成本开启敏捷开发 五人以下团队免费

免费体验

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 视图与网址进阶

[« Django 视图与网址](#)  
[Django URL name详解 »](#)

## 一、在网页上做加减法

### 1. 采用 `/add/?a=4&b=5` 这样GET方法进行

```
django-admin.py startproject zqxt_views
cd zqxt_views
python3 manage.py startapp calc
```

自动生成目录大致如下（因不同的 Django 版本有一些差异，如果差异与这篇文章相关，我会主动提出来，没有说的，暂时可以忽略他们之间的差异，后面的教程也是这样做）：

```
zqxt_views
├── calc
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── manage.py
├── zqxt_views
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
```

我们修改一下 **calc/views.py**文件

```
from django.shortcuts import render
from django.http import HttpResponse

def add(request):
    a = request.GET['a']
    b = request.GET['b']
    c = int(a)+int(b)
    return HttpResponse(str(c))
```

注: `request.GET` 类似于一个字典, 更好的办法是用 `request.GET.get('a', 0)` 当没有传递 `a` 的时候默认 `a` 为 0

接着修改 `zqxt_views/urls.py` 文件, 添加一个网址来对应我们刚才新建的视图函数。

```
from django.contrib import admin
from django.urls import path
from calc import views as calc_views # new

urlpatterns = [
```

```
path('add/', calc_views.add, name='add'), # new
path('admin/', admin.site.urls),
]
```

我们打开开发服务器并访问

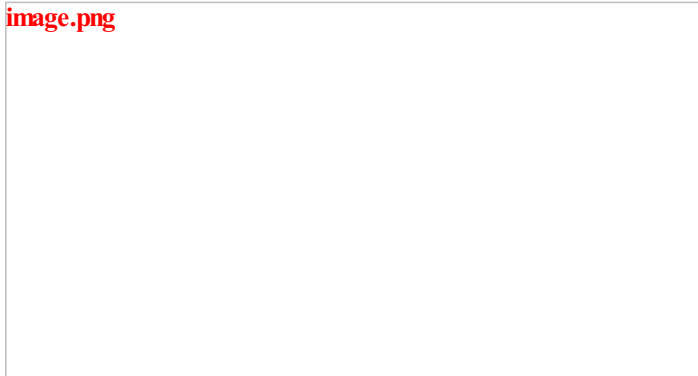
```
python3 manage.py runserver 8002
```

默认端口是 8000，上面使用了自定义端口 8002

如果提示 Error: That port is already in use. 我们可以在后面加上端口号8001, 8888等

打开网址: <http://127.0.0.1:8002/add/> 就可以看到

## MultiValueDictKeyError at /add/



这是因为我们并没有传值进去，我们在后面加上 **?a=4&b=5**，即访问 <http://127.0.0.1:8002/add/?a=4&b=5>

就可以看到网页上显示一个 9，试着改变一下a和b对应的值试试看？

□

□

## 2. 采用 **/add/3/4/** 这样的网址的方式

前面介绍的时候就说过 Django 支持优雅的网址

我们接着修改 **calc/views.py** 文件，再新定义一个 **add2** 函数，原有部分不再贴出

```
def add2(request, a, b):
    c = int(a) + int(b)
    return HttpResponse(str(c))
```

接着修改 **zqxt\_views/urls.py** 文件，再添加一个新的 url

```
path('add/<int:a>/<int:b>/', calc_views.add2, name='add2'),
```

我们再访问 <http://127.0.0.1:8002/add/4/5/> 就可以看到和刚才同样的效果，但是这回网址更优雅了



源代码下载: [zqxt\\_views.zip](#) 示例代码用 Django2.2 创建。

[小额赞助，支持作者！](#)

[« Django 视图与网址  
Django URL name 详解 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 视图与网址进阶

[« Django 视图与网址  
Django URL name 详解 »](#)

### 一、在网页上做加减法

#### 1. 采用 `/add/?a=4&b=5` 这样GET方法进行

```
django-admin.py startproject zqxt_views
cd zqxt_views
python3 manage.py startapp calc
```

自动生成目录大致如下（因不同的 Django 版本有一些差异，如果差异与这篇文章相关，我会主动提出来，没有说的，暂时可以忽略他们之间的差异，后面的教程也是这样做）：

```
zqxt_views
├── calc
│   └── __init__.py
```

```

├── admin.py
├── apps.py
├── migrations
│   └── __init__.py
├── models.py
├── tests.py
├── views.py
├── manage.py
├── zqxt_views
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py

```

我们修改一下 **calc/views.py**文件

```

from django.shortcuts import render
from django.http import HttpResponse

def add(request):
    a = request.GET['a']
    b = request.GET['b']
    c = int(a)+int(b)
    return HttpResponse(str(c))

```

注：**request.GET** 类似于一个字典，更好的办法是用 **request.GET.get('a', 0)** 当没有传递 **a** 的时候默认 **a** 为 0

接着修改 **zqxt\_views/urls.py** 文件，添加一个网址来对应我们刚才新建的视图函数。

```

from django.contrib import admin
from django.urls import path
from calc import views as calc_views # new

urlpatterns = [
    path('add/', calc_views.add, name='add'), # new
    path('admin/', admin.site.urls),
]

```

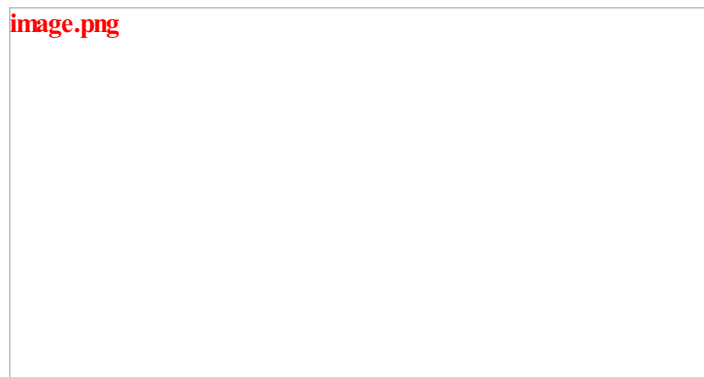
我们打开开发服务器并访问

```
python3 manage.py runserver 8002
```

默认端口是 8000，上面使用了自定义端口 8002  
如果提示 **Error: That port is already in use.** 我们可以在后面加上端口号8001，8888等

打开网址：<http://127.0.0.1:8002/add/> 就可以看到

**MultiValueDictKeyError at /add/**



这是因为我们并没有传值进去，我们在后面加上 `?a=4&b=5`，即访问 <http://127.0.0.1:8002/add/?a=4&b=5> 就可以看到网页上显示一个 9，试着改变一下a和b对应的值试试看？

□

□

## 2. 采用 `/add/3/4/` 这样的网址的方式

前面介绍的时候就说过 Django 支持优雅的网址

我们接着修改 `calc/views.py` 文件，再新定义一个 `add2` 函数，原有部分不再贴出

```
def add2(request, a, b):  
    c = int(a) + int(b)  
    return HttpResponse(str(c))
```

接着修改 `zqxt_views/urls.py` 文件，再添加一个新的 url

```
path('add/<int:a>/<int:b>/', calc_views.add2, name='add2'),
```

我们再访问 <http://127.0.0.1:8002/add/4/5/> 就可以看到和刚才同样的效果，但是这回网址更优雅了

源代码下载: [zqxt\\_views.zip](#) 示例代码用 Django2.2 创建。

[小额赞助，支持作者！](#)

[« Django 视图与网址](#)

[Django URL name 详解 »](#)

 CODING  
CLOUD DEVELOPMENT

零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django URL name详解

[« Django 视图与网址进阶](#)  
[Django 模板 »](#)

我们基于上一节的代码来开始这一节的内容。

上节源代码: [zqxt\\_views.zip](#)

教程中所有的文件, 没有特别说明的, 都是以 **utf8** 格式编码的, 请养成这个习惯。

## 1. 打开 `zqxt_views/urls.py`

```
from django.contrib import admin
from django.urls import path
from calc import views as calc_views # new

urlpatterns = [
    path('add/', calc_views.add, name='add'), # new
    path('add/<int:a>/<int:b>/', calc_views.add2, name='add2'),
    path('admin/', admin.site.urls),
]
```

`url(r'^add/$', calc_views.add, name='add')`, 这里的 **name='add'** 是用来干什么的呢?

简单说, **name** 可以用于在 `templates`, `models`, `views` .....中得到对应的网址, 相当于“**给网址取了个名字**”, 只要这个名字不变, 网址变了也能通过名字获取到。

为了进一步弄清这个问题, 我们先建一个首页的视图和url

## 2. 修改 `calc/views.py`

```
from django.http import HttpResponse
from django.shortcuts import render

def index(request):
    return render(request, 'home.html')
```

...此处省去一些代码

`render` 是渲染模板, 不懂先照着打就好。

## 3. 将 'calc' 这个 app 加入到 `zqxt_views/settings.py` 中

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```

'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',

'calc',
]

```

这样，使用render的时候，Django 会自动找到 INSTALLED\_APPS 中列出的各个 app 下的 templates 中的文件。

小提示，DEBUG=True 的时候，Django 还可以自动找到 各 app 下 static 文件夹中的[静态文件](#)（js，css，图片等资源），方便开发，后面有[专门的章节](#)会讲这些。

4. 我们在 calc 这个 app 中新建一个 templates 文件夹，在templates中新建一个 home.html（关于模板更详细的可以稍后看下一节）

文件 calc/templates/home.html 中写入以下内容（**保存时用 utf8 编码**）

```

<!DOCTYPE html>
<html>
<head>
    <title>自学学堂</title>
</head>
<body>

<a href="/add/4/5/">计算 4+5</a>

</body>
</html>

```

修改 `zqxt_views/urls.py`，添加对应的访问路由

...此处省去一些代码

```

urlpatterns = [
    path('', calc_views.index, name='home'),
    path('add/', calc_views.add, name='add'), # new
    path('add/<int:a>/<int:b>/', calc_views.add2, name='add2'),
    path('admin/', admin.site.urls),
]

```

运行开发服务器 `python3 manage.py runserver`

我们访问 <http://127.0.0.1:8000/> 可以看到



我们计算加法的时候用的是 `/add/4/5/`，**后来需求发生变化**，比如改成 `/4_add_5/`，但在网页中，代码中很多地方都写死的 `/add/4/5/`，比如模板中可能是这么写的

```

<a href="/add/4/5/">计算 4+5</a>

```

如果这样写“死网址”，会使得在改了网址（正则）后，模板（template），视图（views.py，比如用于URL跳转），模



型(models.py, 获取记录访问地址等)用了此网址的, 都必须进行相应的更改, 修改的代价很大, 一不小心, 有的地方没改过来, 就不能用了。

那么有没有更优雅的方式来解决这个问题呢? 当然答案是肯定的。

我们先说一下如何用 Python 代码获取对应的网址(可以用在 views.py, models.py等各种需要转换得到网址的地方):

我们在终端上输入(推荐安装 bpython, 这样Django会用 bpython的 shell)

```
$ python3 manage.py shell

>>> from django.urls import reverse

>>> reverse('add2', args=(4,5))
'/add/4/5/'
>>> reverse('add2', args=(444,555))
'/add/444/555/'
```

**reverse** 接收 **url** 中的 **name** 作为第一个参数, 我们在代码中就可以通过 **reverse()** 来获取对应的网址(这个网址可以用来跳转, 也可以用来计算相关页面的地址), 只要对应的 **url** 的**name**不改, 就不用改代码中的网址。

在网页模板中也是一样, 可以很方便的使用。

不带参数的:

```
{% url 'name' %}
```

带参数的: 参数可以是变量名

```
{% url 'name' 参数 %}
```

例如:

```
<a href="{% url 'add2' 4 5 %}">link</a>
```

上面的代码渲染成最终的页面是

```
<a href="/add/4/5/">link</a>
```

这样就可以通过 {% url 'add2' 4 5 %} 获取到对应的网址 /add/4/5/

当 **urls.py** 进行更改, 前提是不改 **name** (这个参数设定好后不要轻易改), 获取的网址也会动态地跟着变, 比如改成:

```
url(r'^new_add/(\d+)/(\d+)/$', calc_views.add2, name='add2'),
```

注意看重点 **add** 变成了 **new\_add**, 但是后面的 **name='add2'** 没改, 这时 {% url 'add2' 4 5 %} 就会渲染对应的网址成 **/new\_add/4/5/**

用在 **views.py** 或 **models.py** 等地方的 **reverse**函数, 同样会根据 **name** 对应的**url**获取到新的网址。

想要改网址的时候, 修改 **urls.py** 中的正则表达式部分 (**url** 参数第一部分), **name** 不变的前提下, 其它地方都不需要修改。

另外, 比如用户收藏夹中收藏的**URL**是旧的, 如何让以前的 **/add/3/4/**自动跳转到现在新的网址呢?

要知道**Django**不会帮你做这个, 这个**需要自己来写**一个跳转方法:

具体思路是, 在 **views.py** 写一个跳转的函数(已有代码未给出, 以下是新加的代码):

```
from django.http import HttpResponseRedirect
from django.urls import reverse
```


```
def old_add2_redirect(request, a, b):
    return HttpResponseRedirect(
        reverse('add2', args=(a, b))
    )
```

urls.py中:

```
path('add/<int:a>/<int:b>/', calc_views.old_add2_redirect),
path('new_add/<int:a>/<int:b>/', calc_views.add2, name='add2'),
```

这样, 假如用户收藏夹中有 /add/4/5/, 访问时就会自动跳转到新的 /new\_add/4/5/ 了

开始可能觉得直接写网址简单, 但是用多了你一定会发现, 用“死网址”的方法很糟糕。

 [zqxt\\_views\\_for\\_urls\\_name.zip](#) [更新于 2019-11-10 17:48:19] 示例代码用 Django 2.2 创建。

[小额赞助, 支持作者!](#)

[« Django 视图与网址进阶](#)

[Django 模板 »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django URL name详解

[« Django 视图与网址进阶](#)  
[Django 模板 »](#)

我们基于上一节的代码来开始这一节的内容。

上节源代码:  [zqxt\\_views.zip](#)

教程中所有的文件，没有特别说明的，都是以 **utf8** 格式编码的，请养成这个习惯。

### 1. 打开 `zqxt_views/urls.py`

```
from django.contrib import admin
from django.urls import path
from calc import views as calc_views # new

urlpatterns = [
    path('add/', calc_views.add, name='add'), # new
    path('add/<int:a>/<int:b>/', calc_views.add2, name='add2'),
    path('admin/', admin.site.urls),
]
```

`url(r'^add/$', calc_views.add, name='add')`, 这里的 **name='add'** 是用来干什么的呢?

简单说, **name** 可以用于在 `templates`, `models`, `views` .....中得到对应的网址, 相当于“**给网址取了个名字**”, 只要这个名字不变, 网址变了也能通过名字获取到。

为了进一步弄清这个问题, 我们先建一个首页的视图和url

### 2. 修改 `calc/views.py`

```
from django.http import HttpResponse
from django.shortcuts import render

def index(request):
    return render(request, 'home.html')
```

...此处省去一些代码

`render` 是渲染模板, 不懂先照着打就好。

### 3. 将 'calc' 这个 app 加入到 `zqxt_views/settings.py` 中

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'calc',
]
```

这样, 使用`render`的时候, Django 会自动找到 `INSTALLED_APPS` 中列出的各个 app 下的 `templates` 中的文件。

小提示, `DEBUG=True` 的时候, Django 还可以自动找到 各 app 下 `static` 文件夹中的[静态文件](#) (js, css, 图片等资源),

方便开发，后面有[专门的章节](#)会讲这些。

4. 我们在 `calc` 这个 app 中新建一个 `templates` 文件夹，在 `templates` 中新建一个 `home.html`（关于模板更详细的可以稍后看下一节）

文件 `calc/templates/home.html` 中写入以下内容（保存时用 **utf8 编码**）

```
<!DOCTYPE html>
<html>
<head>
    <title>自强学堂</title>
</head>
<body>

<a href="/add/4/5/">计算 4+5</a>

</body>
</html>
```

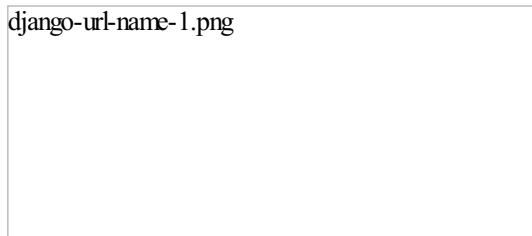
修改 `zqxt_views/urls.py`，添加对应的访问路由

...此处省去一些代码

```
urlpatterns = [
    path('', calc_views.index, name='home'),
    path('add/', calc_views.add, name='add'), # new
    path('add/<int:a>/<int:b>/', calc_views.add2, name='add2'),
    path('admin/', admin.site.urls),
]
```

运行开发服务器 `python3 manage.py runserver`

我们访问 <http://127.0.0.1:8000/> 可以看到



我们计算加法的时候用的是 `/add/4/5/`，后来需求发生变化，比如改成 `/4_add_5/`，但在网页中，代码中很多地方都写死的 `/add/4/5/`，比如模板中可能是这么写的

```
<a href="/add/4/5/">计算 4+5</a>
```

如果这样写“死网址”，会使得在改了网址（正则）后，模板（`template`），视图（`views.py`，比如用于URL跳转），模型（`models.py`，获取记录访问地址等）用了此网址的，都必须进行相应的更改，修改的代价很大，一不小心，有的地方没改过来，就不能用了。

**那么有没有更优雅的方式来解决这个问题呢？当然答案是肯定的。**

我们先说一下如何用 `Python` 代码获取对应的网址（可以用在 `views.py`，`models.py`等各种需要转换得到网址的地方）：

我们在终端上输入(推荐安装 `bpython`，这样 `Django` 会用 `bpython` 的 `shell`)

```
$ python3 manage.py shell
```

```
>>> from django.urls import reverse

>>> reverse('add2', args=(4,5))
'/add/4/5/'
>>> reverse('add2', args=(444,555))
'/add/444/555/'
```

**reverse** 接收 **url** 中的 **name** 作为第一个参数，我们在代码中就可以通过 **reverse()** 来获取对应的网址（这个网址可以用来跳转，也可以用来计算相关页面的地址），只要对应的 **url** 的 **name** 不改，就不用改代码中的网址。

在网页模板中也是一样，可以很方便的使用。

不带参数的：

```
{% url 'name' %}
```

带参数的：参数可以是变量名

```
{% url 'name' 参数 %}
```

例如：

```
<a href="{% url 'add2' 4 5 %}">link</a>
```

上面的代码渲染成最终的页面是

```
<a href="/add/4/5/">link</a>
```

这样就可以通过 `{% url 'add2' 4 5 %}` 获取到对应的网址 `/add/4/5/`

当 **urls.py** 进行更改，前提是不改 **name**（这个参数设定好后不要轻易改），获取的网址也会动态地跟着变，比如改成：

```
url(r'^new_add/(\d+)/(\d+)/$', calc_views.add2, name='add2'),
```

**注意看重点** **add** 变成了 **new\_add**，但是后面的 **name='add2'** 没改，这时 `{% url 'add2' 4 5 %}` 就会渲染对应的网址成 `/new_add/4/5/`

用在 **views.py** 或 **models.py** 等地方的 **reverse** 函数，同样会根据 **name** 对应的 **url** 获取到新的网址。

想要改网址的时候，修改 **urls.py** 中的正则表达式部分（**url** 参数第一部分），**name** 不变的前提下，其它地方都不需要修改。

另外，比如用户收藏夹中收藏的 **URL** 是旧的，如何让以前的 `/add/3/4/` 自动跳转到现在新的网址呢？

要知道 **Django** 不会帮你做这个，这个 **需要自己来写** 一个跳转方法：

具体思路是，在 **views.py** 写一个跳转的函数(已有代码未给出，以下是新加的代码)：

```
from django.http import HttpResponseRedirect
from django.urls import reverse


def old_add2_redirect(request, a, b):
    return HttpResponseRedirect(
        reverse('add2', args=(a, b))
    )
```

**urls.py** 中：

```
path('add/<int:a>/<int:b>/', calc_views.old_add2_redirect),
path('new_add/<int:a>/<int:b>/', calc_views.add2, name='add2'),
```

这样，假如用户收藏夹中有 `/add/4/5/`，访问时就会自动跳转到新的 `/new_add/4/5/` 了

开始可能觉得直接写网址简单，但是用多了你一定会发现，用“死网址”的方法很糟糕。

 [zqxt\\_views\\_for\\_urls\\_name.zip](#) [更新于 2019-11-10 17:48:19] 示例代码用 Django 2.2 创建。

[小额赞助，支持作者！](#)

[« Django 视图与网址进阶](#)

[Django 模板 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 模板

[« Django URL name详解](#)  
[Django 模板进阶 »](#)

在前面的几节中我们都是用简单的 `django.http.HttpResponse` 来把内容显示到网页上。

本节将讲解如何使用渲染模板的方法来显示内容。

这里**强调一点**, 一点要动手敲代码! 不要偷懒, 动手才能学到真东西。

1. 创建一个 `zqxt_tmpl` 项目, 和一个 名称为 `learn` 的应用, 并且

```
django-admin.py startproject zqxt_tmpl
cd zqxt_tmpl
python3 manage.py startapp learn
```

2. 把 `learn` 加入到 `settings.INSTALLED_APPS` 中

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'learn',
]
```

3. 打开 `learn/views.py` 写一个首页的视图

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')
```

4. 在 `learn` 目录下新建一个 `templates` 文件夹, 里面新建一个 `home.html`

**默认配置下, Django 的模板系统会自动找到 app 下面的 templates 文件夹中的模板文件。**

目录的结构是这样的:

```
zqxt_tmpl
├── learn
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── templates
│   │   └── home.html
│   └── tests.py
```

```

├── views.py
├── manage.py
├── zqxt_tmpl
│   ├── __init__.py
│   ├── pycache
│   │   ├── __init__.cpython-37.pyc
│   │   └── settings.cpython-37.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py

```

5 directories, 15 files

## 5. 在 `home.html` 中写一些内容

```

<!DOCTYPE html>
<html>
<head>
    <title>欢迎光临</title>
</head>
<body>
    欢迎光临自强学堂
</body>
</html>

```

## 6. 将视图函数对应到网址，更改 `zqxt_tmpl/urls.py`

```

from django.contrib import admin
from django.urls import path
from learn import views as learn_views # new

urlpatterns = [
    path('', learn_views.home, name="home"), # new
    path('admin/', admin.site.urls),
]

```

## 7. [可选] 创建数据库表

```

$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying sessions.0001_initial... OK

```


创建数据库虽然本节不会用到，但是可以让一些提示消失（提示你要创建数据库之类的）

## 8. 运行开发服务器，看看效果

```
python3 manage.py runserver
```



图和前面有一节的一样，不在放上来了，把代码放上来

 [zqxt\\_tpl\(Django2.2\).zip](#) [更新于 2019-11-10 17:58:46]

模板中的一些循环，条件判断，标签，过滤器等使用请看下一节的内容。

[加餐] 模板补充知识：网站模板的设计，一般的，我们做网站有一些通用的部分，比如 导航，底部，访问统计代码等等

**nav.html, bottom.html, tongji.html**

可以写一个 **base.html** 来包含这些通用文件（**include**）

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}默认标题{% endblock %} - 自强学堂</title>
</head>
<body>

{% include 'nav.html' %}

{% block content %}
<div>这里是默认内容，所有继承自这个模板的，如果不覆盖就显示这里的默认内容。</div>
{% endblock %}

{% include 'bottom.html' %}

{% include 'tongji.html' %}

</body>
</html>
```

如果需要，写足够多的 **block** 以便继承的模板可以重写该部分，**include** 是包含其它文件的内容，就是把一些网页共用的部分拿出来，重复利用，改动的时候也方便一些，还可以把广告代码放在一个单独的**html**中，改动也方便一些，在用到的地方**include**进去。其它的页面继承自 **base.html** 就好了，继承后的模板也可以在 **block** 块中 **include** 其它的模板文件。

比如我们的首页 **home.html**，继承或者说扩展(**extends**)原来的 **base.html**，可以简单这样写，重写部分代码（默认值的那一部分不用改）

```
{% extends 'base.html' %}

{% block title %}欢迎光临首页{% endblock %}

{% block content %}
{% include 'ad.html' %}
这里是首页，欢迎光临
{% endblock %}
```

**注意：**模板一般放在**app**下的**templates**中，**Django**会自动去这个文件夹中找。但假如我们每个**app**的**templates**中都有一个 **index.html**，当我们在**views.py**中使用的时候，直接写一个 **render(request, 'index.html')**，**Django**能不能找到当前 **app** 的 **templates** 文件夹中的 **index.html** 文件夹呢？（答案是不一定能，有可能找错）

**Django 模板查找机制：****Django** 查找模板的过程是在每个 **app** 的 **templates** 文件夹中找（而不只是当前 **app** 中的代码只在当前的 **app** 的 **templates** 文件夹中找）。各个 **app** 的 **templates** 形成一个文件夹列表，**Django** 遍历这个列表，一个个文件夹进行查找，当在某一个文件夹找到的时候就停止，所有的都遍历完了还找不到指定的模板的时候就是 **Template Not Found**（过程类似于**Python**找包）。这样设计有利当然也有弊，有利的是地方是一个**app** 可以用另一个**app**的模板文件，弊是有可能找错了。所以我们使用的时候在 **templates** 中建立一个 **app** 同名的

文件夹，这样就好了。

这就需要把每个app中的 `templates` 文件夹中再建一个 `app` 的名称，仅和该app相关的模板放在 `app/templates/app/` 目录下，

例如：项目 `zqxt` 有两个 `app`，分别为 `tutorial` 和 `tryit`

```
zqxt
├── tutorial
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py
│   ├── templates
│   │   └── tutorial
│   │       ├── index.html
│   │       └── search.html
│   ├── tests.py
│   └── views.py
├── tryit
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py
│   ├── templates
│   │   └── tryit
│   │       ├── index.html
│   │       └── poll.html
│   ├── tests.py
│   └── views.py
├── manage.py
└── zqxt
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

这样，使用的时候，模板就是 `"tutorial/index.html"` 和 `"tryit/index.html"` 这样有app作为名称的一部分，就不会混淆。

模板中的一些循环，条件判断，标签，过滤器等使用请看[Django 模板进阶](#)。

[小额赞助，支持作者！](#)

[« Django URL name 详解](#)

[Django 模板进阶 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云ECS](#)

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 模板

[« Django URL name详解](#)  
[Django 模板进阶 »](#)

在前面的几节中我们都是用简单的 `django.http.HttpResponse` 来把内容显示到网页上。

本节将讲解如何使用渲染模板的方法来显示内容。

这里强调一点，一点要动手敲代码！不要偷懒，动手才能学到真东西。

1. 创建一个 `zqxt_tmpl` 项目，和一个 名称为 `learn` 的应用，并且

```
django-admin.py startproject zqxt_tmpl
cd zqxt_tmpl
python3 manage.py startapp learn
```

2. 把 `learn` 加入到 `settings.INSTALLED_APPS` 中

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'learn',
]
```

3. 打开 `learn/views.py` 写一个首页的视图

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')
```

4. 在 `learn` 目录下新建一个 `templates` 文件夹，里面新建一个 `home.html`

默认配置下，**Django 的模板系统会自动找到 `app` 下面的 `templates` 文件夹中的模板文件。**

目录的结构是这样的：

```
zqxt_tmpl
├── learn
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── templates
│   │   └── home.html
│   ├── tests.py
│   └── views.py
├── manage.py
└── zqxt_tmpl
    ├── __init__.py
    ├── pycache
    │   ├── __init__.cpython-37.pyc
    │   └── settings.cpython-37.pyc
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

5 directories, 15 files

## 5. 在 `home.html` 中写一些内容

```
<!DOCTYPE html>
<html>
<head>
  <title>欢迎光临</title>
</head>
<body>
  欢迎光临自強学堂
</body>
</html>
```

## 6. 将视图函数对应到网址，更改 `zqxt_tmpl/urls.py`

```
from django.contrib import admin
from django.urls import path
from learn import views as learn_views # new

urlpatterns = [
    path('', learn_views.home, name="home"), # new
    path('admin/', admin.site.urls),
]
```

## 7. [可选] 创建数据库表

```
$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
```


```
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0011_update_proxy_permissions... OK
Applying sessions.0001_initial... OK
```

创建数据库虽然本节不会用到，但是可以让一些提示消失（提示你要创建数据库之类的）

## 8. 运行开发服务器，看看效果

```
python3 manage.py runserver
```

图和前面有一节的一样，不在放上来了，把代码放上来

 [zqxz\\_tmpl\(Django2.2\).zip](#) [更新于 2019-11-10 17:58:46]

模板中的一些循环，条件判断，标签，过滤器等使用请看下一节的内容。

[加餐] 模板补充知识：网站模板的设计，一般的，我们做网站有一些通用的部分，比如 **导航**，**底部**，**访问统计代码** 等等

### nav.html, bottom.html, tongji.html

可以写一个 **base.html** 来包含这些通用文件（**include**）

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}默认标题{% endblock %} - 自强学堂</title>
</head>
<body>

{% include 'nav.html' %}

{% block content %}
<div>这里是默认内容，所有继承自这个模板的，如果不覆盖就显示这里的默认内容。</div>
{% endblock %}

{% include 'bottom.html' %}

{% include 'tongji.html' %}

</body>
</html>
```

如果需要，写足够多的 **block** 以便继承的模板可以重写该部分，**include** 是包含其它文件的内容，就是把一些网页共用的部分拿出来，重复利用，改动的时候也方便一些，还可以把广告代码放在一个单独的**html**中，改动也方便一些，在用到的地方**include**进去。其它的页面继承自 **base.html** 就好了，继承后的模板也可以在 **block** 块中 **include** 其它的模板文件。

比如我们的首页 **home.html**，继承或者说扩展(**extends**)原来的 **base.html**，可以简单这样写，重写部分代码（默认值的那一部分不用改）

```
{% extends 'base.html' %}

{% block title %}欢迎光临首页{% endblock %}

{% block content %}
{% include 'ad.html' %}
```

```
这里是首页，欢迎光临
{% endblock %}
```

**注意：**模板一般放在app下的templates中，Django会自动去这个文件夹中找。但假如我们每个app的templates中都有一个index.html，当我们在views.py中使用的时候，直接写一个render(request, 'index.html')，Django能不能找到当前app的templates文件夹中的index.html文件呢？（答案是不一定能，有可能找错）

**Django 模板查找机制：**Django 查找模板的过程是在每个app的templates文件夹中找（而不只是当前app中的代码只在当前的app的templates文件夹中找）。各个app的templates形成一个文件夹列表，Django遍历这个列表，一个个文件夹进行查找，当在某一个文件夹找到的时候就停止，所有的都遍历完了还找不到指定的模板的时候就是Template Not Found（过程类似于Python找包）。这样设计有利当然也有弊，有利的是地方是一个app可以用另一个app的模板文件，弊是有可能会找错了。所以我们使用的时候在templates中建立一个app同名的文件夹，这样就好了。

这就需要把每个app中的templates文件夹中再建一个app的名称，仅和该app相关的模板放在app/templates/app/目录下面，

例如：项目zqxt有两个app，分别为tutorial和tryit

```
zqxt
├── tutorial
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py
│   ├── templates
│   │   └── tutorial
│   │       ├── index.html
│   │       └── search.html
│   ├── tests.py
│   └── views.py
├── tryit
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py
│   ├── templates
│   │   └── tryit
│   │       ├── index.html
│   │       └── poll.html
│   ├── tests.py
│   └── views.py
├── manage.py
└── zqxt
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

这样，使用的时候，模板就是 "tutorial/index.html" 和 "tryit/index.html" 这样有app作为名称的一部分，就不会混淆。

模板中的一些循环，条件判断，标签，过滤器等使用请看[Django 模板进阶](#)。

[小额赞助，支持作者！](#)

[« Django URL name详解](#)

[Django 模板进阶 »](#)



↑上方为自学学堂赞助商，非常荣幸有他们支持自学学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 模板进阶

[« Django 模板](#)  
[Django 模型 \(数据库\) »](#)

本节主要讲 Django模板中的循环, 条件判断, 常用的标签, 过滤器的使用。

1. 列表, 字典, 类的实例的使用
2. 循环: 迭代显示列表, 字典等中的内容
3. 条件判断: 判断是否显示该内容, 比如判断是手机访问, 还是电脑访问, 给出不一样的代码。
4. 标签: for, if这样的功能都是标签。
5. 过滤器: 管道符号后面的功能, 比如{{ var|length }}, 求变量长度的 length 就是一个过滤器。

如果需要将一个或多个变量共享给多个网页或者所有网页使用, 比如在网页上显示来访者的IP, 这个可以使用 Django 上下文渲染器 来做。

### 实例一, 显示一个基本的字符串在网页上

views.py

```
# -*- coding: utf-8 -*-
from django.shortcuts import render

def home(request):
    string = u"我在自强学堂学习Django, 用它来建网站"
    return render(request, 'home.html', {'string': string})
```

在视图中我们传递了一个字符串名称是 string 到模板 home.html, 在模板中这样使用它:

home.html

```
{{ string }}
```



下载: [zqxt\\_tmpl\(Django2.2\).zip](#) (项目名: zqxt\_tmpl, 修改 learn/views.py 和 learn/templates/home.html 来测试后面的示例)

### 实例二, 讲解了基本的 for 循环 和 List 内容的显示

views.py

```
def home(request):
    TutorialList = ["HTML", "CSS", "jQuery", "Python", "Django"]
    return render(request, 'home.html', {'TutorialList': TutorialList})
```

在视图中我们传递了一个List到模板 home.html, 在模板中这样使用它:

home.html

教程列表:

```
{% for i in TutorialList %}
{{ i }}
{% endfor %}
```

for 循环要有一个结束标记，上面的代码假如我们对应的是首页的网址（自己修改urls.py），显示在网页上就是：

简单总结一下：一般的变量之类的用 `{{ }}`（变量），功能类的，比如循环，条件判断是用 `{% %}`（标签）

**实例三，显示字典中内容：**

views.py

```
def home(request):
    info_dict = {'site': u'自强学堂', 'content': u'各种IT技术教程'}
    return render(request, 'home.html', {'info_dict': info_dict})
```

home.html

站点: `{{ info_dict.site }}` 内容: `{{ info_dict.content }}`

在模板中取字典的键是用点`info_dict.site`，而不是Python中的 `info_dict['site']`，效果如下：

还可以这样遍历字典：

```
{% for key, value in info_dict.items %}
    {{ key }}: {{ value }}
{% endfor %}
```

其实就是遍历这样一个 List: `[('site', u'自强学堂'), ('content', u'各种IT技术教程')]`

**实例四，在模板进行 条件判断和 for 循环的详细操作：**

views.py

```
def home(request):
    List = map(str, range(100))# 一个长度为100的 List
    return render(request, 'home.html', {'List': List})
```

假如我们想用逗号将这些元素连接起来：

home.html

```
{% for item in List %}
    {{ item }},
{% endfor %}
```

效果如下：

我们会发现最后一个元素后面也有一个逗号，这样肯定不爽，如果判断是不是遍历到了最后一个元素了呢？

用变量 `forloop.last` 这个变量，如果是最后一项其为真，否则为假，更改如下：

```
{% for item in List %}
    {{ item }}{% if not forloop.last %},{% endif %}
```



```
{% endfor %}
```

在for循环中还有很多有用的东西，如下：

| 变量                         | 描述                                      |
|----------------------------|---|
| <b>forloop.counter</b>     | 索引从 1 开始算                               |
| <b>forloop.counter0</b>    | 索引从 0 开始算                               |
| <b>forloop.revcounter</b>  | 索引从最大长度到 1                              |
| <b>forloop.revcounter0</b> | 索引从最大长度到 0                              |
| <b>forloop.first</b>       | 当遍历的元素为第一项时为真                           |
| <b>forloop.last</b>        | 当遍历的元素为最后一项时为真                          |
| <b>forloop.parentloop</b>  | 用在嵌套的 for 循环中，<br>获取上一层 for 循环的 forloop |

当列表中可能为空值时用 for empty

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% empty %}
    <li>抱歉，列表为空</li>
{% endfor %}
</ul>
```

**实例五，模板上得到视图对应的网址：**

**前面讲过，学会了的可以跳过**

```
# views.py
def add(request, a, b):
    c = int(a) + int(b)
    return HttpResponse(str(c))

# urls.py
urlpatterns = patterns('',
    path('add/<int:a>/<int:b>/', calc_views.add2, name='add'),
)

# template html
{% url 'add' 4 5 %}
```

这样网址上就会显示出：/add/4/5/ 这个网址，假如我们以后修改 urls.py 中的

```
'add/<int:a>/<int:b>/'
```

这一部分，改成另的，比如：

```
'jia/<int:a>/<int:b>/'
```

这样，我们不需要再次修改模板，当再次访问的时候，网址会自动变成 /jia/4/5/

还可以使用 as 语句将内容取别名（相当于定义一个变量），多次使用（但视图名称到网址转换只进行了一次）

```
{% url 'some-url-name' arg arg2 as the_url %}
```

```
<a href="{{ the_url }}">链接到: {{ the_url }}</a>
```

### 实例六，模板中的逻辑操作：

6.1、 `=, !=, >=, <=, >, <` 这些比较都可以在模板中使用，比如：

```
{% if var >= 90 %}  
成绩优秀，自强学堂你没少去吧！学得不错  
{% elif var >= 80 %}  
成绩良好  
{% elif var >= 70 %}  
成绩一般  
{% elif var >= 60 %}  
需要努力  
{% else %}  
不及格啊，大哥！多去自强学堂学习啊！  
{% endif %}
```

（注意：比较符号前后必须有至少一个空格！）

`and, or, not, in, not in` 也可以在模板中使用

假如我们判断 `num` 是不是在 0 到 100 之间：

```
{% if num <= 100 and num >= 0 %}  
num在0到100之间  
{% else %}  
数值不在范围之内！  
{% endif %}
```

假如我们判断 `'zqiangxuetang'` 不在一个列表变量 `List` 中：

```
{% if 'zqiangxuetang' in List %}  
自强学堂在名单中  
{% endif %}
```

### 实例七，模板中 获取当前网址，当前用户等：

建议在 `views.py` 中用的 [render](#) 函数，而不是 [render\\_to\\_response](#)，这样在 模板中我们就可以获取到 `request` 变量。

#### 7.1 获取当前用户：

```
{{ request.user }}
```

如果登陆就显示内容，不登陆就不显示内容：

```
{% if request.user.is_authenticated %}  
    {{ request.user.username }}, 您好!  
{% else %}  
    请登陆，这里放登陆链接  
{% endif %}
```

#### 7.2.1 获取当前网址：

```
{{ request.path }}
```

#### 7.2.2 获取当前 GET 参数：

```
{{ request.GET.urlencode }}
```

#### 7.2.3 合并到一起用的一个例子：

```
<a href="{{ request.path }}?{{ request.GET.urlencode }}&delete=1">当前网址加参数 delete</a>
```

比如我们可以判断 `delete` 参数是不是 1 来删除当前的页面内容。

完整的内容参考官方文档: <https://docs.djangoproject.com/en/2.2/ref/templates/builtins/>

[小额赞助, 支持作者!](#)

[« Django 模板](#)

[Django 模型 \(数据库\) »](#)



↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 模板进阶

[« Django 模板](#)

[Django 模型 \(数据库\) »](#)

本节主要讲 Django模板中的循环, 条件判断, 常用的标签, 过滤器的使用。

1. 列表, 字典, 类的实例的使用
2. 循环: 迭代显示列表, 字典等中的内容
3. 条件判断: 判断是否显示该内容, 比如判断是手机访问, 还是电脑访问, 给出不一样的代码。
4. 标签: for, if这样的功能都是标签。

5. 过滤器：管道符号后面的功能，比如`{{ var|length }}`，求变量长度的 `length` 就是一个过滤器。

如果要将一个或多个变量共享给多个网页或者所有网页使用，比如在网页上显示来访者的IP，这个可以使用 Django 上下文渲染器 来做。

### 实例一，显示一个基本的字符串在网页上

views.py

```
# -*- coding: utf-8 -*-
from django.shortcuts import render


def home(request):
    string = u"我在自强学堂学习Django，用它来建网站"
    return render(request, 'home.html', {'string': string})
```

在视图中我们传递了一个字符串名称是 `string` 到模板 `home.html`，在模板中这样使用它：

home.html

```
{{ string }}
```



下载： [zqxt\\_tmpl\(Django2.2\).zip](#) （项目名：zqxt\_tmpl, 修改 `learn/views.py` 和 `learn/templates/home.html` 来测试后面的示例）

### 实例二，讲解了基本的 for 循环 和 List 内容的显示

views.py

```
def home(request):
    TutorialList = ["HTML", "CSS", "jQuery", "Python", "Django"]
    return render(request, 'home.html', {'TutorialList': TutorialList})
```

在视图中我们传递了一个List到模板 `home.html`，在模板中这样使用它：

home.html

```
教程列表：
{% for i in TutorialList %}
{{ i }}
{% endfor %}
```

for 循环要有一个结束标记，上面的代码假如我们对应的是首页的网址（自己修改`urls.py`），显示在网页上就是：



简单总结一下：一般的变量之类的用 `{{ }}`（变量），功能类的，比如循环，条件判断是用 `{% %}`（标签）

### 实例三，显示字典中内容：

views.py

```
def home(request):
    info_dict = {'site': u'自强学堂', 'content': u'各种IT技术教程'}
    return render(request, 'home.html', {'info_dict': info_dict})
```

home.html

```
站点: {{ info_dict.site }} 内容: {{ info_dict.content }}
```

在模板中取字典的键是用点`info_dict.site`，而不是Python中的 `info_dict['site']`，效果如下：

还可以这样遍历字典：

```
{% for key, value in info_dict.items %}
    {{ key }}: {{ value }}
{% endfor %}
```

其实就是遍历这样一个 List: `[('site', u'自强学堂'), ('content', u'各种IT技术教程')]`

**实例四，在模板进行条件判断和 for 循环的详细操作：**

**views.py**

```
def home(request):
    List = map(str, range(100))# 一个长度为100的 List
    return render(request, 'home.html', {'List': List})
```

假如我们想用逗号将这些元素连接起来：

**home.html**

```
{% for item in List %}
    {{ item }},
{% endfor %}
```

效果如下：

我们会发现最后一个元素后面也有一个逗号，这样肯定不爽，如果判断是不是遍历到了最后一个元素了呢？

用变量 `forloop.last` 这个变量，如果是最后一项其为真，否则为假，更改如下：

```
{% for item in List %}
    {{ item }}{% if not forloop.last %},{% endif %}
{% endfor %}
```

在for循环中还有很多有用的东西，如下：

| 变量                         | 描述                                      |
|----------------------------|---|
| <b>forloop.counter</b>     | 索引从 1 开始算                               |
| <b>forloop.counter0</b>    | 索引从 0 开始算                               |
| <b>forloop.revcounter</b>  | 索引从最大长度到 1                              |
| <b>forloop.revcounter0</b> | 索引从最大长度到 0                              |
| <b>forloop.first</b>       | 当遍历的元素为第一项时为真                           |
| <b>forloop.last</b>        | 当遍历的元素为最后一项时为真                          |
| <b>forloop.parentloop</b>  | 用在嵌套的 for 循环中，<br>获取上一层 for 循环的 forloop |

当列表中可能为空值时用 `for empty`

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
```

```
{% empty %}
    <li>抱歉，列表为空</li>
{% endfor %}
</ul>
```

### 实例五，模板上得到视图对应的网址：

前面讲过，学会了的可以跳过

```
# views.py
def add(request, a, b):
    c = int(a) + int(b)
    return HttpResponse(str(c))

# urls.py
urlpatterns = patterns('',
    path('add/<int:a>/<int:b>/', calc_views.add2, name='add'),
)

# template html
{% url 'add' 4 5 %}
```

这样网址上就会显示出：/add/4/5/ 这个网址，假如我们以后修改 `urls.py` 中的

```
'add/<int:a>/<int:b>/'
```

这一部分，改成另的，比如：

```
'jia/<int:a>/<int:b>/'
```

这样，我们不需要再次修改模板，当再次访问的时候，网址会自动变成 /jia/4/5/

还可以使用 `as` 语句将内容取别名（相当于定义一个变量），多次使用（但视图名称到网址转换只进行了一次）

```
{% url 'some-url-name' arg arg2 as the_url %}

<a href="{{ the_url }}">链接到: {{ the_url }}</a>
```

### 实例六，模板中的逻辑操作：

6.1、`==, !=, >=, <=, >, <` 这些比较都可以在模板中使用，比如：

```
{% if var >= 90 %}
成绩优秀，自强学堂你没少去吧！学得不错
{% elif var >= 80 %}
成绩良好
{% elif var >= 70 %}
成绩一般
{% elif var >= 60 %}
需要努力
{% else %}
不及格啊，大哥！多去自强学堂学习啊！
{% endif %}
```

（注意：比较符号前后必须有至少一个空格！）

`and, or, not, in, not in` 也可以在模板中使用

假如我们判断 `num` 是不是在 0 到 100 之间：

```
{% if num <= 100 and num >= 0 %}
num在0到100之间
{% else %}
数值不在范围之内！
```

```
{% endif %}
```

假如我们判断 'ziqiangxuetang' 不在一个列表变量 List 中：

```
{% if 'ziqiangxuetang' in List %}
自强学堂在名单中
{% endif %}
```

### 实例七，模板中 获取当前网址，当前用户等：

建议在 views.py 中用的 [render](#) 函数，而不是 [render\\_to\\_response](#)，这样在 模板中我们就可以获取到 request 变量。

#### 7.1 获取当前用户：

```
{{ request.user }}
```

如果登陆就显示内容，不登陆就不显示内容：

```
{% if request.user.is_authenticated %}
    {{ request.user.username }}，您好！
{% else %}
    请登陆，这里放登陆链接
{% endif %}
```

#### 7.2.1 获取当前网址：

```
{{ request.path }}
```

#### 7.2.2 获取当前 GET 参数：

```
{{ request.GET.urlencode }}
```

#### 7.2.3 合并到一起用的一个例子：

```
<a href="{{ request.path }}?{{ request.GET.urlencode }}&delete=1">当前网址加参数 delete</a>
```

比如我们可以判断 delete 参数是不是 1 来删除当前的页面内容。

完整的内容参考官方文档：<https://docs.djangoproject.com/en/2.2/ref/templates/builtins/>

[小额赞助，支持作者！](#)

[« Django 模板](#)

[Django 模型（数据库） »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 模型（数据库）

[« Django 模板进阶](#)  
[Django QuerySet API »](#)

Django 模型是与数据库相关的, 与数据库相关的代码一般写在 `models.py` 中, Django 支持 `sqlite3`, `MySQL`, `PostgreSQL` 等数据库, 只需要在 `settings.py` 中配置即可, 不用更改 `models.py` 中的代码, 丰富的 API 极大的方便了使用。

本节的最后有源代码, 但建议初学者按照代码操作, 有问题再下载源代码和自己的代码进行比较。

多动手, 这是学习编程最好的方法!

### 1. 新建项目和应用

```
django-admin.py startproject learn_models # 新建一个项目
cd learn_models # 进入到该项目的文件夹
django-admin.py startapp people # 新建一个 people 应用 (app)
```

补充: 新建 app 也可以用 `python3 manage.py startapp people`, 需要指出的是, `django-admin.py` 是安装 Django 后多出的一个命令, 并不是运行的当前目录下的 `django-admin.py` (当前目录下也没有), 但创建项目会生成一个 `manage.py` 文件。

那 `project` 和 `app` 什么关系呢?

一个项目一般包含多个应用, 一个应用也可以用在多个项目中。

### 2. 添加应用

将我们新建的应用 (`people`) 添加到 `settings.py` 中的 `INSTALLED_APPS` 中, 也就是告诉 Django 有这么一个应用。

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'people',
]
```

### 3. 修改 `models.py`

我们打开 `people/models.py` 文件, 修改其中的代码如下:

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()
```

我们新建了一个 `Person` 类, 继承自 `models.Model`, 一个人有姓名和年龄。

这里用到了两种 **Field**, 更多 **Field** 类型可以参考教程最后的链接。

### 4. 创建数据表



我们来同步一下数据库（我们使用默认的数据库 **SQLite3**，无需配置）

先 **cd** 进入 **manage.py** 所在的那个文件夹下，输入下面的命令

#### 4.1 第一步，生成迁移文件

生成的 **python migration** 文件是描述数据库结构变化的

```
python3 manage.py makemigrations
```

image.png

需要记住，这时候，数据库还没真正变化，只是生成了描述数据库变化的文件。

感兴趣的同学可以打开 生成的文件，在 **migrations** 文件夹中，内容如下（目前我们不需要看懂每一行，大致看一看，知道这个文件是做什么的就行）

```
# Generated by Django 2.2.7 on 2019-11-24 05:57

from django.db import migrations, models

class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Person',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=30)),
                ('age', models.IntegerField()),
            ],
        ),
    ]
```

好了，我们继续下一步。

#### 4.2 将结构变化应用到数据库

```
python3 manage.py migrate
```

image.png

我们会看到，Django将一系列变化应用到了数据库中。

细心的读者可能会发现，除了 `people.0001_initial` 那一条，还有很多 `django` 内置的应用的表，他们是用户及用户认证等相关的，我们可以先不用管它，不影响本节的学习。

那接下来如何使用呢？

## 5. 使用 Django 提供的 QuerySet API

Django提供了丰富的API, 下面演示如何使用它。

[可选]推荐安装 `bpython` 或 `ipython`，它们会使用你在终端上调试更加方便。

```
$ python3 manage.py shell

>>> from people.models import Person
>>> Person.objects.create(name="WeizhongTu", age=24)
<Person: Person object>
>>>
```

我们新建了一个用户WeizhongTu 那么如何从数据库是查询到它呢？

```
>>> Person.objects.get(name="WeizhongTu")
<Person: Person object>
>>>
```

我们用了一个 `.objects.get()` 方法查询出来符合条件的对象，但是大家注意到了没有，查询结果中显示<Person: **Person object**>，这里并没有显示出与WeizhongTu的相关信息，如果用户多了就无法知道查询出来的到底是谁，查询结果是否正确，我们重新修改一下 `people/models.py`

**name** 和 **age** 等字段中不能有 `__`（双下划线 在Django QuerySet API中有特殊含义（用于关系，包含，不区分大小写，以什么开头或结尾，日期的大于小于，正则等）

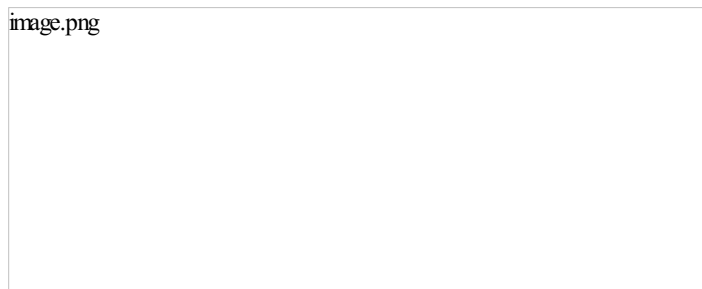
也不能有Python中的关键字，**name** 是合法的，**student\_name** 也合法，但是**student\_\_name**不合法。**try**, **class**, **continue** 也不合法，因为它是Python的关键字( `import keyword; print(keyword.kwlist)` 可以打出所有的关键字)

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()

    def __str__(self):
        return self.name
```

按 `CTRL + C` 退出当前的 `Python shell`, 重复上面的操作，我们就可以看到:



**新建一个对象的方法有以下几种：**

1. `Person.objects.create(name=name,age=age)`
2. `p = Person(name="WZ", age=23)`

```
p.save()
```

```
3. p = Person(name='TWZ')
```

```
p.age = 23
```

```
p.save()
```

```
4. Person.objects.get_or_create(name='WZT', age=23)
```

这种方法是防止重复很好的方法，但是速度要相对慢些，返回一个元组，第一个为Person对象，第二个为True或False, 新建时返回的是True, 已经存在时返回False.

### 获取对象有以下方法：

```
1. Person.objects.all()
```

```
2. Person.objects.all()[0:10] 切片操作，获取10个人，不支持负索引，切片可以节约内存
```

```
3. Person.objects.get(name=name)
```

get是用来获取一个对象的，如果需要获取满足条件的一些人，就要用到filter

```
4. Person.objects.filter(name='abc') # 等于Person.objects.filter(name__exact='abc') 名称严格等于 'abc' 的人
```

```
5. Person.objects.filter(name__iexact='abc') # 名称为 abc 但是不区分大小写，可以找到 ABC, Abc, aBC，这些都符合条件
```

```
6. Person.objects.filter(name__contains='abc') # 名称中包含 'abc'的人
```

```
7. Person.objects.filter(name__icontains='abc') # 名称中包含 'abc'，且abc不区分大小写
```

```
8. Person.objects.filter(name__regex='^abc') # 正则表达式查询
```

```
9. Person.objects.filter(name__iregex='^abc') # 正则表达式不区分大小写
```

filter是找出满足条件的，当然也有排除符合某条件的

```
10. Person.objects.exclude(name__contains='WZ') # 排除包含 WZ 的Person对象
```

```
11. Person.objects.filter(name__contains='abc').exclude(age=23) # 找出名称含有abc, 但是排除年龄是23岁的
```

### 源代码下载：

 [learn\\_models\\_Django2.2.zip](#)

### 参考文档：

更多相关内容：<http://code.ziqiangxuetang.com/django/django-queryset-api.html>

Django models 官方教程：<https://docs.djangoproject.com/en/2.2/topics/db/models/>

Fields相关官方文档：<https://docs.djangoproject.com/en/2.2/ref/models/fields/>

[小额赞助，支持作者！](#)  
[« Django 模板进阶](#)  
[Django QuerySet API »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信，随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 模型（数据库）

[« Django 模板进阶](#)  
[Django QuerySet API »](#)

Django 模型是与数据库相关的，与数据库相关的代码一般写在 `models.py` 中，Django 支持 `sqlite3`, `MySQL`, `PostgreSQL`等数据库，只需要在`settings.py`中配置即可，不用更改`models.py`中的代码，丰富的API极大的方便了使用。

本节的最后有源代码，但建议初学者按照代码操作，有问题再下载源代码和自己的代码进行比较。

多动手，这是学习编程最好的方法！

### 1. 新建项目和应用

```
django-admin.py startproject learn_models # 新建一个项目
cd learn_models # 进入到该项目的文件夹
django-admin.py startapp people # 新建一个 people 应用 (app)
```

补充：新建app也可以用 `python3 manage.py startapp people`，需要指出的是，`django-admin.py` 是安装Django后多出的一个命令，并不是运行的当前目录下的`django-admin.py`（当前目录下也没有），但创建项目会生成一个 `manage.py` 文件。

那project和app什么关系呢？

一个项目一般包含多个应用，一个应用也可以用在多个项目中。

## 2. 添加应用

将我们新建的应用（people）添加到 settings.py 中的 INSTALLED\_APPS 中，也就是告诉 Django 有这么一个应用。

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'people',
]
```

## 3. 修改 models.py

我们打开 people/models.py 文件，修改其中的代码如下：

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()
```

我们新建了一个 Person 类，继承自 models.Model，一个人有姓名和年龄。

这里用到了两种 Field，更多 Field 类型可以参考教程最后的链接。

## 4. 创建数据表

我们来同步一下数据库（我们使用默认的数据库 SQLite3，无需配置）

先 cd 进入 manage.py 所在的那个文件夹下，输入下面的命令

### 4.1 第一步，生成迁移文件

生成的 python migration 文件是描述数据库结构变化的

```
python3 manage.py makemigrations
```

image.png

需要记住，这时候，数据库还没真正变化，只是生成了描述数据库变化的文件。

感兴趣的同学可以打开生成的文件，在 migrations 文件夹中，内容如下（目前我们不需要看懂每一行，大致看一看，知道这个文件是做什么的就行）

```
# Generated by Django 2.2.7 on 2019-11-24 05:57

from django.db import migrations, models

class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Person',
```

```

        fields=[
            ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
            ('name', models.CharField(max_length=30)),
            ('age', models.IntegerField()),
        ],
    ),
]

```

好了，我们继续下一步。

## 4.2 将结构变化应用到数据库

```
python3 manage.py migrate
```



我们会看到，Django将一系列变化应用到了数据库中。

细心的读者可能会发现，除了 `people.0001_initial` 那一条，还有很多 `django` 内置的应用的表，他们是用户及用户认证等相关的，我们可以先不用管它，不影响本节的学习。

那接下来如何使用呢？

## 5. 使用 Django 提供的 QuerySet API

Django提供了丰富的API, 下面演示如何使用它。

[可选]推荐安装 `bpython` 或 `ipython`，它们会使用你在终端上调试更加方便。

```
$ python3 manage.py shell
```

```

>>> from people.models import Person
>>> Person.objects.create(name="WeizhongTu", age=24)
<Person: Person object>
>>>

```

我们新建了一个用户WeizhongTu 那么如何从数据库是查询到它呢？

```

>>> Person.objects.get(name="WeizhongTu")
<Person: Person object>
>>>

```

我们用了一个 `.objects.get()` 方法查询出来符合条件的对象，但是大家注意到了没有，查询结果中显示 `<Person: Person object>`，这里并没有显示出与WeizhongTu的相关信息，如果用户多了就无法知道查询出来的到底是谁，查询结果是否正确，我们重新修改一下 `people/models.py`

`name` 和 `age` 等字段中不能有 `__`（双下划线 在Django QuerySet API中有特殊含义（用于关系，包含，不区分大小写，以什么开头或

结尾，日期的大于小于，正则等)

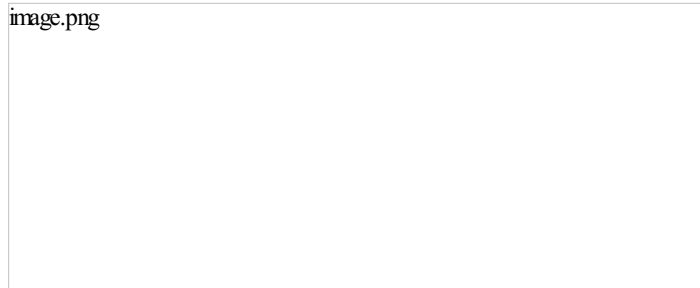
也不能有Python中的关键字，`name` 是合法的，`student_name` 也合法，但是`student__name`不合法。`try`, `class`, `continue` 也不合法，因为它是Python的关键字(`import keyword; print(keyword.kwlist)` 可以打出所有的关键字)

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()

    def __str__(self):
        return self.name
```

按 CTRL + C 退出当前的 Python shell, 重复上面的操作，我们就可以看到:



新建一个对象的方法有以下几种:

1. `Person.objects.create(name=name, age=age)`
2. `p = Person(name="WZ", age=23)`  
`p.save()`
3. `p = Person(name="TWZ")`  
`p.age = 23`  
`p.save()`
4. `Person.objects.get_or_create(name="WZT", age=23)`

这种方法是防止重复很好的方法，但是速度要相对慢些，返回一个元组，第一个为Person对象，第二个为True或False, 新建时返回的是True, 已经存在时返回False.

获取对象有以下方法:

1. `Person.objects.all()`
2. `Person.objects.all()[0:10]` 切片操作，获取10个人，不支持负索引，切片可以节约内存
3. `Person.objects.get(name=name)`

`get`是用来获取一个对象的，如果需要获取满足条件的一些人，就要用到`filter`

4. `Person.objects.filter(name="abc")` # 等于`Person.objects.filter(name__exact="abc")` 名称严格等于 "abc" 的人
5. `Person.objects.filter(name__iexact="abc")` # 名称为 abc 但是不区分大小写，可以找到 ABC, Abc, aBC，这些都符合条件

6. `Person.objects.filter(name__contains="abc")` # 名称中包含 "abc"的人
7. `Person.objects.filter(name__icontains="abc")` #名称中包含 "abc", 且abc不区分大小写
8. `Person.objects.filter(name__regex="^abc")` # 正则表达式查询
9. `Person.objects.filter(name__iregex="^abc")` # 正则表达式不区分大小写

`filter`是找出满足条件的, 当然也有排除符合某条件的

10. `Person.objects.exclude(name__contains="WZ")` # 排除包含 WZ 的Person对象
11. `Person.objects.filter(name__contains="abc").exclude(age=23)` # 找出名称含有abc, 但是排除年龄是23岁的

源代码下载:

 [learn\\_models Django2.2.zip](#)

参考文档:

更多相关内容: <http://code.ziqiangxuetang.com/django/django-queryset-api.html>

Django models 官方教程: <https://docs.djangoproject.com/en/2.2/topics/db/models/>

Fields相关官方文档: <https://docs.djangoproject.com/en/2.2/ref/models/fields/>

小额赞助, 支持作者!  
« Django 模板进阶  
Django QuerySet API »

 CODING  
LESSON RECOMMEND

零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django QuerySet API

[« Django 模型 \(数据库\)](#)  
[Django QuerySet 进阶 »](#)

[Django 模型](#)中我们学习了一些基本的创建与查询。这里专门来讲一下数据库接口相关的接口 (QuerySet API), 当然您也可以选择暂时跳过此节, 如果以后用到数据库相关的时候再看也是可以的。

从数据库中查询出来的结果一般是一个集合, 这个集合叫做 QuerySet。

文中的例子大部分是基于这个 `blog/models.py`

```
from django.db import models

class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def __str__(self):
        return self.name

class Author(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField()

    def __str__(self):
        return self.name

class Entry(models.Model):
    blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    mod_date = models.DateField()
    authors = models.ManyToManyField(Author)
    n_comments = models.IntegerField()
    n_pingbacks = models.IntegerField()
    rating = models.IntegerField()

    def __str__(self):
        return self.headline
```

## 1. QuerySet 创建对象的方法

```
>>> from blog.models import Blog
>>> b = Blog(name='Beatles Blog', tagline='All the latest Beatles news.')
>>> b.save()
```

总之, 一共有四种方法

# 方法 1

```
Author.objects.create(name="WeizhongTu", email="tuweizhong@163.com")
```

# 方法 2

```
twz = Author(name="WeizhongTu", email="tuweizhong@163.com")
twz.save()
```

# 方法 3

```
twz = Author()
twz.name="WeizhongTu"
```

```
twz.email="tuweizhong@163.com"
twz.save()
```

```
# 方法 4, 首先尝试获取, 不存在就创建, 可以防止重复
Author.objects.get_or_create(name="WeizhongTu", email="tuweizhong@163.com")
# 返回值(object, True/False)
```

备注: 前三种方法返回的都是对应的 **object**, 最后一种方法返回的是一个元组, (**object, True/False**), 创建时返回 **True**, 已经存在时返回 **False**

当有一对多, 多对一, 或者多对多的关系的时候, 先把相关的对象查询出来

```
>>> from blog.models import Entry
>>> entry = Entry.objects.get(pk=1)
>>> cheese_blog = Blog.objects.get(name="Cheddar Talk")
>>> entry.blog = cheese_blog
>>> entry.save()
```

## 2. 获取对象的方法 (上一篇的部分代码)

```
Person.objects.all() # 查询所有
Person.objects.all()[:10] 切片操作, 获取10个人, 不支持负索引, 切片可以节约内存, 不支持负索引, 后面有相应解决办法, 第7条
Person.objects.get(name="WeizhongTu") # 名称为 WeizhongTu 的一条, 多条会报错
```

get是用来获取一个对象的, 如果需要获取满足条件的一些人, 就要用到filter

```
Person.objects.filter(name="abc") # 等于Person.objects.filter(name__exact="abc") 名称严格等于 "abc" 的人
Person.objects.filter(name__iexact="abc") # 名称为 abc 但是不区分大小写, 可以找到 ABC, Abc, aBC, 这些都符合条件
```

```
Person.objects.filter(name__contains="abc") # 名称中包含 "abc"的人
Person.objects.filter(name__icontains="abc") # 名称中包含 "abc", 且abc不区分大小写
```

```
Person.objects.filter(name__regex="^abc") # 正则表达式查询
Person.objects.filter(name__iregex="^abc") # 正则表达式不区分大小写
```

# filter是找出满足条件的, 当然也有排除符合某条件的

```
Person.objects.exclude(name__contains="WZ") # 排除包含 WZ 的Person对象
Person.objects.filter(name__contains="abc").exclude(age=23) # 找出名称含有abc, 但是排除年龄是23岁的
```

## 3. 删除符合条件的结果

和上面类似, 得到满足条件的结果, 然后 **delete** 就可以(危险操作, 正式场合操作务必谨慎), 比如:

```
Person.objects.filter(name__contains="abc").delete() # 删除 名称中包含 "abc"的人
```

如果写成

```
people = Person.objects.filter(name__contains="abc")
people.delete()
```

效果也是一样的, Django实际只执行一条 SQL 语句。

## 4. 更新某个内容

(1) 批量更新, 适用于 **.all()** **.filter()** **.exclude()** 等后面 (危险操作, 正式场合操作务必谨慎)

```
Person.objects.filter(name__contains="abc").update(name='xxx') # 名称中包含 "abc"的人 都改成 xxx
Person.objects.all().delete() # 删除所有 Person 记录
```

(2) 单个 **object** 更新, 适合于 **.get()**, **get\_or\_create()**, **update\_or\_create()** 等得到的 **obj**, 和新建很类似。

```
twz = Author.objects.get(name="WeizhongTu")
twz.name="WeizhongTu"
twz.email="tuweizhong@163.com"
twz.save() # 最后不要忘了保存!!!
```

## 5. QuerySet 是可迭代的, 比如:

```
es = Entry.objects.all()
for e in es:
    print(e.headline)
```

`Entry.objects.all()` 或者 `es` 就是 `QuerySet` 是查询所有的 `Entry` 条目。

注意事项:

(1). 如果只是检查 `Entry` 中是否有对象, 应该用 `Entry.objects.all().exists()`

(2). `QuerySet` 支持切片 `Entry.objects.all()[:10]` 取出10条, 可以节省内存

(3). 用 `len(es)` 可以得到`Entry`的数量, 但是推荐用 `Entry.objects.count()`来查询数量, 后者用的是SQL: `SELECT COUNT(*)`

(4). `list(es)` 可以强行将 `QuerySet` 变成 列表

## 6. `QuerySet` 是可以 pickle 序列化到硬盘再读取出来的

```
>>> import pickle
>>> query = pickle.loads(s)      # Assuming 's' is the pickled string.
>>> qs = MyModel.objects.all()
>>> qs.query = query             # Restore the original 'query'.
```

## 7. `QuerySet` 查询结果排序

作者按照名称排序

```
Author.objects.all().order_by('name')
Author.objects.all().order_by('-name') # 在 column name 前加一个负号, 可以实现倒序
```

## 8. `QuerySet` 支持链式查询

```
Author.objects.filter(name__contains="WeizhongTu").filter(email="tuweizhong@163.com")
Author.objects.filter(name__contains="Wei").exclude(email="tuweizhong@163.com")

# 找出名称含有abc, 但是排除年龄是23岁的
Person.objects.filter(name__contains="abc").exclude(age=23)
```

## 9. `QuerySet` 不支持负索引

```
Person.objects.all()[:10] 切片操作, 前10条
Person.objects.all()[-10:] 会报错!!!

# 1. 使用 reverse() 解决
Person.objects.all().reverse()[:2] # 最后两条
Person.objects.all().reverse()[0] # 最后一条

# 2. 使用 order_by, 在栏目名 (column name) 前加一个负号
Author.objects.order_by('-id')[:20] # id最大的20条
```

## 10. `QuerySet` 重复的问题, 使用 `.distinct()` 去重

一般的情况下, `QuerySet` 中不会出来重复的, 重复是很罕见的, 但是当跨越多张表进行检索后, 结果并到一起, 可能会出来重复的值 (我最近就遇到过这样的问题)

```
qs1 = Pathway.objects.filter(label__name='x')
qs2 = Pathway.objects.filter(reaction__name='A + B >> C')
qs3 = Pathway.objects.filter(inputer__name='WeizhongTu')

# 合并到一起
qs = qs1 | qs2 | qs3
这个时候就有可能出现重复的

# 去重方法
qs = qs.distinct()
```

参考: Django数据库操作官方文档: QuerySet API: <https://docs.djangoproject.com/en/dev/ref/models/queries/>

[小额赞助, 支持作者!](#)  
[« Django 模型 \(数据库\)](#)  
[Django QuerySet 进阶 »](#)



↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!   
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。  
自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django QuerySet API

[« Django 模型 \(数据库\)](#)  
[Django QuerySet 进阶 »](#)

[Django 模型](#)中我们学习了一些基本的创建与查询。这里专门来讲一下数据库接口相关的接口 (QuerySet API), 当然您也可以选择暂时跳过此节, 如果以后用到数据库相关的时候再看也是可以的。

从数据库中查询出来的结果一般是一个集合, 这个集合叫做 QuerySet。

文中的例子大部分是基于这个 `blog/models.py`

```
from django.db import models

class Blog(models.Model):
    name = models.CharField(max_length=100)
```

```

tagline = models.TextField()

def __str__(self):
    return self.name

class Author(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField()

    def __str__(self):
        return self.name

class Entry(models.Model):
    blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    mod_date = models.DateField()
    authors = models.ManyToManyField(Author)
    n_comments = models.IntegerField()
    n_pingbacks = models.IntegerField()
    rating = models.IntegerField()

    def __str__(self):
        return self.headline

```

## 1. QuerySet 创建对象的方法

```

>>> from blog.models import Blog
>>> b = Blog(name='Beatles Blog', tagline='All the latest Beatles news.')
>>> b.save()

```

总之，一共有四种方法

# 方法 1

```
Author.objects.create(name="WeizhongTu", email="tuweizhong@163.com")
```

# 方法 2

```
twz = Author(name="WeizhongTu", email="tuweizhong@163.com")
twz.save()
```

# 方法 3

```
twz = Author()
twz.name="WeizhongTu"
twz.email="tuweizhong@163.com"
twz.save()
```

# 方法 4，首先尝试获取，不存在就创建，可以防止重复

```
Author.objects.get_or_create(name="WeizhongTu", email="tuweizhong@163.com")
# 返回值 (object, True/False)
```

**备注：**前三种方法返回的都是对应的 **object**，最后一种方法返回的是一个元组，**(object, True/False)**，创建时返回 **True**，已经存在时返回 **False**

当有一对多，多对一，或者多对多的关系的时候，先把相关的对象查询出来

```

>>> from blog.models import Entry
>>> entry = Entry.objects.get(pk=1)
>>> cheese_blog = Blog.objects.get(name="Cheddar Talk")
>>> entry.blog = cheese_blog
>>> entry.save()

```

## 2. 获取对象的方法（上一篇的部分代码）

Person.objects.all() # 查询所有

Person.objects.all()[:10] 切片操作，获取10个人，不支持负索引，切片可以节约内存，不支持负索引，后面有相应解决办法，第7条

```
Person.objects.get(name="WeizhongTu") # 名称为 WeizhongTu 的一条, 多条会报错
```

get是用来获取一个对象的, 如果需要获取满足条件的一些人, 就要用到filter

```
Person.objects.filter(name="abc") # 等于Person.objects.filter(name__exact="abc") 名称严格等于 "abc" 的人
Person.objects.filter(name__iexact="abc") # 名称为 abc 但是不区分大小写, 可以找到 ABC, Abc, aBC, 这些都符合条件
```

```
Person.objects.filter(name__contains="abc") # 名称中包含 "abc"的人
Person.objects.filter(name__icontains="abc") # 名称中包含 "abc", 且abc不区分大小写
```

```
Person.objects.filter(name__regex="^abc") # 正则表达式查询
Person.objects.filter(name__iregex="^abc") # 正则表达式不区分大小写
```

# filter是找出满足条件的, 当然也有排除符合某条件的

```
Person.objects.exclude(name__contains="WZ") # 排除包含 WZ 的Person对象
```

```
Person.objects.filter(name__contains="abc").exclude(age=23) # 找出名称含有abc, 但是排除年龄是23岁的
```

### 3. 删除符合条件的结果

和上面类似, 得到满足条件的结果, 然后 **delete** 就可以(危险操作, 正式场合操作务必谨慎), 比如:

```
Person.objects.filter(name__contains="abc").delete() # 删除 名称中包含 "abc"的人
```

如果写成

```
people = Person.objects.filter(name__contains="abc")
people.delete()
```

效果也是一样的, Django实际只执行一条 SQL 语句。

### 4. 更新某个内容

(1) 批量更新, 适用于 **.all()** **.filter()** **.exclude()** 等后面 (危险操作, 正式场合操作务必谨慎)

```
Person.objects.filter(name__contains="abc").update(name='xxx') # 名称中包含 "abc"的人 都改成 xxx
Person.objects.all().delete() # 删除所有 Person 记录
```

(2) 单个 object 更新, 适合于 **.get()**, **get\_or\_create()**, **update\_or\_create()** 等得到的 obj, 和新建很类似。

```
twz = Author.objects.get(name="WeizhongTu")
twz.name="WeizhongTu"
twz.email="tuweizhong@163.com"
twz.save() # 最后不要忘了保存!!!
```

### 5. QuerySet 是可迭代的, 比如:

```
es = Entry.objects.all()
for e in es:
    print(e.headline)
```

Entry.objects.all() 或者 es 就是 QuerySet 是查询所有的 Entry 条目。

注意事项:

(1). 如果只是检查 Entry 中是否有对象, 应该用 **Entry.objects.all().exists()**

(2). QuerySet 支持切片 **Entry.objects.all()[0:10]** 取出10条, 可以节省内存

(3). 用 **len(es)** 可以得到Entry的数量, 但是推荐用 **Entry.objects.count()**来查询数量, 后者用的是SQL: **SELECT COUNT(\*)**

(4). **list(es)** 可以强行将 QuerySet 变成 列表

### 6. QuerySet 是可以 pickle 序列化到硬盘再读取出来的

```
>>> import pickle
>>> query = pickle.loads(s) # Assuming 's' is the pickled string.
>>> qs = MyModel.objects.all()
>>> qs.query = query # Restore the original 'query'.
```

## 7. QuerySet 查询结果排序

### 作者按照名称排序

```
Author.objects.all().order_by('name')
Author.objects.all().order_by('-name') # 在 column name 前加一个负号, 可以实现倒序
```

## 8. QuerySet 支持链式查询

```
Author.objects.filter(name__contains="WeizhongTu").filter(email="tuweizhong@163.com")
Author.objects.filter(name__contains="Wei").exclude(email="tuweizhong@163.com")

# 找出名称含有abc, 但是排除年龄是23岁的
Person.objects.filter(name__contains="abc").exclude(age=23)
```

## 9. QuerySet 不支持负索引

```
Person.objects.all()[0:10] 切片操作, 前10条
Person.objects.all()[-10:] 会报错!!!

# 1. 使用 reverse() 解决
Person.objects.all().reverse()[0:2] # 最后两条
Person.objects.all().reverse()[0] # 最后一条

# 2. 使用 order_by, 在栏目名 (column name) 前加一个负号
Author.objects.order_by('-id')[0:20] # id最大的20条
```

## 10. QuerySet 重复的问题, 使用 .distinct() 去重

一般的情况下, **QuerySet** 中不会出来重复的, 重复是很罕见的, 但是当跨越多张表进行检索后, 结果并到一起, 可能会出来重复的值 (我最近就遇到过这样的问题)

```
qs1 = Pathway.objects.filter(label__name='x')
qs2 = Pathway.objects.filter(reaction__name='A + B >> C')
qs3 = Pathway.objects.filter(inputer__name='WeizhongTu')

# 合并到一起
qs = qs1 | qs2 | qs3
这个时候就有可能出现重复的

# 去重方法
qs = qs.distinct()
```

参考: Django数据库操作官方文档: QuerySet API: <https://docs.djangoproject.com/en/dev/ref/models/queries/>

[小额赞助, 支持作者!](#)

[« Django 模型 \(数据库\)](#)

[Django QuerySet 进阶 »](#)

 CODING  
LEARN. DEVELOP. DEPLOY.

零成本开启敏捷开发 五人以下团队免费

免费体验

↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django QuerySet 进阶

[« Django QuerySet API](#)  
[BVDN-1 这个教程要做什么 »](#)

阅读本文你可以学习到什么

1. 查看 Django queryset 执行的 SQL (1部分)
2. 获得的查询结果直接以类似list方式展示(2, 3 部分)
3. 如何在django中给一个字段取一个别名 (4. 部分)
4. annotate 聚合 计数, 求和, 求平均数等 (5. 部分)
5. 优化SQL, 减少多对一, 一对多, 多对多时查询次数(6, 7 部分)
6. 如何只取出需要的字段, 排除某些字段 (8, 9部分)
7. 自定义一个自定义聚合功能, 比如 group\_concat (10. 部分)

本节属于进阶教程, 初学者可以跳过这节, 如果你觉得看不懂, 也可以先跳过, 后面再学。

准备阶段, 可以直接在下面下载代码开始练习, 建议看一下 `models.py` 的代码。

新建一个项目 `zqxt`, 建一个 `app` 名称是 `blog`

```
django-admin startproject zqxt
python manage.py startapp blog
```

把 `blog` 加入到 `settings.py` 中的 `INSTALL_APPS` 中

`blog/models.py` 代码如下

```
# -*- coding: utf-8 -*-
# @Date      : 2019-11-24 14:18:12
# @Author    : Weizhong Tu (mail@tuweizhong.com)
# @Link      : http://www.tuweizhong.com
# @Version   : 0.0.1

from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)
    qq = models.CharField(max_length=10)
    addr = models.TextField()
    email = models.EmailField()

    def __str__(self):
        return self.name

class Article(models.Model):
```



```

    title = models.CharField(max_length=50)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    content = models.TextField()
    score = models.IntegerField() # 文章的打分
    tags = models.ManyToManyField('Tag')

    def __str__(self):
        return self.title

class Tag(models.Model):
    name = models.CharField(max_length=50)

    def __str__(self):
        return self.name

```

比较简单，假设一篇文章只有一个作者(Author)，一个作者可以有多篇文章(Article)，一篇文章可以有多个标签(Tag)。

创建 migrations 然后 migrate 在数据库中生成相应的表

```

python3 manage.py makemigrations
python3 manage.py migrate

```

```
tu@pro ~/zqxt $ python3 manage.py makemigrations
```

Migrations for 'blog':

blog/migrations/0001\_initial.py

- Create model Author

- Create model Tag

- Create model Article

```
tu@pro ~/zqxt $ python3 manage.py migrate
```

Operations to perform:

Apply all migrations: admin, auth, blog, contenttypes, sessions

Running migrations:

Applying contenttypes.0001\_initial... OK

Applying auth.0001\_initial... OK

Applying admin.0001\_initial... OK

Applying admin.0002\_logentry\_remove\_auto\_add... OK

Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK

Applying contenttypes.0002\_remove\_content\_type\_name... OK

Applying auth.0002\_alter\_permission\_name\_max\_length... OK

Applying auth.0003\_alter\_user\_email\_max\_length... OK

Applying auth.0004\_alter\_user\_username\_opts... OK

Applying auth.0005\_alter\_user\_last\_login\_null... OK

Applying auth.0006\_require\_contenttypes\_0002... OK

Applying auth.0007\_alter\_validators\_add\_error\_messages... OK

Applying auth.0008\_alter\_user\_username\_max\_length... OK

Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK

Applying auth.0010\_alter\_group\_name\_max\_length... OK

Applying auth.0011\_update\_proxy\_permissions... OK

**Applying blog.0001\_initial... OK**

Applying sessions.0001\_initial... OK

生成一些示例数据，运行 `initdb.py`（有疑问的可以参考 [数据导入](#)）

```
# @Date      : 2019-11-24 14:19:00
# @Author    : Weizhong Tu (mail@tuweizhong.com)
# @Link      : http://www.tuweizhong.com
# @Version   : 0.0.1

import random
from zqxt.wsgi import *
from blog.models import Author, Article, Tag

author_name_list = ['WeizhongTu', 'twz915', 'dachui', 'zhe', 'zhen']
article_title_list = ['Django 教程', 'Python 教程', 'HTML 教程']

def create_authors():
    for author_name in author_name_list:
        author, created = Author.objects.get_or_create(name=author_name)
        # 随机生成9位数的QQ,
        author.qq = ''.join(
            str(random.choice(range(10))) for _ in range(9)
        )
        author.addr = 'addr_%s' % (random.randrange(1, 3))
        author.email = '%s@ziquangxuetang.com' % (author.addr)
        author.save()

def create_articles_and_tags():
    # 随机生成文章
    for article_title in article_title_list:
        # 从文章标题中得到 tag
        tag_name = article_title.split(' ', 1)[0]
        tag, created = Tag.objects.get_or_create(name=tag_name)

        random_author = random.choice(Author.objects.all())

        for i in range(1, 21):
            title = '%s_%s' % (article_title, i)
            article, created = Article.objects.get_or_create(
                title=title, defaults={
                    'author': random_author, # 随机分配作者
                    'content': '%s 正文' % title,
                    'score': random.randrange(70, 101), # 随机给文章一个打分
                }
            )
            article.tags.add(tag)

def main():
    create_authors()
    create_articles_and_tags()
```

```
if __name__ == '__main__':
    main()
    print("Done!")
```

```
tu@pro ~/zqxt $ python3 initdb.py
Done!
```

导入数据后，我们确认一下数据是不是已经导入。

```
tu@pro ~/zqxt $ python3 manage.py shell
```

```
In [1]: from blog.models import Article, Author, Tag
```

```
In [2]: Article.objects.all()
```

```
Out[2]: <QuerySet [<Article: Django 教程_1>, <Article: Django 教程_2>, <Article: Django 教程_3>, <Article: Django 教程_4>, <Article: Django 教程_5>,
<Article: Django 教程_6>, <Article: Django 教程_7>, <Article: Django 教程_8>, <Article: Django 教程_9>, <Article: Django 教程_10>, <Article: Django
教程_11>, <Article: Django 教程_12>, <Article: Django 教程_13>, <Article: Django 教程_14>, <Article: Django 教程_15>, <Article: Django 教程_16>,
<Article: Django 教程_17>, <Article: Django 教程_18>, <Article: Django 教程_19>, <Article: Django 教程_20>, '...(remaining elements truncated)...']>
```

```
In [3]: Author.objects.all()
```

```
Out[3]: <QuerySet [<Author: WeizhongTu>, <Author: twz915>, <Author: wangdachui>, <Author: xiaoming>]>
```

```
In [4]: Tag.objects.all()
```

```
Out[4]: <QuerySet [<Tag: Django>, <Tag: Python>, <Tag: HTML>]>
```

准备阶段结束，代码下载：

 [zqxt\\_queryset\\_advance.zip](#)

我们开始正式本节的学习，学习一些比较高级的查询方法

## 1. 查看 Django queryset 执行的 SQL

```
In [1]: print str(Author.objects.all().query)
```

```
SELECT "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_author"
```

简化一下，就是：SELECT id, name, qq, addr, email FROM blog\_author;

```
In [2]: print str(Author.objects.filter(name="WeizhongTu").query)
```

```
SELECT "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_author" WHERE
"blog_author"."name" = WeizhongTu
```

简化一下，就是：SELECT id, name, qq, addr, email FROM blog\_author WHERE name=WeizhongTu;

所以，当不知道Django做了什么时，你可以把执行的 SQL 打出来看看，也可以借助 [django-debug-toolbar](#) 等工具在页面上看到访问当前页面执行了哪些SQL，耗时等。

还有一种办法就是修改一下 log 的设置，后面会讲到。

## 2. values\_list 获取元组形式结果

2.1 比如我们要获取作者的名字和 qq

```
In [6]: authors = Author.objects.values_list('name', 'qq')
```

```
In [7]: authors
```

```
Out[7]: <QuerySet [(u'WeizhongTu', u'336643078'), (u'twz915', u'915792575'), (u'wangdachui', u'353506297'), (u'xiaoming', u'004466315')]>
```

```
In [8]: list(authors)
```

```
Out[8]:
```

```
[(u'WeizhongTu', u'336643078'),
 (u'twz915', u'915792575'),
 (u'wangdachui', u'353506297'),
 (u'xiaoming', u'004466315')]
```

如果只需要 1 个字段，可以指定 flat=True

```
In [9]: Author.objects.values_list('name', flat=True)
Out[9]: <QuerySet [u'WeizhongTu', u'twz915', u'wangdachui', u'xiaoming']>
```

```
In [10]: list(Author.objects.values_list('name', flat=True))
Out[10]: [u'WeizhongTu', u'twz915', u'wangdachui', u'xiaoming']
```

2.2 查询 twz915 这个人的文章标题

```
In [11]: Article.objects.filter(author__name='twz915').values_list('title', flat=True)
Out[11]: <QuerySet [u'HTML \u6559\u7a0b_1', u'HTML \u6559\u7a0b_2', u'HTML \u6559\u7a0b_3', u'HTML \u6559\u7a0b_4', u'HTML \u6559\u7a0b_5',
u'HTML \u6559\u7a0b_6', u'HTML \u6559\u7a0b_7', u'HTML \u6559\u7a0b_8', u'HTML \u6559\u7a0b_9', u'HTML \u6559\u7a0b_10', u'HTML
\u6559\u7a0b_11', u'HTML \u6559\u7a0b_12', u'HTML \u6559\u7a0b_13', u'HTML \u6559\u7a0b_14', u'HTML \u6559\u7a0b_15', u'HTML
\u6559\u7a0b_16', u'HTML \u6559\u7a0b_17', u'HTML \u6559\u7a0b_18', u'HTML \u6559\u7a0b_19', u'HTML \u6559\u7a0b_20']>
```

## 3. values 获取字典形式的结果

3.1 比如我们要获取作者的名字和 qq

```
In [13]: Author.objects.values('name', 'qq')
Out[13]: <QuerySet [{ 'qq': u'336643078', 'name': u'WeizhongTu'}, { 'qq': u'915792575', 'name': u'twz915'}, { 'qq': u'353506297', 'name': u'wangdachui'}, { 'qq':
u'004466315', 'name': u'xiaoming'}]>
```

```
In [14]: list(Author.objects.values('name', 'qq'))
Out[14]:
[{'name': u'WeizhongTu', 'qq': u'336643078'},
 {'name': u'twz915', 'qq': u'915792575'},
 {'name': u'wangdachui', 'qq': u'353506297'},
 {'name': u'xiaoming', 'qq': u'004466315'}]
```

3.2 查询 twz915 这个人的文章标题

```
In [23]: Article.objects.filter(author__name='twz915').values('title')
Out[23]: <QuerySet [{ 'title': u'HTML \u6559\u7a0b_1'}, { 'title': u'HTML \u6559\u7a0b_2'}, { 'title': u'HTML \u6559\u7a0b_3'}, { 'title': u'HTML
\u6559\u7a0b_4'}, { 'title': u'HTML \u6559\u7a0b_5'}, { 'title': u'HTML \u6559\u7a0b_6'}, { 'title': u'HTML \u6559\u7a0b_7'}, { 'title': u'HTML \u6559\u7a0b_8'},
{ 'title': u'HTML \u6559\u7a0b_9'}, { 'title': u'HTML \u6559\u7a0b_10'}, { 'title': u'HTML \u6559\u7a0b_11'}, { 'title': u'HTML \u6559\u7a0b_12'}, { 'title':
u'HTML \u6559\u7a0b_13'}, { 'title': u'HTML \u6559\u7a0b_14'}, { 'title': u'HTML \u6559\u7a0b_15'}, { 'title': u'HTML \u6559\u7a0b_16'}, { 'title': u'HTML
\u6559\u7a0b_17'}, { 'title': u'HTML \u6559\u7a0b_18'}, { 'title': u'HTML \u6559\u7a0b_19'}, { 'title': u'HTML \u6559\u7a0b_20'}]>
```

注意：

1. `values_list` 和 `values` 返回的并不是真正的列表或字典，也是 `queryset`，他们也是 `lazy evaluation` 的（惰性评估，通俗地说，就是用的时候才真正的去数据库查）
2. 如果查询后没有使用，在数据库更新后再使用，你发现得到的是新内容！！！如果想要旧内容保持着，数据库更新后不要变，可以 `list` 一下
3. 如果只是遍历这些结果，没有必要 `list` 它们转成列表（浪费内存，数据量大的时候要更谨慎！！！）

## 4. extra 实现 别名，条件，排序等

`extra` 中可实现别名，条件，排序等，后面两个用 `filter`, `exclude` 一般都能实现，排序用 `order_by` 也能实现。我们主要看一下别名这个

比如 `Author` 中有 `name`，`Tag` 中有 `name` 我们想执行

```
SELECT name AS tag_name FROM blog_tag;
```

这样的语句，就可以用 `select` 来实现，如下：

```
In [44]: tags = Tag.objects.all().extra(select={'tag_name': 'name'})
```

```
In [45]: tags[0].name
Out[45]: u'Django'
```

```
In [46]: tags[0].tag_name
```

```
Out[46]: u'Django'
```

我们发现 `name` 和 `tag_name` 都可以使用，确认一下执行的 SQL

```
In [47]: Tag.objects.all().extra(select={'tag_name': 'name'}).query.__str__()
Out[47]: u'SELECT (name) AS "tag_name", "blog_tag"."id", "blog_tag"."name" FROM "blog_tag"'
```

我们发现查询的时候弄了两次 `(name) AS "tag_name"` 和 `"blog_tag"."name"`

如果我们只想其中一个能用，可以用 `defer` 排除掉原来的 `name`（后面有讲）

```
In [49]: Tag.objects.all().extra(select={'tag_name': 'name'}).defer('name').query.__str__()
Out[49]: u'SELECT (name) AS "tag_name", "blog_tag"."id" FROM "blog_tag"'
```

也许你会说为什么要改个名称，最常见的需求就是数据转变成 `list`，然后可视化等，我们在下面一个里面讲。

## 5. annotate 聚合 计数，求和，平均数等

### 5.1 计数

我们来计算一下每个作者的文章数（我们每个作者都导入的 `Article` 的篇数一样，所以下面的每个都一样）

```
In [66]: from django.db.models import Count

In [66]: Article.objects.all().values('author_id').annotate(count=Count('author')).values('author_id', 'count')
Out[66]: <QuerySet [{ 'count': 20, 'author_id': 1}, { 'count': 20, 'author_id': 2}, { 'count': 20, 'author_id': 4}]>
```

这是怎么工作的呢？

```
In [67]: Article.objects.all().values('author_id').annotate(count=Count('author')).values('author_id', 'count').query.__str__()
Out[67]: u'SELECT "blog_article"."author_id", COUNT("blog_article"."author_id") AS "count" FROM "blog_article" GROUP BY "blog_article"."author_id"'
```

简化一下 SQL: `SELECT author_id, COUNT(author_id) AS count FROM blog_article GROUP BY author_id`

我们也可以获取作者的名称 及 作者的文章数

```
In [72]: Article.objects.all().values('author__name').annotate(count=Count('author')).values('author__name', 'count')
Out[72]: <QuerySet [{ 'count': 20, 'author__name': u'WeizhongTu'}, { 'count': 20, 'author__name': u'twz915'}, { 'count': 20, 'author__name': u'xiaoming'}]>
```

细心的同学会发现，这时候实际上查询两张表，因为作者名称(`author__name`)在 `blog_author` 这张表中，而上一个例子中的 `author_id` 是 `blog_article` 表本身就有的字段

### 5.2 求和 与 平均值

#### 5.2.1 求一个作者的所有文章的得分(score)平均值

```
In [6]: from django.db.models import Avg

In [7]: Article.objects.values('author_id').annotate(avg_score=Avg('score')).values('author_id', 'avg_score')
Out[7]: <QuerySet [{ 'author_id': 1, 'avg_score': 86.05}, { 'author_id': 2, 'avg_score': 83.75}, { 'author_id': 5, 'avg_score': 85.65}]>
```

执行的SQL

```
In [8]: Article.objects.values('author_id').annotate(avg_score=Avg('score')).values('author_id', 'avg_score').query.__str__()
Out[8]: u'SELECT "article"."author_id", AVG("article"."score") AS "avg_score" FROM "article" GROUP BY "article"."author_id"'
```

```
Out[8]: u'SELECT "blog_article"."author_id", AVG("blog_article"."score") AS "avg_score" FROM "blog_article" GROUP BY "blog_article"."author_id"'
```

### 5.2.2 求一个作者所有文章的总分

```
In [12]: from django.db.models import Sum
```

```
In [13]: Article.objects.values('author__name').annotate(sum_score=Sum('score')).values('author__name', 'sum_score')
```

```
Out[13]: <QuerySet [{ 'author__name': u'WeizhongTu', 'sum_score': 1721}, { 'author__name': u'twz915', 'sum_score': 1675}, { 'author__name': u'zhen', 'sum_score': 1713}]>
```

执行的SQL

```
In [14]: Article.objects.values('author__name').annotate(sum_score=Sum('score')).values('author__name', 'sum_score')
...: re).query.__str__()
```

```
Out[14]: u'SELECT "blog_author"."name", SUM("blog_article"."score") AS "sum_score" FROM "blog_article" INNER JOIN "blog_author" ON ("blog_article"."author_id" = "blog_author"."id") GROUP BY "blog_author"."name"'
```

## 6. select\_related 优化一对一，多对一查询

开始之前我们修改一个 settings.py 让 Django 打印出在数据库中执行的语句

settings.py 尾部加上

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        'django.db.backends': {
            'handlers': ['console'],
            'level': 'DEBUG' if DEBUG else 'INFO',
        },
    },
}
```

这样当 DEBUG 为 True 的时候，我们可以看出 django 执行了什么 SQL 语句

```
tu@pro ~/zqxt $ python manage.py shell
```

```
In [1]: from blog.models import *
```

```
In [2]: Author.objects.all()
```

```
Out[2]: (0.001) SELECT "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_author" LIMIT 21; args=()
<QuerySet [<Author: WeizhongTu>, <Author: twz915>, <Author: dachui>, <Author: zhe>, <Author: zhen>]>
```

标记背景为 黄色的部分就是打出的 log。

假如，我们取出10篇Django相关的文章，并需要用到作者的姓名

```
In [13]: articles = Article.objects.all()[:10]
```

```
In [14]: a1 = articles[0] # 取第一篇
```

```
(0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM "blog_article" LIMIT 1; args=()
```

```
In [15]: a1.title
```

```
Out[15]: u'Django \u6559\u7a0b_1'
```

```
In [16]: a1.author_id
```

```
Out[16]: 5
```

```
In [17]: a1.author.name # 再次查询了数据库，注意！！
(0.000) SELECT "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_author"
WHERE "blog_author"."id" = 5; args=(5,)
Out[17]: u'zhen'
```

这样的话我们遍历查询结果的时候就会查询很多次数据库，能不能只查询一次，把作者的信息也查出来呢？

当然可以，这时就用到 `select_related`，我们的数据库设计的是一篇文章只能有一个作者，一个作者可以有多篇文章。

现在要查询文章的时候连同作者一起查询出来，“文章”和“作者”的关系就是**多对一**，换句话说，就是一篇文章只可能有一个作者。

```
In [18]: articles = Article.objects.all().select_related('author')[:10]
```

```
In [19]: a1 = articles[0] # 取第一篇
(0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score", "blog_author"."id",
"blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_article" INNER JOIN "blog_author" ON
("blog_article"."author_id" = "blog_author"."id") LIMIT 1; args=()
```

```
In [20]: a1.title
Out[20]: u'Django \u6559\u7a0b_1'
```

```
In [21]: a1.author.name # 嘻嘻，没有再次查询数据库！！
Out[21]: u'zhen'
```

## 7. prefetch\_related 优化一对多，多对多查询

和 `select_related` 功能类似，但是实现不同。

`select_related` 是使用 SQL JOIN 一次性取出相关的内容。

`prefetch_related` 用于一对多，多对多的情况，这时 `select_related` 用不了，因为当前一条有好几条与之相关的内容。

`prefetch_related` 是通过再执行一条额外的 SQL 语句，然后用 Python 把两次 SQL 查询的内容关联（**joining**）到一起

我们来看个例子，查询文章的同时，查询文章对应的标签。“文章”与“标签”是多对多的关系。

```
In [24]: articles = Article.objects.all().prefetch_related('tags')[:10]
```

```
In [25]: articles
Out[25]: (0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM
"blog_article" LIMIT 10; args=()
(0.001) SELECT ("blog_article_tags"."article_id") AS "_prefetch_related_val_article_id", "blog_tag"."id", "blog_tag"."name" FROM "blog_tag"
INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" IN (1, 2, 3, 4, 5, 6, 7,
8, 9, 10); args=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
<QuerySet [<Article: Django 教程_1>, <Article: Django 教程_2>, <Article: Django 教程_3>, <Article: Django 教程_4>, <Article: Django 教程_5>,
<Article: Django 教程_6>, <Article: Django 教程_7>, <Article: Django 教程_8>, <Article: Django 教程_9>, <Article: Django 教程_10>]>
```

遍历查询的结果：

不用 `prefetch_related` 时

```
In [9]: articles = Article.objects.all()[:3]
```

```
In [10]: for a in articles:
...:     print a.title, a.tags.all()
...:
(0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM
"blog_article" LIMIT 3; args=()

(0.000) SELECT "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" =
"blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" = 1 LIMIT 21; args=(1,)
```

```
Django 教程_1 <QuerySet [<Tag: Django>]>
```

```
(0.000) SELECT "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" = 2 LIMIT 21; args=(2,)
```

Django 教程\_2 <QuerySet [Tag: Django]>

```
(0.000) SELECT "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" = 3 LIMIT 21; args=(3,)
```

Django 教程\_3 <QuerySet [Tag: Django]>

用 `prefetch_related` 我们看一下是什么样子

```
In [11]: articles = Article.objects.all().prefetch_related('tags')[3]
```

```
In [12]: for a in articles:
```

```
...:     print a.title, a.tags.all()
```

```
...:
```

```
(0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM "blog_article" LIMIT 3; args=()
```

```
(0.000) SELECT ("blog_article_tags"."article_id") AS "_prefetch_related_val_article_id", "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" IN (1, 2, 3); args=(1, 2, 3)
```

Django 教程\_1 <QuerySet [Tag: Django]>

Django 教程\_2 <QuerySet [Tag: Django]>

Django 教程\_3 <QuerySet [Tag: Django]>

我们可以看到第二条 SQL 语句，一次性查出了所有相关的内容。

## 8. defer 排除不需要的字段

在复杂的情况下，表中可能有些字段内容非常多，取出来转化成 Python 对象会占用大量的资源。

这时候可以用 `defer` 来排除这些字段，比如我们在文章列表页，只需要文章的标题和作者，没有必要把文章的内容也获取出来（因为会转换成 python 对象，浪费内存）

```
In [13]: Article.objects.all()
```

```
Out[13]: (0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM "blog_article" LIMIT 21; args=()
```

```
<QuerySet [Article: Django 教程_1, Article: Django 教程_2, Article: Django 教程_3, Article: Django 教程_4, Article: Django 教程_5, Article: Django 教程_6, Article: Django 教程_7, Article: Django 教程_8, Article: Django 教程_9, Article: Django 教程_10, Article: Django 教程_11, Article: Django 教程_12, Article: Django 教程_13, Article: Django 教程_14, Article: Django 教程_15, Article: Django 教程_16, Article: Django 教程_17, Article: Django 教程_18, Article: Django 教程_19, Article: Django 教程_20, ...(remaining elements truncated)...]>
```

```
In [14]: Article.objects.all().defer('content')
```

```
Out[14]: (0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."score" FROM "blog_article" LIMIT 21; args=() # 注意这里没有查 content 字段了
```

```
<QuerySet [Article: Django 教程_1, Article: Django 教程_2, Article: Django 教程_3, Article: Django 教程_4, Article: Django 教程_5, Article: Django 教程_6, Article: Django 教程_7, Article: Django 教程_8, Article: Django 教程_9, Article: Django 教程_10, Article: Django 教程_11, Article: Django 教程_12, Article: Django 教程_13, Article: Django 教程_14, Article: Django 教程_15, Article: Django 教程_16, Article: Django 教程_17, Article: Django 教程_18, Article: Django 教程_19, Article: Django 教程_20, ...(remaining elements truncated)...]>
```

## 9. only 仅选择需要的字段

和 `defer` 相反，`only` 用于取出需要的字段，假如我们只需要查出 作者的名称

```
In [15]: Author.objects.all().only('name')
```

```
Out[15]: (0.000) SELECT "blog_author"."id", "blog_author"."name" FROM "blog_author" LIMIT 21; args=()
```

```
<QuerySet [Author: WeizhongTu, Author: twz915, Author: dachu, Author: zhe, Author: zhen]>
```

细心的同学会发现，我们让查 `name`，`id` 也查了，这个 `id` 是主键，能不能没有这个 `id` 呢？

试一下原生的 SQL 查询

```
In [26]: authors = Author.objects.raw('select name from blog_author limit 1')
```



```
In [27]: author = authors[0]
(0.000) select name from blog_author limit 1; args=()
```

```
InvalidQuery Traceback (most recent call last)
<ipython-input-27-51c5f914ff2> in <module>()
----> 1author = authors[0]
```

```
/usr/local/lib/python2.7/site-packages/django/db/models/query.pyc in __getitem__(self, k)
1275
1276 def __getitem__(self, k):
-> 1277     return list(self)[k]
1278
1279 @property
```

```
/usr/local/lib/python2.7/site-packages/django/db/models/query.pyc in __iter__(self)
1250     if skip:
1251         if self.model._meta.pk.attname in skip:
-> 1252             raise InvalidQuery('Raw query must include the primary key')
1253     model_cls = self.model
1254     fields = [self.model._fields.get(c) for c in self.columns]
```

InvalidQuery: Raw query must include the primary key

报错信息说 非法查询，原生SQL查询必须包含 主键！

再试试直接执行 SQL

```
tu@pro ~/zqxt $ python manage.py dbshell
SQLite version 3.14.0 2016-07-26 15:17:14
Enter ".help" for usage hints.
sqlite> select name from blog_author limit 1;
WeizhongTu <-- 成功!!!
```

虽然直接执行SQL语句可以这样，但是 `django queryset` 不允许这样做，一般也不需要关心，反正 **only** 一定会取出你指定的字段。

## 10. 自定义聚合功能

我们前面看到了 `django.db.models` 中有 `Count`, `Avg`, `Sum` 等，但是有一些没有的，比如 `GROUP_CONCAT`，它用来聚合时将符合某分组条件(`group by`)的不同的值，连到一起，作为整体返回。

我们来演示一下，如果实现 `GROUP_CONCAT` 功能。

新建一个文件 比如 `my_aggregate.py`

```
from django.db.models import Aggregate, CharField
```

```
class GroupConcat(Aggregate):
    function = 'GROUP_CONCAT'
    template = '%(function)s(%(distinct)s%(expressions)s%(ordering)s%(separator)s)'

    def __init__(self, expression, distinct=False, ordering=None, separator=',', **extra):
        super(GroupConcat, self).__init__(
            expression,
            distinct='DISTINCT ' if distinct else '',
            ordering=' ORDER BY %s' % ordering if ordering is not None else '',
            separator=' SEPARATOR "%s"' % separator,
            output_field=CharField(),
            **extra
        )
```

代码来自: <http://stackoverflow.com/a/40478702/2714931> (我根据一个回复改写的增强版本)

使用时先引入 GroupConcat 这个类，比如聚合后的错误日志记录有这些字段 time, level, info

我们想把 level, info 一样的 聚到一起，按时间和发生次数倒序排列，并含有每次日志发生的时间。

```
ErrorLogModel.objects.values('level', 'info').annotate(  
    count=Count(1), time=GroupConcat('time', ordering='time DESC', separator=' | ')  
) .order_by('-time', '-count')
```

参考资源：

QuerySet API: <https://docs.djangoproject.com/en/2.2/ref/models/queries/>

聚合相关: <https://docs.djangoproject.com/en/2.2/topics/db/aggregation/>

[小额赞助，支持作者！](#)

[« Django QuerySet API](#)

[BVDN-1 这个教程要做什么 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django QuerySet 进阶

[« Django QuerySet API](#)

## [BVDN-1 这个教程要做什么»](#)

阅读本文你可以学习到什么

1. 查看 Django queryset 执行的 SQL (1部分)
2. 获得的查询结果直接以类似list方式展示(2, 3 部分)
3. 如何在django中给一个字段取一个别名 (4. 部分)
4. `annotate` 聚合 计数, 求和, 求平均数等 (5. 部分)
5. 优化SQL, 减少多对一, 一对多, 多对多时查询次数(6, 7 部分)
6. 如何只取出需要的字段, 排除某些字段 (8, 9部分)
7. 自定义一个自定义聚合功能, 比如 `group_concat` (10. 部分)

本节属于进阶教程, 初学者可以跳过这节, 如果你觉得看不懂, 也可以先跳过, 后面再学。

准备阶段, 可以直接在下面下载代码开始练习, 建议看一下 `models.py` 的代码。

新建一个项目 `zqxt`, 建一个 `app` 名称是 `blog`

```
django-admin startproject zqxt
python manage.py startapp blog
```

把 `blog` 加入到 `settings.py` 中的 `INSTALL_APPS` 中

`blog/models.py` 代码如下

```
# -*- coding: utf-8 -*-
# @Date      : 2019-11-24 14:18:12
# @Author    : Weizhong Tu (mail@tuweizhong.com)
# @Link      : http://www.tuweizhong.com
# @Version   : 0.0.1

from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)
    qq = models.CharField(max_length=10)
    addr = models.TextField()
    email = models.EmailField()

    def __str__(self):
        return self.name

class Article(models.Model):
    title = models.CharField(max_length=50)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    content = models.TextField()
    score = models.IntegerField() # 文章的打分
    tags = models.ManyToManyField('Tag')

    def __str__(self):
        return self.title

class Tag(models.Model):
    name = models.CharField(max_length=50)
```

```
def __str__(self):  
    return self.name
```

比较简单，假设一篇文章只有一个作者(Author)，一个作者可以有多篇文章(Article)，一篇文章可以有多个标签 (Tag)。

创建 **migrations** 然后 **migrate** 在数据库中生成相应的表

```
python3 manage.py makemigrations  
python3 manage.py migrate
```

```
tu@pro ~/zqxt $ python3 manage.py makemigrations
```

Migrations for 'blog':

blog/migrations/0001\_initial.py

- Create model Author

- Create model Tag

- Create model Article

```
tu@pro ~/zqxt $ python3 manage.py migrate
```

Operations to perform:

Apply all migrations: admin, auth, blog, contenttypes, sessions

Running migrations:

Applying contenttypes.0001\_initial... OK

Applying auth.0001\_initial... OK

Applying admin.0001\_initial... OK

Applying admin.0002\_logentry\_remove\_auto\_add... OK

Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK

Applying contenttypes.0002\_remove\_content\_type\_name... OK

Applying auth.0002\_alter\_permission\_name\_max\_length... OK

Applying auth.0003\_alter\_user\_email\_max\_length... OK

Applying auth.0004\_alter\_user\_username\_opts... OK

Applying auth.0005\_alter\_user\_last\_login\_null... OK

Applying auth.0006\_require\_contenttypes\_0002... OK

Applying auth.0007\_alter\_validators\_add\_error\_messages... OK

Applying auth.0008\_alter\_user\_username\_max\_length... OK

Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK

Applying auth.0010\_alter\_group\_name\_max\_length... OK

Applying auth.0011\_update\_proxy\_permissions... OK

Applying blog.0001\_initial... OK

Applying sessions.0001\_initial... OK

生成一些示例数据，运行 `initdb.py`（有疑问的可以参考 [数据导入](#)）

```
# @Date      : 2019-11-24 14:19:00
# @Author    : Weizhong Tu (mail@tuweizhong.com)
# @Link      : http://www.tuweizhong.com
# @Version   : 0.0.1

import random
from zqxt.wsgi import *
from blog.models import Author, Article, Tag

author_name_list = ['WeizhongTu', 'twz915', 'dachui', 'zhe', 'zhen']
article_title_list = ['Django 教程', 'Python 教程', 'HTML 教程']

def create_authors():
    for author_name in author_name_list:
        author, created = Author.objects.get_or_create(name=author_name)
        # 随机生成9位数的QQ,
        author.qq = ''.join(
            str(random.choice(range(10))) for _ in range(9)
        )
        author.addr = 'addr_%s' % (random.randrange(1, 3))
        author.email = '%s@ziquangxuetang.com' % (author.addr)
        author.save()

def create_articles_and_tags():
    # 随机生成文章
    for article_title in article_title_list:
        # 从文章标题中得到 tag
        tag_name = article_title.split(' ', 1)[0]
        tag, created = Tag.objects.get_or_create(name=tag_name)

        random_author = random.choice(Author.objects.all())

        for i in range(1, 21):
            title = '%s_%s' % (article_title, i)
            article, created = Article.objects.get_or_create(
                title=title, defaults={
                    'author': random_author, # 随机分配作者
                    'content': '%s 正文' % title,
                    'score': random.randrange(70, 101), # 随机给文章一个打分
                }
            )
            article.tags.add(tag)

def main():
    create_authors()
    create_articles_and_tags()

if __name__ == '__main__':
    main()
    print("Done!")

tu@pro ~/zqxt $ python3 initdb.py
Done!
```

导入数据后，我们确认一下数据是不是已经导入。

```
tu@pro ~/zqxt $ python3 manage.py shell
```

```
In [1]: from blog.models import Article, Author, Tag
```

```
In [2]: Article.objects.all()
```

```
Out[2]: <QuerySet [<Article: Django 教程_1>, <Article: Django 教程_2>, <Article: Django 教程_3>, <Article: Django 教程_4>, <Article: Django 教程_5>, <Article: Django 教程_6>, <Article: Django 教程_7>, <Article: Django 教程_8>, <Article: Django 教程_9>, <Article: Django 教程_10>, <Article: Django 教程_11>, <Article: Django 教程_12>, <Article: Django 教程_13>, <Article: Django 教程_14>, <Article: Django 教程_15>, <Article: Django 教程_16>, <Article: Django 教程_17>, <Article: Django 教程_18>, <Article: Django 教程_19>, <Article: Django 教程_20>, '...(remaining elements truncated)...']>
```

```
In [3]: Author.objects.all()
```

```
Out[3]: <QuerySet [<Author: WeizhongTu>, <Author: twz915>, <Author: wangdachui>, <Author: xiaoming>]>
```

```
In [4]: Tag.objects.all()
```

```
Out[4]: <QuerySet [<Tag: Django>, <Tag: Python>, <Tag: HTML>]>
```

准备阶段结束，代码下载：

 [zqxt\\_queryset\\_advance.zip](#)

我们开始正式本节的学习，学习一些比较高级的查询方法

## 1. 查看 Django queryset 执行的 SQL

```
In [1]: print str(Author.objects.all().query)
```

```
SELECT "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_author"
```

简化一下，就是：SELECT id, name, qq, addr, email FROM blog\_author;

```
In [2]: print str(Author.objects.filter(name="WeizhongTu").query)
```

```
SELECT "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_author" WHERE "blog_author"."name" = WeizhongTu
```

简化一下，就是：SELECT id, name, qq, addr, email FROM blog\_author WHERE name=WeizhongTu;

所以，当不知道Django做了什么时，你可以把执行的SQL打出来看看，也可以借助 [django-debug-toolbar](#) 等工具在页面上看到访问当前页面执行了哪些SQL，耗时等。

还有一种办法就是修改一下 log 的设置，后面会讲到。

## 2. values\_list 获取元组形式结果

### 2.1 比如我们要获取作者的名字和 qq

```
In [6]: authors = Author.objects.values_list('name', 'qq')
```

```
In [7]: authors
```

```
Out[7]: <QuerySet [(u'WeizhongTu', u'336643078'), (u'twz915', u'915792575'), (u'wangdachui', u'353506297'), (u'xiaoming', u'004466315')]>
```

```
In [8]: list(authors)
```

```
Out[8]: [(u'WeizhongTu', u'336643078'), (u'twz915', u'915792575'), (u'wangdachui', u'353506297'), (u'xiaoming', u'004466315')]
```

如果只需要 1 个字段，可以指定 flat=True

```
In [9]: Author.objects.values_list('name', flat=True)
```

```
Out[9]: <QuerySet [u'WeizhongTu', u'twz915', u'wangdachui', u'xiaoming']>
```

```
In [10]: list(Author.objects.values_list('name', flat=True))
```

```
Out[10]: [u'WeizhongTu', u'twz915', u'wangdachui', u'xiaoming']
```

### 2.2 查询 twz915 这个人的文章标题

```
In [11]: Article.objects.filter(author__name='twz915').values_list('title', flat=True)
```

```
Out[11]: <QuerySet [u'HTML \u6559\u7a0b_1', u'HTML \u6559\u7a0b_2', u'HTML \u6559\u7a0b_3', u'HTML \u6559\u7a0b_4', u'HTML \u6559\u7a0b_5', u'HTML \u6559\u7a0b_6', u'HTML \u6559\u7a0b_7', u'HTML \u6559\u7a0b_8', u'HTML \u6559\u7a0b_9', u'HTML \u6559\u7a0b_10', u'HTML
```

```
\u6559\u7a0b_11', u'HTML\u6559\u7a0b_12', u'HTML\u6559\u7a0b_13', u'HTML\u6559\u7a0b_14', u'HTML\u6559\u7a0b_15', u'HTML\u6559\u7a0b_16', u'HTML\u6559\u7a0b_17', u'HTML\u6559\u7a0b_18', u'HTML\u6559\u7a0b_19', u'HTML\u6559\u7a0b_20']>
```

### 3. values 获取字典形式的结果

3.1 比如我们要获取作者的 `name` 和 `qq`

```
In [13]: Author.objects.values('name', 'qq')
```

```
Out[13]: <QuerySet [{ 'qq': u'336643078', 'name': u'WeizhongTu'}, { 'qq': u'915792575', 'name': u'twz915'}, { 'qq': u'353506297', 'name': u'wangdachui'}, { 'qq': u'004466315', 'name': u'xiaoming'}]>
```

```
In [14]: list(Author.objects.values('name', 'qq'))
```

```
Out[14]:
[{'name': u'WeizhongTu', 'qq': u'336643078'},
 {'name': u'twz915', 'qq': u'915792575'},
 {'name': u'wangdachui', 'qq': u'353506297'},
 {'name': u'xiaoming', 'qq': u'004466315'}]
```

3.2 查询 `twz915` 这个人的文章标题

```
In [23]: Article.objects.filter(author__name='twz915').values('title')
```

```
Out[23]: <QuerySet [{ 'title': u'HTML\u6559\u7a0b_1'}, { 'title': u'HTML\u6559\u7a0b_2'}, { 'title': u'HTML\u6559\u7a0b_3'}, { 'title': u'HTML\u6559\u7a0b_4'}, { 'title': u'HTML\u6559\u7a0b_5'}, { 'title': u'HTML\u6559\u7a0b_6'}, { 'title': u'HTML\u6559\u7a0b_7'}, { 'title': u'HTML\u6559\u7a0b_8'}, { 'title': u'HTML\u6559\u7a0b_9'}, { 'title': u'HTML\u6559\u7a0b_10'}, { 'title': u'HTML\u6559\u7a0b_11'}, { 'title': u'HTML\u6559\u7a0b_12'}, { 'title': u'HTML\u6559\u7a0b_13'}, { 'title': u'HTML\u6559\u7a0b_14'}, { 'title': u'HTML\u6559\u7a0b_15'}, { 'title': u'HTML\u6559\u7a0b_16'}, { 'title': u'HTML\u6559\u7a0b_17'}, { 'title': u'HTML\u6559\u7a0b_18'}, { 'title': u'HTML\u6559\u7a0b_19'}, { 'title': u'HTML\u6559\u7a0b_20'}]>
```

注意：

1. `values_list` 和 `values` 返回的并不是真正的 列表 或 字典，也是 `queryset`，他们也是 `lazy evaluation` 的（惰性评估，通俗地说，就是用的时候才真正的去数据库查）
2. 如果查询后没有使用，在数据库更新后再使用，你发现得到的是新内容！！！如果想要旧内容保持着，数据库更新后不要变，可以 `list` 一下
3. 如果只是遍历这些结果，没有必要 `list` 它们转成列表（浪费内存，数据量大的时候要更谨慎！！！）

### 4. extra 实现 别名，条件，排序等

`extra` 中可实现别名，条件，排序等，后面两个用 `filter`, `exclude` 一般都能实现，排序用 `order_by` 也能实现。我们主要看一下别名这个

比如 `Author` 中有 `name`，`Tag` 中有 `name` 我们想执行

```
SELECT name AS tag_name FROM blog_tag;
```

这样的语句，就可以用 `select` 来实现，如下：

```
In [44]: tags = Tag.objects.all().extra(select={'tag_name': 'name'})
```

```
In [45]: tags[0].name
```

```
Out[45]: u'Django'
```

```
In [46]: tags[0].tag_name
```

```
Out[46]: u'Django'
```

我们发现 `name` 和 `tag_name` 都可以使用，确认一下执行的 SQL

```
In [47]: Tag.objects.all().extra(select={'tag_name': 'name'}).query.__str__()
```

```
Out[47]: u'SELECT (name) AS "tag_name", "blog_tag"."id", "blog_tag"."name" FROM "blog_tag"
```

我们发现查询的时候弄了两次 **(name) AS "tag\_name"** 和 **"blog\_tag"."name"**

如果我们只想其中一个能用，可以用 **defer** 排除掉原来的 **name**（后面有讲）

```
In [49]: Tag.objects.all().extra(select={'tag_name': 'name'}).defer('name').query.__str__()
Out[49]: u'SELECT (name) AS "tag_name", "blog_tag"."id" FROM "blog_tag"'
```

也许你会说为什么要改个名称，最常见的需求就是数据转变成 **list**，然后可视化等，我们在下面一个里面讲。

## 5. annotate 聚合 计数，求和，平均数等

### 5.1 计数

我们来计算一下每个作者的文章数（我们每个作者都导入的Article的篇数一样，所以下面的每个都一样）

```
In [66]: from django.db.models import Count
```

```
In [66]: Article.objects.all().values('author_id').annotate(count=Count('author')).values('author_id', 'count')
Out[66]: <QuerySet [{ 'count': 20, 'author_id': 1}, { 'count': 20, 'author_id': 2}, { 'count': 20, 'author_id': 4}]>
```

这是怎么工作的呢？

```
In [67]: Article.objects.all().values('author_id').annotate(count=Count('author')).values('author_id', 'count').query.__str__()
Out[67]: u'SELECT "blog_article"."author_id", COUNT("blog_article"."author_id") AS "count" FROM "blog_article" GROUP BY "blog_article"."author_id"'
```

简化一下SQL: `SELECT author_id, COUNT(author_id) AS count FROM blog_article GROUP BY author_id`

我们也可以获取作者的名称 及 作者的文章数

```
In [72]: Article.objects.all().values('author__name').annotate(count=Count('author')).values('author__name', 'count')
Out[72]: <QuerySet [{ 'count': 20, 'author__name': u'WeizhongTu'}, { 'count': 20, 'author__name': u'twz915'}, { 'count': 20, 'author__name': u'xiaoming'}]>
```

细心的同学会发现，这时候实际上查询两张表，因为作者名称(**author\_\_name**)在 **blog\_author** 这张表中，而上一个例子中的 **author\_id** 是 **blog\_article** 表本身就有的字段

### 5.2 求和 与 平均值

#### 5.2.1 求一个作者的所有文章的得分(score)平均值

```
In [6]: from django.db.models import Avg
```

```
In [7]: Article.objects.values('author_id').annotate(avg_score=Avg('score')).values('author_id', 'avg_score')
Out[7]: <QuerySet [{ 'author_id': 1, 'avg_score': 86.05}, { 'author_id': 2, 'avg_score': 83.75}, { 'author_id': 5, 'avg_score': 85.65}]>
```

执行的SQL

```
In [8]: Article.objects.values('author_id').annotate(avg_score=Avg('score')).values('author_id', 'avg_score').query.__str__()
Out[8]: u'SELECT "blog_article"."author_id", AVG("blog_article"."score") AS "avg_score" FROM "blog_article" GROUP BY "blog_article"."author_id"'
```

#### 5.2.2 求一个作者所有文章的总分

```
In [12]: from django.db.models import Sum
```

```
In [13]: Article.objects.values('author__name').annotate(sum_score=Sum('score')).values('author__name', 'sum_score')
Out[13]: <QuerySet [{ 'author__name': u'WeizhongTu', 'sum_score': 1721}, { 'author__name': u'twz915', 'sum_score': 1675}, { 'author__name': u'zhen',
```



```
'sum_score': 1713}}]>
```

执行的SQL

```
In [14]: Article.objects.values('author__name').annotate(sum_score=Sum('score')).values('author__name', 'sum_score')
Out[14]: u'SELECT "blog_author"."name", SUM("blog_article"."score") AS "sum_score" FROM "blog_article" INNER JOIN "blog_author" ON ("blog_article"."author_id" = "blog_author"."id") GROUP BY "blog_author"."name"
```

## 6. select\_related 优化一对一，多对一查询

开始之前我们修改一个 settings.py 让Django打印出在数据库中执行的语句

settings.py 尾部加上

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        'django.db.backends': {
            'handlers': ['console'],
            'level': 'DEBUG' if DEBUG else 'INFO',
        },
    },
}
```

这样当 DEBUG 为 True 的时候，我们可以看出 django 执行了什么 SQL 语句

```
tu@pro ~/zqxt $ python manage.py shell
```

```
In [1]: from blog.models import *
```

```
In [2]: Author.objects.all()
```

```
Out[2]: (0.001) SELECT "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_author" LIMIT 21; args=()
<QuerySet [<Author: WeizhongTu>, <Author: twz915>, <Author: dachui>, <Author: zhe>, <Author: zhen>]>
```

标记背景为 黄色的部分就是打出的 log。

假如，我们取出10篇Django相关的文章，并需要用到作者的姓名

```
In [13]: articles = Article.objects.all()[0:10]
```

```
In [14]: a1 = articles[0] # 取第一篇
```

```
(0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM "blog_article" LIMIT 1; args=()
```

```
In [15]: a1.title
```

```
Out[15]: u'Django \u6559\u7a0b_1'
```

```
In [16]: a1.author_id
```

```
Out[16]: 5
```

```
In [17]: a1.author.name # 再次查询了数据库，注意！！
```

```
(0.000) SELECT "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_author" WHERE "blog_author"."id" = 5; args=(5,)
Out[17]: u'zhen'
```

这样的话我们遍历查询结果的时候就会查询很多次数据库，能不能只查询一次，把作者的信息也查出来呢？

当然可以，这时就用到 select\_related，我们的数据库设计的是一篇文章只能有一个作者，一个作者可以有多篇文章。

现在要查询文章的时候连同作者一起查询出来，“文章”和“作者”的关系就是多对一，换句话说，就是一篇文章只可能有一个作者。

```
In [18]: articles = Article.objects.all().select_related('author')[:10]
```

```
In [19]: a1 = articles[0] # 取第一篇
```

```
(0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score", "blog_author"."id", "blog_author"."name", "blog_author"."qq", "blog_author"."addr", "blog_author"."email" FROM "blog_article" INNER JOIN "blog_author" ON ("blog_article"."author_id" = "blog_author"."id") LIMIT 1; args=()
```

```
In [20]: a1.title
```

```
Out[20]: u'Django \u6559\u7a0b_1'
```

```
In [21]: a1.author.name # 嘻嘻，没有再次查询数据库！！
```

```
Out[21]: u'zhen'
```

## 7. prefetch\_related 优化一对多，多对多查询

和 select\_related 功能类似，但是实现不同。

select\_related 是使用 SQL JOIN 一次性取出相关的内容。

prefetch\_related 用于一对多，多对多的情况，这时 select\_related 用不了，因为当前一条有好几条与之相关的内容。

prefetch\_related 是通过再执行一条额外的 SQL 语句，然后用 Python 把两次 SQL 查询的内容关联（joining）到一起

我们来看个例子，查询文章的同时，查询文章对应的标签。“文章”与“标签”是多对多的关系。

```
In [24]: articles = Article.objects.all().prefetch_related('tags')[:10]
```

```
In [25]: articles
```

```
Out[25]: (0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM "blog_article" LIMIT 10; args=()
```

```
(0.001) SELECT ("blog_article_tags"."article_id") AS "_prefetch_related_val_article_id", "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10); args=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
<QuerySet [<Article: Django 教程_1>, <Article: Django 教程_2>, <Article: Django 教程_3>, <Article: Django 教程_4>, <Article: Django 教程_5>, <Article: Django 教程_6>, <Article: Django 教程_7>, <Article: Django 教程_8>, <Article: Django 教程_9>, <Article: Django 教程_10>]>
```

遍历查询的结果：

不用 prefetch\_related 时

```
In [9]: articles = Article.objects.all()[:3]
```

```
In [10]: for a in articles:
```

```
...:     print a.title, a.tags.all()
```

```
...:
```

```
(0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM "blog_article" LIMIT 3; args=()
```

```
(0.000) SELECT "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" = 1 LIMIT 21; args=(1,)
```

```
Django 教程_1 <QuerySet [<Tag: Django>]>
```

```
(0.000) SELECT "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" = 2 LIMIT 21; args=(2,)
```

```
Django 教程_2 <QuerySet [<Tag: Django>]>
```

```
(0.000) SELECT "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" = 3 LIMIT 21; args=(3,)
```

```
Django 教程_3 <QuerySet [<Tag: Django>]>
```

用 `prefetch_related` 我们看一下是什么样子

```
In [11]: articles = Article.objects.all().prefetch_related(tags)[:3]
```

```
In [12]: for a in articles:
```

```
...:     print a.title, a.tags.all()
```

```
...:
```

```
(0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM "blog_article" LIMIT 3; args=()
```

```
(0.000) SELECT ("blog_article_tags"."article_id") AS "_prefetch_related_val_article_id", "blog_tag"."id", "blog_tag"."name" FROM "blog_tag" INNER JOIN "blog_article_tags" ON ("blog_tag"."id" = "blog_article_tags"."tag_id") WHERE "blog_article_tags"."article_id" IN (1, 2, 3); args=(1, 2, 3)
```

```
Django 教程_1 <QuerySet [<Tag: Django>]>
```

```
Django 教程_2 <QuerySet [<Tag: Django>]>
```

```
Django 教程_3 <QuerySet [<Tag: Django>]>
```

我们可以看到第二条 SQL 语句，一次性查出了所有相关的内容。

## 8. defer 排除不需要的字段

在复杂的情况下，表中可能有些字段内容非常多，取出来转化成 Python 对象会占用大量的资源。

这时候可以用 `defer` 来排除这些字段，比如我们在文章列表页，只需要文章的标题和作者，没有必要把文章的内容也获取出来（因为会转换成 python 对象，浪费内存）

```
In [13]: Article.objects.all()
```

```
Out[13]: (0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."content", "blog_article"."score" FROM "blog_article" LIMIT 21; args=()
```

```
<QuerySet [<Article: Django 教程_1>, <Article: Django 教程_2>, <Article: Django 教程_3>, <Article: Django 教程_4>, <Article: Django 教程_5>, <Article: Django 教程_6>, <Article: Django 教程_7>, <Article: Django 教程_8>, <Article: Django 教程_9>, <Article: Django 教程_10>, <Article: Django 教程_11>, <Article: Django 教程_12>, <Article: Django 教程_13>, <Article: Django 教程_14>, <Article: Django 教程_15>, <Article: Django 教程_16>, <Article: Django 教程_17>, <Article: Django 教程_18>, <Article: Django 教程_19>, <Article: Django 教程_20>, '...(remaining elements truncated)...']>
```

```
In [14]: Article.objects.all().defer('content')
```

```
Out[14]: (0.000) SELECT "blog_article"."id", "blog_article"."title", "blog_article"."author_id", "blog_article"."score" FROM "blog_article" LIMIT 21; args=() # 注意这里没有查 content 字段了
```

```
<QuerySet [<Article: Django 教程_1>, <Article: Django 教程_2>, <Article: Django 教程_3>, <Article: Django 教程_4>, <Article: Django 教程_5>, <Article: Django 教程_6>, <Article: Django 教程_7>, <Article: Django 教程_8>, <Article: Django 教程_9>, <Article: Django 教程_10>, <Article: Django 教程_11>, <Article: Django 教程_12>, <Article: Django 教程_13>, <Article: Django 教程_14>, <Article: Django 教程_15>, <Article: Django 教程_16>, <Article: Django 教程_17>, <Article: Django 教程_18>, <Article: Django 教程_19>, <Article: Django 教程_20>, '...(remaining elements truncated)...']>
```

## 9. only 仅选择需要的字段

和 `defer` 相反，`only` 用于取出需要的字段，假如我们只需要查出 作者的名称

```
In [15]: Author.objects.all().only('name')
```

```
Out[15]: (0.000) SELECT "blog_author"."id", "blog_author"."name" FROM "blog_author" LIMIT 21; args=()
```

```
<QuerySet [<Author: WeizhongTu>, <Author: twz915>, <Author: dachui>, <Author: zhe>, <Author: zhen>]>
```

细心的同学会发现，我们让查 `name`，`id` 也查了，这个 `id` 是主键，能不能没有这个 `id` 呢？

试一下原生的 SQL 查询

```
In [26]: authors = Author.objects.raw('select name from blog_author limit 1')
```

```
In [27]: author = authors[0]
```

```
(0.000) select name from blog_author limit 1; args=()
```

```
InvalidQuery Traceback (most recent call last)
```

```
<ipython-input-27-51c5f914fff2> in <module>()
```

```
----> 1author = authors[0]
```

```
/usr/local/lib/python2.7/site-packages/django/db/models/query.py in __getitem__(self, k)
```

```
1275
```

```
1276 def __getitem__(self, k):
```

```

-> 1277     return list(self)[k]
    1278
    1279     @property

/usr/local/lib/python2.7/site-packages/django/db/models/query.py in __iter__(self)
    1250         if skip:
    1251             if self.model._meta.pk.attname in skip:
-> 1252                 raise InvalidQuery('Raw query must include the primary key')
    1253         model_cls = self.model
    1254         fields = [self.model._fields.get(c) for c in self.columns]

```

InvalidQuery: Raw query must include the primary key

报错信息说 非法查询，原生SQL查询必须包含 主键！

再试试直接执行 SQL

```

tu@pro ~/zqxt $ python manage.py dbshell
SQLite version 3.14.0 2016-07-26 15:17:14
Enter ".help" for usage hints.
sqlite> select name from blog_author limit 1;
WeizhongTu    <-- 成功!!!

```

虽然直接执行SQL语句可以这样，但是 `django queryset` 不允许这样做，一般也不需要关心，反正 **only** 一定会取出你指定了的字段。

## 10. 自定义聚合功能

我们前面看到了 `django.db.models` 中有 `Count`, `Avg`, `Sum` 等，但是有一些没有的，比如 `GROUP_CONCAT`，它用来聚合时将符合某分组条件(`group by`)的不同的值，连到一起，作为整体返回。

我们来演示一下，如果实现 `GROUP_CONCAT` 功能。

新建一个文件 比如 `my_aggregate.py`

```

from django.db.models import Aggregate, CharField

class GroupConcat(Aggregate):
    function = 'GROUP_CONCAT'
    template = '%(function)s(%(distinct)s%(expressions)s%(ordering)s%(separator)s)'

    def __init__(self, expression, distinct=False, ordering=None, separator=',', **extra):
        super(GroupConcat, self).__init__(
            expression,
            distinct='DISTINCT ' if distinct else '',
            ordering=' ORDER BY %s' % ordering if ordering is not None else '',
            separator=' SEPARATOR "%s"' % separator,
            output_field=CharField(),
            **extra
        )

```

代码来自: <http://stackoverflow.com/a/40478702/2714931> (我根据一个回复改写的增强版本)

使用时先引入 `GroupConcat` 这个类，比如聚合后的错误日志记录有这些字段 `time`, `level`, `info`

我们想把 `level`, `info` 一样的 聚到一起，按时间和发生次数倒序排列，并含有每次日志发生的时间。

```

ErrorLogModel.objects.values('level', 'info').annotate(
    count=Count(1), time=GroupConcat('time', ordering='time DESC', separator=' | ')
).order_by('-time', '-count')

```

参考资源:

QuerySet API: <https://docs.djangoproject.com/en/2.2/ref/models/queriesets/>

聚合相关: <https://docs.djangoproject.com/en/2.2/topics/db/aggregation/>

[小额赞助, 支持作者!](#)

[« Django QuerySet API](#)

[BVDN-1 这个教程要做什么 »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 自定义Field

[« Django QuerySet 进阶](#)  
[Django 数据表更改 »](#)

Django 的官方提供了很多的 Field, 但是有时候还是不能满足我们的需求, 不过Django提供了自定义 Field 的方法:

**提示:** 如果现在用不到可以跳过这一节, 不影响后面的学习, 等用到的时候再来学习不迟。

来一个简单的例子吧。

**1. 减少文本的长度, 保存数据的时候压缩, 读取的时候解压缩, 如果发现压缩后更长, 就用原文本直接存储:**

## Django 1.7 以下

```
from django.db import models

class CompressedTextField(models.TextField):
    """ model Fields for storing text in a compressed format (bz2 by default) """
    __metaclass__ = models.SubfieldBase

    def to_python(self, value):
        if not value:
            return value

        try:
            return value.decode('base64').decode('bz2').decode('utf-8')
        except Exception:
            return value

    def get_prep_value(self, value):
        if not value:
            return value

        try:
            value.decode('base64')
            return value
        except Exception:
            try:
                tmp = value.encode('utf-8').encode('bz2').encode('base64')
            except Exception:
                return value
            else:
                if len(tmp) > len(value):
                    return value

            return tmp
```

**to\_python** 函数用于转化数据库中的字符到 Python 的变量, **get\_prep\_value** 用于将 Python 变量处理后(此处为压缩) 保存到数据库, 使用和 Django 自带的 Field 一样。

## Django 1.8 以上版本, 可以用

```
#coding:utf-8
from django.db import models

class CompressedTextField(models.TextField):
    """
```

```

model Fields for storing text in a compressed format (bz2 by default)
"""

def from_db_value(self, value, expression, connection, context):
    if not value:
        return value
    try:
        return value.decode('base64').decode('bz2').decode('utf-8')
    except Exception:
        return value

def to_python(self, value):
    if not value:
        return value
    try:
        return value.decode('base64').decode('bz2').decode('utf-8')
    except Exception:
        return value

def get_prep_value(self, value):
    if not value:
        return value
    try:
        value.decode('base64')
        return value
    except Exception:
        try:
            return value.encode('utf-8').encode('bz2').encode('base64')
        except Exception:
            return value

```

Django 1.8及以上版本中, `from_db_value` 函数用于转化数据库中的字符到 **Python** 的变量。

## 2. 比如我们想保存一个 列表到数据库中, 在读取用的时候要是 **Python** 的列表的形式, 我们来自自己写一个 **ListField**:

这个 **ListField** 继承自 **TextField**, 代码如下:

```

from django.db import models
import ast

class ListField(models.TextField):
    __metaclass__ = models.SubfieldBase
    description = "Stores a python list"

    def __init__(self, *args, **kwargs):
        super(ListField, self).__init__(*args, **kwargs)

    def to_python(self, value):
        if not value:
            value = []

        if isinstance(value, list):
            return value

        return ast.literal_eval(value)

    def get_prep_value(self, value):
        if value is None:
            return value

        return unicode(value) # use str(value) in Python 3

    def value_to_string(self, obj):
        value = self._get_val_from_obj(obj)
        return self.get_db_prep_value(value)

```

使用它很简单，首先导入 `ListField`，像自带的 `Field` 一样使用：

```
class Article(models.Model):  
    labels = ListField()
```

在终端上尝试（运行 `python manage.py shell` 进入）：

```
>>> from app.models import Article  
>>> d = Article()  
>>> d.labels  
[]  
>>> d.labels = ["Python", "Django"]  
>>> d.labels  
["Python", "Django"]
```

示例代码：

 [zqxt\\_custom\\_fields\\_project.zip](#)

下载上面的代码，解压，进入项目目录，输入 `python manage.py shell` 搞起

```
>>> from blog.models import Article  
  
>>> a = Article()  
>>> a.labels.append('Django')  
>>> a.labels.append('custom fields')  
  
>>> a.labels  
['Django', 'custom fields']  
  
>>> type(a.labels)  
<type 'list'>  
  
>>> a.content = u'我正在写一篇关于自定义Django Fields的教程'  
>>> a.save()
```

参考网址：

<https://djangosnippets.org/snippets/2014/>

<https://docs.djangoproject.com/en/dev/howto/custom-model-fields/>

[小额赞助，支持作者！](#)

[« Django QuerySet 进阶](#)

[Django 数据表更改 »](#)

 **零成本开启敏捷开发 五人以下团队免费**

免费体验

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)



[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 自定义Field

[« Django QuerySet 进阶](#)  
[Django 数据表更改 »](#)

Django 的官方提供了很多的 Field，但是有时候还是不能满足我们的需求，不过Django提供了自定义 Field 的方法：

**提示：**如果现在用不到可以跳过这一节，不影响后面的学习，等用到的时候再来学习不迟。

来一个简单的例子吧。

**1. 减少文本的长度，保存数据的时候压缩，读取的时候解压缩，如果发现压缩后更长，就用原文本直接存储：**

### Django 1.7 以下

```
from django.db import models

class CompressedTextField(models.TextField):
    """    model Fields for storing text in a compressed format (bz2 by default)    """
    __metaclass__ = models.SubfieldBase

    def to_python(self, value):
        if not value:
            return value

        try:
            return value.decode('base64').decode('bz2').decode('utf-8')
        except Exception:
            return value

    def get_prep_value(self, value):
        if not value:
            return value

        try:
            value.decode('base64')
            return value
        except Exception:
            try:
                tmp = value.encode('utf-8').encode('bz2').encode('base64')
            except Exception:
                return value
            else:
                return tmp
```

```

        if len(tmp) > len(value):
            return value

    return tmp

```

**to\_python** 函数用于转化数据库中的字符到 **Python** 的变量，**get\_prep\_value** 用于将 **Python** 变量处理后(此处为压缩) 保存到数据库，使用 **和 Django 自带的 Field 一样**。

**Django 1.8 以上版本，可以用**

```

#coding:utf-8
from django.db import models

class CompressedTextField(models.TextField):
    """
    model Fields for storing text in a compressed format (bz2 by default)
    """

    def from_db_value(self, value, expression, connection, context):
        if not value:
            return value
        try:
            return value.decode('base64').decode('bz2').decode('utf-8')
        except Exception:
            return value

    def to_python(self, value):
        if not value:
            return value
        try:
            return value.decode('base64').decode('bz2').decode('utf-8')
        except Exception:
            return value

    def get_prep_value(self, value):
        if not value:
            return value
        try:
            value.decode('base64')
            return value
        except Exception:
            try:
                return value.encode('utf-8').encode('bz2').encode('base64')
            except Exception:
                return value

```

Django 1.8及以上版本中，**from\_db\_value** 函数用于转化数据库中的字符到 **Python** 的变量。

**2. 比如我们想保存一个 列表到数据库中，在读取用的时候要是 Python 的列表的形式，我们来自自己写一个 ListField:**

这个 **ListField** 继承自 **TextField**，代码如下：

```

from django.db import models
import ast

class ListField(models.TextField):
    __metaclass__ = models.SubfieldBase
    description = "Stores a python list"

    def __init__(self, *args, **kwargs):
        super(ListField, self).__init__(*args, **kwargs)

    def to_python(self, value):
        if not value:
            value = []

```

```

        if isinstance(value, list):
            return value

        return ast.literal_eval(value)

    def get_prep_value(self, value):
        if value is None:
            return value

        return unicode(value) # use str(value) in Python 3

    def value_to_string(self, obj):
        value = self._get_val_from_obj(obj)
        return self.get_db_prep_value(value)

```

使用它很简单，首先导入 **ListField**，像自带的 **Field** 一样使用：

```

class Article(models.Model):
    labels = ListField()

```

在终端上尝试（运行 `python manage.py shell` 进入）：

```

>>> from app.models import Article
>>> d = Article()
>>> d.labels
[]
>>> d.labels = ["Python", "Django"]
>>> d.labels
["Python", "Django"]

```

示例代码：

 [zqxt\\_custom\\_fields\\_project.zip](#)

下载上面的代码，解压，进入项目目录，输入 `python manage.py shell` 搞起

```

>>> from blog.models import Article

>>> a = Article()
>>> a.labels.append('Django')
>>> a.labels.append('custom fields')

>>> a.labels
['Django', 'custom fields']

>>> type(a.labels)
<type 'list'>

>>> a.content = u'我正在写一篇关于自定义Django Fields的教程'
>>> a.save()

```

参考网址：

<https://djangosnippets.org/snippets/2014/>

<https://docs.djangoproject.com/en/dev/howto/custom-model-fields/>

小额赞助，支持作者！

« Django QuerySet 进阶

Django 数据表更改 »



零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

# Django 数据表更改

[« Django 开发内容管理系统（第四天） Django 后台 »](#)

我们设计数据库的时候，早期设计完后，后期会发现不完善，要对数据表进行更改，这时候就要用到本节的知识。

## Django 1.7.x 和后来的版本：

Django 1.7.x 及以后的版本集成了 South 的功能，在修改 `models.py` 后运行：

```
python manage.py makemigrations
python manage.py migrate
```

这两行命令就会对我们的 `models.py` 进行检测，自动发现需要更改的，应用到数据库中去。

## Django 1.6.x 及以前：

写过 Django 项目的同学，必然会遇到这个问题：

在 Django 1.6 以及以前的版本中，我们测试，当发现 `model` 要改，怎么办？

我们修改了 `models.py` 之后，我们运行：

```
python manage.py syncdb
```

这句话只会将我们在 `models.py` 中新加的类创建相应的表。

对于原来有的，现在删除了的类，Django 会询问是否要删除数据库中已经存在的相关数据表。

如果在原来的类上增加字段或者删除字段，可以参考这个命令：

```
python manage.py sql appname
```

给出的 SQL 语句，然后自己手动到数据库执行 SQL。但是这样非常容易出错！

Django 的第三方 app South 就是专门做数据库表结构自动迁移工作，Jacob Kaplan-Moss 曾做过一次调查，South 名列最受欢迎的第三方 app。事实上，它现在已经俨然成为 Django 事实上的数据库表迁移标准，很多第三方 app 都会带 South migrations 脚本，Django 1.7 中集成了 South 的功能。

## 1, 安装 South

```
(sudo) pip install South
```

## 2. 使用方法

一个好的程序使用起来必定是简单的，South和它的宗旨一样，使用简单。只需要简单几步，针对已经建好model和创建完表的应用。

把south加入到settings.py中的INSTALL\_APPS中

```
# Application definition
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'blog',
    'south',
)
```

修改好后运行一次 `python manage.py syncdb`，Django会新建一个 `south_migrationhistory` 表，用来记录数据表更改(Migration)的历史纪录。

```
$ python manage.py syncdb
Syncing...
Creating tables ...
Creating table south_migrationhistory
Installing custom SQL ...
Installing indexes ...
No fixtures found.
```

```
Synced:
> django.contrib.admin
> django.contrib.auth
> django.contrib.contenttypes
> django.contrib.sessions
> django.contrib.messages
> django.contrib.staticfiles
> blog
> south
```

```
Not synced (use migrations):
```

如果要把之前建好的比如 `blog` 这个 app 使用 South 来管理：

```
$ python manage.py convert_to_south blog
```

你会发现`blog`文件夹中多了一个 `migrations` 目录，里面有一个 `0001_initial.py` 文件。

注：如果 `blog` 这个 app 之前就创建过相关的表，可以用下面的来“假装”用 South 创建（伪创建，在改动 `models.py` 之前运行这个）

```
python manage.py migrate blog --fake
```

意思是这个表我以前已经建好了，用 South 只是纪一下这个创建记录，下次 `migrate` 的时候不必再创建了。

原理就是 `south_migrationhistory` 中记录下了 `models.py` 的修改的历史，下次再修改时会和最近一次记录比较，发现改变了什么，然后生成相应的对应文件，最终执行相应的 SQL 更改原有的数据表。

接着，当你对 `Blog` models 做任何修改后，只要执行：

```
$ python manage.py schemamigration blog --auto
```

South 就会帮助我们找出哪些地方做了修改，如果你新增的数据表没有给 `default` 值，并且没有设置 `null=True`, south 会问你一些问题，因为新增的 `column` 对于原来的旧的数据不能为 `Null` 的话就得有一个值。顺利的话，在 `migrations` 文件夹下会产生一个 `0002_add_mobile_column.py`，但是这一步并没有真正修改数据库的表，我们需要执行 `python manage.py migrate`：

```
$ python manage.py migrate
Running migrations for blog:
- Migrating forwards to 0002_add_mobile_column.
> blog:0002_add_mobile_column
- Loading initial data for blog.
No fixtures found.
```

这样所做的更改就写入到了数据库中了。

## 恢复到以前

South 好处就是可以随时恢复到之前的一个版本，比如我们想要回到最开始的那个版本：

```
> python manage.py migrate blog 0001
- Soft matched migration 0001 to 0001_initial.
Running migrations for blog:
- Migrating backwards to just after 0001_initial.
< blog:0002_add_mobile_column
```

这样就搞定了，数据库就恢复到以前了，比你手动更改要方便太多了。

[小额赞助，支持作者！](#)

[« Django 开发内容管理系统（第四天）](#)  
[Django 后台 »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

---

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在 [阿里云](#) ECS

推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 数据表更改

[« Django 开发内容管理系统（第四天） Django 后台 »](#)

我们设计数据库的时候，早期设计完后，后期会发现不完善，要对数据表进行更改，这时候就要用到本节的知识。

### Django 1.7.x 和后来的版本：

Django 1.7.x 及以后的版本集成了 South 的功能，在修改 `models.py` 后运行：

```
python manage.py makemigrations
python manage.py migrate
```

这两行命令就会对我们的 `models.py` 进行检测，自动发现需要更改的，应用到数据库中去。

### Django 1.6.x 及以前：

写过 Django 项目的同学，必然会遇到这个问题：

在 Django 1.6 以及以前的版本中，我们测试，当发现 `model` 要改，怎么办？

我们修改了 `models.py` 之后，我们运行：

```
python manage.py syncdb
```

这句话只会将我们在 `models.py` 中新加的类创建相应的表。

对于原来有的，现在删除了的类，Django 会询问是否要删除数据库中已经存在的相关数据表。

如果在原来的类上增加字段或者删除字段，可以参考这个命令：

```
python manage.py sql appname
```

给出的 SQL 语句，然后自己手动到数据库执行 SQL。但是这样非常容易出错！

Django 的第三方 app South 就是专门做数据库表结构自动迁移工作，Jacob Kaplan-Moss 曾做过一次调查，South 名列最受欢迎的第三方 app。事实上，它现在已经俨然成为 Django 事实上的数据库表迁移标准，很多第三方 app 都会带 South migrations 脚本，Django 1.7 中集成了 South 的功能。

### 1, 安装 South



```
(sudo) pip install South
```

## 2. 使用方法

一个好的程序使用起来必定是简单的，**South**和它的宗旨一样，使用简单。只需要简单几步，针对已经建好model和创建完表的应用。

把**south**加入到**settings.py**中的**INSTALLED\_APPS**中

```
# Application definition
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'blog',
    'south',
)
```

修改好后运行一次 `python manage.py syncdb`，Django会新建一个 `south_migrationhistory` 表，用来记录数据表更改(Migration)的历史纪录。

```
$ python manage.py syncdb
Syncing...
Creating tables ...
Creating table south_migrationhistory
Installing custom SQL ...
Installing indexes ...
No fixtures found.
```

```
Synced:
> django.contrib.admin
> django.contrib.auth
> django.contrib.contenttypes
> django.contrib.sessions
> django.contrib.messages
> django.contrib.staticfiles
> blog
> south
```

```
Not synced (use migrations):
```

如果要把之前建好的比如 **blog** 这个 app 使用 **South** 来管理：

```
$ python manage.py convert_to_south blog
```

你会发现**blog**文件夹中多了一个 **migrations** 目录，里面有一个 `0001_initial.py` 文件。

注：如果 **blog** 这个 app 之前就创建过相关的表，可以用下面的来“假装”用 **South** 创建（伪创建，在改动 `models.py` 之前运行这个）

```
python manage.py migrate blog --fake
```

意思是这个表我以前已经建好了，用 **South** 只是纪一下这个创建记录，下次 **migrate** 的时候不必再创建了。

原理就是 `south_migrationhistory` 中记录下了 `models.py` 的修改的历史，下次再修改时会和最近一次记录比较，发现改变了

什么，然后生成相应的对应文件，最终执行相应的 SQL 更改原有的数据表。

接着，当你对 `Blog` models 做任何修改后，只要执行：

```
$ python manage.py schemamigration blog --auto
```

South 就会帮助我们找出哪些地方做了修改，如果你新增的数据表没有给 `default` 值，并且没有设置 `null=True`, south 会问你一些问题，因为新增的 `column` 对于原来的旧的数据不能为 `Null` 的话就得有一个值。顺利的话，在 `migrations` 文件夹下会产生一个 `0002_add_mobile_column.py`，但是这一步并没有真正修改数据库的表，我们需要执行 `python manage.py migrate`：

```
$ python manage.py migrate
Running migrations for blog:
- Migrating forwards to 0002_add_mobile_column.
> blog:0002_add_mobile_column
- Loading initial data for blog.
No fixtures found.
```

这样所做的更改就写入到了数据库中了。

## 恢复到以前

South 好处就是可以随时恢复到之前的一个版本，比如我们想要回到最开始的那个版本：

```
> python manage.py migrate blog 0001
- Soft matched migration 0001 to 0001_initial.
Running migrations for blog:
- Migrating backwards to just after 0001_initial.
< blog:0002_add_mobile_column
```

这样就搞定了，数据库就恢复到以前了，比你手动更改要方便太多了。

[小额赞助，支持作者！](#)

[« Django 开发内容管理系统（第四天）](#)

[Django 后台 »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 后台

[« Django 数据表更改 Django 表单 »](#)

django的后台我们只要加少些代码, 就可以实现强大的功能。

与后台相关文件: 每个app中的 `admin.py` 文件与后台相关。

下面示例是做一个后台添加博客文章的例子:

## 一, 新建一个 名称为 `zqxt_admin` 的项目

```
django-admin.py startproject zqxt_admin
```

## 二, 新建一个 叫做 `blog` 的app

```
# 进入 zqxt_admin 文件夹
cd zqxt_admin

# 创建 blog 这个 app
python manage.py startapp blog
```

注意: 不同版本的 Django 创建 project 和 app 出来的文件会有一些不同

## 三, 修改 `blog` 文件夹中的 `models.py`

```
# coding:utf-8
from django.db import models

class Article(models.Model):
    title = models.CharField(u'标题', max_length=256)
    content = models.TextField(u'内容')

    pub_date = models.DateTimeField(u'发表时间', auto_now_add=True, editable = True)
    update_time = models.DateTimeField(u'更新时间', auto_now=True, null=True)
```

## 四, 把 `blog` 加入到`settings.py`中的`INSTALLED_APPS`中

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'blog',
)
```

提示: `INSTALLED_APPS` 是一个元组, 每次加入新的app的时候, 在后面都加一个逗号, 这是一个好习惯。

## 五, 同步所有的数据表

```
# 进入包含有 manage.py 的文件夹
python manage.py makemigrations
```

```
python manage.py migrate
```

注意: Django 1.6.x 及以下的版本需要用以下命令  
`python manage.py syncdb`

可以看到:

Creating tables ...

Creating table django\_admin\_log

Creating table auth\_permission

Creating table auth\_group\_permissions

Creating table auth\_group

Creating table auth\_user\_groups

Creating table auth\_user\_user\_permissions

Creating table auth\_user

Creating table django\_content\_type

Creating table django\_session

Creating table blog\_article

You just installed Django's auth system, which means you don't have any superusers defined.

Would you like to create one now? (yes/no): yes

Username (leave blank to use 'tu'): tu

Email address:

Password:

Password (again):

Superuser created successfully.

Installing custom SQL ...

Installing indexes ...

Installed 0 object(s) from 0 fixture(s)

如果是 Django 不主动提示创建管理员 (Django 1.9不提示) 用下面的命令创建一个帐号

```
python manage.py createsuperuser
```

## 六, 修改 admin.py

进入 blog 文件夹, 修改 admin.py 文件 (如果没有新建一个), 内容如下

```
from django.contrib import admin
from .models import Article
```

```
admin.site.register(Article)
```

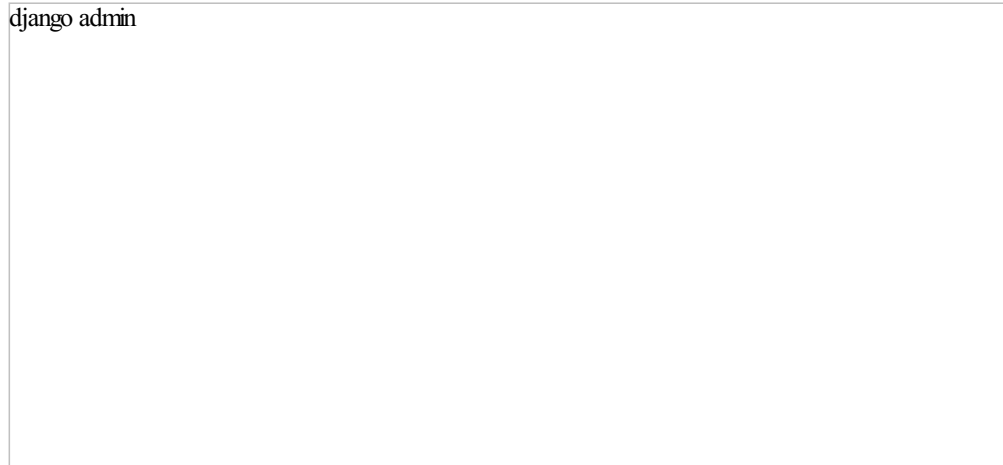
只需要这三行代码, 我们就可以拥有一个强大的后台!

提示: `urls.py`中关于 `admin`的已经默认开启, 如果没有, [参考这里](#)。

## 七, 打开 开发服务器

```
python manage.py runserver
# 如果提示 8000 端口已经被占用, 可以用 python manage.py runserver 8001 以此类推
```

访问 <http://localhost:8000/admin/> 输入设定的帐号和密码, 就可以看到:



点击 **Articles**, 动手输入 添加几篇文章, 就可以看到:

我们会发现所有的文章都是叫 **Article object**, 这样肯定不好, 比如我们要修改, 如何知道要修改哪个呢?

我们修改一下 `blog` 中的 `models.py`

```
# coding:utf-8
from django.db import models

class Article(models.Model):
    title = models.CharField(u'标题', max_length=256)
    content = models.TextField(u'内容')

    pub_date = models.DateTimeField(u'发表时间', auto_now_add=True, editable = True)
    update_time = models.DateTimeField(u'更新时间', auto_now=True, null=True)

    def __unicode__(self):# 在Python3中用 __str__ 代替 __unicode__
        return self.title
```

我们加了一个 `__unicode__` 函数, 刷新后台网页, 会看到:

所以**推荐定义 Model 的时候 写一个 `__unicode__` 函数(或 `__str__` 函数)**

**技能提升:** 如何兼容python2.x和python3.x呢?

示例如下:

```
# coding:utf-8
from __future__ import unicode_literals

from django.db import models
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible
class Article(models.Model):
```

```

title = models.CharField('标题', max_length=256)
content = models.TextField('内容')

pub_date = models.DateTimeField('发表时间', auto_now_add=True, editable = True)
update_time = models.DateTimeField('更新时间', auto_now=True, null=True)

def __str__(self):
    return self.title

```

`python_2_unicode_compatible` 会自动做一些处理去适应python不同的版本，本例中的 `unicode_literals` 可以让python2.x 也像python3 那个处理 `unicode` 字符，以便有更好地兼容性。

## 八，在列表显示与字段相关的其它内容

后台已经基本上做出来了，可是如果我们还需要显示一些其它的fields，如何做呢？

```

from django.contrib import admin
from .models import Article

class ArticleAdmin(admin.ModelAdmin):
    list_display = ('title', 'pub_date', 'update_time',)

admin.site.register(Article, ArticleAdmin)

```

`list_display` 就是来配置要显示的字段的，当然也可以显示非字段内容，或者字段相关的内容，比如：

```

class Person(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)

    def my_property(self):
        return self.first_name + ' ' + self.last_name
    my_property.short_description = "Full name of the person"

    full_name = property(my_property)

```

在admin.py中

```

from django.contrib import admin
from .models import Article, Person

class ArticleAdmin(admin.ModelAdmin):
    list_display = ('title', 'pub_date', 'update_time',)

class PersonAdmin(admin.ModelAdmin):
    list_display = ('full_name',)

admin.site.register(Article, ArticleAdmin)
admin.site.register(Person, PersonAdmin)

```

到这里我们发现我们又有新的需求，比如要改 `models.py` 中的字段，添加一个文章的状态（草稿，正式发布），这时候我们就需要更改表，`django 1.7`以前的都不会自动更改表，我们需要用第三方插件 `South`，参见 [Django 迁移数据](#)。

Django 1.7 及以上用以下命令来同步数据库表的更改


```

python manage.py makemigrations
python manage.py migrate

```

本节代码下载：

 [zqxt\\_admin \(Django 1.6\).zip](#) (基于Django 1.6，后台帐号 tu 密码 zqxt)

 [zqxt\\_admin \(Django 1.9\).zip](#) (基于 Django 1.9 后台帐号 tu 密码 ziqiangxuetang)

其它一些常用的功能:

搜索功能: `search_fields = ('title', 'content',)` 这样就可以按照 标题或内容搜索了

[https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.search\\_fields](https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.search_fields)

筛选功能: `list_filter = ('status',)` 这样就可以根据文章的状态去筛选, 比如找出是草稿的文章

[https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.list\\_filter](https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.list_filter)

新增或修改时的布局顺序: <https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.fieldsets>

有时候我们需要对 `django admin site` 进行修改以满足自己的需求, 那么我们可以从哪些地方入手呢?

以下举例说明:

**1.定制加载的列表, 根据不同的人显示不同的内容列表, 比如输入员只能看见自己输入的, 审核员能看到所有的草稿, 这时候就需要重写 `get_queryset` 方法**

```
class MyModelAdmin(admin.ModelAdmin):
    def get_queryset(self, request):
        qs = super(MyModelAdmin, self).get_queryset(request)
        if request.user.is_superuser:
            return qs
        else:
            return qs.filter(author=request.user)
```

该类实现的功能是如果是超级管理员就列出所有的, 如果不是, 就仅列出访问者自己相关的

**2.定制搜索功能 (django 1.6及以上才有)**

```
class PersonAdmin(admin.ModelAdmin):
    list_display = ('name', 'age')
    search_fields = ('name',)

    def get_search_results(self, request, queryset, search_term):
        queryset, use_distinct = super(PersonAdmin, self).get_search_results(request, queryset, search_term)
        try:
            search_term_as_int = int(search_term)
            queryset |= self.model.objects.filter(age=search_term_as_int)
        except:
            pass
        return queryset, use_distinct
```

`queryset` 是默认的结果, `search_term` 是在后台搜索的关键词

**3.修改保存时的一些操作, 可以检查用户, 保存的内容等, 比如保存时加上添加人**

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        obj.user = request.user
        obj.save()
```

其中 `obj` 是修改后的对象, `form` 是返回的表单 (修改后的), 当新建一个对象时 `change = False`, 当修改一个对象时 `change = True`

如果需要获取修改前的对象的内容可以用

```
from django.contrib import admin
```

```
class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        obj_original = self.model.objects.get(pk=obj.pk)
        obj.user = request.user
        obj.save()
```

那么又有问题了，这里如果原来的obj不存在，也就是如果我们是新建的一个怎么办呢，这时候可以用try,except的方法尝试获取,当然更好的方法是判断一下这个对象是新建还是修改，是新建就没有 obj\_original，是修改就有

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        if change:# 更改的时候
            obj_original = self.model.objects.get(pk=obj.pk)
        else:# 新增的时候
            obj_original = None

        obj.user = request.user
        obj.save()
```

#### 4, 删除时做一些处理,

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def delete_model(self, request, obj):
        """
        Given a model instance delete it from the database.
        """
        # handle something here
        obj.delete()
```

Django的后台非常强大，这个只是帮助大家入门，更完整的还需要查看官方文档，如果你有更好的方法或不懂的问题，欢迎评论！

参考: [官方文档](#)

[小额赞助，支持作者！](#)

[« Django 数据表更改  
Django 表单 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实



例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 后台

[« Django 数据表更改 Django 表单 »](#)

django的后台我们只要加少些代码，就可以实现强大的功能。

与后台相关文件：每个app中的 `admin.py` 文件与后台相关。

下面示例是做一个后台添加博客文章的例子：

### 一，新建一个 名称为 `zqxt_admin` 的项目

```
django-admin.py startproject zqxt_admin
```

### 二，新建一个 叫做 `blog` 的app

```
# 进入 zqxt_admin 文件夹
cd zqxt_admin

# 创建 blog 这个 app
python manage.py startapp blog
```

注意：不同版本的 Django 创建 project 和 app 出来的文件会有一些不同

### 三，修改 `blog` 文件夹中的 `models.py`

```
# coding:utf-8
from django.db import models

class Article(models.Model):
    title = models.CharField(u'标题', max_length=256)
    content = models.TextField(u'内容')

    pub_date = models.DateTimeField(u'发表时间', auto_now_add=True, editable = True)
    update_time = models.DateTimeField(u'更新时间', auto_now=True, null=True)
```

### 四，把 `blog` 加入到`settings.py`中的`INSTALLED_APPS`中

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

```
)  
'blog',  
)
```

提示：INSTALLED\_APPS 是一个元组，每次加入新的app的时候，在后面都加一个逗号，这是一个好习惯。

## 五，同步所有的数据表

```
# 进入包含有 manage.py 的文件夹  
python manage.py makemigrations  
python manage.py migrate
```

注意：Django 1.6.x 及以下的版本需要用以下命令  
python manage.py syncdb

可以看到：

Creating tables ...

Creating table django\_admin\_log

Creating table auth\_permission

Creating table auth\_group\_permissions

Creating table auth\_group

Creating table auth\_user\_groups

Creating table auth\_user\_user\_permissions

Creating table auth\_user

Creating table django\_content\_type

Creating table django\_session

Creating table blog\_article

You just installed Django's auth system, which means you don't have any superusers defined.

Would you like to create one now? (yes/no): yes

Username (leave blank to use 'tu'): tu

Email address:

Password:

Password (again):

Superuser created successfully.

Installing custom SQL ...

Installing indexes ...

Installed 0 object(s) from 0 fixture(s)

如果是 Django 不主动提示创建管理员（Django 1.9不提示）用下面的命令创建一个帐号

```
python manage.py createsuperuser
```

## 六，修改 admin.py

进入 **blog** 文件夹，修改 **admin.py** 文件（如果没有新建一个），内容如下

```
from django.contrib import admin
from .models import Article
```

```
admin.site.register(Article)
```

只需要这三行代码，我们就可以拥有一个强大的后台！

提示：**urls.py**中关于 **admin**的已经默认开启，如果没有，[参考这里](#)。

## 七，打开 开发服务器

```
python manage.py runserver
# 如果提示 8000 端口已经被占用，可以用 python manage.py runserver 8001 以此类推
```

访问 <http://localhost:8000/admin/> 输入设定的帐号和密码, 就可以看到:

django admin

点击 **Articles**，动手输入 添加几篇文章，就可以看到：

我们会发现所有的文章都是叫 **Article object**，这样肯定不好，比如我们要修改，如何知道要修改哪个呢？

我们修改一下 **blog** 中的 **models.py**

```
# coding:utf-8
from django.db import models

class Article(models.Model):
    title = models.CharField(u'标题', max_length=256)
    content = models.TextField(u'内容')

    pub_date = models.DateTimeField(u'发表时间', auto_now_add=True, editable = True)
    update_time = models.DateTimeField(u'更新时间', auto_now=True, null=True)

    def __unicode__(self):# 在Python3中用 __str__ 代替 __unicode__
        return self.title
```

我们加了一个 **\_\_unicode\_\_** 函数，刷新后台网页，会看到：

所以**推荐定义 Model 的时候 写一个 \_\_unicode\_\_ 函数(或 \_\_str\_\_ 函数)**

技能提升：如何兼容python2.x和python3.x呢？

示例如下：

```
# coding:utf-8
from __future__ import unicode_literals

from django.db import models
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible
class Article(models.Model):
    title = models.CharField('标题', max_length=256)
    content = models.TextField('内容')

    pub_date = models.DateTimeField('发表时间', auto_now_add=True, editable = True)
    update_time = models.DateTimeField('更新时间', auto_now=True, null=True)

    def __str__(self):
        return self.title
```

`python_2_unicode_compatible` 会自动做一些处理去适应python不同的版本，本例中的 `unicode_literals` 可以让python2.x 也像python3 那个处理 `unicode` 字符，以便有更好地兼容性。

## 八，在列表显示与字段相关的其它内容

后台已经基本上做出来了，可是如果还需要显示一些其它的fields，如何做呢？

```
from django.contrib import admin
from .models import Article

class ArticleAdmin(admin.ModelAdmin):
    list_display = ('title', 'pub_date', 'update_time',)

admin.site.register(Article, ArticleAdmin)
```

`list_display` 就是来配置要显示的字段的，当然也可以显示非字段内容，或者字段相关的内容，比如：

```
class Person(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)

    def my_property(self):
        return self.first_name + ' ' + self.last_name
    my_property.short_description = "Full name of the person"

    full_name = property(my_property)
```

在`admin.py`中

```
from django.contrib import admin
from .models import Article, Person

class ArticleAdmin(admin.ModelAdmin):
    list_display = ('title', 'pub_date', 'update_time',)

class PersonAdmin(admin.ModelAdmin):
    list_display = ('full_name',)

admin.site.register(Article, ArticleAdmin)
admin.site.register(Person, PersonAdmin)
```

到这里我们发现我们又有新的需求，比如要改 `models.py` 中的字段，添加一个文章的状态（草稿，正式发布），这时候我们就需要更改表，`django 1.7`以前的都不会自动更改表，我们需要用第三方插件 `South`，参见 [Django 迁移数据](#)。

Django 1.7 及以上用以下命令来同步数据库表的更改

```
python manage.py makemigrations
```

```
python manage.py migrate
```

本节代码下载:

 [zqxt\\_admin \(Django 1.6\).zip](#) (基于 Django 1.6, 后台帐号 tu 密码 zqxt)

 [zqxt\\_admin \(Django 1.9\).zip](#) (基于 Django 1.9 后台帐号 tu 密码 ziqiangxuetang)

其它一些常用的功能:

搜索功能: `search_fields = ('title', 'content',)` 这样就可以按照 标题或内容搜索了

[https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.search\\_fields](https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.search_fields)

筛选功能: `list_filter = ('status',)` 这样就可以根据文章的状态去筛选, 比如找出是草稿的文章

[https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.list\\_filter](https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.list_filter)

新增或修改时的布局顺序: <https://docs.djangoproject.com/en/dev/ref/contrib/admin/#django.contrib.admin.ModelAdmin.fieldsets>

有时候我们需要对 `django admin site` 进行修改以满足自己的需求, 那么我们可以从哪些地方入手呢?

以下举例说明:

**1. 定制加载的列表, 根据不同的人显示不同的内容列表, 比如输入员只能看见自己输入的, 审核员能看到所有的草稿, 这时候就需要重写 `get_queryset` 方法**

```
class MyModelAdmin(admin.ModelAdmin):
    def get_queryset(self, request):
        qs = super(MyModelAdmin, self).get_queryset(request)
        if request.user.is_superuser:
            return qs
        else:
            return qs.filter(author=request.user)
```

该类实现的功能是如果是超级管理员就列出所有的, 如果不是, 就仅列出访问者自己相关的

## 2. 定制搜索功能 (django 1.6及以上才有)

```
class PersonAdmin(admin.ModelAdmin):
    list_display = ('name', 'age')
    search_fields = ('name',)

    def get_search_results(self, request, queryset, search_term):
        queryset, use_distinct = super(PersonAdmin, self).get_search_results(request, queryset, search_term)
        try:
            search_term_as_int = int(search_term)
            queryset |= self.model.objects.filter(age=search_term_as_int)
        except:
            pass
        return queryset, use_distinct
```

`queryset` 是默认的结果, `search_term` 是在后台搜索的关键词

**3. 修改保存时的一些操作, 可以检查用户, 保存的内容等, 比如保存时加上添加人**

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
```

```
def save_model(self, request, obj, form, change):
    obj.user = request.user
    obj.save()
```

其中obj是修改后的对象，form是返回的表单（修改后的），当新建一个对象时 **change = False**, 当修改一个对象时 **change = True**

如果需要获取修改前的对象的内容可以用

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        obj_original = self.model.objects.get(pk=obj.pk)
        obj.user = request.user
        obj.save()
```

那么又有问题了，这里如果原来的obj不存在，也就是如果是新建的一个怎么办呢，这时候可以用try,except的方法尝试获取,当然更好的方法是判断一下这个对象是新建还是修改，是新建就没有 obj\_original，是修改就有

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        if change:# 更改的时候
            obj_original = self.model.objects.get(pk=obj.pk)
        else:# 新增的时候
            obj_original = None

        obj.user = request.user
        obj.save()
```

#### 4, 删除时做一些处理,

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def delete_model(self, request, obj):
        """
        Given a model instance delete it from the database.
        """
        # handle something here
        obj.delete()
```

Django的后台非常强大，这个只是帮助大家入门，更完整的还需要查看官方文档，如果你有更好的方法或不懂的问题，欢迎评论！

参考: [官方文档](#)

[小额赞助, 支持作者!](#)

[«Django 数据表更改](#)

[Django 表单»](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 表单

[« Django 后台](#)  
[Django 配置 »](#)

有时候我们需要在前台用 `get` 或 `post` 方法提交一些数据, 所以自己写一个网页, 用到 [html 表单](#) 的知识。

## 第一节:

比如写一个计算 `a` 和 `b` 之和的简单应用, 网页上这么写

```
<!DOCTYPE html>
<html>
<body>
<p>请输入两个数字</p>

<form action="/add/" method="get">
  a: <input type="text" name="a"> <br>
  b: <input type="text" name="b"> <br>

  <input type="submit" value="提交">
</form>

</body>
</html>
```

把这些代码保存成一个 `index.html`, 放在 `templates` 文件夹中。

网页的值传到服务器是通过 `<input>` 或 `<textarea>` 标签中的 `name` 属性来传递的, 在服务器端这么接收:

```
from django.http import HttpResponse
from django.shortcuts import render

def index(request):
    return render(request, 'index.html')

def add(request):
    a = request.GET['a']
    b = request.GET['b']
    a = int(a)
    b = int(b)
    return HttpResponse(str(a+b))
```

`request.GET` 可以看成字典, 用 `GET` 方法传递的值都会保存到其中, 可以用 `request.GET.get('key', None)` 来取值, 没有时不报错。

再将函数和网址对应上, 就可以访问了, 详情参见源码。这样就完成了基本的功能, 基本上可以用了。

但是, 比如用户输入的不是数字, 而是字母, 就出错了, 还有就是提交后再回来已经输入的数据也会没了。

当然如果我们手动将输入之后的数据在 `views` 中都获取到再传递到网页, 这样是可行的, 但是很不方便, 所以 Django 提供了更简单易用的 `forms` 来解决验证等这系列的问题。

源码下载: (建议在不同环境下按照上面的说明自己打一遍, 所以 Python 环境, Django 版本应该是类似的。

 [zqxt\\_form\\_learn1.zip](#) (Django 1.4-Django 1.7参考)

## 第二节，使用 Django 的 表单 (forms)

例子足够简单，但是能说明问题

```
新建一个 zqxt_form2 项目
django-admin.py startproject zqxt_form2
# 进入到 zqxt_form2 文件夹，新建一个 tools APP
python manage.py startapp tools
```

在tools文件夹中新建一个 **forms.py** 文件

```
from django import forms

class AddForm(forms.Form):
    a = forms.IntegerField()
    b = forms.IntegerField()
```

我们的视图函数 **views.py** 中

```
# coding:utf-8
from django.shortcuts import render
from django.http import HttpResponse

# 引入我们创建的表单类
from .forms import AddForm

def index(request):
    if request.method == 'POST':# 当提交表单时

        form = AddForm(request.POST) # form 包含提交的数据

        if form.is_valid():# 如果提交的数据合法
            a = form.cleaned_data['a']
            b = form.cleaned_data['b']
            return HttpResponse(str(int(a) + int(b)))

    else:# 当正常访问时
        form = AddForm()
        return render(request, 'index.html', {'form': form})
```

对应的模板文件 **index.html**

```
<form method='post'>
{% csrf_token %}
{{ form }}
<input type="submit" value="提交">
</form>
```

再在 **urls.py** 中对应写上这个函数

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # 注意下面这一行
    url(r'^$', 'tools.views.index', name='home'),
    url(r'^admin/', include(admin.site.urls)),
)
```

源码下载:  [zqxt\\_forms2.zip](#)



新手可能觉得这样变得更麻烦了，有些情况是这样的，但是 Django 的 forms 提供了：

1. 模板中表单的渲染
2. 数据的验证工作，某一些输入不合法也不会丢失已经输入的数据。
3. 还可以定制更复杂的验证工作，如果提供了10个输入框，必须必须要输入其中两个以上，在 forms.py 中都很容易实现

也有一些将 Django forms 渲染成 Bootstrap 的插件，也很好用，很方便。

扩展：如果提交后在同一个页面显示结果呢？请看 [Django Ajax](#)

[小额赞助，支持作者！](#)

[« Django 后台](#)

[Django 配置 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 表单

[« Django 后台](#)

## [Django 配置 »](#)

有时候我们需要在前台用 `get` 或 `post` 方法提交一些数据，所以自己写一个网页，用到 [html 表单](#) 的知识。

### 第一节：

比如写一个计算 `a` 和 `b` 之和的简单应用，网页上这么写

```
<!DOCTYPE html>
<html>
<body>
<p>请输入两个数字</p>

<form action="/add/" method="get">
  a: <input type="text" name="a"> <br>
  b: <input type="text" name="b"> <br>

  <input type="submit" value="提交">
</form>

</body>
</html>
```

把这些代码保存成一个 `index.html`，放在 `templates` 文件夹中。

网页的值传到服务器是通过 `<input>` 或 `<textarea>` 标签中的 `name` 属性来传递的，在服务器端这么接收：

```
from django.http import HttpResponse
from django.shortcuts import render

def index(request):
    return render(request, 'index.html')

def add(request):
    a = request.GET['a']
    b = request.GET['b']
    a = int(a)
    b = int(b)
    return HttpResponse(str(a+b))
```

`request.GET` 可以看成是一个字典，用 `GET` 方法传递的值都会保存到其中，可以用 `request.GET.get('key', None)` 来取值，没有时不报错。


再将函数和网址对应上，就可以访问了，详情参见源码。这样就完成了基本的功能，基本上可以用了。

但是，比如用户输入的不是数字，而是字母，就出错了，还有就是提交后再回来已经输入的数据也会没了。

当然如果我们手动将输入之后的数据在 `views` 中都获取到再传递到网页，这样是可行的，但是很不方便，所以 Django 提供了更简单易用的 `forms` 来解决验证等这一系列的问题。

源码下载：（建议在不同环境下按照上面的说明自己打一遍，所以 Python 环境，Django 版本应该是类似的。）

 [zqxt\\_form\\_learn1.zip](#)（Django 1.4-Django 1.7 参考）

 [zqxt4396.zip](#) 基于 Django 1.11 [更新于 2017-08-25 01:07:41]

### 第二节，使用 Django 的 表单 (forms)

例子足够简单，但是能说明问题

```
新建一个 zqxt_form2 项目
django-admin.py startproject zqxt_form2
```

```
# 进入到 zqxt_form2 文件夹, 新建一个 tools APP
python manage.py startapp tools
```

在tools文件夹中新建一个 **forms.py** 文件

```
from django import forms

class AddForm(forms.Form):
    a = forms.IntegerField()
    b = forms.IntegerField()
```

我们的视图函数 **views.py** 中

```
# coding:utf-8
from django.shortcuts import render
from django.http import HttpResponse

# 引入我们创建的表单类
from .forms import AddForm

def index(request):
    if request.method == 'POST':# 当提交表单时

        form = AddForm(request.POST) # form 包含提交的数据

        if form.is_valid():# 如果提交的数据合法
            a = form.cleaned_data['a']
            b = form.cleaned_data['b']
            return HttpResponse(str(int(a) + int(b)))

        else:# 当正常访问时
            form = AddForm()
            return render(request, 'index.html', {'form': form})
```

对应的模板文件 **index.html**

```
<form method='post'>
{% csrf_token %}
{{ form }}
<input type="submit" value="提交">
</form>
```

再在 **urls.py** 中对应写上这个函数

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # 注意下面这一行
    url(r'^$', 'tools.views.index', name='home'),
    url(r'^admin/', include(admin.site.urls)),
)
```

源码下载:  [zqxt\\_forms2.zip](#)

新手可能觉得这样变得更麻烦了, 有些情况是这样的, 但是 **Django** 的 **forms** 提供了:

1. 模板中表单的渲染
2. 数据的验证工作, 某一些输入不合法也不会丢失已经输入的数据。
3. 还可以定制更复杂的验证工作, 如果提供了10个输入框, 必须必须要输入其中两个以上, 在 **forms.py** 中都很容易实现

也有一些将 Django forms 渲染成 Bootstrap 的插件，也很好用，很方便。

扩展：如果提交后在同一个页面显示结果呢？请看 [Django Ajax](#)

[小额赞助，支持作者！](#)

[«Django 后台](#)

[Django 配置»](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 配置

[« Django 表单](#)  
[Django 静态文件 »](#)

运行 `django-admin.py startproject [project-name]` 命令会生成一系列文件, 在Django 1.6版本以后的 `settings.py` 文件中有以下语句:

```
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
```

这里用到了python中一个神奇的变量 `__file__` 这个变量可以获取到当前文件(包含这个代码的文件)的路径。  
`os.path.dirname(__file__)` 得到文件所在目录, 再来一个 `os.path.dirname()` 就是目录的上一级, `BASE_DIR` 即为 项目 所在目录。我们在后面的与目录有关的变量都用它, 这样使得移植性更强。

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
TEMPLATE_DEBUG = True
```

**DEBUG=True** 时, 如果出现 bug 便于我们看见问题所在, 但是部署时最好不要让用户看见bug的详情, 可能一些不怀好心的人攻击网站, 造成不必要的麻烦。

```
ALLOWED_HOSTS = ['*.besttome.com', 'www.ziqiangxuetang.com']
```

`ALLOWED_HOSTS` 允许你设置哪些域名可以访问, 即使在 Apache 或 Nginx 等中绑定了, 这里不允许的话, 也是不能访问的。

当 **DEBUG=False** 时, 这个为必填项, 如果不想输入, 可以用 `ALLOW_HOSTS = ['*']` 来允许所有的。

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

`static` 是静态文件所有目录, 比如 `jquery.js`, `bootstrap.min.css` 等文件。

一般来说我们只要把静态文件放在 APP 中的 `static` 目录下, 部署时用 `python manage.py collectstatic` 就可以把静态文件收集到(复制到) `STATIC_ROOT` 目录, 但是有时我们有一些共用的静态文件, 这时候可以设置 `STATICFILES_DIRS` 另外弄一个文件夹, 如下:

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
    '/var/www/static/',
)
```

这样我们就可以把静态文件放在 `common_static` 和 `/var/www/static/` 中了, Django也能找到它们。

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

media文件夹用来存放用户上传的文件，与权限有关，详情见 [Django 静态文件](#) 和 [Django 部署](#)

有时候有一些模板不是属于app的，比如 baidutongji.html, share.html等，

### Django 1.5 - Django 1.7

```
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates').replace('\\', '/'),
    os.path.join(BASE_DIR, 'templates2').replace('\\', '/'),
    # ...
)
```

### Django 1.8 及以上版本

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'templates').replace('\\', '/'),
            os.path.join(BASE_DIR, 'templates2').replace('\\', '/'),
        ],
        'APP_DIRS': True,
    }
]
```

这样 就可以把模板文件放在 templates 和 templates2 文件夹中了。

[小额赞助，支持作者！](#)

[« Django 表单](#)

[Django 静态文件 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云ECS](#)

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 配置

[« Django 表单](#)  
[Django 静态文件 »](#)

运行 `django-admin.py startproject [project-name]` 命令会生成一系列文件, 在Django 1.6版本以后的 `settings.py` 文件中有以下语句:

```
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
```

这里用到了python中一个神奇的变量 `__file__` 这个变量可以获取到当前文件(包含这个代码的文件)的路径。  
`os.path.dirname(__file__)` 得到文件所在目录, 再来一个 `os.path.dirname()` 就是目录的上一级, `BASE_DIR` 即为 项目 所在目录。我们在后面的与目录有关的变量都用它, 这样使得移植性更强。

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
TEMPLATE_DEBUG = True
```

**DEBUG=True** 时, 如果出现 bug 便于我们看见问题所在, 但是部署时最好不要让用户看见bug的详情, 可能一些不怀好心的人攻击网站, 造成不必要的麻烦。

```
ALLOWED_HOSTS = ['*.besttome.com', 'www.ziqiangxuetang.com']
```

`ALLOWED_HOSTS` 允许你设置哪些域名可以访问, 即使在 Apache 或 Nginx 等中绑定了, 这里不允许的话, 也是不能访问的。

当 **DEBUG=False** 时, 这个为必填项, 如果不想输入, 可以用 `ALLOW_HOSTS=[*]` 来允许所有的。

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

`static` 是静态文件所有目录, 比如 `jquery.js`, `bootstrap.min.css` 等文件。

一般来说我们只要把静态文件放在 APP 中的 `static` 目录下, 部署时用 `python manage.py collectstatic` 就可以把静态文件收集到(复制到) `STATIC_ROOT` 目录, 但是有时我们有一些共用的静态文件, 这时候可以设置 `STATICFILES_DIRS` 另外弄一个文件夹, 如下:

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
    '/var/www/static/',
)
```

这样我们就可以把静态文件放在 `common_static` 和 `/var/www/static/` 中了, Django也能找到它们。

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

media文件夹用来存放用户上传的文件，与权限有关，详情见 [Django 静态文件](#) 和 [Django 部署](#)

有时候有一些模板不是属于app的，比如 baidutongji.html, share.html等，

### Django 1.5 - Django 1.7

```
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates').replace('\\', '/'),
    os.path.join(BASE_DIR, 'templates2').replace('\\', '/'),
    # ...
)
```

### Django 1.8 及以上版本

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'templates').replace('\\', '/'),
            os.path.join(BASE_DIR, 'templates2').replace('\\', '/'),
        ],
        'APP_DIRS': True,
    }
]
```

这样 就可以把模板文件放在 templates 和 templates2 文件夹中了。

[小额赞助，支持作者！](#)

[« Django 表单](#)

[Django 静态文件 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。



推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 静态文件

[« Django 配置](#)  
[Django 部署基础 »](#)

静态文件是指 网站中的 js, css, 图片, 视频等文件

## 开发阶段

推荐用新版本的Django进行开发, 可以肯定的是 Django 1.4 以后的版本应该都支持下面的设置

注意: Django 1.4 版本需要在 `project/urls.py` 底部加上:

```
from django.contrib.staticfiles.urls import staticfiles_urlpatterns

urlpatterns += staticfiles_urlpatterns()
```

Django 1.4 静态文件相关文档: <https://docs.djangoproject.com/en/1.4/howto/static-files/>

Django 1.5 - Django 1.8 不需要添加上面的代码。

`settings.py` 静态文件相关示例代码及说明:

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.8/howto/static-files/

STATIC_URL = '/static/'

# 当运行 python manage.py collectstatic 的时候
# STATIC_ROOT 文件夹 是用来将所有STATICFILES_DIRS中所有文件夹中的文件, 以及各app中static中的文件都复制过来
# 把这些文件放到一起是为了用apache等部署的时候更方便
STATIC_ROOT = os.path.join(BASE_DIR, 'collected_static')

# 其它 存放静态文件的文件夹, 可以用来存放项目中公用的静态文件, 里面不能包含 STATIC_ROOT
# 如果不想用 STATICFILES_DIRS 可以不用, 都放在 app 里的 static 中也可以
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
    '/path/to/others/static/', # 用不到的时候可以不写这一行
)

# 这个是默认设置, Django 默认会在 STATICFILES_DIRS中的文件夹 和 各app下的static文件夹中找文件
# 注意有先后顺序, 找到了就不再继续找了
STATICFILES_FINDERS = (
    "django.contrib.staticfiles.finders.FileSystemFinder",
    "django.contrib.staticfiles.finders.AppDirectoriesFinder"
)
```

静态文件放在对应的 app 下的 static 文件夹中 或者 STATICFILES\_DIRS 中的文件夹中。

当 `DEBUG = True` 时, Django 就能自动找到放在里面的静态文件。(Django 通过 `STATICFILES_FINDERS` 中的“查找器”, 找到符合的就停下来, 寻找的过程 类似于 Python 中使用 `import xxx` 时, 找 `xxx` 这个包的过程)。

示例项目 `dj18static`, 应用 app 下面有一个 static 里面有一个 `zqxt.png` 图片:

```

dj18static
├── blog
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── static # 应用 blog 下的 static, 默认会找这个文件夹
│   │   └── 【zqxt.png】
│   └── tests.py
├── views.py
├── common_static # 已经添加到了 STATICFILES_DIRS 的文件夹
│   └── js
│       └── 【jquery.js】
└── dj18static
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    ├── wsgi.py
    └── manage.py

```

当 `settings.py` 中的 `DEBUG = True` 时，打开开发服务器 `python manage.py runserver` 直接访问 `/static/zqxt.png` 就可以找到这个静态文件。

也可以在 `settings.py` 中指定所有 app 共用的静态文件，比如 `jquery.js` 等

```

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
)

```

把 `jquery.js` 放在 `common_static/js/` 下，这样就可以在 `/static/js/jquery.js` 中访问到它！

示例下载: [📄 dj18static.zip](#)

其它参考办法（当你想为静态文件分配多个不同的网址时，可能会用上这个）：

当然也可以自己指定静态文件夹，在 `urls.py` 的最后边这样写

```

# static files
import os
from django.conf.urls.static import static
from django.conf import settings
if settings.DEBUG:
    media_root = os.path.join(settings.BASE_DIR, 'media2')
    urlpatterns += static('/media2/', document_root=media_root)

```

也可以这样

```

from django.conf.urls.static import static

urlpatterns = ...

urlpatterns += static('/media2/', document_root=media_root)

```

## 部署时

### 1. 收集静态文件

```
python manage.py collectstatic
```

这一句话就会把以前放在 app 下 `static` 中的静态文件全部拷贝到 `settings.py` 中设置的 `STATIC_ROOT` 文件夹中

## 2. 用 apache2 或 nginx 示例代码

### apache2配置文件

```
Alias /static/ /path/to/collected_static/

<Directory /path/to/collected_static>
    Require all granted
</Directory>
```

### nginx 示例代码:

```
location /media {
    alias /path/to/project/media;
}

location /static {
    alias /path/to/project/collected_static;
}
```

### Apache 完整的示例代码:

```
<VirtualHost *:80>
    ServerName www.ziqiangxuetang.com
    ServerAlias ziqiangxuetang.com
    ServerAdmin tuweizhong@163.com

    Alias /media/ /path/to/media/
    Alias /static/ /path/to/collected_static/

    <Directory /path/to/media>
        Require all granted
    </Directory>

    <Directory /path/to/collected_static>
        Require all granted
    </Directory>

    WSGIScriptAlias / /path/to/prj/prj/wsgi.py
    <Directory /path/to/prj/prj>
    <Files wsgi.py>
        Require all granted
    </Files>
    </Directory>
</VirtualHost>
```

如果你用的是apache 2.2 版本 用下面的代替 `Require all granted` 赋予权限

```
Order allow,deny
Allow from all
```

备注: (用 `apachectl -v` 命令查看 apache2版本号)

一般不推荐下面的方法, 除非有特殊需求, 有时候会很有用。

补充: 有没有不把静态文件放在static和media文件夹中, 放在任何地方都能访问到的方法呢?

刚开始搞Django的时候其实我也一直在找这样的东西, 因为毕竟asp, php等可以直接访问任何路径中的静态文件。

下面是一个例子

```
<VirtualHost *:80>
```

```
ServerName www.ziqiangxuetang.com
ServerAdmin tuweizhong@163.com
```

```
AliasMatch "(?i)^/(.+)\.ico$" "/path/to/project/$1.ico"
AliasMatch "(?i)^/(.+)\.js$" "/path/to/project/$1.js"
AliasMatch "(?i)^/(.+)\.css$" "/path/to/project/$1.css"
AliasMatch "(?i)^/(.+)\.jpg$" "/path/to/project/$1.jpg"
AliasMatch "(?i)^/(.+)\.jpeg$" "/path/to/project/$1.jpeg"
AliasMatch "(?i)^/(.+)\.png$" "/path/to/project/$1.png"
AliasMatch "(?i)^/(.+)\.gif$" "/path/to/project/$1.gif"
AliasMatch "(?i)^/(.+)\.xml$" "/path/to/project/$1.xml"
AliasMatch "(?i)^/(.+)\.xsl$" "/path/to/project/$1.xsl"
AliasMatch "(?i)^/(.+)\.txt$" "/path/to/project/$1.txt"
AliasMatch "(?i)^/(.+)\.zip$" "/path/to/project/$1.zip"
AliasMatch "(?i)^/(.+)\.rar$" "/path/to/project/$1.txt"
```

```
<Directory /path/to/project>
    Require all granted
</Directory>
```

```
WSGIScriptAlias / /path/to/project/project/wsgi.py
```

```
ErrorLog /path/to/project/error.log
CustomLog /path/to/project/access.log common
```

```
</VirtualHost>
```

更详细的部署讲解请查看 [Django 部署 \(apache2\)](#)

nginx 详细完整的代码看 [Django 部署 \(nginx\)](#)

[小额赞助，支持作者！](#)

[« Django 配置](#)

[Django 部署基础 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 静态文件

[« Django 配置](#)  
[Django 部署基础 »](#)

静态文件是指 网站中的 js, css, 图片, 视频等文件

## 开发阶段

推荐用新版本的Django进行开发, 可以肯定的是 Django 1.4 以后的版本应该都支持下面的设置

注意: Django 1.4 版本需要在 `project/urls.py` 底部加上:

```
from django.contrib.staticfiles.urls import staticfiles_urlpatterns

urlpatterns += staticfiles_urlpatterns()
```

Django 1.4 静态文件相关文档: <https://docs.djangoproject.com/en/1.4/howto/static-files/>

Django 1.5 - Django 1.8 不需要添加上面的代码。

`settings.py` 静态文件相关示例代码及说明:

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.8/howto/static-files/

STATIC_URL = '/static/'

# 当运行 python manage.py collectstatic 的时候
# STATIC_ROOT 文件夹 是用来将所有STATICFILES_DIRS中所有文件夹中的文件, 以及各app中static中的文件都复制过来
# 把这些文件放到一起是为了用apache等部署的时候更方便
STATIC_ROOT = os.path.join(BASE_DIR, 'collected_static')

# 其它 存放静态文件的文件夹, 可以用来存放项目中公用的静态文件, 里面不能包含 STATIC_ROOT
# 如果不想用 STATICFILES_DIRS 可以不用, 都放在 app 里的 static 中也可以
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
    '/path/to/others/static/', # 用不到的时候可以不写这一行
)

# 这个是默认设置, Django 默认会在 STATICFILES_DIRS中的文件夹 和 各app下的static文件夹中找文件
# 注意有先后顺序, 找到了就不再继续找了
STATICFILES_FINDERS = (
    "django.contrib.staticfiles.finders.FileSystemFinder",
    "django.contrib.staticfiles.finders.AppDirectoriesFinder"
)
```

静态文件放在对应的 app 下的 `static` 文件夹中 或者 `STATICFILES_DIRS` 中的文件夹中。

当 `DEBUG = True` 时, Django 就能自动找到放在里面的静态文件。(Django 通过 `STATICFILES_FINDERS` 中的“查找器”, 找到符合的就停下来, 寻找的过程类似于 Python 中使用 `import xxx` 时, 找 `xxx` 这个包的过程)。

示例项目 **djl8static**, 应用 **app** 下面有一个 **static** 里面有一个 **zqxt.png** 图片:

```
djl8static
├── blog
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── static # 应用 blog 下的 static, 默认会找这个文件夹
│   │   └── 【zqxt.png】
│   ├── tests.py
│   └── views.py
├── common_static # 已经添加到了 STATICFILES_DIRS 的文件夹
│   └── js
│       └── 【jquery.js】
├── djl8static
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

当 **settings.py** 中的 **DEBUG = True** 时, 打开开发服务器 **python manage.py runserver** 直接访问 **/static/zqxt.png** 就可以找到这个静态文件。

也可以在 **settings.py** 中指定所有 **app** 共用的静态文件, 比如 **jquery.js** 等

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
)
```

把 **jquery.js** 放在 **common\_static/js/** 下, 这样就可以在 **/static/js/jquery.js** 中访问到它!

示例下载: [📦 djl8static.zip](#)

其它参考办法 (当你想为静态文件分配多个不同的网址时, 可能会用上这个):

当然也可以自己指定静态文件夹, 在 **urls.py** 的最后边这样写

```
# static files
import os
from django.conf.urls.static import static
from django.conf import settings
if settings.DEBUG:
    media_root = os.path.join(settings.BASE_DIR, 'media2')
    urlpatterns += static('/media2/', document_root=media_root)
```

也可以这样

```
from django.conf.urls.static import static

urlpatterns = ...

urlpatterns += static('/media2/', document_root=media_root)
```

## 部署时

### 1. 收集静态文件

```
python manage.py collectstatic
```

这一句话就会把以前放在app下static中的静态文件全部拷贝到 settings.py 中设置的 **STATIC\_ROOT** 文件夹中

## 2. 用 apache2 或 nginx 示例代码

### apache2配置文件

```
Alias /static/ /path/to/collected_static/

<Directory /path/to/collected_static>
    Require all granted
</Directory>
```

### nginx 示例代码:

```
location /media {
    alias /path/to/project/media;
}

location /static {
    alias /path/to/project/collected_static;
}
```

### Apache 完整的示例代码:

```
<VirtualHost *:80>
    ServerName www.ziqiangxuetang.com
    ServerAlias ziqiangxuetang.com
    ServerAdmin tuweizhong@163.com

    Alias /media/ /path/to/media/
    Alias /static/ /path/to/collected_static/

    <Directory /path/to/media>
        Require all granted
    </Directory>

    <Directory /path/to/collected_static>
        Require all granted
    </Directory>

    WSGIScriptAlias / /path/to/prj/prj/wsgi.py
    <Directory /path/to/prj/prj>
    <Files wsgi.py>
        Require all granted
    </Files>
    </Directory>
</VirtualHost>
```

如果你用的是apache 2.2 版本 用下面的代替 **Require all granted** 赋予权限

```
Order allow,deny
Allow from all
```

备注: (用 **apachectl -v** 命令查看 apache2版本号)

一般不推荐下面的方法, 除非有特殊需求, 有时候会很有用。

补充: 有没有不把静态文件放在static和media文件夹中, 放在任何地方都能访问到的方法呢?

刚开始搞Django的时候其实我也一直在找这样的东西, 因为毕竟asp, php等可以直接访问任何路径中的静态文件。

下面是一个例子

```
<VirtualHost *:80>
    ServerName www.ziqiangxuetang.com
    ServerAdmin tuweizhong@163.com

    AliasMatch "(?i)^(.+)\.ico$" "/path/to/project/$1.ico"
    AliasMatch "(?i)^(.+)\.js$" "/path/to/project/$1.js"
    AliasMatch "(?i)^(.+)\.css$" "/path/to/project/$1.css"
    AliasMatch "(?i)^(.+)\.jpg$" "/path/to/project/$1.jpg"
    AliasMatch "(?i)^(.+)\.jpeg$" "/path/to/project/$1.jpeg"
    AliasMatch "(?i)^(.+)\.png$" "/path/to/project/$1.png"
    AliasMatch "(?i)^(.+)\.gif$" "/path/to/project/$1.gif"
    AliasMatch "(?i)^(.+)\.xml$" "/path/to/project/$1.xml"
    AliasMatch "(?i)^(.+)\.xsl$" "/path/to/project/$1.xsl"
    AliasMatch "(?i)^(.+)\.txt$" "/path/to/project/$1.txt"
    AliasMatch "(?i)^(.+)\.zip$" "/path/to/project/$1.zip"
    AliasMatch "(?i)^(.+)\.rar$" "/path/to/project/$1.rar"

    <Directory /path/to/project>
        Require all granted
    </Directory>

    WSGIScriptAlias / /path/to/project/project/wsgi.py

    ErrorLog /path/to/project/error.log
    CustomLog /path/to/project/access.log common
</VirtualHost>
```

更详细的部署讲解请查看 [Django 部署 \(apache2\)](#)

nginx 详细完整的代码看 [Django 部署 \(nginx\)](#)

[小额赞助，支持作者！](#)

[« Django 配置](#)

[Django 部署基础 »](#)

 CODING  
GROUP. ENVIRONMENT

零成本开启敏捷开发 五人以下团队免费

免费体验

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

# Django 部署基础

« [Django 静态文件](#)  
[Django 部署\(Apache\)](#) »

此文摘录于即将完稿的Django书籍“服务部署”章节中的一部分，先给关注公众号和网站上的读者尝个鲜。

本文干货较多，对部署有困惑的坚持读完，一定会有收获的，建议把此文分享给有同样困惑的小伙伴吧。

本文主要讲解在 Linux 平台下，使用 Nginx + uWSGI 的方式来部署来 Django，这是目前比较主流的方式。当然你也可以使用 Gunicorn 代替 uWSGI，不过原理都是类似的，弄懂了其中一种，其它的方式理解起来问题也不会很大。

有很多人曾经在邮件中咨询过我如何部署，部署确实对开发者有着较高的要求，尤其是初学者来说，部署比较难，这很正常，**千万不要轻易就放弃了。**

回想我2013年刚开始接触也是一头雾水，整整弄了四五天才完全搞懂，部署好了真是欣喜若狂。我自认为算不上聪明，我都能搞明白，你也一定能行。最关键和宝贵的品质就是坚持不懈和勇于试错。下面就让我们一起开始正式的部署教程吧。

## 基础知识储备

当我们发现用浏览器不能访问的时候，我们需要一步步排查问题。

整个部署的链路是 Nginx -> uWSGI -> Python Web程序，通常还会提到supervisord工具。

uWSGI是一个软件，部署服务的工具，了解整个过程，我们先了解一下WSGI规范，uwsgi协议等内容。

**WSGI** (Web Server Gateway Interface)规范，WSGI规定了Python Web应用和Python Web服务器之间的通讯方式。目前主流的Python Web框架，比如Django, Flask, Tornado等都是基于这个规范实现的。

**uwsgi协议**是uWSGI工具独有的协议，简洁高效的uwsgi协议是选择uWSGI作为部署工具的重要理由之一，详细的 uwsgi协议 可以参考uWSGI的文档。

uWSGI是 实现了uwsgi协议，WSGI规范和HTTP协议的一个C语言实现的软件。

**Nginx**是一个Web服务器，是一个反向代理工具，我们通常用它来部署静态文件。主流的Python Web开发框架都遵循WSGI规范。uWSGI通过WSGI规范和我们编写的服务进程通讯，然后通过自带的高效的 uwsgi 协议和 Nginx进行通讯，最终Nginx通过HTTP协议将服务对外透出。

当一个访问进来的时候，首先到 Nginx，Nginx会把请求（HTTP协议）转换uwsgi协议传递给uWSGI，uWSGI通过WSGI和web server进行通讯取到响应结果，再通过uwsgi协议发给Nginx，最终Nginx以HTTP协议发现响应给用户。

有些同学可能会说，uWSGI不是支持HTTP协议么，也支持静态文件部署，我不用Nginx行不行？

当然可以，这么做没问题，但目前主流的做法是用Nginx，毕竟它久经考验，更稳定，当然也更值得我们信赖。

**supervisor** 是一个进程管理工具。任何人都不能保证程序不异常退出，不别被人误杀，所以一个典型的工程做法就是使用supervisor看守着你的进程，一旦异常退出它会立马进程重新启动起来。

如果服务部署后出现异常，不能访问。我们需要分析每一步有没有问题，这时候就不得不用到Linux中一些命令。

# 进程分析

进程是计算机分配资源的最小单位，我们的程序至少是运行在一个进程中。

## 1. 查看进程信息

通常我们使用 `ps aux | grep python` 来查看系统中运行的 python 进程，输出结果如下：

```
tu@linux / $ ps aux | grep python
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        1780  0.0  0.0  58888 10720 ?        Ss   Jan15   8:46 /usr/bin/python /usr/local/bin/supervisord -c
/etc/supervisord.conf
tu         4491  0.0 18.0 3489820 2960628 ?        Sl   Jan29   0:19 python a.py
tu        12602  5.3 26.4 4910444 4343708 pts/1  Sl+  12:34   4:07 python b.py
```

有些同学习惯使用 `ps -ef | grep xxx` 结果也是类似的，读者可以自行尝试。

输出结果中 USER 后面的 PID 代表进程编号。

我们可以通过查看 `/proc/PID/` 目录的文件信息来得到这个进程的一些信息（Linux中一切皆文件，进程信息也在文件中），比如它是在哪个目录启动的，启动命令是什么等信息。执行命令后输入内容如下：

```
tu@linux /proc/4491 $ sudo ls -ahl
...
dr-xr-xr-x  2 tu tu 0 Feb 17 13:32 attr
-rw-r--r--  1 tu tu 0 Feb 17 13:32 autogroup
-r-----  1 tu tu 0 Feb 17 13:32 auxv
-r--r--r--  1 tu tu 0 Feb 17 13:32 cgroup
--w-----  1 tu tu 0 Feb 17 13:32 clear_refs
-r--r--r--  1 tu tu 0 Feb 17 12:49 cmdline 这个文件中有启动进程具体的命令
-rw-r--r--  1 tu tu 0 Feb 17 13:32 comm
-rw-r--r--  1 tu tu 0 Feb 17 13:32 coredump_filter
-r--r--r--  1 tu tu 0 Feb 17 13:32 cpuset
lrwxrwxrwx  1 tu tu 0 Feb 17 13:32 cwd -> /home/tu 启动进程时的工作目录
-r-----  1 tu tu 0 Feb 17 13:32 environ 进程的环境变量列表
lrwxrwxrwx  1 tu tu 0 Feb 17 12:00 exe -> /usr/bin/python2.7 链接到进程的执行命令文件
...省去了部分内容
```

## 2. 向进程发送信号

我们可以使用 `kill PID` 杀死一个进程，或者使用 `kill -9 PID` 强制杀死一个进程。

记得以前在研究生的时候师弟和师妹经常问我，`kill -9` 里面的 `-9` 是什么意思，我告诉他们，这是强制杀死进程的意思，让这个进程“九死一生”。当然这是开玩笑，这里的 `-9` 是信号的一种，`kill` 命令会向进程发送一个信号，`-9`代表 `SIGKILL` 之意，用于强制终止某个进程，当然这是一种无情地，野蛮地方式干掉进程。

我们可以通过 `kill -l` 命令查看到所有的信号

```
HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU
XFSZ VTALRM PROF WINCH POLL PWR SYS
```

上面的信号是有顺序的，比如第1个是 HUP，第9个是 KILL，下面两种方式是等价的：

`kill -1 PID` 和 `kill -HUP PID`

`kill -9 PID` 和 `kill -KILL PID`

信号HUP通常程序用这个信号进行优雅重载配置文件，重新启动并且不影响正在运行的服务。比如

`pkill -1 uwsgi` 优雅重启uwsgi 进程，对服务器没有影响

`kill -1 NGINX_PID` 优雅重启nginx进程，对服务器没有影响

除了知道可以这么使用之外，感兴趣的读者还可以自行学习，深入了解uwsgi和nginx无损reload的机制。

我们常用CTRL+C中断一个命令的执行，其实就是发送了一个信号到该进程

CTRL-C 发送 SIGINT 信号给前台进程组中的所有进程，常用于终止正在运行的程序。

CTRL-Z 发送 SIGTSTP 信号给前台进程组中的所有进程，常用于挂起一个进程。

每个程序可能对部分信号的功能定义不一致，其它信号的含义大家可以自行学习。

### 3. 查看进程打开了哪些文件

`sudo lsof -p PID`

如果是分析一个你不太了解的进程，这个命令比较有用。

可以使用 `lsof -p PID | grep TCP` 查看进程中的 TCP 连接信息。

### 4. 查看文件被哪个进程使用

使用这个命令查看一个文件被哪些进程正在使用 `sudo lsof /path/to/file`，示例如下：

```
> sudo lsof /home/tu/.virtualenvs/mic/bin/uwsgi
COMMAND  PID USER  FD   TYPE DEVICE SIZE/OFF      NODE NAME
uwsgi    2071 tu  txt    REG 253,17 1270899 13240576 /home/tu/.virtualenvs/mic/bin/uwsgi
uwsgi    13286 tu  txt    REG 253,17 1270899 13240576 /home/tu/.virtualenvs/mic/bin/uwsgi
uwsgi    13287 tu  txt    REG 253,17 1270899 13240576 /home/tu/.virtualenvs/mic/bin/uwsgi
uwsgi    13288 tu  txt    REG 253,17 1270899 13240576 /home/tu/.virtualenvs/mic/bin/uwsgi
```

### 5. 查看进程当前状态

当我们发现一个进程启动了，端口也是正常的，但好像这个进程就是不“干活”。比如我们执行的是数据更新进程，这个进程不更新数据了，但还是在跑着。可能数据源有问题，可能我们写的程序有BUG，也可能是更新时要写入到的数据库出问题了（数据库连接不上，写数据死锁了）。我们这里主要说下第二种，我们自己的程序如果有BUG，导致工作不正常，我们怎么知道它当前正在干什么呢，这时候就要用到Linux中的调试分析诊断strace，可以使用 `sudo strace -p PID`这个命令。

通过执行后输出的一些信息，推测分析看是哪些出了问题。

这里我们讲了一些进程分析的工具和方法，关于进程分析工具和方法还有许多，大家需要不断练习，熟练运用这些工具去排查遇到的问题。

## 端口分析

比如我们在服务器上运行 Nginx，访问的时候就是连接不上，我们可以使用 `ps aux | grep nginx`看下nginx进程是不是启动了，也可以看下 80端口有没有被占用。换句话说，如果没有任何程序跑在这个端口上（或者说没有任何程序使用这个端口），证明忘了启动相关程序或者没能启动成功，或者说程序使用的端口被修改了，不是80了，那又怎么可能访问到呢？

#### 1. 查看全部端口占用情况

Linux我们可以使用 `netstat` 工具来进程网络分析，`netstat` 命令有非常多选项，这里只列出了常用的一部分

`-a`或`--all` 显示所有连接中的Socket，默认不显示 LISTEN 相关的。

-c或--continuous 持续列出网络状态，不断自动刷新输出。  
-l或--listening 显示监听中的服务器的Socket。  
-n或--numeric 直接使用IP地址，而不是展示域名。  
-p或--programs 显示正在使用Socket的程序进程PID和名称。  
-t或--tcp 显示TCP传输协议的连接。  
-u或--udp 显示UDP传输协议的连接。

比如我们可以查看服务器中监控了哪些端口，如果我们的nginx是使用80端口，uwsgi使用的是7001端口，我们就知道通过下面的命令

```
> netstat -nlt  
Active Internet connections (only servers)  
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name  
tcp        0      0 0.0.0.0:7001            0.0.0.0:*               LISTEN      2070/uwsgi  
tcp        0      0 127.0.0.1:6379          0.0.0.0:*               LISTEN      1575/redis-server 1
```

就能知道80端口的nginx是不是启动成功了，7001端口的uwsgi是不是启动成功了。

注意：如果PID和Program Name显示不出来，证明是权限不够，可以使用sudo运行

## 2. 查看具体端口占用情况

```
> sudo lsof -i :80 (注意端口80前面有个英文的冒号)
```

```
COMMAND  PID    USER   FD   TYPE DEVICE SIZE/OFF NODE NAME  
nginx    4123   admin   3u    IPv4  13031      0t0  TCP *:http (LISTEN)  
nginx    4124   admin   3u    IPv4  13031      0t0  TCP *:http (LISTEN)
```

我们可以通过这个方法查询出占用端口的程序，如果遇到端口已经被占用，原来的进程没有正确地终止，可以使用kill命令停掉原来的进程，这样我们就又可以使用这个端口了。

除了上面讲的一些命令，在部署过程中会经常用到下面的一些Linux命令，如果不清楚它们是做什么的，可以提前自行学习下这些Linux基础命令：

ls, touch, mkdir, mv, cp, ps, chmod, chown

学习完了这些内容，我们应该就具备了部署Linux服务器的基础知识了，在遇到问题后，应该也会有一些调查思路。

剩余部分大家再去 自强学堂 上看 Django 部署，应该会容易懂的多。

**创作不易，觉得有用记得分享。关注微信，提前知道书籍出版情况。**



[小额赞助，支持作者！](#)

[« Django 静态文件](#)

[Django 部署\(Apache\) »](#)

 **零成本开启敏捷开发 五人以下团队免费**

免费体验

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)

[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 部署基础

« [Django 静态文件](#)  
[Django 部署\(Apache\)](#) »

此文摘录于即将完稿的Django书籍“服务部署”章节中的一部分，先给关注公众号和网站上的读者尝个鲜。

本文干货较多，对部署有困惑的坚持读完，一定会有收获的，建议把此文分享给有同样困惑的小伙伴吧。

本文主要讲解在 Linux 平台下，使用 Nginx + uWSGI 的方式来部署来 Django，这是目前比较主流的方式。当然你也可以使用 Gunicorn 代替 uWSGI，不过原理都是类似的，弄懂了其中一种，其它的方式理解起来问题也不会很大。

有很多人曾经在邮件中咨询过我如何部署，部署确实对开发者有着较高的要求，尤其是初学者来说，部署比较难，这很正常，**千万不要轻易就放弃了。**

回想我2013年刚开始接触也是一头雾水，整整弄了四五天才完全搞懂，部署好了真是欣喜若狂。我自认为算不上聪明，我都能够搞明白，你也一定能行。最关键和宝贵的品质就是坚持不懈和勇于试错。下面就让我们一起开始正式的部署教程吧。

### 基础知识储备

当我们发现用浏览器不能访问的时候，我们需要一步步排查问题。

整个部署的链路是 Nginx -> uWSGI -> Python Web程序，通常还会提到supervisord工具。

uWSGI是一个软件，部署服务的工具，了解整个过程，我们先了解一下WSGI规范，uwsgi协议等内容。

**WSGI** (Web Server Gateway Interface)规范，WSGI规定了Python Web应用和Python Web服务器之间的通讯方式。目前主流的Python Web框架，比如Django，Flask，Tornado等都是基于这个规范实现的。

**uwsgi协议**是uWSGI工具独有的协议，简洁高效的uwsgi协议是选择uWSGI作为部署工具的重要理由之一，详细的 uwsgi协议 可以参考uWSGI的文档。

uWSGI是实现了uwsgi协议，WSGI规范和HTTP协议的一个C语言实现的软件。

**Nginx**是一个Web服务器，是一个反向代理工具，我们通常用它来部署静态文件。主流的Python Web开发框架都遵循WSGI规范。

uWSGI通过WSGI规范和我们编写的服务进程通讯，然后通过自带的高效的 uwsgi 协议和 Nginx进行通讯，最终Nginx通过HTTP协议将服务对外透出。

当一个访问进来的时候，首先到 Nginx，Nginx会把请求（HTTP协议）转换uwsgi协议传递给uWSGI，uWSGI通过WSGI和web server进行通讯取到响应结果，再通过uwsgi协议发给Nginx，最终Nginx以HTTP协议发现响应给用户。

有些同学可能会说，uWSGI不是支持HTTP协议么，也支持静态文件部署，我不用Nginx行不行？

当然可以，这么做没问题，但目前主流的做法是用Nginx，毕竟它久经考验，更稳定，当然也更值得我们信赖。

**supervisor** 是一个进程管理工具。任何人都不能保证程序不异常退出，不别被人误杀，所以一个典型的工程做法就是使用supervisor看守着你的进程，一旦异常退出它会立马进程重新启动起来。

如果服务部署后出现异常，不能访问。我们需要分析每一步有没有问题，这时候就不得不用到Linux中一些命令。

## 进程分析

进程是计算机分配资源的最小单位，我们的程序至少是运行在一个进程中。

### 1. 查看进程信息

通常我们使用 `ps aux | grep python` 来查看系统中运行的 python 进程，输出结果如下：

```
tu@linux / $ ps aux | grep python
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1780  0.0  0.0  58888 10720 ?        Ss   Jan15    8:46 /usr/bin/python /usr/local/bin/supervisord -c /etc/supervisord.conf
tu         4491  0.0 18.0 3489820 2960628 ?        Sl   Jan29    0:19 python a.py
tu        12602  5.3 26.4 4910444 4343708 pts/1  Sl+  12:34    4:07 python b.py
```

有些同学习惯使用 `ps -ef | grep xxx` 结果也是类似的，读者可以自行尝试。

输出结果中 USER 后面的 PID 代表进程编号。

我们可以通过查看 `/proc/PID/` 目录的文件信息来得到这个进程的一些信息（Linux中一切皆文件，进程信息也在文件中），比如它是在哪个目录启动的，启动命令是什么等信息。执行命令后输入内容如下：

```
tu@linux /proc/4491 $ sudo ls -ahl
...
dr-xr-xr-x  2 tu tu 0 Feb 17 13:32 attr
-rw-r--r--  1 tu tu 0 Feb 17 13:32 autogroup
-r-----  1 tu tu 0 Feb 17 13:32 auxv
-r--r--r--  1 tu tu 0 Feb 17 13:32 cgroup
-w-----  1 tu tu 0 Feb 17 13:32 clear_refs
-r--r--r--  1 tu tu 0 Feb 17 12:49 cmdline 这个文件中有启动进程具体的命令
-rw-r--r--  1 tu tu 0 Feb 17 13:32 comm
-rw-r--r--  1 tu tu 0 Feb 17 13:32 coredump_filter
-r--r--r--  1 tu tu 0 Feb 17 13:32 cpuset
lrwxrwxrwx  1 tu tu 0 Feb 17 13:32 cwd -> /home/tu 启动进程时的工作目录
-r-----  1 tu tu 0 Feb 17 13:32 environ 进程的环境变量列表
lrwxrwxrwx  1 tu tu 0 Feb 17 12:00 exe -> /usr/bin/python2.7 链接到进程的执行命令文件
...省去了部分内容
```

### 2. 向进程发送信号

我们可以使用 `kill PID` 杀死一个进程，或者使用 `kill -9 PID` 强制杀死一个进程。

记得以前在研究生的时候师弟和师妹经常问我，kill -9 里面的 -9 是什么意思，我告诉他们，这是强制杀死进程的意思，让这个进程“九死一生”。当然这是开玩笑，这里的 -9 是信号的一种，kill 命令会向进程发送一个信号，-9代表 SIGKILL 之意，用于强制终止某个进程，当然这是一种无情地，野蛮地方式干掉进程。

我们可以通过 kill -l 命令查看到所有的信号

```
HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU
XFSZ VTALRM PROF WINCH POLL PWR SYS
```

上面的信号是有顺序的，比如第1个是 HUP，第9个是 KILL，下面两种方式是等价的：

kill -1 PID 和 kill -HUP PID

kill -9 PID 和 kill -KILL PID

信号HUP通常程序用这个信号进行优雅重载配置文件，重新启动并且不影响正在运行的服务。比如

pkill -1 uwsgi 优雅重启uwsgi 进程，对服务器没有影响

kill -1 NGINX\_PID 优雅重启nginx进程，对服务器没有影响

除了知道可以这么使用之外，感兴趣的读者还可以自行学习，深入了解下uwsgi和nginx无损reload的机制。

我们常用CTRL+C中断一个命令的执行，其实就是发送了一个信号到该进程

CTRL-C 发送 SIGINT 信号给前台进程组中的所有进程，常用于终止正在运行的程序。

CTRL-Z 发送 SIGTSTP 信号给前台进程组中的所有进程，常用于挂起一个进程。

每个程序可能对部分信号的功能定义不一致，其它信号的含义大家可以自行学习。

### 3. 查看进程打开了哪些文件

sudo lsof -p PID

如果是分析一个你不太了解的进程，这个命令比较有用。

可以使用 lsof -p PID | grep TCP 查看进程中的 TCP 连接信息。

### 4. 查看文件被哪个进程使用

使用这个命令查看一个文件被哪些进程正在使用 sudo lsof /path/to/file，示例如下：

```
> sudo lsof /home/tu/.virtualenvs/mic/bin/uwsgi
COMMAND  PID USER  FD   TYPE DEVICE SIZE/OFF      NODE NAME
uwsgi    2071 tu  txt    REG 253,17 1270899 13240576 /home/tu/.virtualenvs/mic/bin/uwsgi
uwsgi    13286 tu  txt    REG 253,17 1270899 13240576 /home/tu/.virtualenvs/mic/bin/uwsgi
uwsgi    13287 tu  txt    REG 253,17 1270899 13240576 /home/tu/.virtualenvs/mic/bin/uwsgi
uwsgi    13288 tu  txt    REG 253,17 1270899 13240576 /home/tu/.virtualenvs/mic/bin/uwsgi
```

### 5. 查看进程当前状态

当我们发现一个进程启动了，端口也是正常的，但好像这个进程就是不“干活”。比如我们执行的是数据更新进程，这个进程不更新数据了，但还是在跑着。可能数据源有问题，可能我们写的程序有BUG，也可能是更新时要写入到的数据库出问题了（数据库连接不上，写数据死锁了）。我们这里主要说下第二种，我们自己的程序如果有BUG，导致工作不正常，我们怎么知道它当前正在干什么呢，这时候就要用到Linux中的调试分析诊断strace，可以使用 **sudo strace -p PID**这个命令。

通过执行后输出的一些信息，推测分析看是哪些出了问题。

这里我们讲了一些进程分析的工具和方法，关于进程分析工具和方法还有许多，大家需要不断练习，熟练运用这些工具去排查遇到的问题。

# 端口分析

比如我们在服务器上运行 Nginx，访问的时候就是连接不上，我们可以使用 `ps aux | grep nginx`看下nginx进程是不是启动了，也可以看下 80端口有没有被占用。换句话说，如果没有任何程序跑在这个端口上（或者说没有任何程序使用这个端口），证明忘了启动相关程序或者没能启动成功，或者说程序使用的端口被修改了，不是80了，那又怎么可能访问到呢？

## 1. 查看全部端口占用情况

Linux中我们可以使用 `netstat` 工具来进程网络分析，`netstat` 命令有非常多选项，这里只列出了常用的一部分

-a或--all 显示所有连接中的Socket，默认不显示 LISTEN 相关的。

-c或--continuous 持续列出网络状态，不断自动刷新输出。

-l或--listening 显示监听中的服务器的Socket。

-n或--numeric 直接使用IP地址，而不是展示域名。

-p或--programs 显示正在使用Socket的程序进程PID和名称。

-t或--tcp 显示TCP传输协议的连接。

-u或--udp 显示UDP传输协议的连接。

比如我们可以查看服务器中监控了哪些端口，如果我们的nginx是使用80端口，uwsgi使用的是7001端口，我们就知道通过下面的命令

```
> netstat -nltp
```

Active Internet connections (only servers)

| Proto | Recv-Q | Send-Q | Local Address  | Foreign Address | State  | PID/Program name    |
|-------|--------|--------|----------------|-----------------|--------|---------------------|
| tcp   | 0      | 0      | 0.0.0.0:7001   | 0.0.0.0:*       | LISTEN | 2070/uwsgi          |
| tcp   | 0      | 0      | 127.0.0.1:6379 | 0.0.0.0:*       | LISTEN | 1575/redis-server 1 |

就能知道80端口的 nginx 是不是启动成功了，7001端口的uwsgi是不是启动成功了。

注意：如果PID和Program Name显示不出来，证明是权限不够，可以使用sudo运行

## 2. 查看具体端口占用情况

```
> sudo lsof -i :80（注意端口80前面有个英文的冒号）
```

| COMMAND | PID  | USER  | FD | TYPE | DEVICE | SIZE/OFF | NODE | NAME            |
|---------|------|-------|----|------|--------|----------|------|-----------------|
| nginx   | 4123 | admin | 3u | IPv4 | 13031  | 0t0      | TCP  | *:http (LISTEN) |
| nginx   | 4124 | admin | 3u | IPv4 | 13031  | 0t0      | TCP  | *:http (LISTEN) |

我们可以通过这个方法查询出占用端口的程序，如果遇到端口已经被占用，原来的进程没有正确地终止，可以使用kill命令杀掉原来的进程，这样我们就又可以使用这个端口了。

除了上面讲的一些命令，在部署过程中会经常用到下面的一些Linux命令，如果不清楚它们是做什么的，可以提前自行学习下这些Linux基础命令：

ls, touch, mkdir, mv, cp, ps, chmod, chown

学习完了这些内容，我们应该就具备了部署Linux服务器的基础知识了，在遇到问题后，应该也会有一些调查思路。

剩余部分大家再去 自强学堂 上看 Django部署，应该会容易懂的多。

创作不易，觉得有用记得分享。关注微信，提前知道书籍出版情况。



[小额赞助，支持作者！](#)

[« Django 静态文件](#)

[Django 部署\(Apache\)»](#)



 CODING  
Cloud Development

**零成本开启敏捷开发 五人以下团队免费**

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的,自强学堂网站源代码免费下载](#)

## Django 部署(Apache)

«[Django 部署基础](#)  
[Django 部署\(Nginx\)](#)»

本文讲述部署的方法和常见的问题,并给出了在 **BAE**, **SAE** 等上面部署的实例。

如果不是在自己的服务器上部署,当开发完成后可以部署到 [BAE](#), [SAE](#), 也可以使用阿里云的服务器。

推荐: [使用Code Studio 在线代码编辑,新人更有免费一个月云主机,可以用来实战体验本节的部署!](#)

部署到**BAE**的例子: [https://github.com/twz915/BAE\\_Django](https://github.com/twz915/BAE_Django) (不推荐,BAE已经要关闭了)

部署到**SAE**的例子: <https://github.com/twz915/django-sae> (Fork smallcode同学的,没有测试过,SAE有一定的免费份额)

**Django + nginx + Gunicorn/uwsgi 部署方式, 参见另一篇: [Django 部署 \(nginx\)](#)**

自己的服务器(比如用的[阿里云服务器](#))请看下文:

如果是新手,个人推荐用ubuntu,除非你对linux非常熟悉,ubuntu服务器的优点:

- 一, 开机apache2等都自动启动,不需要额外设置
- 二, 安装软件非常方便 apt-get 搞定
- 三, 安装ssh, git等也非常容易,几乎是傻瓜化

如果你在虚拟机或个人电脑中安装,也可以试试 **Linux Mint**, 它用起来更简单, 和ubuntu兼容。

下面是ubuntu上的部署详细步骤:

### 1. 安装 apache2 和 mod\_wsgi

```
sudo apt-get install apache2

# Python 2
sudo apt-get install libapache2-mod-wsgi

# Python 3
sudo apt-get install libapache2-mod-wsgi-py3
```

### 2. 确认安装的apache2版本号

```
apachectl -v

Server version: Apache/2.4.6 (ubuntu)
Server built: Dec 5 2013 18:32:22
```

### 3. 准备一个新网站

ubuntu的apache2配置文件在 /etc/apache2/ 下

备注: centos 用户 apache 名称为 httpd 在 /etc/httpd/ 中 (可以参考文章下面置顶的评论)

新建一个网站配置文件

```
sudo vi /etc/apache2/sites-available/sitename.conf
```

示例内容如下：

```
<VirtualHost *:80>
    ServerName www.yourdomain.com
    ServerAlias otherdomain.com
    ServerAdmin tuweizhong@163.com

    Alias /media/ /home/tu/blog/media/
    Alias /static/ /home/tu/blog/static/

    <Directory /home/tu/blog/media>
        Require all granted
    </Directory>

    <Directory /home/tu/blog/static>
        Require all granted
    </Directory>

    WSGIScriptAlias / /home/tu/blog/blog/wsgi.py
    # WSGIDaemonProcess ziqiangxuetang.com python-path=/home/tu/blog:/home/tu/.virtualenvs/blog/lib/python2.7/site-packages
    # WSGIProcessGroup ziqiangxuetang.com

    <Directory /home/tu/blog/blog>
    <Files wsgi.py>
        Require all granted
    </Files>
    </Directory>
</VirtualHost>
```

如果你的apache版本号是 2.2.x（第二步有方法判断）

用下面的代替 `Require all granted`

```
Order deny,allow
Allow from all
```

备注：把上面配置文件中这两句的备注去掉，可以使用 **virtualenv** 来部署网站，当然也可以只写一个 **/home/tu/blog**

```
# WSGIDaemonProcess ziqiangxuetang.com python-path=/home/tu/blog:/home/tu/.virtualenvs/blog/lib/python2.7/site-packages
# WSGIProcessGroup ziqiangxuetang.com
```

## 4. 修改wsgi.py文件

注意：上面如果写了 **WSGIDaemonProcess** 的话，这一步可以跳过，即可以不修改 **wsgi.py** 文件。

上面的配置中写的 `WSGIScriptAlias / /home/tu/blog/blog/wsgi.py`

就是把apache2和你的网站project联系起来了

```
import os
from os.path import join,dirname,abspath

PROJECT_DIR = dirname(dirname(abspath(__file__)))#3
import sys # 4
sys.path.insert(0,PROJECT_DIR) # 5

os.environ["DJANGO_SETTINGS_MODULE"] = "blog.settings" # 7

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

第 3, 4, 5 行为新加的内容，作用是让脚本找到django项目的位置，也可以在sitemame.conf中做，用WSGI PythonPath,想了解的自行搜索, 第 7 行如果一台服务器有多个django project时一定要修改成上面那样，否则访问的时候会发生网站互相串的情况，即访问A网站到了B网站，一会儿正常，一会儿又不正常（当然也可以使用 `mod_wsgi daemon` 模式, [点击这里查看](#)）

## 5. 设置目录和文件权限

一般目录权限设置为 755，文件权限设置为 644

假如项目位置在 `/home/tu/zqxt`（在`zqxt`下面有一个`manage.py`，`zqxt`是项目名称）

```
cd /home/tu/  
sudo chmod -R 644 zqxt  
sudo find zqxt -type d | xargs chmod 755
```

**apache** 服务器运行用户可以在 `/etc/apache2/envvars` 文件里面改，这里使用的是默认值，当然也可以更改成自己的当前用户，这样的话权限问题就简单很多，但在服务器上推荐有 **www-data** 用户，更安全。以下是默认设置：

```
# Since there is no sane way to get the parsed apache2 config in scripts, some  
# settings are defined via environment variables and then used in apache2ctl,  
# /etc/init.d/apache2, /etc/logrotate.d/apache2, etc.  
  
export APACHE_RUN_USER=www-data  
export APACHE_RUN_GROUP=www-data
```

## 上传文件夹权限

**media** 文件夹一般用来存放用户上传文件，**static** 一般用来放自己网站的js，css，图片等，在`settings.py`中的相关设置

**STATIC\_URL** 为静态文件的网址 **STATIC\_ROOT** 为静态文件的根目录，

**MEDIA\_URL** 为用户上传文件夹的根目录，**MEDIA\_URL**为对应的访问网址

在`settings.py`中设置：

```
# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/dev/howto/static-files/  
STATIC_URL = '/static/'  
STATIC_ROOT = os.path.join(BASE_DIR, 'static')  
  
# upload folder  
MEDIA_URL = '/media/'  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

在 **Linux** 服务器上，**用户上传目录**还要设置给 **www-data** 用户的写权限，下面的方法比较好，不影响原来的用户的编辑。

假如上传目录为 `zqxt/media/uploads` 文件夹,进入`media`文件夹，将 `uploads` 用户组改为`www-data`，并且赋予该组写权限：

```
cd media/ # 进入media文件夹  
sudo chgrp -R www-data uploads  
sudo chmod -R g+w uploads
```

**备注：**这两条命令，比直接用`sudo chown -R www-data:www-data uploads`好，因为下面的命令不影响文件原来所属用户编辑文件，**fedora**系统应该不用设置上面的权限，但是个人强烈推荐用**ubuntu**,除非你对**linux**非常熟悉，你自己选择。

如果你使用的是**sqlite3**数据库，还会提示 **Attempt to write a readonly database**,同样要给**www-data**写数据库的权限

进入项目目录的上一级，比如`project`目录为 `/home/tu/blog` 那就进入 `/home/tu` 执行下面的命令（和修改上传文件夹类似）

```
sudo chgrp www-data blog  
sudo chmod g+w blog  
sudo chgrp www-data blog/db.sqlite3 # 更改为你的数据库名称  
sudo chmod g+w blog/db.sqlite3
```

**备注：**上面的不要加 `-R`，`-R`是更改包括所有的子文件夹和文件，这样不安全。个人建议可以专门弄一个文件夹,用它来放**sqlite3**数据库，给该文件夹**www-data**写权限，而不是整个项目给写权限，有些文件只要读的权限就够了，给写权限会造成不安全。

## 6. 激活新网站

```
sudo a2ensite sitename 或 sudo a2ensite sitename.conf
```

如果顺利，这样网站就搭建成功，访问你的网址试试看，如果出现问题就接着看下面的。

## 7. 错误排查

一，没有静态文件，网站打开很乱，没有布局，多半是静态文件没有生效。

1. 确保你的配置文件中的路径是正确的
2. 确保你的`settings.py`中的文件设置正确
3. 收集静态文件(详细[静态文件部署](#)教程)

```
python manage.py collectstatic
```

## 二，网站打开后报错

这时你可以把settings.py更改

```
DEBUG = True
```

重启服务器

```
sudo service apache2 restart
```

再访问网站 来查看具体的出错信息。

如果这样做还看不到出错信息，只是显示一个服务器错误，你可以查看apache2的错误日志

```
cat /var/log/apache2/error.log
```

根据错误日志里面的内容进行修正！

### 总结：

部署时文件对应关系：

siteName.conf --> wsgi.py --> settings.py --> urls.py --> views.py

扩展

明白了上面的关系，一个 Django project 使用多个域名或让app使用子域名很简单，只要新建一个 wsgi.py 文件，更改里面对应的settings文件，新的settings文件可以对应新的urls.py，从而做到访问与原来不同的地址！

[小额赞助，支持作者！](#)

[« Django 部署基础](#)

[Django 部署\(Nginx\)»](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信，随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [京ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 部署(Apache)

[« Django 部署基础](#)  
[Django 部署\(Nginx\) »](#)

本文讲述部署的方法和常见的问题, 并给出了在 BAE, SAE 等上面部署的实例。

如果不是在自己的服务器上部署, 当开发完成后可以部署到 [BAE](#), [SAE](#), 也可以使用阿里云的服务器。

推荐: [使用Code Studio 在线代码编辑, 新人更有免费一个月云主机, 可以用来实战体验本节的部署!](#)

部署到BAE的例子: [https://github.com/twz915/BAE\\_Django](https://github.com/twz915/BAE_Django) (不推荐, BAE已经要关闭了)

部署到SAE的例子: <https://github.com/twz915/django-sae> (Fork smallcode同学的, 没有测试过, SAE有一定的免费份额)

[Django + nginx + Gunicorn/uwsgi 部署方式, 参见另一篇: Django 部署 \(nginx\)](#)

自己的服务器 (比如用的[阿里云服务器](#)) 请看下文:

如果是新手, 个人推荐用ubuntu, 除非你对linux非常熟悉, ubuntu服务器的优点:

一, 开机apache2等都自动启动, 不需要额外设置

二, 安装软件非常方便 apt-get 搞定

三, 安装ssh, git等也非常容易, 几乎是傻瓜化

如果你在虚拟机或个人电脑中安装, 也可以试试 Linux Mint, 它用起来更简单, 和ubuntu兼容。

下面是ubuntu上的部署详细步骤:

### 1. 安装 apache2 和 mod\_wsgi

```
sudo apt-get install apache2

# Python 2
sudo apt-get install libapache2-mod-wsgi

# Python 3
sudo apt-get install libapache2-mod-wsgi-py3
```

### 2. 确认安装的apache2版本号

```
apachectl -v

Server version: Apache/2.4.6 (ubuntu)

Server built: Dec 5 2013 18:32:22
```

### 3. 准备一个新网站

ubuntu的apache2配置文件在 /etc/apache2/ 下

备注: centos 用户 apache 名称为 httpd 在 /etc/httpd/ 中 (可以参考文章下面置顶的评论)

新建一个网站配置文件

```
sudo vi /etc/apache2/sites-available/sitename.conf
```

示例内容如下:

```
<VirtualHost *:80>
    ServerName www.yourdomain.com
```

```

ServerAlias otherdomain.com
ServerAdmin tuweizhong@163.com

Alias /media/ /home/tu/blog/media/
Alias /static/ /home/tu/blog/static/

<Directory /home/tu/blog/media>
    Require all granted
</Directory>

<Directory /home/tu/blog/static>
    Require all granted
</Directory>

WSGIScriptAlias / /home/tu/blog/blog/wsgi.py
# WSGIDaemonProcess ziqiangxuetang.com python-path=/home/tu/blog:/home/tu/.virtualenvs/blog/lib/python2.7/site-packages
# WSGIProcessGroup ziqiangxuetang.com

<Directory /home/tu/blog/blog>
<Files wsgi.py>
    Require all granted
</Files>
</Directory>
</VirtualHost>

```

如果你的apache版本号是 2.2.x（第二步有方法判断）

用下面的代替 `Require all granted`

```

Order deny,allow
Allow from all

```

备注：把上面配置文件中这两句的备注去掉，可以使用 `virtualenv` 来部署网站，当然也可以只写一个 `/home/tu/blog`

```

# WSGIDaemonProcess ziqiangxuetang.com python-path=/home/tu/blog:/home/tu/.virtualenvs/blog/lib/python2.7/site-packages
# WSGIProcessGroup ziqiangxuetang.com

```

## 4. 修改wsgi.py文件

注意：上面如果写了 `WSGIDaemonProcess` 的话，这一步可以跳过，即可以不修改 `wsgi.py` 文件。

上面的配置中写的 `WSGIScriptAlias /home/tu/blog/blog/wsgi.py`

就是把apache2和你的网站project联系起来了

```

import os
from os.path import join,dirname,abspath

PROJECT_DIR = dirname(dirname(abspath(__file__)))#3
import sys # 4
sys.path.insert(0,PROJECT_DIR) # 5

os.environ["DJANGO_SETTINGS_MODULE"] = "blog.settings" # 7

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()

```

第 3, 4, 5 行为新加的内容，作用是让脚本找到django项目的位置，也可以在`sitemame.conf`中做，用`WSGI PythonPath`，想了解的自行搜索，第 7 行如果一台服务器有多个django project时一定要修改成上面那样，否则访问的时候会发生网站互相串的情况，即访问A网站到了B网站，一会儿正常，一会儿又不正常（当然也可以使用 `mod_wsgi daemon` 模式，[点击这里查看](#)）

## 5. 设置目录和文件权限

一般目录权限设置为 755，文件权限设置为 644

假如项目位置在 `/home/tu/zqxt`（在zqxt 下面有一个 `manage.py`，zqxt 是项目名称）

```

cd /home/tu/
sudo chmod -R 644 zqxt
sudo find zqxt -type d | xargs chmod 755

```

**apache** 服务器运行用户可以在 `/etc/apache2/envvars` 文件里面改，这里使用的是默认值，当然也可以更改成自己的当前用户，这样的话权限问题就简单很多，但在服务器上推荐有 **www-data** 用户，更安全。以下是默认设置：

```
# Since there is no sane way to get the parsed apache2 config in scripts, some
# settings are defined via environment variables and then used in apache2ctl,
# /etc/init.d/apache2, /etc/logrotate.d/apache2, etc.
```

```
export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
```

## 上传文件夹权限

**media** 文件夹一般用来存放用户上传文件，**static** 一般用来放自己网站的js，css，图片等，在**settings.py**中的相关设置

**STATIC\_URL** 为静态文件的网址 **STATIC\_ROOT** 为静态文件的根目录，

**MEDIA\_URL** 为用户上传文件夹的根目录，**MEDIA\_URL**为对应的访问网址

在**settings.py**中设置：

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/dev/howto/static-files/
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

# upload folder
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

在 **Linux** 服务器上，**用户上传目录**还要设置给 **www-data** 用户的写权限，下面的方法比较好，不影响原来的用户的编辑。

假如上传目录为 **zqxt/media/uploads** 文件夹,进入**media**文件夹，将 **uploads** 用户组改为**www-data**，并且赋予该组写权限：

```
cd media/ # 进入media文件夹
sudo chgrp -R www-data uploads
sudo chmod -R g+w uploads
```

**备注：**这两条命令，比直接用**sudo chown -R www-data:www-data uploads** 好，因为下面的命令不影响文件原来所属用户编辑文件，**fedora**系统应该不用设置上面的权限，但是个人强烈推荐用**ubuntu**,除非你对**linux**非常熟悉，你自己选择。

如果你使用的是**sqlite3**数据库，还会提示 **Attempt to write a readonly database**,同样要给**www-data**写数据库的权限

进入项目目录的上一级，比如**project**目录为 **/home/tu/blog** 那就进入 **/home/tu** 执行下面的命令（和修改上传文件夹类似）

```
sudo chgrp www-data blog
sudo chmod g+w blog
sudo chgrp www-data blog/db.sqlite3 # 更改为你的数据库名称
sudo chmod g+w blog/db.sqlite3
```

**备注：**上面的不要加 **-R**,-R是更改包括所有的子文件夹和文件，这样不安全。个人建议可以专门弄一个文件夹,用它来放**sqlite3**数据库，给该文件夹**www-data**写权限，而不是整个项目给写权限，有些文件只要读的权限就够了，给写权限会造成不安全。

## 6. 激活新网站

```
sudo a2ensite sitename 或 sudo a2ensite sitename.conf
```

如果顺利，这样网站就搭建成功，访问你的网址试试看，如果出现问题就接着看下面的。

## 7. 错误排查

一，没有静态文件，网站打开很乱，没有布局，多半是静态文件没有生效。

1. 确保你的配置文件中的路径是正确的
2. 确保你的**settings.py**中的文件设置正确
3. 收集静态文件(详细[静态文件部署](#)教程)

```
python manage.py collectstatic
```

### 二，网站打开后报错

这时你可以把**settings.py**更改



```
DEBUG = True
```

重启服务器

```
sudo service apache2 restart
```

再访问网站 来查看具体的出错信息。

如果这样做还看不到出错信息，只是显示一个服务器错误，你可以查看apache2的错误日志

```
cat /var/log/apache2/error.log
```

根据错误日志里面的内容进行修正！

### 总结：

部署时文件对应关系：

sitename.conf --> wsgi.py --> settings.py --> urls.py --> views.py

扩展

明白了上面的关系，一个 Django project 使用多个域名或让app使用子域名很简单，只要新建一个 wsgi.py 文件，更改里面对应的settings文件，新的 settings文件可以对应新的urls.py,从而做到访问与原来不同的地址！

[小额赞助，支持作者！](#)

[« Django 部署基础](#)

[Django 部署\(Nginx\)»](#)

 CODING  
Cloud Development Environment

零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 部署(Nginx)

[« Django 部署\(Apache\)](#)  
[Django 发送邮件 »](#)

觉得本文比较难的先学学[基础知识](#), 再来看这个

本文主要讲解 nginx + uwsgi socket 的方式来部署 Django, 比 Apache mod\_wsgi 要复杂一些, 但这是目前主流的方法。

[推荐: 使用Code Studio 云端开发, 新人更有免费一个月云主机, 可以用来实战体验本节的部署!](#)

## 1. 运行开发服务器测试

```
cd zqxt # 进入项目 zqxt 目录
python manage.py runserver
```

运行开发服务器测试, 确保开发服务器下能正常打开网站。

## 2. 安装 nginx 和 需要的包

### 2.1 安装 nginx 等软件

ubuntu / Linux Mint 等, 下面简写为 (ubuntu):

```
sudo apt-get install python-dev nginx
```

centos / Fedora / redhat 等, 下面简写为 (centos)

```
sudo yum install epel-release
sudo yum install python-devel nginx
```

### 2.2 安装 supervisor, 一个专门用来管理进程的工具, 我们用它来管理 uwsgi 进程

```
sudo pip install supervisor
```

Ubuntu用户 请直接看 3, 以下是CentOS 注意事项:

CentOS下, 如果不是非常懂 SELinux 和 iptables 的话, 为了方便调试, 可以先临时关闭它们, 如果发现部署了之后出不来结果, 可以临时关闭测试一下, 这样就知道是不是 SELinux 和 iptables 的问题

CentOS 7 iptables 如何使用: <http://stackoverflow.com/questions/24756240/>

将 SELinux 设置为宽容模式, 方便调试:

```
sudo setenforce 0
```

防火墙相关的设置:

可以选择临时关闭防火墙  

```
sudo service iptables stop
```

或者开放一些需要的端口, 比如 80  

```
sudo iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
```

上面的两条命令, 如果是 CentOS 7 用

临时关闭防火墙  

```
sudo systemctl stop firewalld
```

或者 开放需要的端口  

```
sudo firewall-cmd --zone=public --add-port=80/tcp --permanent
sudo firewall-cmd --reload
```

备注：由于我还没有用 最新版本的 Fedora，新版 Fedora 需要用 dnf 来安装包，有需求的同学自测，可以[参考这里](#)。

### 3. 使用 uwsgi 来部署

安装 uwsgi

```
sudo pip install uwsgi --upgrade
```

使用 uwsgi 运行项目

```
uwsgi --http :8001 --chdir /path/to/project --home=/path/to/env --module project.wsgi
```

这样就可以跑了，--home 指定 virtualenv 路径，如果没有可以去掉。project.wsgi 指的是 project/wsgi.py 文件

如果提示端口已经被占用：

```
probably another instance of uWSGI is running on the same address (:8002).
bind(): Address already in use [core/socket.c line 764]
```

这时可以把相关的进程 kill 掉：

按照端口进行查询：

```
lsof -i :8002
```

可以查出：

| COMMAND | PID  | USER | FD | TYPE | DEVICE             | SIZE/OFF | NODE | NAME                      |
|---------|------|------|----|------|--------------------|----------|------|---------------------------|
| uwsgi   | 2208 | tu   | 4u | IPv4 | 0x53492abadb5c9659 | 0t0      | TCP  | *:teradataordbms (LISTEN) |
| uwsgi   | 2209 | tu   | 4u | IPv4 | 0x53492abadb5c9659 | 0t0      | TCP  | *:teradataordbms (LISTEN) |

这时根据 PID 可以用下面的命令 kill 掉相关程序：

```
sudo kill -9 2208 2209
```

按照程序名称查询：

```
ps aux | grep uwsgi
```

补充内容：

使用 gunicorn 代替 uwsgi 的方法

```
sudo pip install gunicorn
```

在项目目录下运行下面的命令进行测试：

```
gunicorn -w4 -b0.0.0.0:8001 zqxt.wsgi
```

-w 表示开启多少个 worker，-b 表示要使用的 ip 和 port，我们这里用的是 8001，0.0.0.0 代表监控电脑的所有 ip。

如果使用了 virtualenv 可以这样

```
/path/to/env/bin/gunicorn --chdir /path/to/project --pythonpath /path/to/env/ -w4 -b0.0.0.0:8017 project.wsgi:application
```

用 --pythonpath 指定依赖包路径，多个的时候用逗号隔开，如：'/path/to/lib/home/tu/lib'

### 4. 使用 supervisor 来管理进程

安装 supervisor 软件包

```
(sudo) pip install supervisor
```

生成 supervisor 默认配置文件，比如我们放在 /etc/supervisord.conf 路径中：

```
(sudo) echo_supervisord_conf > /etc/supervisord.conf
```

打开 supervisord.conf 在最底部添加（每一行前面不要有空格，防止报错）：

```
[program:zqxt]
command=/path/to/uwsgi --http :8003 --chdir /path/to/zqxt --module zqxt.wsgi
directory=/path/to/zqxt
startsecs=0
stopwaitsecs=0
autostart=true
autorestart=true
```

command 中写上对应的命令，这样，就可以用 supervisor 来管理了。

启动 supervisor

```
(sudo) supervisord -c /etc/supervisord.conf
```

重启 zqxt 程序（项目）：

```
(sudo) supervisorctl -c /etc/supervisord.conf restart zqxt
```

启动，停止，或重启 supervisor 管理的某个程序 或 所有程序：

```
(sudo) supervisorctl -c /etc/supervisord.conf [start|stop|restart] [program-name|all]
```

以 uwsgi 为例，上面这样使用一行命令太长了，我们使用 ini 配置文件来搞定，比如项目在 /home/tu/zqxt 这个位置，

在其中新建一个 uwsgi.ini 全路径为 /home/tu/zqxt/uwsgi.ini

```
[uwsgi]
socket = /home/tu/zqxt/zqxt.sock
chdir = /home/tu/zqxt
wsgi-file = zqxt/wsgi.py
touch-reload = /home/tu/zqxt/reload
```

```
processes = 2
threads = 4
```

```
chmod-socket = 664
chown-socket = tu:www-data
```

```
vacuum = true
```

注意上面的 **/home/tu/zqxt/zqxt.sock**，一会儿我们把它和 nginx 关联起来。

在项目上新建一个空白的 reload 文件，只要 touch 一下这个文件（touch reload）项目就会重启。

**注意：不建议把 sock 文件放在 /tmp 下，比如 /tmp/xxx.sock (不建议)!** 有些系统的临时文件是 namespaced 的，进程只能看到自己的临时文件，导致 nginx 找不到 uwsgi 的 socket 文件，访问时显示 502，nginx 的 access log 中显示 `unix: /tmp/xxx.sock failed (2: No such file or directory)`，所以部署的时候建议用其它目录来放 socket 文件，比如放在运行 nginx 用户目录中，也可以专门弄一个目录来存放 sock 文件，比如 /tmp2/

```
sudo mkdir -p /tmp2/ && sudo chmod 777 /tmp2/
然后可以用 /tmp2/zqxt.sock 这样的路径了
```

详细参考 <http://stackoverflow.com/questions/32974204/got-no-such-file-or-directory-error-while-configuring-nginx-and-uwsgi>

修改 supervisor 配置文件中的 command 一行：

```
[program:zqxt]
command=/path/to/uwsgi --ini /home/tu/zqxt/uwsgi.ini
directory=/path/to/zqxt
startsecs=0
```

然后重启一下 supervisor：

```
(sudo) supervisorctl -c /etc/supervisord.conf restart zqxt
或者
(sudo) supervisorctl -c /etc/supervisord.conf restart all
```

## 5. 配置 Nginx

新建一个网站 zqxt

```
sudo vim /etc/nginx/sites-available/zqxt.conf
```

写入以下内容：

```
server {
    listen      80;
    server_name www.ziqiangxuetang.com;
    charset     utf-8;
```

```
client_max_body_size 75M;

location /media {
    alias /path/to/project/media;
}

location /static {
    alias /path/to/project/static;
}

location / {
    uwsgi_pass unix:///home/tu/zqxt/zqxt.sock;
    include /etc/nginx/uwsgi_params;
}
}
```

激活网站:

```
sudo ln -s /etc/nginx/sites-available/zqxt.conf /etc/nginx/sites-enabled/zqxt.conf
```

测试配置语法问题

```
sudo service nginx configtest 或 /path/to/nginx -t
```

重启 nginx 服务器:

```
sudo service nginx reload 或 sudo service nginx restart 或 /path/to/nginx -s reload
```

一些有用的参考教程:

Django 官网部署教程:

<https://docs.djangoproject.com/en/dev/howto/deployment/wsgi/gunicorn/>

<https://docs.djangoproject.com/en/dev/howto/deployment/wsgi/uwsgi/>

nginx 与 socket

[http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django\\_and\\_nginx.html#configure-nginx-for-your-site](http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html#configure-nginx-for-your-site)

防火墙:

iptables: <https://www.digitalocean.com/community/tutorials/how-to-setup-a-basic-ip-tables-configuration-on-centos-6>

centos 7 FireWald: <http://stackoverflow.com/questions/24756240/how-can-i-use-iptables-on-centos-7>

ubuntu ufw 防火墙: <http://wiki.ubuntu.org.cn/Ufw%E4%BD%BF%E7%94%A8%E6%8C%87%E5%8D%97>

uwsgi ini 配置文件: [http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django\\_and\\_nginx.html#configuring-uwsgi-to-run-with-a-ini-file](http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html#configuring-uwsgi-to-run-with-a-ini-file)

[小额赞助, 支持作者!](#)

[« Django 部署 \(Apache\)](#)

[Django 发送邮件 »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。  
自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 部署(Nginx)

[« Django 部署\(Apache\)](#)  
[Django 发送邮件 »](#)

觉得本文比较难的先学学[基础知识](#)，再来看这个

本文主要讲解 `nginx + uwsgi socket` 的方式来部署 Django，比 `Apache mod_wsgi` 要复杂一些，但这是目前主流的方法。

[推荐: 使用Code Studio 云端开发, 新人更有免费一个月云主机, 可以用来实战体验本节的部署!](#)

### 1. 运行开发服务器测试

```
cd zqxt # 进入项目 zqxt 目录
python manage.py runserver
```

运行开发服务器测试，确保开发服务器下能正常打开网站。

### 2. 安装 nginx 和 需要的包

#### 2.1 安装 nginx 等软件

ubuntu / Linux Mint 等，下面简写为 (ubuntu):

```
sudo apt-get install python-dev nginx
```

centos / Fedora / redhat 等，下面简写为 (centos)

```
sudo yum install epel-release
sudo yum install python-devel nginx
```

#### 2.2 安装 supervisor，一个专门用来管理进程的工具，我们用它来管理 uwsgi 进程

```
sudo pip install supervisor
```

**Ubuntu用户** 请直接看 3，以下是CentOS 注意事项:

CentOS下，如果不是非常懂 SELinux 和 iptables 的话，为了方便调试，可以先临时关闭它们，如果发现部署了之后出不来结果，可以临时关闭测试一下，这样就知道是不是 SELinux 和 iptables 的问题

**CentOS 7 iptables**如何使用: <http://stackoverflow.com/questions/24756240/>

将 SELinux 设置为宽容模式，方便调试:

```
sudo setenforce 0
```

防火墙相关的设置：

可以选择临时关闭防火墙  

```
sudo service iptables stop
```

或者开放一些需要的端口，比如 80  

```
sudo iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
```

上面的两条命令，如果是 CentOS 7 用

临时关闭防火墙  

```
sudo systemctl stop firewalld
```

或者 开放需要的端口  

```
sudo firewall-cmd --zone=public --add-port=80/tcp --permanent  
sudo firewall-cmd --reload
```

备注：由于我还没有用 最新版本的 Fedora，新版 Fedora 需要用 dnf 来安装包，有需求的同学自测，可以[参考这里](#)。

### 3. 使用 uwsgi 来部署

安装 uwsgi

```
sudo pip install uwsgi --upgrade
```

使用 uwsgi 运行项目

```
uwsgi --http :8001 --chdir /path/to/project --home=/path/to/env --module project.wsgi
```

这样就可以跑了，--home 指定 virtualenv 路径，如果没有可以去掉。project.wsgi 指的是 project/wsgi.py 文件

如果提示端口已经被占用：

```
probably another instance of uWSGI is running on the same address (:8002).  
bind(): Address already in use [core/socket.c line 764]
```

这时可以把相关的进程 kill 掉：

按照端口进行查询：

```
lsof -i :8002
```

可以查出：

| COMMAND | PID  | USER | FD | TYPE | DEVICE             | SIZE/OFF | NODE | NAME                      |
|---------|------|------|----|------|--------------------|----------|------|---------------------------|
| uwsgi   | 2208 | tu   | 4u | IPv4 | 0x53492abadb5c9659 | 0t0      | TCP  | *:teradataordbms (LISTEN) |
| uwsgi   | 2209 | tu   | 4u | IPv4 | 0x53492abadb5c9659 | 0t0      | TCP  | *:teradataordbms (LISTEN) |

这时根据 PID 可以用下面的命令 kill 掉相关程序：

```
sudo kill -9 2208 2209
```

按照程序名称查询：

```
ps aux | grep uwsgi
```

补充内容：

使用 gunicorn 代替 uwsgi 的方法

```
sudo pip install gunicorn
```

在项目目录下运行下面的命令进行测试：  

```
gunicorn -w4 -b0.0.0.0:8001 zqxt.wsgi
```

-w 表示开启多少个 worker，-b 表示要使用的 ip 和 port，我们这里用的是 8001，0.0.0.0 代表监控电脑的所有 ip。

如果使用了 virtualenv 可以这样

```
/path/to/env/bin/gunicorn --chdir /path/to/project --pythonpath /path/to/env/ -w4 -b0.0.0.0:8017 project.wsgi:application
```

用 --pythonpath 指定依赖包路径，多个的时候用逗号隔开，如：'/path/to/lib/home/tu/lib'

### 4. 使用 supervisor 来管理进程

安装 supervisor 软件包

```
(sudo) pip install supervisor
```

生成 supervisor 默认配置文件，比如我们放在 /etc/supervisord.conf 路径中：

```
(sudo) echo_supervisord_conf > /etc/supervisord.conf
```

打开 supervisor.conf 在最底部添加（**每一行前面不要有空格，防止报错**）：

```
[program:zqxt]
command=/path/to/uwsgi --http :8003 --chdir /path/to/zqxt --module zqxt.wsgi
directory=/path/to/zqxt
startsecs=0
stopwaitsecs=0
autostart=true
autorestart=true
```

command 中写上对应的命令，这样，就可以用 supervisor 来管理了。

启动 supervisor

```
(sudo) supervisord -c /etc/supervisord.conf
```

重启 zqxt 程序（项目）：

```
(sudo) supervisorctl -c /etc/supervisord.conf restart zqxt
```

启动，停止，或重启 supervisor 管理的某个程序 或 所有程序：

```
(sudo) supervisorctl -c /etc/supervisord.conf [start|stop|restart] [program-name|all]
```

以 uwsgi 为例，上面这样使用一行命令太长了，我们使用 ini 配置文件来搞定，比如项目在 /home/tu/zqxt 这个位置，

在其中新建一个 uwsgi.ini 全路径为 /home/tu/zqxt/uwsgi.ini

```
[uwsgi]
socket = /home/tu/zqxt/zqxt.sock
chdir = /home/tu/zqxt
wsgi-file = zqxt/wsgi.py
touch-reload = /home/tu/zqxt/reload
```

```
processes = 2
threads = 4
```

```
chmod-socket = 664
chown-socket = tu:www-data
```

```
vacuum = true
```

注意上面的 **/home/tu/zqxt/zqxt.sock**，一会儿我们把它和 nginx 关联起来。

在项目上新建一个空白的 reload 文件，只要 touch 一下这个文件（touch reload）项目就会重启。

**注意：不建议把 sock 文件放在 /tmp 下，比如 /tmp/xxx.sock (不建议)!** 有些系统的临时文件是 namespaced 的，进程只能看到自己的临时文件，导致 nginx 找不到 uwsgi 的 socket 文件，访问时显示 502，nginx 的 access log 中显示 `unix: /tmp/xxx.sock failed (2: No such file or directory)`，所以部署的时候建议用其它目录来放 socket 文件，比如放在运行 nginx 用户目录中，也可以专门弄一个目录来存放 sock 文件，比如 /tmp2/

```
sudo mkdir -p /tmp2/ && sudo chmod 777 /tmp2/
然后可以用 /tmp2/zqxt.sock 这样的路径了
```

详细参考 <http://stackoverflow.com/questions/32974204/got-no-such-file-or-directory-error-while-configuring-nginx-and-uwsgi>

修改 supervisor 配置文件中的 command 一行：

```
[program:zqxt]
command=/path/to/uwsgi --ini /home/tu/zqxt/uwsgi.ini
directory=/path/to/zqxt
startsecs=0
```

然后重启一下 supervisor：



```
(sudo) supervisorctl -c /etc/supervisord.conf restart zqxt
或者
(sudo) supervisorctl -c /etc/supervisord.conf restart all
```

## 5. 配置 Nginx

新建一个网站 **zqxt**

```
sudo vim /etc/nginx/sites-available/zqxt.conf
```

写入以下内容：

```
server {
    listen      80;
    server_name www.ziqiangxuetang.com;
    charset     utf-8;

    client_max_body_size 75M;

    location /media {
        alias /path/to/project/media;
    }

    location /static {
        alias /path/to/project/static;
    }

    location / {
        uwsgi_pass  unix:///home/tu/zqxt/zqxt.sock;
        include     /etc/nginx/uwsgi_params;
    }
}
```

激活网站：

```
sudo ln -s /etc/nginx/sites-available/zqxt.conf /etc/nginx/sites-enabled/zqxt.conf
```

测试配置语法问题

```
sudo service nginx configtest 或 /path/to/nginx -t
```

重启 **nginx** 服务器：

```
sudo service nginx reload 或 sudo service nginx restart 或 /path/to/nginx -s reload
```

一些有用的参考教程：

Django 官网部署教程：

<https://docs.djangoproject.com/en/dev/howto/deployment/wsgi/gunicorn/>

<https://docs.djangoproject.com/en/dev/howto/deployment/wsgi/uwsgi/>

nginx 与 socket

[http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django\\_and\\_nginx.html#configure-nginx-for-your-site](http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html#configure-nginx-for-your-site)

防火墙：

iptables: <https://www.digitalocean.com/community/tutorials/how-to-setup-a-basic-ip-tables-configuration-on-centos-6>

centos 7 FireWall: <http://stackoverflow.com/questions/24756240/how-can-i-use-iptables-on-centos-7>

ubuntu ufw 防火墙: <http://wiki.ubuntu.org.cn/Ufw%E4%BD%BF%E7%94%A8%E6%8C%87%E5%8D%97>

uwsgi ini 配置文件: [http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django\\_and\\_nginx.html#configuring-uwsgi-to-run-with-a-ini-file](http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html#configuring-uwsgi-to-run-with-a-ini-file)

[小额赞助，支持作者！](#)

[« Django 部署\(Apache\)](#)  
[Django 发送邮件 »](#)

 CODING  
Cloud Development

**零成本开启敏捷开发 五人以下团队免费**

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 发送邮件

[« Django 部署\(Nginx\)](#)

[Django 数据导入 »](#)

我们常常会用到一些发送邮件的功能, 比如有人提交了应聘的表单, 可以向HR的邮箱发邮件, 这样, HR不看网站就可以知道有人在网站上提交了应聘信息。

## 1. 配置相关参数

如果用的是 阿里云的企业邮箱, 则类似于下面:

在 `settings.py` 的最后面加上类似这些

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL_USE_TLS = False
EMAIL_HOST = 'smtp.tuweizhong.com'
EMAIL_PORT = 25
EMAIL_HOST_USER = 'mail@tuweizhong.com'
EMAIL_HOST_PASSWORD = 'xxxx'
DEFAULT_FROM_EMAIL = 'mail@tuweizhong.com'
```

或者

```
EMAIL_USE_SSL = True
EMAIL_HOST = 'smtp.qq.com' # 如果是 163 改成 smtp.163.com
EMAIL_PORT = 465
EMAIL_HOST_USER = 'xxx@qq.com' # 帐号
EMAIL_HOST_PASSWORD = 'p@ssw0rd' # 密码
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```

**EMAIL\_USE\_SSL 和 EMAIL\_USE\_TLS 是互斥的, 即只能有一个为 True。**

DEFAULT\_FROM\_EMAIL 还可以写成这样:

```
DEFAULT_FROM_EMAIL = 'tuweizhong <tuweizhong@163.com>'
```

这样别人收到的邮件中就会有设定的名称, 如下图:

发件人: **tuweizhong** <tuweizhong@163.com>

下面是一些常用的邮箱:

[163 邮箱](#) [126 邮箱](#) [QQ 邮箱](#)

其它邮箱参数可能登陆邮箱看寻找帮助信息, 也可以尝试在搜索引擎中搜索: "SMTP 邮箱名称", 比如: "163 SMTP" 进行查找。

## 2. 发送邮件:

2.1 官网的一个例子:

```
from django.core.mail import send_mail

send_mail('Subject here', 'Here is the message.', 'from@example.com',
        ['to@example.com'], fail_silently=False)
```

2.2 一次性发送多个邮件:

```
from django.core.mail import send_mass_mail
```

```
message1 = ('Subject here', 'Here is the message', 'from@example.com', ['first@example.com', 'other@example.com'])
message2 = ('Another Subject', 'Here is another message', 'from@example.com', ['second@test.com'])

send_mass_mail((message1, message2), fail_silently=False)
```

备注：`send_mail`每次发邮件都会建立一个连接，发多封邮件时建立多个连接。而 `send_mass_mail` 是建立单个连接发送多封邮件，所以一次性发送多封邮件时 `send_mass_mail` 要优于 `send_mail`。

### 2.3 如果我们想在邮件中添加附件，发送 `html` 格式的内容

```
from django.conf import settings
from django.core.mail import EmailMultiAlternatives

from_email = settings.DEFAULT_FROM_EMAIL
# subject 主题 content 内容 to_addr 是一个列表，发送给哪些人
msg = EmailMultiAlternatives(subject, content, from_email, [to_addr])

msg.content_subtype = "html"

# 添加附件（可选）
msg.attach_file('./twz.pdf')

# 发送
msg.send()
```

上面的做法可能有一些风险，除非你确信你的接收者都可以阅读 `html` 格式的邮件。

为安全起见，你可以弄两个版本，一个纯文本(`text/plain`)的为默认的，另外再提供一个 `html` 版本的（好像好多国外发的邮件都是纯文本的）

```
from __future__ import unicode_literals

from django.conf import settings
from django.core.mail import EmailMultiAlternatives

subject = '来自自强学堂的问候'

text_content = '这是一封重要的邮件。'

html_content = '<p>这是一封<strong>重要的</strong>邮件.</p>'

msg = EmailMultiAlternatives(subject, text_content, from_email, [to@youemail.com])

msg.attach_alternative(html_content, "text/html")

msg.send()
```

[小额赞助，支持作者！](#)

[« Django 部署\(Nginx\)](#)

[Django 数据导入 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [京ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 发送邮件

[« Django 部署\(Nginx\)](#)  
[Django 数据导入 »](#)

我们常常会用到一些发送邮件的功能，比如有人提交了应聘的表单，可以向HR的邮箱发邮件，这样，HR不看网站就可以知道有人在网站上提交了应聘信息。

### 1. 配置相关参数

如果用的是 阿里云的企业邮箱，则类似于下面：

在 `settings.py` 的最后面加上类似这些

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL_USE_TLS = False
EMAIL_HOST = 'smtp.tuweizhong.com'
EMAIL_PORT = 25
EMAIL_HOST_USER = 'mail@tuweizhong.com'
EMAIL_HOST_PASSWORD = 'xxxx'
DEFAULT_FROM_EMAIL = 'mail@tuweizhong.com'
```

或者

```
EMAIL_USE_SSL = True
EMAIL_HOST = 'smtp.qq.com' # 如果是 163 改成 smtp.163.com
EMAIL_PORT = 465
EMAIL_HOST_USER = 'xxx@qq.com' # 帐号
EMAIL_HOST_PASSWORD = 'p@ssw0rd' # 密码
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```

**EMAIL\_USE\_SSL 和 EMAIL\_USE\_TLS 是互斥的，即只能有一个为 True。**

DEFAULT\_FROM\_EMAIL 还可以写成这样：

```
DEFAULT_FROM_EMAIL = 'tuweizhong <tuweizhong@163.com>'
```

这样别人收到的邮件中就会有设定的名称，如下图：

发件人: **tuweizhong** <tuweizhong@163.com>

下面是一些常用的邮箱：

[163 邮箱](#) [126 邮箱](#) [QQ 邮箱](#)

其它邮箱参数可能登陆邮箱看寻找帮助信息，也可以尝试在搜索引擎中搜索：“SMTP 邮箱名称”，比如：“163 SMTP”进行查找。

## 2. 发送邮件：

### 2.1 官网的一个例子：

```
from django.core.mail import send_mail

send_mail('Subject here', 'Here is the message.', 'from@example.com',
        ['to@example.com'], fail_silently=False)
```

### 2.2 一次性发送多个邮件：

```
from django.core.mail import send_mass_mail

message1 = ('Subject here', 'Here is the message', 'from@example.com', ['first@example.com', 'other@example.com'])
message2 = ('Another Subject', 'Here is another message', 'from@example.com', ['second@test.com'])

send_mass_mail((message1, message2), fail_silently=False)
```

备注：`send_mail` 每次发邮件都会建立一个连接，发多封邮件时建立多个连接。而 `send_mass_mail` 是建立单个连接发送多封邮件，所以一次性发送多封邮件时 `send_mass_mail` 要优于 `send_mail`。

### 2.3 如果我们想在邮件中添加附件，发送 `html` 格式的内容

```
from django.conf import settings
from django.core.mail import EmailMultiAlternatives

from_email = settings.DEFAULT_FROM_EMAIL
# subject 主题 content 内容 to_addr 是一个列表，发送给哪些人
msg = EmailMultiAlternatives(subject, content, from_email, [to_addr])

msg.content_subtype = "html"

# 添加附件（可选）
msg.attach_file('./twz.pdf')

# 发送
msg.send()
```

上面的做法可能有一些风险，除非你确信你的接收者都可以阅读 `html` 格式的 邮件。

为安全起见，你可以弄两个版本，一个纯文本(`text/plain`)的为默认的，另外再提供一个 `html` 版本的（好像好多国外发的邮件都是纯文本的）

```
from __future__ import unicode_literals

from django.conf import settings
from django.core.mail import EmailMultiAlternatives

subject = '来自自强学堂的问候'

text_content = '这是一封重要的邮件。'

html_content = '<p>这是一封<strong>重要的</strong>邮件.</p>'

msg = EmailMultiAlternatives(subject, text_content, from_email, [to@youemail.com])

msg.attach_alternative(html_content, "text/html")

msg.send()
```

[小额赞助，支持作者！](#)  
[« Django 部署\(Nginx\)](#)  
[Django 数据导入 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的,自强学堂网站源代码免费下载](#)

# Django 数据导入

[« Django 发送邮件](#)  
[Django 数据迁移 »](#)

从网上下载的一些数据, excel表格, xml文件, txt文件等有时候我们想把它导入数据库, 应该如何操作呢?

以下操作符合 Django版本为 1.6, 兼顾 Django 1.7, Django 1.8 版本, 理论上Django 1.4, 1.5 也没有问题, 没有提到的都是默认值

备注: 你可能会问数据从哪儿来的, 比如你用python从以前的blog上获取过来的, 想导入现在的博客, 或者别人整理好的数据, 或者你自己整理的excel表, 一个个地在网站后台复制粘贴你觉得好么? 这就是批量导入的必要性。

下载本教程源代码: [mysite.zip](#)

建议先不要看源码, 按教程一步步做下去, 遇到问题再试试源代码, 直接复制粘贴, 很快就会忘掉, 自己动手打一遍

我们新建一个项目 mysite, 再新建一个 app, 名称为blog

```
django-admin.py startproject mysite
cd mysite
python manage.py startapp blog
```

把 blog 中的 models.py 更改为以下内容

```
#!/usr/bin/python
#coding:utf-8

from django.db import models

class Blog(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()

    def __unicode__(self):
        return self.title
```

不要忘了把 blog 加入到 settings.py 中的 INSTALLED\_APPS 中。

```
# Application definition
INSTALLED_APPS = (
    ...

    # 添加上 blog 这个 app
    'blog',
)
```

## 一, 同步数据库, 创建相应的表

```
python manage.py syncdb
```

Django 1.6以下版本会看到:



djangog 同步创建数据库表

Django 创建了一些默认的表，注意后面那个红色标记的**blog\_blog**是appname\_classname的样式，这个表是我们自己写的Blog类创建的

Django 1.7.6及以上的版本会看到：（第六行即为创建了对应的**blog\_blog**表）

```
Operations to perform:
  Synchronize unmigrated apps: blog
  Apply all migrations: admin, contenttypes, auth, sessions
Synchronizing apps without migrations:
  Creating tables...
    Creating table blog_blog
  Installing custom SQL...
  Installing indexes...
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial... OK

You have installed Django's auth system, and don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'tu'): tu
Email address:
Password:
Password (again):
Superuser created successfully.
```

## 二，输入 python manage.py shell

进入该项目的django环境的终端（windows如何进入对应目录？看 [Django环境搭建](#) 的 3.2 部分）

先说如何用命令新增一篇文章：

```
$ python manage.py shell
>>> from blog.models import Blog
>>> Blog.objects.create(title="The first blog of my site",
                        content="I am writing my blog on Terminal")
```

这样就新增了一篇博文，我们查看一下

```
>>> Blog.objects.all() # 获取所有blog
[<Blog: The first blog of my site>]
```

还有两种方法(这两种差不多):

```
>>> blog2 = Blog()
>>> blog2.title = "title 2"
>>> blog2.content = "content 2"
>>> blog2.save()
或者
```

```
>>> blog2 = Blog(title="title 2",content="content 2")
>>> blog2.save()
```

后面两种方法也很重要，尤其是用在修改数据的时候，要记得最后要保存一下 `blog.save()`，第一种 `Blog.objects.create()` 是自动保存的。

### 三，批量导入

比如我们要导入一个文本，里面是标题和内容，中间用四个\*隔开的，示例(`oldblog.txt`):

```
title 1****content 1
title 2****content 2
title 3****content 3
title 4****content 4
title 5****content 5
title 6****content 6
title 7****content 7
title 8****content 8
title 9****content 9
```

在终端导入有时候有些不方便，我们在 最外面那个 `mysite` 目录下写一个脚本，叫 `txt2db.py`，把 `oldblog.txt` 也放在 `mysite` 下

```
#!/usr/bin/env python
#coding:utf-8

import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")

'''
Django 版本大于等于1.7的时候，需要加上下面两句
import django
django.setup()
否则会抛出错误 django.core.exceptions.AppRegistryNotReady: Models aren't loaded yet.
'''

import django
if django.VERSION >= (1, 7):#自动判断版本
    django.setup()

def main():
    from blog.models import Blog
    f = open('oldblog.txt')
    for line in f:
        title,content = line.split('****')
        Blog.objects.create(title=title,content=content)
    f.close()

if __name__ == "__main__":
    main()
    print('Done!')
```

好了，我们在终端运行它

```
python txt2db.py
# 运行完后显示 一个 Done! 导入完成!
```

运行完毕后会打出一个 `"Done!"`，数据已经全部导入！

### 四，导入数据重复 解决办法

如果你导入数据过多，导入时出错了，或者你手动停止了，导入了一部分，还有一部分没有导入。或者你再次运行上面的命令，你会发现数据重复了，怎么办呢？

`django.db.models` 中还有一个函数叫 `get_or_create()` 有就获取过来，没有就创建，用它可以避免重复，但是速度可能会慢些，因为要先尝试获取，看看有没有

只要把上面的

```
Blog.objects.create(title=title,content=content)
```

换成下面的就不会重复导入数据了

```
Blog.objects.get_or_create(title=title,content=content)
```

返回值是 (`BlogObject`, `True/False`) 新建时返回 `True`, 已经存在时返回 `False`。

更多数据库API的知识请参见官网文档: [QuerySet API](#)

## 五, 用fixture导入

最常见的fixture文件就是用 `python manage.py dumpdata` 导出的文件, 示例如下:

```
[
  {
    "model": "myapp.person",
    "pk": 1,
    "fields": {
      "first_name": "John",
      "last_name": "Lennon"
    }
  },
  {
    "model": "myapp.person",
    "pk": 2,
    "fields": {
      "first_name": "Paul",
      "last_name": "McCartney"
    }
  }
]
```

你也可以根据自己的models, 创建这样的json文件, 然后用 `python manage.py loaddata fixture.json` 导入

详见 <https://docs.djangoproject.com/en/dev/howto/initial-data/>

可以写一个脚本, 把要导入的数据转化成 json 文件, 这样导入也会更快些!

## 六, `Model.objects.bulk_create()` 更快更方便

```
#!/usr/bin/env python
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")

def main():
    from blog.models import Blog
    f = open('oldblog.txt')
    BlogList = []
    for line in f:
        title,content = line.split('****')
        blog = Blog(title=title,content=content)
        BlogList.append(blog)
    f.close()

    Blog.objects.bulk_create(BlogList)
```

```
if __name__ == "__main__":
    main()
    print('Done!')
```

由于`Blog.objects.create()`每保存一条就执行一次SQL，而`bulk_create()`是执行一条SQL存入多条数据，做会快很多！当然用列表解析代替for循环会更快！！

```
#!/usr/bin/env python
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")

def main():
    from blog.models import Blog
    f = open('oldblog.txt')

    BlogList = []
    for line in f:
        parts = line.split('****')
        BlogList.append(Blog(title=parts[0], content=parts[1]))

    f.close()

    # 以上四行 也可以用 列表解析 写成下面这样
    # BlogList = [Blog(title=line.split('****')[0], content=line.split('****')[1]) for line in f]

    Blog.objects.bulk_create(BlogList)

if __name__ == "__main__":
    main()
    print('Done!')
```

当然也可以利用数据中的导出，再导入的方法，见下一节。

[小额赞助，支持作者！](#)

[« Django 发送邮件](#)

[Django 数据迁移 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 数据导入

[« Django 发送邮件](#)  
[Django 数据迁移 »](#)

从网上下载的一些数据, excel表格, xml文件, txt文件等有时候我们想把它导入数据库, 应该如何操作呢?

以下操作符合 **Django**版本为 **1.6**, 兼顾 **Django 1.7**, **Django 1.8** 版本, 理论上**Django 1.4**, **1.5** 也没有问题, 没有提到的都是默认值

备注: 你可能会问数据从哪儿来的, 比如你用python从以前的blog上获取过来的, 想导入现在的博客, 或者别人整理好的数据, 或者你自己整理的excel表, 一个个地在网站后台复制粘贴你觉得好么? 这就是批量导入的必要性。

下载本教程源代码: [mysite.zip](#)

建议先不要看源码, 按教程一步步做下去, 遇到问题再试试源代码, 直接复制粘贴, 很快就会忘掉, 自己动手打一遍

我们新建一个项目 **mysite**, 再新建一个 **app**, 名称为**blog**

```
django-admin.py startproject mysite
cd mysite
python manage.py startapp blog
```

把 **blog** 中的 **models.py** 更改为以下内容

```
#!/usr/bin/python
#coding:utf-8

from django.db import models

class Blog(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()

    def __unicode__(self):
        return self.title
```

不要忘了把 **blog** 加入到 **settings.py** 中的 **INSTALLED\_APPS** 中。

```
# Application definition
INSTALLED_APPS = (
    ...
    # 添加上 blog 这个 app
    'blog',
)
```

一, 同步数据库, 创建相应的表

```
python manage.py syncdb
```

Django 1.6以下版本会看到:

djanog 同步创建数据库表

Django 创建了一些默认的表, 注意后面那个红色标记的**blog\_blog**是appname\_classname的样式, 这个表是我们自己写的Blog类创建的

Django 1.7.6及以上的版本会看到: (第六行即为创建了对应的blog\_blog表)

```
Operations to perform:
  Synchronize unmigrated apps: blog
  Apply all migrations: admin, contenttypes, auth, sessions
Synchronizing apps without migrations:
  Creating tables...
    Creating table blog_blog
  Installing custom SQL...
  Installing indexes...
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial... OK
```

```
You have installed Django's auth system, and don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'tu'): tu
Email address:
Password:
Password (again):
Superuser created successfully.
```

## 二, 输入 python manage.py shell

进入该项目的django环境的终端 (windows如何进入对应目录? 看 [Django环境搭建](#) 的 3.2 部分)

先说如何用命令新增一篇文章:

```
$ python manage.py shell
>>> from blog.models import Blog
>>> Blog.objects.create(title="The first blog of my site",
                        content="I am writing my blog on Terminal")
```

这样就新增了一篇博文, 我们查看一下

```
>>> Blog.objects.all() # 获取所有blog
[<Blog: The first blog of my site>]
```

还有两种方法(这两种差不多):

```
>>> blog2 = Blog()
```

```
>>> blog2.title = "title 2"
>>> blog2.content = "content 2"
>>> blog2.save()
或者
>>> blog2 = Blog(title="title 2",content="content 2")
>>> blog2.save()
```

后面两种方法也很重要，尤其是用在修改数据的时候，要记得最后要保存一下 `blog.save()`，第一种 `Blog.objects.create()` 是自动保存的。

### 三，批量导入

比如我们要导入一个文本，里面是标题和内容，中间用四个\*隔开的，示例(`oldblog.txt`):

```
title 1****content 1
title 2****content 2
title 3****content 3
title 4****content 4
title 5****content 5
title 6****content 6
title 7****content 7
title 8****content 8
title 9****content 9
```

在终端导入有时候有些不方便，我们在 最外面那个 `mysite` 目录下写一个脚本，叫 `txt2db.py`，把 `oldblog.txt` 也放在 `mysite` 下

```
#!/usr/bin/env python
#coding:utf-8

import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")

'''
Django 版本大于等于1.7的时候，需要加上下面两句
import django
django.setup()
否则会抛出错误 django.core.exceptions.AppRegistryNotReady: Models aren't loaded yet.
'''

import django
if django.VERSION >= (1, 7):#自动判断版本
    django.setup()

def main():
    from blog.models import Blog
    f = open('oldblog.txt')
    for line in f:
        title,content = line.split('****')
        Blog.objects.create(title=title,content=content)
    f.close()

if __name__ == "__main__":
    main()
    print('Done!')
```

好了，我们在终端运行它

```
python txt2db.py
# 运行完后显示 一个 Done! 导入完成!
```

运行完毕后会打出一个 "Done!", 数据已经全部导入!

## 四，导入数据重复 解决办法

如果你导入数据过多，导入时出错了，或者你手动停止了，导入了一部分，还有一部分没有导入。或者你再次运行上面的命令，你会发现数据重复了，怎么办呢？

`django.db.models` 中还有一个函数叫 `get_or_create()` 有就获取过来，没有就创建，用它可以避免重复，但是速度可能会慢些，因为要先尝试获取，看看有没有

只要把上面的

```
Blog.objects.create(title=title,content=content)
```

换成下面的就不会重复导入数据了

```
Blog.objects.get_or_create(title=title,content=content)
```

返回值是 (`BlogObject`, `True/False`) 新建时返回 `True`, 已经存在时返回 `False`。

更多数据库API的知识请参见官网文档: [QuerySet API](#)

## 五, 用fixture导入

最常见的fixture文件就是用 `python manage.py dumpdata` 导出的文件,示例如下:

```
[
  {
    "model": "myapp.person",
    "pk": 1,
    "fields": {
      "first_name": "John",
      "last_name": "Lennon"
    }
  },
  {
    "model": "myapp.person",
    "pk": 2,
    "fields": {
      "first_name": "Paul",
      "last_name": "McCartney"
    }
  }
]
```

你也可以根据自己的models,创建这样的json文件,然后用 `python manage.py loaddata fixture.json` 导入

详见 <https://docs.djangoproject.com/en/dev/howto/initial-data/>

可以写一个脚本,把要导入的数据转化成 json 文件,这样导入也会更快些!

## 六， `Model.objects.bulk_create()` 更快更方便

```
#!/usr/bin/env python
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")

def main():
    from blog.models import Blog
    f = open('oldblog.txt')
    BlogList = []
    for line in f:
        title,content = line.split('*****')
        blog = Blog(title=title,content=content)
        BlogList.append(blog)
    f.close()
```



```
Blog.objects.bulk_create(BlogList)

if __name__ == "__main__":
    main()
    print('Done!')
```

由于`Blog.objects.create()`每保存一条就执行一次SQL，而`bulk_create()`是执行一条SQL存入多条数据，做会快很多！当然用列表解析代替for循环会更快！！

```
#!/usr/bin/env python
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")

def main():
    from blog.models import Blog
    f = open('oldblog.txt')

    BlogList = []
    for line in f:
        parts = line.split('****')
        BlogList.append(Blog(title=parts[0], content=parts[1]))

    f.close()

    # 以上四行 也可以用 列表解析 写成下面这样
    # BlogList = [Blog(title=line.split('****')[0], content=line.split('****')[1]) for line in f]

    Blog.objects.bulk_create(BlogList)

if __name__ == "__main__":
    main()
    print('Done!')
```

当然也可以利用数据中的导出，再导入的方法，见下一节。

[小额赞助，支持作者！](#)

[« Django 发送邮件](#)

[Django 数据迁移 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 数据迁移

[« Django 数据导入](#)  
[Django 多数据库联用 »](#)

本文主要讲数据库的迁移方法, 包含不同数据库, 如 SQLite3, MySQL, PostgreSQL 之间数据迁移方案, 以及数据在不同机器上迁移方案

## 一, 简单的数据导出与导入 (简单的迁移)

1. django 项目提供了一个导出的方法 `python manage.py dumpdata`, 不指定 `appname` 时默认为导出所有的app

```
python manage.py dumpdata [appname] > appname_data.json
```

比如我们有一个项目叫 `mysite`, 里面有一个 app 叫 `blog`, 我们想导出 `blog` 的所有数据

```
python manage.py dumpdata blog > blog_dump.json
```

2. 数据导入, 不需要指定 `appname`

```
python manage.py loaddata blog_dump.json
```

备注: 一些常用的

```
python manage.py dumpdata auth > auth.json # 导出用户数据
```

优点: 可以兼容各种支持的数据库, 也就是说, 以前用的是 SQLite3, 可以导出后, 用这种方法导入到 MySQL, PostgreSQL 等数据库, 反过来也可以。

缺点: 数据量大的时候, 速度相对较慢, 表的关系比较复杂的时候可以导入不成功。

## 二, 数据库的迁移

### 2.1. 用 Django 自带的命令

比如早期我们为了开发方便, 用的 sqlite3 数据库, 后来发现网站数据太多, sqlite3 性能有点跟不上了, 想换成 postgresql, 或者 MySQL 的时候。

如果我还使用上面的命令, 如果你运气好的话, 也许会导入成功, 流程如下:

#### 2.1.1. 从原来的整个数据库导出所有数据

```
python manage.py dumpdata > mysite_all_data.json
```

#### 2.1.2. 将 mysite\_all\_data.json 传送到另一个服务器或电脑上导入

```
python manage.py loaddata mysite_all_data.json
```

如果你运气好的话可能会导入完成, 但是往往不那么顺利, 原因如下:

a) 我们在写 models 的时候如果用到 CharField, 就一定要写 `max_length`, 在 sqlite3 中是不检查这个最大长度的, 你写最大允许

长度为100，你往数据库放10000个，sqlite3都不报错，而且不截断数据的长度，这似乎是slite3的优点，但是也给从sqlite3导入其它数据库带来了困难,因为MySQL和PostgreSQL数据库都会检查最大长度，超出时就报错！

b) Django 自带的contentType会导致出现一些问题

用上面的方法只迁移一个app应该问题不大，但是如果有用户，用户组挂钩，事情往往变得糟糕！如果导入后没有对数据进行修改，你可以考虑重新导入，可能还要快一些，如果是手动在后台输入或者修改过，这种方法就不适用了

## 2.2, 用数据库自带的导出导入命令

预备知识：

先输入 mysql (比如 mysql -u root -p) 进入数据库 shell

创建 GBK 格式的数据库 zqxt  
create database `zqxt` DEFAULT CHARACTER SET gbk COLLATE gbk\_chinese\_ci;

创建 UTF8 格式的数据库 zqxt  
CREATE DATABASE `zqxt` DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;

赋予数据库（zqxt）权限给某用户，可以是已经存在的用户或新用户名  
GRANT ALL PRIVILEGES ON zqxt.\* TO "任意用户名"@"localhost" IDENTIFIED BY "新密码";

刷新权限  
FLUSH PRIVILEGES;

退出数据库shell  
EXIT;

假定 Django 用的数据库名称为 zqxt

### 2.2.1 在 PostgreSQL 中：

# 导出数据库 zqxt 到 zqxt.sql 文件中  
pg\_dump zqxt > zqxt.sql

# 导入数据库到 新的服务器  
psql zqxt -f zqxt.sql

#注意：数据导入导出可能需要数据库超级权限,用 sudo su postgres 切换到数据库超级用户 postgres

### 2.2.2 在 MySQL 中：

使用网页工具，比如phpMyAdmin导入导出很简单，这里就不说了，主要说一下命令行如何操作：

# 导出数据库 zqxt 到 zqxt.sql 文件中  
mysqldump -u username -p --database zqxt > zqxt.sql

# 导入数据库到 新的服务器（假设数据库已经创建好）  
cat /path/to/zqxt.sql | mysql -u username -p zqxt  
或 mysql -u username -p zqxt < /path/to/zqxt.sql  
或 mysql -u username -p zqxt 进入 mysql shell 后，执行 source /path/to/zqxt.sql  
# 输入密码开始导入数据

**总结：**其它的数据库，请自行搜索如何导入导出，整个数据库导出的好处就是对数据之间的关系处理比较省事，比如自学学堂里面的很多教程，上一篇和下一篇是一个一对一的关系，这样的话用 python manage.py dumpdata 无法导出教程与教程的关系，但是数据库整个导出就没有任何问题，当然也可以写一个脚本去导出关系再导入。Django 自带的 python manage.py dumpdata 和 python manage.py loaddata 最大的好处就是可以跨数据库进行导入导出。

[小额赞助，支持作者！](#)

[«Django 数据导入](#)  
[Django 多数据库联用»](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 数据迁移

[«Django 数据导入](#)  
[Django 多数据库联用»](#)

本文主要讲[数据库的迁移方法](#)，包含不同数据库，如 SQLite3, MySQL, PostgreSQL 之间数据迁移方案，以及数据在不同机器上迁移方案

### 一，简单的数据导出与导入（简单的迁移）

1. django 项目提供了一个导出的方法 `python manage.py dumpdata`, 不指定 `appname` 时默认为导出所有的app

```
python manage.py dumpdata [appname] > appname_data.json
```

比如我们有一个项目叫 `mysite`, 里面有一个 app 叫 `blog`, 我们想导出 `blog` 的所有数据

```
python manage.py dumpdata blog > blog_dump.json
```

2. 数据导入,不需要指定 `appname`

```
python manage.py loaddata blog_dump.json
```

备注：一些常用的

```
python manage.py dumpdata auth > auth.json # 导出用户数据
```

优点：可以兼容各种支持的数据库，也就是说，以前用的是 SQLite3，可以导出后，用这种方法导入到 MySQL, PostgreSQL等数据库，反过来也可以。

缺点：数据量大的时候，速度相对较慢，表的关系比较复杂的时候可以导入不成功。

## 二，数据库的迁移

### 2.1. 用 Django 自带的命令

比如早期我们为了开发方便，用的sqlite3数据库，后来发现网站数据太多，sqlite3性能有点跟不上了，想换成 PostgreSQL,或者 MySQL的时候。

如果还使用上面的命令，如果你运气好的话，也许会导入成功，流程如下：

#### 2.1.1. 从原来的整个数据库导出所有数据

```
python manage.py dumpdata > mysite_all_data.json
```

#### 2.1.2. 将mysite\_all\_data.json传送到另一个服务器或电脑上导入

```
python manage.py loaddata mysite_all_data.json
```

如果你运气好的话可能会导入完成，但是往往不那么顺利，原因如下：

a) 我们在写models的时候如果用到CharField,就一定要写max\_length,在sqlite3中是不检查这个最大长度的，你写最大允许长度为100，你往数据库放10000个，sqlite3都不报错，而且不截断数据的长度，这似乎是sqlite3的优点，但是也给从sqlite3导入其它数据库带来了困难,因为MySQL和PostgreSQL数据库都会检查最大长度，超出时就报错！

b) Django 自带的contentType会导致出现一些问题

用上面的方法只迁移一个app应该问题不大，但是如果有用户，用户组挂钩，事情往往变得糟糕！如果导入后没有对数据进行修改，你可以考虑重新导入，可能还要快一些，如果是手动在后台输入或者修改过，这种方法就不适用了

### 2.2, 用数据库自带的导出导入命令

预备知识：

先输入 mysql (比如 mysql -u root -p) 进入数据库 shell

```
创建 GBK 格式的数据库 zqxt
create database `zqxt` DEFAULT CHARACTER SET gbk COLLATE gbk_chinese_ci;
```

```
创建 UTF8 格式的数据库 zqxt
CREATE DATABASE `zqxt` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

```
赋予数据库（zqxt）权限给某用户，可以是已经存在的用户或新用户名
GRANT ALL PRIVILEGES ON zqxt.* TO "任意用户名"@"localhost" IDENTIFIED BY "新密码";
```

```
刷新权限
FLUSH PRIVILEGES;
```

```
退出数据库shell
EXIT;
```

假定 Django 用的数据库名称为 **zqxt**

2.2.1 在 **PostgreSQL** 中:

```
# 导出数据库 zqxt 到 zqxt.sql 文件中
pg_dump zqxt > zqxt.sql
```

```
# 导入数据库到 新的服务器
psql zqxt -f zqxt.sql
```

#注意: 数据导入导出可能需要数据库超级权限,用 `sudo su postgres` 切换到数据库超级用户 `postgres`

2.2.2 在 **MySQL** 中:

使用网页工具, 比如 **phpMyAdmin** 导入导出很简单, 这里就不说了, 主要说一下命令行如何操作:

```
# 导出数据库 zqxt 到 zqxt.sql 文件中
mysqldump -u username -p --database zqxt > zqxt.sql

# 导入数据库到 新的服务器 (假设数据库已经创建好)
cat /path/to/zqxt.sql | mysql -u username -p zqxt
或 mysql -u username -p zqxt < /path/to/zqxt.sql
或 mysql -u username -p zqxt 进入 mysql shell 后, 执行 source /path/to/zqxt.sql
# 输入密码开始导入数据
```

**总结:** 其它的数据库, 请自行搜索如何导入导出, 整个数据库导出的好处就是对数据之间的关系处理比较省事, 比如 自强学堂里面的很多教程, 上一篇和下一篇是一个一对一的关系, 这样的话用 `python manage.py dumpdata` 无法导出教程与教程的关系, 但是数据库整个导出就没有任何问题, 当然也可以写一个脚本去导出关系再导入。Django 自带的 `python manage.py dumpdata` 和 `python manage.py loaddata` 最大的好处就是可以跨数据库进行导入导出。

[小额赞助, 支持作者!](#)

[« Django 数据导入](#)

[Django 多数据库联用 »](#)

 CODING  
GROUP DEVELOPMENT

零成本开启敏捷开发 五人以下团队免费

免费体验

↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 多数据库联用

[« Django 数据迁移](#)  
[Django 用户注册系统 »](#)

本文讲述在一个 django project 中使用多个数据库的方法, 多个数据库的联用 以及多数据库时数据导入导出的方法。

直接给出一种简单的方法吧, 想了解更多的到官方教程, [点击此处](#)

代码文件下载: [project\\_name.zip](#) (2017年05月01日更新)

## 1. 每个app都可以单独设置一个数据库

settings.py中有数据库的相关设置, 有一个默认的数据库 default,我们可以再加一些其它的, 比如:

```
# Database
# https://docs.djangoproject.com/en/1.8/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
    'db1': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dbname1',
        'USER': 'your_db_user_name',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
    },
    'db2': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dbname2',
        'USER': 'your_db_user_name',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
    },
}

# use multi-database in django
# add by WeizhongTu
DATABASE_ROUTERS = ['project_name.database_router.DatabaseAppsRouter']
DATABASE_APPS_MAPPING = {
    # example:
    # 'app_name': 'database_name',
    'app1': 'db1',
    'app2': 'db2',
}
```

在project\_name文件夹中存放 database\_router.py 文件, 内容如下:

```
# -*- coding: utf-8 -*-
from django.conf import settings

DATABASE_MAPPING = settings.DATABASE_APPS_MAPPING

class DatabaseAppsRouter(object):
```

```

"""
A router to control all database operations on models for different
databases.

In case an app is not set in settings.DATABASE_APPS_MAPPING, the router
will fallback to the `default` database.

Settings example:

DATABASE_APPS_MAPPING = {'app1': 'db1', 'app2': 'db2'}
"""

def db_for_read(self, model, **hints):
    """Point all read operations to the specific database."""
    if model._meta.app_label in DATABASE_MAPPING:
        return DATABASE_MAPPING[model._meta.app_label]
    return None

def db_for_write(self, model, **hints):
    """Point all write operations to the specific database."""
    if model._meta.app_label in DATABASE_MAPPING:
        return DATABASE_MAPPING[model._meta.app_label]
    return None

def allow_relation(self, obj1, obj2, **hints):
    """Allow any relation between apps that use the same database."""
    db_obj1 = DATABASE_MAPPING.get(obj1._meta.app_label)
    db_obj2 = DATABASE_MAPPING.get(obj2._meta.app_label)
    if db_obj1 and db_obj2:
        if db_obj1 == db_obj2:
            return True
        else:
            return False
    return None

# for Django 1.4 - Django 1.6
def allow_syncdb(self, db, model):
    """Make sure that apps only appear in the related database."""

    if db in DATABASE_MAPPING.values():
        return DATABASE_MAPPING.get(model._meta.app_label) == db
    elif model._meta.app_label in DATABASE_MAPPING:
        return False
    return None

# Django 1.7 - Django 1.11
def allow_migrate(self, db, app_label, model_name=None, **hints):
    print db, app_label, model_name, hints
    if db in DATABASE_MAPPING.values():
        return DATABASE_MAPPING.get(app_label) == db
    elif app_label in DATABASE_MAPPING:
        return False
    return None

```

这样就实现了指定的 **app** 使用指定的数据库了,当然你也可以多个sqlite3一起使用,相当于可以给每个app都可以单独设置一个数据库! 如果不设置或者没有设置的app就会自动使用默认的数据库。

## 2.使用指定的数据库来执行操作

在查询的语句后面用 **using(dbname)** 来指定要操作的数据库即可

```

# 查询
YourModel.objects.using('db1').all()
或者 YourModel.objects.using('db2').all()

# 保存 或 删除
user_obj.save(using='new_users')

```



```
user_obj.delete(using='legacy_users')
```

### 3.多个数据库联用时数据导入导出

使用的时候和一个数据库的区别是:

如果不是default(默认数据库)要在命令后边加 **--database=数据库对应的settings.py中的名称** 如: **--database=db1** 或 **--database=db2**

#### 数据库同步 (创建表)

# Django 1.6及以下版本

```
python manage.py syncdb #同步默认的数据库, 和原来的没有区别
```

# 同步数据库 db1 (注意: 不是数据库名是db1, 是settings.py中的那个db1, 不过你可以使这两个名称相同, 容易使用)

```
python manage.py syncdb --database=db1
```

# Django 1.7 及以上版本

```
python manage.py migrate --database=db1
```

#### 数据导出

```
python manage.py dumpdata app1 --database=db1 > app1_fixture.json
```

```
python manage.py dumpdata app2 --database=db2 > app2_fixture.json
```

```
python manage.py dumpdata auth > auth_fixture.json
```

#### 数据库导入

```
python manage.py loaddata app1_fixture.json --database=db1
```

```
python manage.py loaddata app2_fixture.json --database=db2
```

[小额赞助, 支持作者!](#)

[« Django 数据迁移](#)

[Django 用户注册系统 »](#)

 **零成本开启敏捷开发 五人以下团队免费**

免费体验

↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云ECS](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 多数据库联用

[« Django 数据迁移](#)  
[Django 用户注册系统 »](#)

本文讲述在一个 django project 中使用多个数据库的方法, 多个数据库的联用 以及多数据库时数据导入导出的方法。

直接给出一种简单的方法吧, 想了解更多的到官方教程, [点击此处](#)

代码文件下载: [project\\_name.zip](#) (2017年05月01日更新)

## 1. 每个app都可以单独设置一个数据库

settings.py中有数据库的相关设置, 有一个默认的数据库 default,我们可以再加一些其它的, 比如:

```
# Database
# https://docs.djangoproject.com/en/1.8/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
    'db1': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dbname1',
        'USER': 'your_db_user_name',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
    },
    'db2': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dbname2',
        'USER': 'your_db_user_name',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
    },
}

# use multi-database in django
# add by WeizhongTu
DATABASE_ROUTERS = ['project_name.database_router.DatabaseAppsRouter']
DATABASE_APPS_MAPPING = {
    # example:
    # 'app_name': 'database_name',
    'app1': 'db1',
    'app2': 'db2',
}
```

在project\_name文件夹中存放 database\_router.py 文件, 内容如下:

```

# -*- coding: utf-8 -*-
from django.conf import settings

DATABASE_MAPPING = settings.DATABASE_APPS_MAPPING

class DatabaseAppsRouter(object):
    """
    A router to control all database operations on models for different
    databases.

    In case an app is not set in settings.DATABASE_APPS_MAPPING, the router
    will fallback to the `default` database.

    Settings example:

    DATABASE_APPS_MAPPING = {'app1': 'db1', 'app2': 'db2'}
    """

    def db_for_read(self, model, **hints):
        """Point all read operations to the specific database."""
        if model._meta.app_label in DATABASE_MAPPING:
            return DATABASE_MAPPING[model._meta.app_label]
        return None

    def db_for_write(self, model, **hints):
        """Point all write operations to the specific database."""
        if model._meta.app_label in DATABASE_MAPPING:
            return DATABASE_MAPPING[model._meta.app_label]
        return None

    def allow_relation(self, obj1, obj2, **hints):
        """Allow any relation between apps that use the same database."""
        db_obj1 = DATABASE_MAPPING.get(obj1._meta.app_label)
        db_obj2 = DATABASE_MAPPING.get(obj2._meta.app_label)
        if db_obj1 and db_obj2:
            if db_obj1 == db_obj2:
                return True
            else:
                return False
        return None

    # for Django 1.4 - Django 1.6
    def allow_syncdb(self, db, model):
        """Make sure that apps only appear in the related database."""

        if db in DATABASE_MAPPING.values():
            return DATABASE_MAPPING.get(model._meta.app_label) == db
        elif model._meta.app_label in DATABASE_MAPPING:
            return False
        return None

    # Django 1.7 - Django 1.11
    def allow_migrate(self, db, app_label, model_name=None, **hints):
        print db, app_label, model_name, hints
        if db in DATABASE_MAPPING.values():
            return DATABASE_MAPPING.get(app_label) == db
        elif app_label in DATABASE_MAPPING:
            return False
        return None

```

这样就实现了指定的 **app** 使用指定的数据库了,当然你也可以多个 **sqlite3** 一起使用,相当于可以给每个 **app** 都可以单独设置一个数据库! 如果不设置或者没有设置的 **app** 就会自动使用默认的数据库。

## 2.使用指定的数据库来执行操作

在查询的语句后面用 **using(dbname)** 来指定要操作的数据库即可

```
# 查询
YourModel.objects.using('db1').all()
或者 YourModel.objects.using('db2').all()

# 保存 或 删除
user_obj.save(using='new_users')
user_obj.delete(using='legacy_users')
```

### 3.多个数据库联用时数据导入导出

使用的时候和一个数据库的区别是：

如果不是default(默认数据库)要在命令后边加 **--database=数据库对应的settings.py中的名称** 如： **--database=db1**  
或 **--database=db2**

#### 数据库同步（创建表）

```
# Django 1.6及以下版本
python manage.py syncdb #同步默认的数据库，和原来的没有区别

# 同步数据库 db1（注意：不是数据库名是db1，是settings.py中的那个db1，不过你可以使这两个名称相同，容易使用）
python manage.py syncdb --database=db1

# Django 1.7 及以上版本
python manage.py migrate --database=db1
```

#### 数据导出

```
python manage.py dumpdata app1 --database=db1 > app1_fixture.json
python manage.py dumpdata app2 --database=db2 > app2_fixture.json
python manage.py dumpdata auth > auth_fixture.json
```

#### 数据库导入

```
python manage.py loaddata app1_fixture.json --database=db1
python manage.py loaddata app2_fixture.json --database=db2
```

[小额赞助，支持作者！](#)

[« Django 数据迁移](#)

[Django 用户注册系统 »](#)

 CODING  
CLOUD DEVELOPMENT

零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 用户注册系统

[« Django 多数据库联用](#)  
[Django 缓存系统 »](#)

直接上代码, 下载代码自己跑起来看看。

链接: <https://pan.baidu.com/s/1fCXe9rTZdHjnvf2sPDla4g>

密码: 9fm3

下面的需要更新, 建议不要看, 可以大致浏览下, 其它可参考的实现 `django-user-accounts`, `django-userena`。

Django 的源码中已经有登陆, 退出, 重设密码等相关的视图函数, 在下面这个 app 中

`django.contrib.auth`

可以点击对应的版本查看相关源代码: [1.9](#) [1.8](#) [1.7](#) [1.6](#) [1.5](#) [1.4](#)

## 一, 创建一个 `zqxt_auth` 项目

```
django-admin startproject zqxt_auth
```

打开 `zqxt_auth/setting.py` 可以看到 `django.contrib.auth` 已经在 `INSTALLED_APPS` 中:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

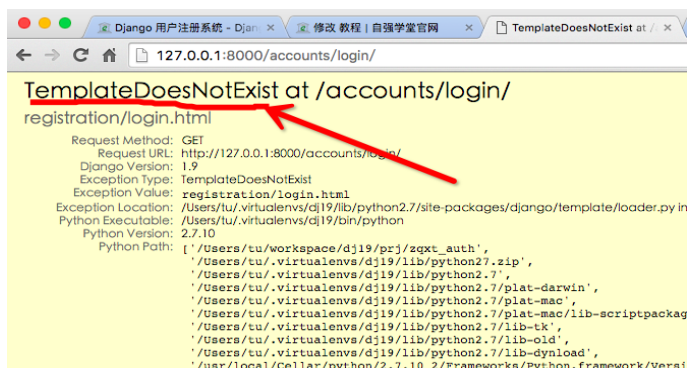
注: 各版本的Django生成的文件可能有些差异, 请按照你的 Django 版本为准。

## 二, 修改 `urls.py` 【此步阅读即可, 不需要照着做】

```
from django.conf.urls import url, include  
from django.contrib import admin  
from django.contrib.auth import urls as auth_urls  
  
urlpatterns = [  
    url(r'^accounts/', include(auth_urls, namespace='accounts')),  
    url(r'^admin/', admin.site.urls),  
]
```

我们引入了 `django.contrib.auth.urls` 中的内容, 改好后, 我们试着访问一下:

<http://127.0.0.1:8000/accounts/login/> 报错信息说:



下面我们的任务就是弄一些相关的模板了。

### 三、准备相关的模板文件【此步阅读即可，不需要照着做】

Django默认配置下会自动寻找 app 下的模板，但是 `django.contrib.auth` 这个是官方提供的，我们修改这个 app 不太容易，我们可以建立一个公用的模板文件夹。

#### 3.1 添加一个公用的放模板的文件夹

Django 1.8 及以上的版本 `settings.py`，修改 `TEMPLATES` 中的 `DIRS`

```
TEMPLATES = [
    {
        ...
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        ...
    },
]
```

Django 1.7 及以下的版本，修改 `TEMPLATE_DIRS`:

```
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates'),
)
```

注：如果是旧的项目已经存在公用的，可以不添加上面的目录

这个文件夹需要我们手工创建出来，创建后如下：









```
tu@mac ~/workspace/dj19/prj/zqxt_auth $ pwd
/Users/tu/workspace/dj19/prj/zqxt_auth

tu@mac ~/workspace/dj19/prj/zqxt_auth $ tree .
.
├── db.sqlite3
├── manage.py
├── templates # 手工创建的模板文件
└── zqxt_auth
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

2 directories, 6 files

#### 3.2 模板文件

熟悉Django的同学知道，Django 的后台是有登陆，重设密码的功能，那么在 `django.contrib.admin` 中应该是有相应的模板文件的，一找，果然有，[查看链接](#)。

|   |  |
|---|--|
| scop committed with timgraham [1.9.x] Fixed #25565 -- Removed action="" from admin forms (invalid i... <span>...</span>       |  |
| ..  |  |
|  <a href="#">logged_out.html</a>             | Fixed #18037 -- Changed behaviour of url and ssi template tags to the... |
|  <a href="#">password_change_done.html</a>   | Fixed #18511 -- Cleaned up admin password reset template titles.         |
|  <a href="#">password_change_form.html</a>   | [1.9.x] Fixed #25565 -- Removed action="" from admin forms (invalid i... |
|  <a href="#">password_reset_complete....</a> | Fixed #18511 -- Cleaned up admin password reset template titles.         |
|  <a href="#">password_reset_confirm.html</a> | [1.9.x] Fixed #25565 -- Removed action="" from admin forms (invalid i... |
|  <a href="#">password_reset_done.html</a>    | Fixed #23793 -- Clarified password reset messages.                       |
|  <a href="#">password_reset_email.html</a>   | Fixed #14881 -- Modified password reset to work with a non-integer Us... |
|  <a href="#">password_reset_form.html</a>    | [1.9.x] Fixed #25565 -- Removed action="" from admin forms (invalid i... |

我们把这些文件拷贝出来。

需要注意的是，有人已经按照类似的方法，做成了一个包了，地址在这里：<https://github.com/mishbahr/django-users2>

这个包比较完善了，我们没必要重新发明轮子，上面的示例只是让你明白，这个包其实是由官方的django.contrib.auth改进后做出来的。

四，用 **django-users2** 这个包来实现登陆注册及找回密码等功能。

**django-users2 这个包在 Django 1.5 - Django 1.9 中使用都没有问题。**

#### 4.1 安装

```
pip install django-users2
```

#### 4.2 把 users 这个 app 加入到 INSTALLED\_APPS

```
INSTALLED_APPS = (
    ...
    'django.contrib.auth',
    'django.contrib.sites',
    'users',
    ...
)
```

```
AUTH_USER_MODEL = 'users.User'
```

AUTH\_USER\_MODEL 是替换成自定义的用户认证。[参考这里](#)

#### 4.3 修改 urls.py

```
urlpatterns = patterns('',
    ...
    url(r'^accounts/', include('users.urls')),
    ...
)
```

#### 4.4 同步数据，创建相应的表

```
python manage.py syncdb
```

Django 1.7 及以上

```
python manage.py makemigrations
python manage.py migrate
```

#### 4.5 配置登陆注册的一些选项，找回密码时发邮件的邮箱

下面的代码加在 `settings.py` 最后面

```
USERS_REGISTRATION_OPEN = True

USERS_VERIFY_EMAIL = True

USERS_AUTO_LOGIN_ON_ACTIVATION = True

USERS_EMAIL_CONFIRMATION_TIMEOUT_DAYS = 3

# Specifies minimum length for passwords:
USERS_PASSWORD_MIN_LENGTH = 5

# Specifies maximum length for passwords:
USERS_PASSWORD_MAX_LENGTH = None

# the complexity validator, checks the password strength
USERS_CHECK_PASSWORD_COMPLEXITY = True

USERS_SPAM_PROTECTION = False # important!

# -----
# Email
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL_USE_TLS = False
EMAIL_HOST = 'smtp.tuweizhong.com'
EMAIL_PORT = 25
EMAIL_HOST_USER = 'mail@tuweizhong.com'
EMAIL_HOST_PASSWORD = 'xxxx'
DEFAULT_FROM_EMAIL = 'mail@tuweizhong.com'
# -----
```

这样登陆注册和找回密码功能应该就没有问题了。

[小额赞助，支持作者！](#)

[« Django 多数据库联用](#)

[Django 缓存系统 »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---



自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 用户注册系统

[« Django 多数据库联用](#)  
[Django 缓存系统 »](#)

直接上代码，下载代码自己跑起来看看。

链接: <https://pan.baidu.com/s/1fCXe9rTZdHjnvf2sPDla4g>

密码: 9fm3

下面的需要更新，建议不要看，可以大致浏览下，其它可参考的实现 [django-user-accounts](#), [django-userena](#)。

Django 的源码中已经有登陆，退出，重设密码等相关的视图函数，在下面这个 app 中

`django.contrib.auth`

可以点击对应的版本查看相关源代码: [1.9](#) [1.8](#) [1.7](#) [1.6](#) [1.5](#) [1.4](#)

### 一，创建一个 `zqxt_auth` 项目

```
django-admin startproject zqxt_auth
```

打开 `zqxt_auth/setting.py` 可以看到 `django.contrib.auth` 已经在 `INSTALLED_APPS` 中：

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

注：各版本的Django生成的文件可能有些差异，请按照你的 Django 版本为准。

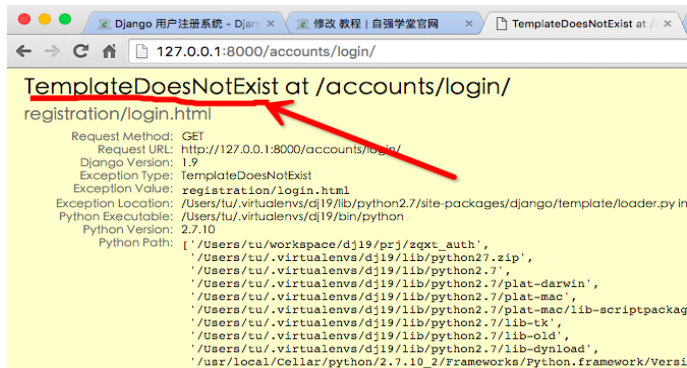
### 二，修改 `urls.py` 【此步阅读即可，不需要照着做】

```
from django.conf.urls import url, include  
from django.contrib import admin  
from django.contrib.auth import urls as auth_urls
```

```
urlpatterns = [
    url(r'^accounts/', include(auth_urls, namespace='accounts')),
    url(r'^admin/', admin.site.urls),
]
```

我们引入了 `django.contrib.auth.urls` 中的内容，改好后，我们试着访问一下：

<http://127.0.0.1:8000/accounts/login/> 报错信息说：



下面我们的任务就是弄一些相关的模板了。

### 三、准备相关的模板文件【此步阅读即可，不需要照着做】

Django默认配置下会自动寻找 app 下的模板，但是 `django.contrib.auth` 这个是官方提供的，我们修改这个 app 不太容易，我们可以建立一个公用的模板文件夹。

#### 3.1 添加一个公用的放模板的文件夹

Django 1.8 及以上的版本 `settings.py`，修改 `TEMPLATES` 中的 `DIRS`

```
TEMPLATES = [
    {
        ...
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        ...
    },
]
```

Django 1.7 及以下的版本，修改 `TEMPLATE_DIRS`：

```
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates'),
)
```

注：如果是旧的项目已经存在公用的，可以不添加上面的目录

这个文件夹需要我们手工创建出来，创建后如下：









```
tu@mac ~/workspace/dj19/prj/zqxt_auth $ pwd
/Users/tu/workspace/dj19/prj/zqxt_auth

tu@mac ~/workspace/dj19/prj/zqxt_auth $ tree .
.
├── db.sqlite3
├── manage.py
├── templates # 手工创建的模板文件
└── zqxt_auth
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

2 directories, 6 files

### 3.2 模板文件

熟悉Django的同学知道，Django的后台是有登陆，重设密码的功能，那么在 `django.contrib.admin` 中应该是有相应的模板文件的，一找，果然有，[查看链接](#)。

|   |  |
|---|--|
| Tag: 1.9 ▾ <a href="#">django</a> / <a href="#">django</a> / <a href="#">contrib</a> / <a href="#">admin</a> / <a href="#">templates</a> / <a href="#">registration</a> / |  |
| scop committed with timgraham [1.9.x] Fixed #25565 -- Removed action="" from admin forms (invalid i... <a href="#">...</a>  |  |
| ..  |  |
|  <a href="#">logged_out.html</a>   | Fixed #18037 -- Changed behaviour of url and ssi template tags to the... |
|  <a href="#">password_change_done.html</a>   | Fixed #18511 -- Cleaned up admin password reset template titles.         |
|  <a href="#">password_change_form.html</a>   | [1.9.x] Fixed #25565 -- Removed action="" from admin forms (invalid i... |
|  <a href="#">password_reset_complete...</a>  | Fixed #18511 -- Cleaned up admin password reset template titles.         |
|  <a href="#">password_reset_confirm.html</a>   | [1.9.x] Fixed #25565 -- Removed action="" from admin forms (invalid i... |
|  <a href="#">password_reset_done.html</a>  | Fixed #23793 -- Clarified password reset messages.                       |
|  <a href="#">password_reset_email.html</a>   | Fixed #14881 -- Modified password reset to work with a non-integer Us... |
|  <a href="#">password_reset_form.html</a>  | [1.9.x] Fixed #25565 -- Removed action="" from admin forms (invalid i... |

我们把这些文件拷贝出来。

需要注意的是，有人已经按照类似的方法，做成了一个包了，地址在这里：<https://github.com/mishbahr/django-users2>

这个包比较完善了，我们没必要重新发明轮子，上面的示例只是让你明白，这个包其实是由官方的 `django.contrib.auth` 改进后做出来的。

四，用 **django-users2** 这个包来实现登陆注册及找回密码等功能。

**django-users2 这个包在 Django 1.5 - Django 1.9 中使用都没有问题。**

#### 4.1 安装

```
pip install django-users2
```

#### 4.2 把 users 这个 app 加入到 INSTALLED\_APPS

```
INSTALLED_APPS = (
    ...
    'django.contrib.auth',
    'django.contrib.sites',
    'users',
    ...
)
```

```
AUTH_USER_MODEL = 'users.User'
```

`AUTH_USER_MODEL` 是替换成自定义的用户认证。[参考这里](#)

#### 4.3 修改 urls.py

```
urlpatterns = patterns('',
    ...
    url(r'^accounts/', include('users.urls')),
    ...
)
```

)

#### 4.4 同步数据，创建相应的表

```
python manage.py syncdb
```

Django 1.7 及以上

```
python manage.py makemigrations
```

```
python manage.py migrate
```

#### 4.5 配置登陆注册的一些选项，找回密码时发邮件的邮箱

下面的代码加在 `settings.py` 最后面

```
USERS_REGISTRATION_OPEN = True

USERS_VERIFY_EMAIL = True

USERS_AUTO_LOGIN_ON_ACTIVATION = True

USERS_EMAIL_CONFIRMATION_TIMEOUT_DAYS = 3

# Specifies minimum length for passwords:
USERS_PASSWORD_MIN_LENGTH = 5

# Specifies maximum length for passwords:
USERS_PASSWORD_MAX_LENGTH = None

# the complexity validator, checks the password strength
USERS_CHECK_PASSWORD_COMPLEXITY = True

USERS_SPAM_PROTECTION = False # important!

# -----
# Email
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL_USE_TLS = False
EMAIL_HOST = 'smtp.tuweizhong.com'
EMAIL_PORT = 25
EMAIL_HOST_USER = 'mail@tuweizhong.com'
EMAIL_HOST_PASSWORD = 'xxxx'
DEFAULT_FROM_EMAIL = 'mail@tuweizhong.com'
# -----
```

这样登陆注册和找回密码功能应该就没有问题了。

[小额赞助，支持作者！](#)

[« Django 多数据库联用](#)

[Django 缓存系统 »](#)

 CODING

零成本开启敏捷开发 五人以下团队免费

免费体验

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 缓存系统

[« Django 用户注册系统](#)  
[Django 生成静态页面 »](#)

Django 官方关于cache的介绍: <https://docs.djangoproject.com/en/dev/topics/cache/>

Django 是动态网站, 一般来说需要实时地生成访问的网页, 展示给访问者, 这样, 内容可以随时变化, 但是从数据库读多次把所需要的数据取出来, 要比从内存或者硬盘等一次读出来 付出的成本大很多。

缓存系统工作原理:

对于给定的网址, 尝试从缓存中找到网址, 如果页面在缓存中, 直接返回缓存的页面, 如果缓存中没有, 一系列操作 (比如查数据库) 后, 保存生成的页面内容到缓存系统以供下一次使用, 然后返回生成的页面内容。

Django settings 中 cache 默认为

```
{
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
    }
}
```

也就是默认利用本地的内存来当缓存, 速度很快。当然可能出来内存不够用的情况, 其它的一些内建可用的 Backends 有

```
'django.core.cache.backends.db.DatabaseCache'
'django.core.cache.backends.dummy.DummyCache'
'django.core.cache.backends.filebased.FileBasedCache'
'django.core.cache.backends.locmem.LocMemCache'
'django.core.cache.backends.memcached.MemcachedCache'
'django.core.cache.backends.memcached.PyLibMCCache'
```

在 github 上也有用 redis 做 Django 的缓存系统的开源项目: <https://github.com/niwibe/django-redis>

利用文件系统来缓存:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': '/var/tmp/django_cache',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    }
}
```

利用数据库来缓存, 利用命令创建相应的表: `python manage.py createcachetable cache_table_name`

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'cache_table_name',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 2000
        }
    }
}
```

```
}
```

下面用一些实例来说明如何使用 Django 缓存系统

一般来说我们用 Django 来搭建一个网站，要用到数据库等。

```
from django.shortcuts import render
def index(request):
    # 读取数据库等 并渲染到网页
    # 数据库获取的结果保存到 queryset 中
    return render(request, 'index.html', {'queryset':queryset})
```

像这样每次访问都要读取数据库，一般的小网站没什么问题，当访问量非常大的时候，就会有很多次的数据库查询，肯定会造成访问速度变慢，服务器资源占用较多等问题。

```
from django.shortcuts import render
from django.views.decorators.cache import cache_page

@cache_page(60 * 15) # 秒数，这里指缓存 15 分钟，不直接写900是为了提高可读性
def index(request):
    # 读取数据库等 并渲染到网页
    return render(request, 'index.html', {'queryset':queryset})
```

当使用了cache后，访问情况变成了如下：

```
访问一个网址时，尝试从 cache 中找有没有缓存内容
如果网页在缓存中显示缓存内容，否则生成访问的页面，保存在缓存中以便下次使用，显示缓存的页面。
given a URL, try finding that page in the cache
if the page is in the cache:
    return the cached page
else:
    generate the page
    save the generated page in the cache (for next time)
    return the generated page
```

**Memcached** 是目前 Django 可用的最快的缓存

另外，Django 还可以共享缓存。

[小额赞助，支持作者！](#)

[« Django 用户注册系统](#)

[Django 生成静态页面 »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云ECS](#)

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 缓存系统

[« Django 用户注册系统](#)  
[Django 生成静态页面 »](#)

Django 官方关于cache的介绍: <https://docs.djangoproject.com/en/dev/topics/cache/>

Django 是动态网站，一般来说需要实时地生成访问的网页，展示给访问者，这样，内容可以随时变化，但是从数据库读多次把所需要的数据取出来，要比从内存或者硬盘等一次读出来 付出的成本大很多。

缓存系统工作原理：

对于给定的网址，尝试从缓存中找到网址，如果页面在缓存中，直接返回缓存的页面，如果缓存中没有，一系列操作（比如查数据库）后，保存生成的页面内容到缓存系统以供下一次使用，然后返回生成的页面内容。

**Django settings 中 cache 默认为**

```
{
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
    }
}
```

也就是默认利用本地的内存来当缓存，速度很快。当然可能出来内存不够用的情况，其它的一些内建可用的 Backends 有

```
'django.core.cache.backends.db.DatabaseCache'
'django.core.cache.backends.dummy.DummyCache'
'django.core.cache.backends.filebased.FileBasedCache'
'django.core.cache.backends.locmem.LocMemCache'
'django.core.cache.backends.memcached.MemcachedCache'
'django.core.cache.backends.memcached.PyLibMCCache'
```

在 [github](#) 上也有用 redis 做 Django 的缓存系统的开源项目: <https://github.com/niwibe/django-redis>

利用文件系统来缓存：

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': '/var/tmp/django_cache',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    }
}
```

```
}
```

利用数据库来缓存，利用命令创建相应的表：`python manage.py createcachetable cache_table_name`

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'cache_table_name',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 2000
        }
    }
}
```

下面用一些实例来说明如何使用 Django 缓存系统

一般来说我们用 Django 来搭建一个网站，要用到数据库等。

```
from django.shortcuts import render
def index(request):
    # 读取数据库等 并渲染到网页
    # 数据库获取的结果保存到 queryset 中
    return render(request, 'index.html', {'queryset':queryset})
```

像这样每次访问都要读取数据库，一般的小网站没什么问题，当访问量非常大的时候，就会有很多次数据库查询，肯定会造成访问速度变慢，服务器资源占用较多等问题。

```
from django.shortcuts import render
from django.views.decorators.cache import cache_page

@cache_page(60 * 15) # 秒数，这里指缓存 15 分钟，不直接写900是为了提高可读性
def index(request):
    # 读取数据库等 并渲染到网页
    return render(request, 'index.html', {'queryset':queryset})
```

当使用了cache后，访问情况变成了如下：

```
访问一个网址时，尝试从 cache 中找有没有缓存内容
如果网页在缓存中显示缓存内容，否则生成访问的页面，保存在缓存中以便下次使用，显示缓存的页面。
given a URL, try finding that page in the cache
if the page is in the cache:
    return the cached page
else:
    generate the page
    save the generated page in the cache (for next time)
    return the generated page
```

**Memcached** 是目前 Django 可用的最快的缓存

另外，Django 还可以共享缓存。

[小额赞助，支持作者！](#)

[«Django 用户注册系统](#)

[Django 生成静态页面»](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 生成静态页面

[« Django 缓存系统](#)  
[Django 安全 »](#)

如果网站的流量过大, 每次访问时都动态生成, 执行SQL语句, 消耗大量服务器资源, 这时候可以考虑生成静态页面。

生成静态很简单, 下面是一个例子:

只要在views.py中这样写就行了

```
from django.shortcuts import render
from django.template.loader import render_to_string
import os

def my_view(request):
    context = {'some_key': 'some_value'}

    static_html = '/path/to/static.html'

    if not os.path.exists(static_html):
        content = render_to_string('template.html', context)
        with open(static_html, 'w') as static_file:
            static_file.write(content)

    return render(request, static_html)
```

上面的例子中, 当用户访问时, 如果判断没有静态页面就自动生成静态页面, 然后返回静态文件, 当文件存在的时候就不再次生成。

也可以用一个文件夹, 比如在project下建一个 static\_html 文件夹, 把生成的静态文件都放里面, 让用户像访问静态文件那样访问页面。

## 更佳办法

但是一般情况下都不需要生成静态页面, 因为Django有缓存功能, 使用 [Django Cache\(缓存\)](#) 就相当于生成静态页面, 而且还有自动更新的功能, 比如30分钟刷新一下页面内容。

## 用Django管理静态网站内容

如果服务器上不支持Django环境, 你可以在本地上搭建一个Django环境, 然后生成静态页面, 把这些页面放到不支持Django的服务器上去, 在本地更新, 然后上传到服务器, 用Django来管理和更新网站的内容, 也是一个不错的做法, 还可以更安全, 听说有很多黑客都是这么做的。

[小额赞助, 支持作者!](#)  
[« Django 缓存系统](#)  
[Django 安全 »](#)

↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习,测试和培训。实例可能为了更容易理解而简化。我们一直对教程,参考手册,在线实例保持修订,但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时,代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的,对任何法律问题及风险不承担任何责任。版权所有,保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点,托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的,自强学堂网站源代码免费下载](#)

## Django 生成静态页面

[« Django 缓存系统](#)  
[Django 安全 »](#)

如果网站的流量过大,每次访问时都动态生成,执行SQL语句,消耗大量服务器资源,这时候可以考虑生成静态页面。

生成静态很简单,下面是一个例子:

只要在views.py中这样写就行了

```
from django.shortcuts import render
from django.template.loader import render_to_string
import os

def my_view(request):
    context = {'some_key': 'some_value'}

    static_html = '/path/to/static.html'

    if not os.path.exists(static_html):
        content = render_to_string('template.html', context)
        with open(static_html, 'w') as static_file:
            static_file.write(content)

    return render(request, static_html)
```

上面的例子中,当用户访问时,如果判断没有静态页面就自动生成静态页面,然后返回静态文件,当文件存在的时候

就不再次生成。

也可以用一个文件夹，比如在project下建一个 static\_html 文件夹，把生成的静态文件都放里面，让用户像访问静态文件那样访问页面。

## 更佳办法

但是一般情况下都不需要生成静态页面，因为Django 有缓存功能，使用 **Django Cache(缓存)** 就相当于生成静态页面，而且还有自动更新的功能，比如30分钟刷新一下页面内容。

## 用Django管理静态网站内容

如果服务器上不支持Django环境，你可以在本地搭建一个Django环境，然后生成静态页面，把这些页面放到不支持Django 的服务器上去，在本地更新，然后上传到服务器，用Django来管理和更新网站的内容，也是一个不错的做法，还可以更安全，听说有很多黑客都是这么做的。

[小额赞助，支持作者！](#)

[« Django 缓存系统](#)

[Django 安全 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 安全

[« Django 生成静态页面](#)  
[Django 国际化 »](#)

本文介绍Django关于安全的一些特征, 包括如何使基于Django的网站的一些建议。

关于安全的官方文档: <https://docs.djangoproject.com/en/dev/#security>

官方文档包括以下几个方面:

- [Security overview](#)
- [Disclosed security issues in Django](#)
- [Clickjacking protection](#)
- [Cross Site Request Forgery protection](#)
- [Cryptographic signing](#)
- [Security Middleware](#)

Django表单用在模板中的时候我们会加一句 `{% csrf_token %}`

[小额赞助, 支持作者!](#)  
[« Django 生成静态页面](#)  
[Django 国际化 »](#)

↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云ECS](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 安全

[« Django 生成静态页面](#)  
[Django 国际化 »](#)

本文介绍Django关于安全的一些特征, 包括如何使基于Django的网站的一些建议。

关于安全的官方文档: <https://docs.djangoproject.com/en/dev/#security>

官方文档包括以下几个方面:

- [Security overview](#)
- [Disclosed security issues in Django](#)
- [Clickjacking protection](#)
- [Cross Site Request Forgery protection](#)
- [Cryptographic signing](#)
- [Security Middleware](#)

Django表单用在模板中的时候我们会加一句 `{% csrf_token %}`

[小额赞助, 支持作者!](#)  
[« Django 生成静态页面](#)  
[Django 国际化 »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 国际化

« [Django 安全](#)  
[Django session](#) »

Django 支持国际化, 多语言。Django的国际化是默认开启的, 如果您不需要国际化支持, 那么您可以在您的设置文件中设置 `USE_I18N = False`, 那么Django会进行一些优化, 不加载国际化支持机制。

NOTE: 18表示Internationalization这个单词首字母I和结尾字母N之间的字母有18个。I18N就是Internationalization (国际化) 的意思。

Django 完全支持文本翻译, 日期时间数字格式和时区。

本质上讲, Django做了两件事:

1. 它允许开发者指定要翻译的字符串
2. Django根据特定的访问者的偏好设置 进行调用相应的翻译文本。

一, 开启国际化的支持, 需要在`settings.py`文件中设置

```
MIDDLEWARE_CLASSES = (
    ...
    'django.middleware.locale.LocaleMiddleware',
)
```

```
LANGUAGE_CODE = 'en'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

```
LANGUAGES = (
    ('en', ('English')),
    ('zh-cn', ('中文简体')),
    ('zh-tw', ('中文繁體')),
)
```

```
#翻译文件所在目录, 需要手工创建
LOCALE_PATHS = (
    os.path.join(BASE_DIR, 'locale'),
)
```

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    "django.core.context_processors.i18n",
)
```

**注意: Django 1.9 及以上版本中, 语言的代码发生变化(详情链接: [github](#), [django ticket](#), 如下**

```
LANGUAGES = (
    ('en', ('English')),
    ('zh-hans', ('中文简体')),
)
```

```
) ('zh-hant', ('中文繁體')),
```

二，生成需要翻译的文件（Django 1.8及以下的版本）：

```
python manage.py makemessages -l zh-cn
python manage.py makemessages -l zh-tw
```

Django 1.9 及以上版本要改成

```
python manage.py makemessages -l zh_hans
python manage.py makemessages -l zh_hant
```

三，手工翻译 locale 中的 django.po

此处省去500字

...

```
#: .\tutorial\models.py:23
msgid "created at"
msgstr "创建于"
```

```
#: .\tutorial\models.py:24
msgid "updated at"
msgstr "更新于"
```

...

此处省去几百字

四，编译一下，这样翻译才会生效

```
python manage.py compilemessages
```

如果翻译不生效，请检查你的语言包的文件夹是不是有中划线，请用下划线代替它。

比如 `zh-hans` 改成 `zh_hans`（但是要注意 `settings.py` 中要用中划线，不要也改了，就这一句话，你可能会浪费几个小时或几天）

Django 官方教程：[internationalization-and-localization](#)

[小额赞助，支持作者！](#)

[« Django 安全](#)

[Django session »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 国际化

[« Django 安全  
Django session »](#)

Django 支持国际化，多语言。Django的国际化是默认开启的，如果您不需要国际化支持，那么您可以在您的设置文件中设置 `USE_I18N = False`，那么Django会进行一些优化，不加载国际化支持机制。

NOTE: 18表示Internationalization这个单词首字母I和结尾字母N之间的字母有18个。I18N就是Internationalization（国际化）的意思。

Django 完全支持文本翻译，日期时间数字格式和时区。

本质上讲，**Django**做了两件事：

1. 它允许开发者指定要翻译的字符串
2. Django根据特定的访问者的偏好设置 进行调用相应的翻译文本。

一，开启国际化的支持，需要在`settings.py`文件中设置

```
MIDDLEWARE_CLASSES = (  
    ...  
    'django.middleware.locale.LocaleMiddleware',  
)
```

```
LANGUAGE_CODE = 'en'  
TIME_ZONE = 'UTC'  
USE_I18N = True  
USE_L10N = True  
USE_TZ = True
```

```
LANGUAGES = (  
    ('en', ('English')),  
    ('zh-cn', ('中文简体')),  
    ('zh-tw', ('中文繁體')),
```



```
)

#翻译文件所在目录，需要手工创建
LOCALE_PATHS = (
    os.path.join(BASE_DIR, 'locale'),
)

TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    "django.core.context_processors.i18n",
)
```

**注意：Django 1.9 及以上版本中，语言的代码发生变化(详情链接：[github](#), [django ticket](#), 如下**

```
LANGUAGES = (
    ('en', ('English')),
    ('zh-hans', ('中文简体')),
    ('zh-hant', ('中文繁體')),
)
```

二，生成需要翻译的文件（Django 1.8及以下的版本）：

```
python manage.py makemessages -l zh-cn
python manage.py makemessages -l zh-tw
```

Django 1.9 及以上版本要改成

```
python manage.py makemessages -l zh_hans
python manage.py makemessages -l zh_hant
```

三，手工翻译 `locale` 中的 `django.po`

此处省去500字  
...

```
#: ./tutorial\models.py:23
msgid "created at"
msgstr "创建于"
```

```
#: ./tutorial\models.py:24
msgid "updated at"
msgstr "更新于"
```

...  
此处省去几百字

四，编译一下，这样翻译才会生效

```
python manage.py compilemessages
```

如果翻译不生效，请检查你的语言包的文件夹是不是有 中划线，请用下划线代替它。

比如 `zh-hans` 改成 `zh_hans`（但是要注意 `settings.py` 中要用 中划线，不要也改了，就这一句话，你可能会浪费几个小时或几天）

Django 官方教程：[internationalization-and-localization](#)

[小额赞助，支持作者！](#)

[« Django 安全](#)

[Django session »](#)

 CODING  
Cloud Development

**零成本开启敏捷开发 五人以下团队免费**

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django session

« [Django 国际化](#)  
[Django传递数据给JS](#) »

Django完全支持也匿名会话,简单说就是使用跨网页之间可以进行通讯,比如显示用户名,用户是否已经发表评论。  
session框架让你存储和获取访问者的数据信息,这些信息保存在服务器上(默认是数据库中),以 cookies 的方式发送和获取一个包含 session ID的值,并不是用cookies传递数据本身。

## 启用session

编辑settings.py中的一些配置

MIDDLEWARE\_CLASSES 确保其中包含以下内容

```
'django.contrib.sessions.middleware.SessionMiddleware',
```

INSTALLED\_APPS 是包含

```
'django.contrib.sessions',
```

这些是默认启用的。如果你不用的话,也可以关掉这个以节省一点服务器的开销。

提示: [您也可以配置使用比如 cache 来存储 session](#)

## 在视图中使用 session

request.session 可以在视图中任何地方使用,它类似于python中的字典

session 默认有效时间为两周,可以在 settings.py 中修改默认值: [参见这里](#)

```
# 创建或修改 session:
request.session[key] = value
# 获取 session:
request.session.get(key,default=None)
# 删除 session
del request.session[key] # 不存在时报错
```

## session 例子

比如写一个不让用户评论两次的应用:

```
from django.http import HttpResponse

def post_comment(request, new_comment):
    if request.session.get('has_commented', False):
        return HttpResponse("You've already commented.")
    c = comments.Comment(comment=new_comment)
    c.save()
    request.session['has_commented'] = True
    return HttpResponse('Thanks for your comment!')
```

一个简化的登陆认证:

```
def login(request):
    m = Member.objects.get(username=request.POST['username'])
    if m.password == request.POST['password']:
        request.session['member_id'] = m.id
        return HttpResponse("You're logged in.")
    else:
        return HttpResponse("Your username and password didn't match.")

def logout(request):
    try:
        del request.session['member_id']
    except KeyError:
        pass
    return HttpResponse("You're logged out.")
```

当登陆时验证用户名和密码，并保存用户id在 session 中，这样就可以在视图中用 request.session['member\_id']来检查用户是否登陆，当退出的时候，删除掉它。

[小额赞助，支持作者！](#)

[« Django 国际化](#)

[Django传递数据给JS »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

Django session

# Django session

[« Django 国际化](#)  
[Django传递数据给JS »](#)

Django完全支持匿名会话，简单说就是使用跨网页之间可以进行通讯，比如显示用户名，用户是否已经发表评论。  
session框架让你存储和获取访问者的数据信息，这些信息保存在服务器上（默认是数据库中），以 cookies 的方式发送和获取一个包含 session ID的值，并不是用cookies传递数据本身。

## 启用session

编辑settings.py中的一些配置

MIDDLEWARE\_CLASSES 确保其中包含以下内容

```
'django.contrib.sessions.middleware.SessionMiddleware',
```

INSTALLED\_APPS 是包含

```
'django.contrib.sessions',
```

这些是默认启用的。如果你不用的话，也可以关掉这个以节省一点服务器的开销。

提示: [您也可以配置使用比如 cache 来存储 session](#)

## 在视图中使用 session

request.session 可以在视图中任何地方使用，它类似于python中的字典

session 默认有效时间为两周，可以在 settings.py 中修改默认值: [参见这里](#)

```
# 创建或修改 session:
request.session[key] = value
# 获取 session:
request.session.get(key, default=None)
# 删除 session
del request.session[key] # 不存在时报错
```

## session 例子

比如写一个不让用户评论两次的应用:

```
from django.http import HttpResponseRedirect

def post_comment(request, new_comment):
    if request.session.get('has_commented', False):
        return HttpResponseRedirect("You've already commented.")
    c = comments.Comment(comment=new_comment)
    c.save()
    request.session['has_commented'] = True
    return HttpResponseRedirect("Thanks for your comment!")
```

一个简化的登陆认证:

```
def login(request):
    m = Member.objects.get(username=request.POST['username'])
    if m.password == request.POST['password']:
        request.session['member_id'] = m.id
        return HttpResponseRedirect("You're logged in.")
    else:
        return HttpResponseRedirect("Your username and password didn't match.")
```

```
def logout(request):  
    try:  
        del request.session['member_id']  
    except KeyError:  
        pass  
    return HttpResponseRedirect("You're logged out.")
```

当登陆时验证用户名和密码，并保存用户id在 session 中，这样就可以在视图中用 request.session['member\_id']来检查用户是否登陆，当退出的时候，删除掉它。

[小额赞助，支持作者！](#)

[«Django 国际化](#)

[Django传递数据给JS»](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django传递数据给JS

[« Django session](#)  
[Django Ajax »](#)

有时候我们想**把一个 list 或 dict等 JSON对象 传到网页的 javascript**, 用 JS 进行处理, 比如用 js 将数据可视化显示到网页上。

请注意: 如果不需要处理, 直接显示到网页上, 用Django模板就可以了, 请看前面的教程。

这里讲述两种方法:

一, 页面加载完成后, 在页面上操作, 在页面上通过 **ajax** 方法得到新的数据 (**再向服务器发送一次请求**) 并显示在网页上, 这种情况适用于页面不刷新的情况下, 动态加载一些内容。比如用户输入一个值或者点击某个地方, 动态地把相应内容显示在网页上。

这种请问详见 [Django Ajax](#) 一节的内容。

二, 直接在视图函数 (**views.py**中的函数) 中将 JSON对象 和网页其它内容一起传递到Django模板 (**一次性地渲染, 还是同一次请求**)。

请看下面的示例:

views.py

```
from __future__ import unicode_literals
from django.shortcuts import render

def home(request):
    List = ['自强学堂', '渲染Json到模板']
    return render(request, 'home.html', {'List': List})
```

home.html 中的一部分

```
<script type="text/javascript">
    var List = {{ List }};
    alert(List);
</script>
```

需要注意的是, 我们如果直接这么做, **传递到 js 的时候, 网页的内容会被转义, 得到的格式会报错**。

访问时会得到 **Uncaught SyntaxError: Unexpected token ILLEGAL**

需要注意两点:

1. 视图函数中的字典或列表要用 **json.dumps()**处理。
2. 在模板上要加 **safe 过滤器**。

views.py

```
# -*- coding: utf-8 -*-
```

```

from __future__ import unicode_literals

import json
from django.shortcuts import render

def home(request):
    List = ['自强学堂', '渲染Json到模板']
    Dict = {'site': '自强学堂', 'author': '涂伟忠'}
    return render(request, 'home.html', {
        'List': json.dumps(List),
        'Dict': json.dumps(Dict)
    })

```

**home.html** 只给出了 **js** 核心部分:

```

//列表
var List = {{ List|safe }};
//字典
var Dict = {{ Dict|safe }};

```

如果你对 **js** 比较熟悉，到此为止，后面的不用看了。

如果不太熟悉，可以参考下面的更详细的代码。

**html** 完全代码及完整代码下载（最后面）：

```

<!DOCTYPE html>
<html>
<head>
<title>欢迎光临 自强学堂! </title>
<script src="http://apps.bdimg.com/libs/jquery/1.10.2/jquery.min.js"></script>
</head>
<body>
<div id="list"> 学习 </div>
<div id="dict"></div>
<script type="text/javascript">
    //列表
    var List = {{ List|safe }};

    //下面的代码把List的每一部分放到头部和尾部
    $('#list').prepend(List[0]);
    $('#list').append(List[1]);

    console.log('--- 遍历 List 方法 1 ---')
    for(i in List){
        console.log(i); // i为索引
    }

    console.log('--- 遍历 List 方法 2 ---')
    for (var i = List.length - 1; i >= 0; i--) {
        // 鼠标右键，审核元素，选择 console 可以看到输入的值。
        console.log(List[i]);
    };

    console.log('--- 同时遍历索引和内容，使用 jQuery.each() 方法 ---')
    $.each(List, function(index, item){
        console.log(index);
        console.log(item);
    });

    // 字典
    var Dict = {{ Dict|safe }};
    console.log("--- 两种字典的取值方式 ---")
    console.log(Dict['site']);
    console.log(Dict.author);

    console.log("--- 遍历字典 ---");

```



```
for(i in Dict) {
    console.log(i + Dict[i]); //注意, 此处 i 为键值
}
</script>
</body>
</html>
```

完整代码下载: [zqxt\\_json.zip](#) (基于Django 1.8, 注意settings.py文件和低版本可能不兼容, 其它部分相同)

[小额赞助, 支持作者!](#)

[« Django session](#)

[Django Ajax »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django传递数据给JS

[« Django session](#)

[Django Ajax »](#)

有时候我们想把一个 list 或 dict等 JSON对象 传到网页的 javascript, 用 JS 进行处理, 比如用 js 将数据可视化显示到网页上。

请注意: 如果不需要处理, 直接显示到网页上, 用Django模板就可以了, 请看前面的教程。

这里讲述两种方法：

一，页面加载完成后，在页面上操作，在页面上通过 ajax 方法得到新的数据（**再向服务器发送一次请求**）并显示在网页上，这种情况适用于页面不刷新的情况下，动态加载一些内容。比如用户输入一个值或者点击某个地方，动态地把相应内容显示在网页上。

这种请问详见 [Django Ajax](#) 一节的内容。

二，直接在视图函数（`views.py`中的函数）中将 JSON对象 和网页其它内容一起传递到Django模板（**一次性地渲染，还是同一次请求**）。

请看下面的示例：

`views.py`

```
from __future__ import unicode_literals
from django.shortcuts import render

def home(request):
    List = ['自强学堂', '渲染Json到模板']
    return render(request, 'home.html', {'List': List})
```

`home.html` 中的一部分

```
<script type="text/javascript">
    var List = {{ List }};
    alert(List);
</script>
```

需要注意的是，我们如果直接这么做，传递到 js 的时候，网页的内容会被转义，得到的格式会报错。

访问时会得到 **Uncaught SyntaxError: Unexpected token ILLEGAL**

需要注意两点：

1. 视图函数中的字典或列表要用 `json.dumps()`处理。
2. 在模板上要加 **safe 过滤器**。

`views.py`

```
# -*- coding: utf-8 -*-

from __future__ import unicode_literals

import json
from django.shortcuts import render

def home(request):
    List = ['自强学堂', '渲染Json到模板']
    Dict = {'site': '自强学堂', 'author': '涂伟忠'}
    return render(request, 'home.html', {
        'List': json.dumps(List),
        'Dict': json.dumps(Dict)
    })
```

`home.html` 只给出了 js 核心部分：

```
//列表
var List = {{ List|safe }};
//字典
var Dict = {{ Dict|safe }};
```

如果你对 js 比较熟悉，到此为止，后面的不用看了。

如果不太熟悉，可以参考下面的更详细的代码。

html 完全代码及完整代码下载（最后面）：

```
<!DOCTYPE html>
<html>
<head>
<title>欢迎光临 自强学堂! </title>
<script src="http://apps.bdimg.com/libs/jquery/1.10.2/jquery.min.js"></script>
</head>
<body>
<div id="list"> 学习 </div>
<div id='dict'></div>
<script type="text/javascript">
    //列表
    var List = [{ List|safe }];

    //下面的代码把List的每一部分放到头部和尾部
    $('#list').prepend(List[0]);
    $('#list').append(List[1]);

    console.log('--- 遍历 List 方法 1 ---')
    for(i in List){
        console.log(i);// i为索引
    }

    console.log('--- 遍历 List 方法 2 ---')
    for (var i = List.length - 1; i >= 0; i--) {
        // 鼠标右键，审核元素，选择 console 可以看到输入的值。
        console.log(List[i]);
    };

    console.log('--- 同时遍历索引和内容，使用 jQuery.each() 方法 ---')
    $.each(List, function(index, item){
        console.log(index);
        console.log(item);
    });

    // 字典
    var Dict = [{ Dict|safe }];
    console.log("--- 两种字典的取值方式 ---")
    console.log(Dict['site']);
    console.log(Dict.author);

    console.log("--- 遍历字典 ---");
    for(i in Dict) {
        console.log(i + Dict[i]);//注意，此处 i 为键值
    }
</script>
</body>
</html>
```

完整代码下载：[zqxt\\_json.zip](#)（基于Django 1.8，注意settings.py文件和低版本可能不兼容，其它部分相同）

[小额赞助，支持作者！](#)

[« Django session](#)

[Django Ajax »](#)

 **零成本开启敏捷开发 五人以下团队免费**

免费体验

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django Ajax

« [Django传递数据给JS](#)  
[Django Ajax CSRF 认证](#) »

有时候我们需要在不刷新的情况下载入一些内容, 在网页的基本知识中我们介绍了 [ajax](#) 技术。

在本文中讲解如何用 Django 来实现 不刷新网页的情况下加载一些内容。

由于用 jQuery 实现 ajax 比较简单, 所以我们用 jQuery 库来实现, 想用原生的 javascript 的同学可以参考: [ajax 教程](#), 下面也有例子提供下载。

本节有多个实例提供下载, 通过看代码可以更快的学习。

第一节, 源代码下载: [zqxt\\_ajax\\_1.zip](#)

这里用 [Django 表单](#) 第一节 中的一个例子, 我们要实现的是在不刷新的情况下显示计算结果到页面上。

修改 index.html 文件

```
<!DOCTYPE html>
<html>
<body>
<p>请输入两个数字</p>
<form action="/add/" method="get">
  a: <input type="text" id="a" name="a"> <br>
  b: <input type="text" id="b" name="b"> <br>
  <p>result: <span id='result'></span></p>
  <button type="button" id='sum'>提交</button>
</form>

<script src="http://apps.bdimg.com/libs/jquery/1.11.1/jquery.min.js"></script>
<script>
  $(document).ready(function() {
    $("#sum").click(function() {
      var a = $("#a").val();
      var b = $("#b").val();

      $.get("/add/", {'a':a, 'b':b}, function(ret) {
        $('#result').html(ret)
      });
    });
  });
</script>
</body>
</html>
```

在原来的基础上, 在一些元素上加了 id, 以便于获取值和绑定数据, 然后我们用了 jQuery.get() 方法, 并用 \$(selector).html() 方法将结果显示在页面上, 如下图:

备注: 关于请求头和 request.is\_ajax() 方法使用

views.py 中可以用 request.is\_ajax() 方法判断是否是 ajax 请求, 需要添加一个 HTTP 请求头:

原生 javascript:

```
xmlhttp.setRequestHeader("X-Requested-With", "XMLHttpRequest");
```

用 jQuery:

用 \$.ajax 方法代替 \$.get, 因为 \$.get 在 IE 中不会发送 ajax header


服务器端会将请求头的值全部大写, 中划线改成下划线, 并在非标准的头前面加上 HTTP\_, 这个过程可以认为相当于以下Python代码:

```
STANDARD_HEADERS = ['REFER', 'HOST', ...] # just for example

def handle_header(value):
    value = value.replace('-', '_').upper()

    if value in STANDARD_HEADERS:
        return value

    return 'HTTP_' + value
```

判断ajax方法, 以及原生的 javascript实现ajax的示例下载:  [zqxt\\_views\\_ajax.zip](#)

第二节, 源代码下载: [zqxt\\_ajax\\_list\\_dict.zip](#)

更复杂的例子, 传递一个数组或字典到网页, 由JS处理, 再显示出来。

views.py

```
from django.http import HttpResponse
import json

def ajax_list(request):
    a = range(100)
    return HttpResponse(json.dumps(a), content_type='application/json')

def ajax_dict(request):
    name_dict = {'twz': 'Love python and Django', 'zqxt': 'I am teaching Django'}
    return HttpResponse(json.dumps(name_dict), content_type='application/json')
```

Django 1.7 及以后的版本有更简单的方法 (使用 [JsonResponse](#)(官方文档)):

```
from django.http import JsonResponse

def ajax_list(request):
    a = range(100)
    return JsonResponse(a, safe=False)

def ajax_dict(request):
    name_dict = {'twz': 'Love python and Django', 'zqxt': 'I am teaching Django'}
    return JsonResponse(name_dict)
```

在 django 1.6 及以前的旧版本中可以自己写一个 JsonResponse 方法, 如下:

```
from django.http import HttpResponse

import json

class JsonResponse(HttpResponse):
    def __init__(self,
                  content={},
                  mimetype=None,
                  status=None,
                  content_type='application/json'):

        super(JsonResponse, self).__init__(
            json.dumps(content),
            mimetype=mimetype,
```

```
status=status,  
content_type=content_type)
```

写好后，我们在 `urls.py` 中添加以下两行：

```
url(r'^ajax_list/$', 'tools.views.ajax_list', name='ajax-list'),  
url(r'^ajax_dict/$', 'tools.views.ajax_dict', name='ajax-dict'),
```

打开开发服务器 `python manage.py runserver`

我们访问对应的网址会看到输出值：

django ajax dict 自强学堂示例

django ajax list 自强学堂示例

下一步就是在无刷新的情况下把内容加载到网页了，我们修改一下首页的模板 `index.html`

```
<!DOCTYPE html>  
<html>  
<body>  
<p>请输入两个数字</p>  
<form action="/add/" method="get">  
  a: <input type="text" id="a" name="a"> <br>  
  b: <input type="text" id="b" name="b"> <br>  
  <p>result: <span id='result'></span></p>  
  <button type="button" id='sum'>提交</button>  
</form>  
  
<div id="dict">Ajax 加载字典</div>  
<p id="dict_result"></p>  
  
<div id="list">Ajax 加载列表</div>  
<p id="list_result"></p>  
  
<script src="http://apps.bdimg.com/libs/jquery/1.11.1/jquery.min.js"></script>  
<script>  
  $(document).ready(function() {  
    // 求和 a + b  
    $("#sum").click(function() {  
      var a = $("#a").val();  
      var b = $("#b").val();  
  
      $.get("/add/", {'a':a, 'b':b}, function(ret) {  
        $('#result').html(ret);  
      })  
    });  
  });  
</script>
```

```

// 列表 list
$('#list').click(function(){
    $.getJSON('/ajax_list/',function(ret){
        //返回值 ret 在这里是一个列表
        for (var i = ret.length - 1; i >= 0; i--) {
            // 把 ret 的每一项显示在网页上
            $('#list_result').append(' ' + ret[i])
        };
    })
});

// 字典 dict
$('#dict').click(function(){
    $.getJSON('/ajax_dict/',function(ret){
        //返回值 ret 在这里是一个字典
        $('#dict_result').append(ret.twz + '<br>');
        // 也可以用 ret['twz']
    })
});
});
</script>
</body>
</html>

```

**技能提升：**getJSON中的写的对应网址，用 `urls.py` 中的 `name` 来获取是一个更好的方法!

**标签：**`{% url 'name' %}`

```

<script>
$(document).ready(function(){
    // 求和 a + b
    $("#sum").click(function(){
        var a = $("#a").val();
        var b = $("#b").val();

        $.get("{% url 'add' %}",{'a':a,'b':b}, function(ret){
            $('#result').html(ret);
        })
    });

    // 列表 list
    $('#list').click(function(){
        $.getJSON("{% url 'ajax-list' %}",function(ret){
            //返回值 ret 在这里是一个列表
            for (var i = ret.length - 1; i >= 0; i--) {
                // 把 ret 的每一项显示在网页上
                $('#list_result').append(' ' + ret[i])
            };
        })
    });

    // 字典 dict
    $('#dict').click(function(){
        $.getJSON("{% url 'ajax-dict' %}",function(ret){
            //返回值 ret 在这里是一个字典
            $('#dict_result').append(ret.twz + '<br>');
            // 也可以用 ret['twz']
        })
    });
});
</script>

```

这样做最大的好处就是在修改 `urls.py` 中的网址后，不用改模板中对应的网址。

**补充：**如果是一个复杂的 列表 或 字典，因为比如如下信息：

```
person_info_dict = [
```

```
[
  {"name": "xiaoming", "age": 20},
  {"name": "tuweizhong", "age": 24},
  {"name": "xiaoli", "age": 33},
]
```

这样我们遍历列表的时候，每次遍历得到一个字典，再用字典的方法去处理，当然有更简单的遍历方法：

用 **\$.each()** 方法代替 for 循环，html 代码 (jQuery)

```
$.getJSON('ajax-url-to-json', function(ret) {
  $.each(ret, function(i,item){
    // i 为索引, item为遍历值
  });
});
```

补充：如果 **ret** 是一个字典，**\$.each** 的参数有所不同，详见：<http://api.jquery.com/jquery.each/>

```
$.getJSON('ajax-get-a-dict', function(ret) {
  $.each(ret, function(key, value){
    // key 为字典的 key, value 为对应的值
  });
});
```

最后，附上一个返回图片并显示的 **ajax** 实例：



自强学堂 Django Ajax 返回图片

代码下载：[zqxt\\_ajax\\_pic.zip](#)

[小额赞助，支持作者！](#)

[« Django 传递数据给 JS](#)

[Django Ajax CSRF 认证 »](#)

 CODING  
LOWE DEVELOPMENT

零成本开启敏捷开发 五人以下团队免费

免费体验

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。



关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。  
自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)

 **零成本开启敏捷开发 五人以下团队免费**

免费体验

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django Ajax

« [Django传递数据给JS](#)  
[Django Ajax CSRF 认证](#) »

有时候我们需要在不刷新的情况下载入一些内容, 在网页的基本知识中我们介绍了 [ajax](#) 技术。

在本文中讲解如何用 Django 来实现 不刷新网页的情况下加载一些内容。

由于用 jQuery 实现 ajax 比较简单, 所以我们用 jQuery 库来实现, 想用原生的 javascript 的同学可以参考: [ajax 教程](#), 下面也有例子提供下载。

本节有多个实例提供下载, 通过看代码可以更快的学习。

第一节, 源代码下载: [📁 zqxt\\_ajax\\_1.zip](#)

这里用 [Django 表单](#) 第一节 中的一个例子, 我们要实现的是在不刷新的情况下显示计算结果到页面上。

修改 `index.html` 文件

```
<!DOCTYPE html>
<html>
<body>
<p>请输入两个数字</p>
<form action="/add/" method="get">
  a: <input type="text" id="a" name="a"> <br>
  b: <input type="text" id="b" name="b"> <br>
```

```

    <p>result: <span id='result'></span></p>
    <button type="button" id='sum'>提交</button>
</form>

<script src="http://apps.bdimg.com/libs/jquery/1.11.1/jquery.min.js"></script>
<script>
    $(document).ready(function() {
        $("#sum").click(function() {
            var a = $("#a").val();
            var b = $("#b").val();

            $.get("/add/", {'a':a, 'b':b}, function(ret) {
                $('#result').html(ret)
            })
        });
    });
</script>
</body>
</html>

```

在原来的基础上，在一些元素上加了 **id**，以便于获取值和绑定数据，然后我们用了 `jQuery.get()` 方法，并用 `$(selector).html()` 方法将结果显示在页面上，如下图：

**备注：**关于请求头和 `request.is_ajax()` 方法使用

`views.py` 中可以用 `request.is_ajax()` 方法判断是否是 `ajax` 请求，需要添加一个 `HTTP` 请求头：

原生 `javascript`:

```
xmlhttp.setRequestHeader("X-Requested-With", "XMLHttpRequest");
```

用 `jQuery`:

用 `$.ajax` 方法代替 `$.get`，因为 `$.get` 在 `IE` 中不会发送 `ajax` header

服务器端会将请求头的值全部大写，中划线改成下划线，并在非标准的头前面加上 `HTTP_`，这个过程可以认为相当于以下 `Python` 代码：

```


STANDARD_HEADERS = ['REFER', 'HOST', ...] # just for example

def handle_header(value):
    value = value.replace('-', '_').upper()

    if value in STANDARD_HEADERS:
        return value

    return 'HTTP_' + value

```

判断 `ajax` 方法，以及原生的 `javascript` 实现 `ajax` 的示例下载： [zqxt\\_views\\_ajax.zip](#)

**第二节，源代码下载：** [zqxt\\_ajax\\_list\\_dict.zip](#)

更复杂的例子，传递一个数组或字典到网页，由 `JS` 处理，再显示出来。

`views.py`

```

from django.http import HttpResponse
import json

def ajax_list(request):
    a = range(100)
    return HttpResponse(json.dumps(a), content_type='application/json')

```

```
def ajax_dict(request):
    name_dict = {'twz': 'Love python and Django', 'zqxt': 'I am teaching Django'}
    return HttpResponse(json.dumps(name_dict), content_type='application/json')
```

Django 1.7 及以后的版本有更简单的方法（使用 [JsonResponse](#)(官方文档)）:

```
from django.http import JsonResponse

def ajax_list(request):
    a = range(100)
    return JsonResponse(a, safe=False)

def ajax_dict(request):
    name_dict = {'twz': 'Love python and Django', 'zqxt': 'I am teaching Django'}
    return JsonResponse(name_dict)
```

在 django 1.6 及以前的旧版本中可以自己写一个 JsonResponse 方法，如下:

```
from django.http import HttpResponse

import json

class JsonResponse(HttpResponse):
    def __init__(self,
                  content={},
                  mimetype=None,
                  status=None,
                  content_type='application/json'):

        super(JsonResponse, self).__init__(
            json.dumps(content),
            mimetype=mimetype,
            status=status,
            content_type=content_type)
```

写好后，我们在 `urls.py` 中添加以下两行:

```
url(r'^ajax_list/$', 'tools.views.ajax_list', name='ajax-list'),
url(r'^ajax_dict/$', 'tools.views.ajax_dict', name='ajax-dict'),
```

打开开发服务器 `python manage.py runserver`

我们访问对应的网址会看到输出值:

django ajax dict 自强学堂示例

django ajax list 自强学堂示例

下一步就是在无刷新的情况下把内容加载到网页了，我们修改一下首页的模板 `index.html`

```

<!DOCTYPE html>
<html>
<body>
<p>请输入两个数字</p>
<form action="/add/" method="get">
  a: <input type="text" id="a" name="a"> <br>
  b: <input type="text" id="b" name="b"> <br>
  <p>result: <span id='result'></span></p>
  <button type="button" id='sum'>提交</button>
</form>

<div id="dict">Ajax 加载字典</div>
<p id="dict_result"></p>

<div id="list">Ajax 加载列表</div>
<p id="list_result"></p>

<script src="http://apps.bdimg.com/libs/jquery/1.11.1/jquery.min.js"></script>
<script>
  $(document).ready(function(){
    // 求和 a + b
    $("#sum").click(function(){
      var a = $("#a").val();
      var b = $("#b").val();

      $.get("/add/", {'a':a, 'b':b}, function(ret){
        $('#result').html(ret);
      })
    });

    // 列表 list
    $('#list').click(function(){
      $.getJSON('/ajax_list/', function(ret){
        //返回值 ret 在这里是一个列表
        for (var i = ret.length - 1; i >= 0; i--) {
          // 把 ret 的每一项显示在网页上
          $('#list_result').append(' ' + ret[i])
        }
      })
    })

    // 字典 dict
    $('#dict').click(function(){
      $.getJSON('/ajax_dict/', function(ret){
        //返回值 ret 在这里是一个字典
        $('#dict_result').append(ret.twz + '<br>');
        // 也可以用 ret['twz']
      })
    })
  });
</script>
</body>
</html>

```

**技能提升：**getJSON中的写的对应网址，用 `urls.py` 中的 `name` 来获取是一个更好的方法!

**标签：** `{% url 'name' %}`

```

<script>
  $(document).ready(function(){
    // 求和 a + b
    $("#sum").click(function(){
      var a = $("#a").val();
      var b = $("#b").val();

      $.get("{% url 'add' %}", {'a':a, 'b':b}, function(ret){
        $('#result').html(ret);
      })
    })
  });
</script>

```

```

});

// 列表 list
$('#list').click(function(){
    $.getJSON("{% url 'ajax-list' %}",function(ret){
        //返回值 ret 在这里是一个列表
        for (var i = ret.length - 1; i >= 0; i--) {
            // 把 ret 的每一项显示在网页上
            $('#list_result').append(' ' + ret[i])
        };
    })
})

// 字典 dict
$('#dict').click(function(){
    $.getJSON("{% url 'ajax-dict' %}",function(ret){
        //返回值 ret 在这里是一个字典
        $('#dict_result').append(ret.twz + '<br>');
        // 也可以用 ret['twz']
    })
})
});
</script>

```

这样做最大的好处就是在修改 `urls.py` 中的网址后，不用改模板中对应的网址。

**补充：**如果是一个复杂的 列表 或 字典，因为比如如下信息：

```

person_info_dict = [
    {"name":"xiaoming", "age":20},
    {"name":"tuweizhong", "age":24},
    {"name":"xiaoli", "age":33},
]

```

这样我们遍历列表的时候，每次遍历得到一个字典，再用字典的方法去处理，当然有更简单的遍历方法：

用 `$.each()` 方法代替 `for` 循环，html 代码 (jQuery)

```

$.getJSON('ajax-url-to-json', function(ret) {
    $.each(ret, function(i,item){
        // i 为索引, item为遍历值
    });
});

```

补充：如果 `ret` 是一个字典，`$.each` 的参数有所不同，详见：<http://api.jquery.com/jquery.each/>

```

$.getJSON('ajax-get-a-dict', function(ret) {
    $.each(ret, function(key, value){
        // key 为字典的 key, value 为对应的值
    });
});

```

最后，附上一个返回图片并显示的 **ajax** 实例：



请输入

color:

number:

result:



自强学堂 Django Ajax 返回图片

代码下载: [zqxt\\_ajax\\_pic.zip](#)

[小额赞助, 支持作者!](#)

[« Django传递数据给JS](#)

[Django Ajax CSRF 认证 »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django Ajax CSRF 认证

[« Django Ajax](#)  
[Django Sitemap 站点地图 »](#)

CSRF (Cross-site request forgery 跨站请求伪造, 也被称为“one click attack”或者 session riding, 通常缩写为 CSRF 或者 XSRF。是一种对网站的恶意利用。

## XSS

假如A网站有XSS漏洞, 访问A网站的攻击用户 用发帖的方式, 在标题或内容等地方植入js代码, 这些代码在某些场景下会被触发执行 (比如点回帖时)。

当A网站的其它用户点回帖后, js运行了, 这段js按道理可以做任何事, 比如将用户的cookie发到指定服务器 (攻击者所有), 这样攻击者就可以使用cookie假冒用户访问A网站了, 可以发帖, 删帖等。

## CSRF

背景知识: 浏览器在发送请求的时候, 会自动带上当前域名对应的cookie内容, 发送给服务端, 不管这个请求是来源A网站还是其它网站, 只要请求的是A网站的链接, 就会带上A网站的cookie。浏览器的同源策略并不能阻止CSRF攻击, 因为浏览器不会停止js发送请求到服务端, 只是在必要的时候拦截了响应的内容。或者说浏览器收到响应之前它不知道该不该拒绝。

攻击过程: 用户登陆A网站后, 攻击者自己开发一个B网站, 这个网站会通过js请求A网站, 比如用户点击了某个按钮, 就触发了js的执行。

预防:

### Double Submit Cookie

攻击者是利用cookie随着http请求发送的特性来攻击。但攻击都不知道 cookie里面是什么。

Django中是在表单中加一个隐藏的 csrfmiddlewaretoken, 在提交表单的时候, 会有 cookie 中的内容做比对, 一致则认为正常, 不一致则认为是攻击。由于每个用户的 token 不一样, B网站上的js代码无法猜出token内容, 对比必然失败, 所以可以起到防范作用。

### Synchronizer Token

和上面的类似, 但不使用 cookie, 服务端的数据库中保存一个 session\_csrf\_token, 表单提交后, 将表单中的 token 和 session 中的对比, 如果不一致则是攻击。

这个方法实施起来并不困难, 但它更安全一些, 因为网站即使有 xss 攻击, 也不会有泄露token的问题。

Django 中自带了 防止CSRF攻击的功能, 但是一些新手不知道如何使用, 给自己编程带来了麻烦。常常会出现下面 django csrf token missing or incorrect的错误。

GET 请求不需要 CSRF 认证, POST 请求需要正确认证才能得到正确的返回结果。一般在POST表单中加入 **{% csrf\_token %}**

```
<form method="POST" action="/post-url/">
    {% csrf_token %}

    <input name='zqxt' value="自强学堂学习Django技术">
</form>
```

如果使用Ajax调用的时候，就要麻烦一些。需要注意以下几点：

1. 在视图中使用 **render**（而不要使用 **render\_to\_response**）
2. 使用 jQuery 的 ajax 或者 post 之前 加入这个 js 代码：<http://www.ziqiangxuetang.com/media/django/csrf.js>

```
jQuery(document).ajaxSend(function(event, xhr, settings) {
    function getCookie(name) {
        var cookieValue = null;
        if (document.cookie && document.cookie != '') {
            var cookies = document.cookie.split(';');
            for (var i = 0; i < cookies.length; i++) {
                var cookie = jQuery.trim(cookies[i]);
                // Does this cookie string begin with the name we want?
                if (cookie.substring(0, name.length + 1) == (name + '=')) {
                    cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
                    break;
                }
            }
        }
        return cookieValue;
    }
    function sameOrigin(url) {
        // url could be relative or scheme relative or absolute
        var host = document.location.host; // host + port
        var protocol = document.location.protocol;
        var sr_origin = '//' + host;
        var origin = protocol + sr_origin;
        // Allow absolute or scheme relative URLs to same origin
        return (url == origin || url.slice(0, origin.length + 1) == origin + '/') ||
            (url == sr_origin || url.slice(0, sr_origin.length + 1) == sr_origin + '/') ||
            // or any other URL that isn't scheme relative or absolute i.e relative.
            !(/^(\/|\/http:|https:).*/.test(url));
    }
    function safeMethod(method) {
        return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
    }

    if (!safeMethod(settings.type) && sameOrigin(settings.url)) {
        xhr.setRequestHeader("X-CSRFToken", getCookie('csrftoken'));
    }
});
```

或者 更为优雅简洁的代码（不能写在 .js 中，要直接写在模板文件中）：

```
$.ajaxSetup({
    data: {csrfmiddlewaretoken: '{{ csrf_token }}' },
});
```

这样之后，就可以像原来一样的使用 `jQuery.ajax()` 和 `jQuery.post()` 了

最后，附上一个 Django Ajax CSRF 实例：[exam.zip](#)

参考：

What is a CSRF token and how does it work? <https://cloudunder.io/blog/csrf-token>

How CSRF Protection Works <https://kylebebak.github.io/post/csrf-protection>

[小额赞助，支持作者！](#)

[« Django Ajax](#)

[Django Sitemap 站点地图 »](#)



零成本开启敏捷开发 五人以下团队免费

免费体验



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django Ajax CSRF 认证

[« Django Ajax](#)

[Django Sitemap 站点地图 »](#)

CSRF (Cross-site request forgery跨站请求伪造，也被称为“one click attack”或者session riding，通常缩写为CSRF或者XSRF。是一种对网站的恶意利用。

## XSS

假如A网站有XSS漏洞，访问A网站的攻击用户 用发帖的方式，在标题或内容等地方植入js代码，这些代码在某些场景下会被触发执行（比如点回帖时）。

当A网站的其它用户点回帖后，js运行了，这段js按道理可以做任何事，比如将用户的cookie发到指定服务器（攻击者所有），这样攻击者就可以使用cookie假冒用户访问A网站了，可以发帖，删帖等。

## CSRF

背景知识：浏览器在发送请求的时候，会自动带上当前域名对应的cookie内容，发送给服务端，不管这个请求是来源A网站还是其它网站，只要请求的是A网站的链接，就会带上A网站的cookie。浏览器的同源策略并不能阻止CSRF攻击，因为浏览器不会停止js发送请求到服务端，只是在必要的时候拦截了响应的内容。或者说浏览器收到响应之前它不知道该不该拒绝。

攻击过程：用户登陆A网站后，攻击者自己开发一个B网站，这个网站会通过js请求A网站，比如用户点击了某个按钮，就触发了js的执

行。

预防：

### Double Submit Cookie

攻击者是利用cookie随着http请求发送的特性来攻击。但攻击都不知道 cookie里面是什么。

Django中是在表单中加一个隐藏的 csrfmiddlewaretoken，在提交表单的时候，会有 cookie 中的内容做比对，一致则认为正常，不一致则认为是攻击。由于每个用户的 token 不一样，B网站上的js代码无法猜出token内容，对比必然失败，所以可以起到防范作用。

### Synchronizer Token

和上面的类似，但不使用 cookie，服务端数据库中保存一个 session\_csrf\_token，表单提交后，将表单中的 token 和 session 中的对比，如果不一致则是攻击。

这个方法实施起来并不困难，但它更安全一些，因为网站即使有 xss 攻击，也不会有泄露token的问题。

Django 中自带了 防止CSRF攻击的功能，但是一些新手不知道如何使用，给自己编程带来了麻烦。常常会出现下面 django csrf token missing or incorrect的错误。

GET 请求不需要 CSRF 认证，POST 请求需要正确认证才能得到正确的返回结果。一般在POST表单中加入 **{% csrf\_token %}**

```
<form method="POST" action="/post-url/">
    {% csrf_token %}

    <input name='zqxt' value="自强学堂学习Django技术">
</form>
```

如果使用Ajax调用的时候，就要麻烦一些。需要注意以下几点：

1. 在视图中使用 **render**（而不要使用 **render\_to\_response**）
2. 使用 jQuery 的 ajax 或者 post 之前 加入这个 js 代码：<http://www.ziqiangxuetang.com/media/django/csrf.js>

```
jQuery(document).ajaxSend(function(event, xhr, settings) {
    function getCookie(name) {
        var cookieValue = null;
        if (document.cookie && document.cookie != '') {
            var cookies = document.cookie.split(';');
            for (var i = 0; i < cookies.length; i++) {
                var cookie = jQuery.trim(cookies[i]);
                // Does this cookie string begin with the name we want?
                if (cookie.substring(0, name.length + 1) == (name + '=')) {
                    cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
                    break;
                }
            }
        }
    }
    return cookieValue;
}

function sameOrigin(url) {
    // url could be relative or scheme relative or absolute
    var host = document.location.host; // host + port
    var protocol = document.location.protocol;
    var sr_origin = '//' + host;
    var origin = protocol + sr_origin;
    // Allow absolute or scheme relative URLs to same origin
    return (url == origin || url.slice(0, origin.length + 1) == origin + '/') ||
        (url == sr_origin || url.slice(0, sr_origin.length + 1) == sr_origin + '/') ||
        // or any other URL that isn't scheme relative or absolute i.e relative.
        !(/^(\:\/\/|http|https):.*\.test(url));
}

function safeMethod(method) {
    return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
}
```

```
    }  
  
    if (!safeMethod(settings.type) && sameOrigin(settings.url)) {  
        xhr.setRequestHeader("X-CSRFToken", getCookie('csrftoken'));  
    }  
});
```

或者 更为优雅简洁的代码（不能写在 .js 中，要直接写在模板文件中）：

```
$.ajaxSetup({  
    data: {csrfmiddlewaretoken: '{{ csrf_token }}' },  
});
```

这样之后，就可以像原来一样的使用 `jQuery.ajax()` 和 `jQuery.post()` 了

最后，附上一个 Django Ajax CSRF 实例： [exam.zip](#)

参考：

What is a CSRF token and how does it work? <https://cloudunder.io/blog/csrf-token>

How CSRF Protection Works <https://kylebebak.github.io/post/csrf-protection>

[小额赞助，支持作者！](#)

[« Django Ajax](#)

[Django Sitemap 站点地图 »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django Sitemap 站点地图

[« Django Ajax CSRF 认证](#)  
[Django ORM »](#)

Django 中自带了 `sitemap` 框架, 用来生成 `xml` 文件

Django sitemap 演示: <http://www.ziqiangxuetang.com/sitemap.xml>

sitemap 很重要, 可以用来通知搜索引擎页面的地址, 页面的重要性, 帮助站点得到比较好的收录。

开启 `sitemap` 功能的步骤

`settings.py` 文件中 `django.contrib.sitemaps` 和 `django.contrib.sites` 要在 `INSTALL_APPS` 中

```
INSTALLED_APPS = (  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'django.contrib.sites',  
    'django.contrib.sitemaps',  
    'django.contrib.redirects',  
  
    #####  
    #othther apps  
    #####  
)
```

**Django 1.7 及以前版本:**

`TEMPLATE_LOADERS` 中要加入 `'django.template.loaders.app_directories.Loader'`, 像这样:

```
TEMPLATE_LOADERS = (  
    'django.template.loaders.filesystem.Loader',  
    'django.template.loaders.app_directories.Loader',  
)
```

**Django 1.8 及以上版本**新加入了 `TEMPLATES` 设置, 其中 `APP_DIRS` 要为 `True`, 比如:

```
# NOTICE: code for Django 1.8, not work on Django 1.7 and below  
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [  
            os.path.join(BASE_DIR, 'templates').replace('\\', '/'),  
        ],  
        'APP_DIRS': True,  
    },  
]
```

然后在 `urls.py` 中如下配置:

```
from django.conf.urls import url  
from django.contrib.sitemaps import GenericSitemap  
from django.contrib.sitemaps.views import sitemap
```

```

from blog.models import Entry

sitemaps = {
    'blog': GenericSitemap({'queryset': Entry.objects.all(), 'date_field': 'pub_date'}, priority=0.6),
    # 如果还要加其它的可以模仿上面的
}

urlpatterns = [
    # some generic view using info_dict
    # ...

    # the sitemap
    url(r'^sitemap\.xml$', sitemap, {'sitemaps': sitemaps},
        name='django.contrib.sitemaps.views.sitemap'),
]

```

但是这样生成的 **sitemap**，如果网站内容太多就很慢，很耗费资源，可以采用分页的功能：

```

from django.conf.urls import url
from django.contrib.sitemaps import GenericSitemap
from django.contrib.sitemaps.views import sitemap

from blog.models import Entry

from django.contrib.sitemaps import views as sitemaps_views
from django.views.decorators.cache import cache_page

sitemaps = {
    'blog': GenericSitemap({'queryset': Entry.objects.all(), 'date_field': 'pub_date'}, priority=0.6),
    # 如果还要加其它的可以模仿上面的
}

urlpatterns = [
    url(r'^sitemap\.xml$',
        cache_page(86400)(sitemaps_views.index),
        {'sitemaps': sitemaps, 'sitemap_url_name': 'sitemaps'}),
    url(r'^sitemap-(?P<section>.)\.xml$',
        cache_page(86400)(sitemaps_views.sitemap),
        {'sitemaps': sitemaps, 'name': 'sitemaps'}),
]

```

这样就可以看到类似如下的 **sitemap**，如果本地测试访问 <http://localhost:8000/sitemap.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=2</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=3</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=4</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=5</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=6</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=7</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=8</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=9</loc></sitemap>
</sitemapindex>

```

查看了分页是实现了，但是全部显示成了 **?p=** 页面数，而且在百度站长平台上测试，发现这样的 **sitemap** 百度报错，于是看了下 **Django** 的源代码：

在这里 <https://github.com/django/django/blob/1.7.7/django/contrib/sitemaps/views.py>

于是对源代码作了修改，变成了本站的 **sitemap** 的样子，比 **?p=2** 这样更优雅

引入下面这个 比如是 **sitemap\_views.py**

```

import warnings
from functools import wraps

from django.contrib.sites.models import get_current_site
from django.core import urlresolvers
from django.core.paginator import EmptyPage, PageNotAnInteger
from django.http import Http404
from django.template.response import TemplateResponse
from django.utils import six

def x_robots_tag(func):
    @wraps(func)
    def inner(request, *args, **kwargs):
        response = func(request, *args, **kwargs)
        response['X-Robots-Tag'] = 'noindex, noodp, noarchive'
        return response
    return inner

@x_robots_tag
def index(request, sitemaps,
          template_name='sitemap_index.xml', content_type='application/xml',
          sitemap_url_name='django.contrib.sitemaps.views.sitemap',
          mimetype=None):

    if mimetype:
        warnings.warn("The mimetype keyword argument is deprecated, use "
                      "content_type instead", DeprecationWarning, stacklevel=2)
        content_type = mimetype

    req_protocol = 'https' if request.is_secure() else 'http'
    req_site = get_current_site(request)

    sites = []
    for section, site in sitemaps.items():
        if callable(site):
            site = site()
        protocol = req_protocol if site.protocol is None else site.protocol
        for page in range(1, site.paginator.num_pages + 1):
            sitemap_url = urlresolvers.reverse(
                sitemap_url_name, kwargs={'section': section, 'page': page})
            absolute_url = '%s://%s%s' % (protocol, req_site.domain, sitemap_url)
            sites.append(absolute_url)

    return TemplateResponse(request, template_name, {'sitemaps': sites},
                           content_type=content_type)

@x_robots_tag
def sitemap(request, sitemaps, section=None, page=1,
           template_name='sitemap.xml', content_type='application/xml',
           mimetype=None):

    if mimetype:
        warnings.warn("The mimetype keyword argument is deprecated, use "
                      "content_type instead", DeprecationWarning, stacklevel=2)
        content_type = mimetype

    req_protocol = 'https' if request.is_secure() else 'http'
    req_site = get_current_site(request)

    if section is not None:
        if section not in sitemaps:
            raise Http404("No sitemap available for section: %r" % section)
        maps = [sitemaps[section]]
    else:
        maps = list(six.itervalues(sitemaps))

    urls = []
    for site in maps:
        try:
            if callable(site):
                site = site()

```

```
urls.extend(site.get_urls(page=page, site=req_site,
                           protocol=req_protocol))
except EmptyPage:
    raise Http404("Page %s empty" % page)
except PageNotAnInteger:
    raise Http404("No page '%s'" % page)
return TemplateResponse(request, template_name, {'urlset': urls},
                        content_type=content_type)
```

如果还是不懂，可以下载附件查看：[zqxt\\_sitemap.zip](#)

更多参考：

官方文档：<https://docs.djangoproject.com/en/dev/ref/contrib/sitemaps/>

[小额赞助，支持作者！](#)

[« Django Ajax CSRF 认证](#)

[Django ORM »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django Sitemap 站点地图

[« Django Ajax CSRF 认证](#)

[Django ORM »](#)

Django 中自帶了 `sitemap` 框架，用来生成 `xml` 文件

Django `sitemap` 演示: <http://www.ziqiangxuetang.com/sitemap.xml>

`sitemap` 很重要，可以用来通知搜索引擎页面的地址，页面的重要性，帮助站点得到比较好的收录。

### 开启 `sitemap` 功能的步骤

`settings.py` 文件中 `django.contrib.sitemaps` 和 `django.contrib.sites` 要在 `INSTALL_APPS` 中

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites',
    'django.contrib.sitemaps',
    'django.contrib.redirects',

    #####
    #othther apps
    #####
)
```

**Django 1.7 及以前版本:**

`TEMPLATE_LOADERS` 中要加入 `'django.template.loaders.app_directories.Loader'`，像这样:

```
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader',
)
```

**Django 1.8 及以上版本**新加入了 `TEMPLATES` 设置，其中 `APP_DIRS` 要为 `True`，比如:

```
# NOTICE: code for Django 1.8, not work on Django 1.7 and below
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'templates').replace('\\', '/'),
        ],
        'APP_DIRS': True,
    },
]
```

然后在 `urls.py` 中如下配置:

```
from django.conf.urls import url
from django.contrib.sitemaps import GenericSitemap
from django.contrib.sitemaps.views import sitemap

from blog.models import Entry

sitemaps = {
    'blog': GenericSitemap({'queryset': Entry.objects.all(), 'date_field': 'pub_date'}, priority=0.6),
    # 如果还要加其它的可以模仿上面的
}

urlpatterns = [
    # some generic view using info_dict
    # ...

    # the sitemap
    url(r'^sitemap\.xml$', sitemap, {'sitemaps': sitemaps},
```



```

        name='django.contrib.sitemaps.views.sitemap'),
    ]

```

但是这样生成的 **sitemap**，如果网站内容太多就很慢，很耗费资源，可以采用分页的功能：

```

from django.conf.urls import url
from django.contrib.sitemaps import GenericSitemap
from django.contrib.sitemaps.views import sitemap

from blog.models import Entry

from django.contrib.sitemaps import views as sitemaps_views
from django.views.decorators.cache import cache_page

sitemaps = {
    'blog': GenericSitemap({'queryset': Entry.objects.all(), 'date_field': 'pub_date'}, priority=0.6),
    # 如果还要加其它的可以模仿上面的
}

urlpatterns = [
    url(r'^sitemap\.xml$',
        cache_page(86400)(sitemaps_views.index),
        {'sitemaps': sitemaps, 'sitemap_url_name': 'sitemaps'}),
    url(r'^sitemap-(?P<section>+)\.xml$',
        cache_page(86400)(sitemaps_views.sitemap),
        {'sitemaps': sitemaps, 'name': 'sitemaps'}),
]

```

这样就可以看到类似如下的 **sitemap**，如果本地测试访问 <http://localhost:8000/sitemap.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=2</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=3</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=4</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=5</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=6</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=7</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=8</loc></sitemap>
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.xml?p=9</loc></sitemap>
</sitemapindex>

```

查看了分页是实现了，但是全部显示成了 **?p=**页面数，而且在百度站长平台上测试，发现这样的 **sitemap** 百度报错，于是看了下 **Django** 的源代码：

在这里 <https://github.com/django/django/blob/1.7.7/django/contrib/sitemaps/views.py>

于是对源代码作了修改，变成了本站的 **sitemap** 的样子，比 **?p=2** 这样更优雅

引入下面这个 比如是 **sitemap\_views.py**

```

import warnings
from functools import wraps

from django.contrib.sites.models import get_current_site
from django.core import urlresolvers
from django.core.paginator import EmptyPage, PageNotAnInteger
from django.http import Http404
from django.template.response import TemplateResponse
from django.utils import six

def x_robots_tag(func):
    @wraps(func)
    def inner(request, *args, **kwargs):
        response = func(request, *args, **kwargs)

```

```

        response['X-Robots-Tag'] = 'noindex, noodp, noarchive'
    return response
return inner

@x_robots_tag
def index(request, sitemaps,
          template_name='sitemap_index.xml', content_type='application/xml',
          sitemap_url_name='django.contrib.sitemaps.views.sitemap',
          mimetype=None):

    if mimetype:
        warnings.warn("The mimetype keyword argument is deprecated, use "
                      "content_type instead", DeprecationWarning, stacklevel=2)
        content_type = mimetype

    req_protocol = 'https' if request.is_secure() else 'http'
    req_site = get_current_site(request)

    sites = []
    for section, site in sitemaps.items():
        if callable(site):
            site = site()
        protocol = req_protocol if site.protocol is None else site.protocol
        for page in range(1, site.paginator.num_pages + 1):
            sitemap_url = urlresolvers.reverse(
                sitemap_url_name, kwargs={'section': section, 'page': page})
            absolute_url = '%s://%s%s' % (protocol, req_site.domain, sitemap_url)
            sites.append(absolute_url)

    return TemplateResponse(request, template_name, {'sitemaps': sites},
                           content_type=content_type)

@x_robots_tag
def sitemap(request, sitemaps, section=None, page=1,
            template_name='sitemap.xml', content_type='application/xml',
            mimetype=None):

    if mimetype:
        warnings.warn("The mimetype keyword argument is deprecated, use "
                      "content_type instead", DeprecationWarning, stacklevel=2)
        content_type = mimetype

    req_protocol = 'https' if request.is_secure() else 'http'
    req_site = get_current_site(request)

    if section is not None:
        if section not in sitemaps:
            raise Http404("No sitemap available for section: %r" % section)
        maps = [sitemaps[section]]
    else:
        maps = list(six.itervalues(sitemaps))

    urls = []
    for site in maps:
        try:
            if callable(site):
                site = site()
            urls.extend(site.get_urls(page=page, site=req_site,
                                     protocol=req_protocol))
        except EmptyPage:
            raise Http404("Page %s empty" % page)
        except PageNotAnInteger:
            raise Http404("No page '%s'" % page)
    return TemplateResponse(request, template_name, {'urlset': urls},
                           content_type=content_type)

```

如果还是不懂，可以下载附件查看：[zqxt\\_sitemap.zip](#)

更多参考：

官方文档: <https://docs.djangoproject.com/en/dev/ref/contrib/sitemaps/>

小额赞助, 支持作者!

«Django Ajax CSRF 认证

Django ORM»



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django ORM

[« Django Sitemap 站点地图](#)  
[Django 通用视图 »](#)

本文介绍如何只使用Django的数据库。

Django 的数据库接口非常好用, 我们甚至不需要知道SQL语句如何书写, 就可以轻松地查询, 创建一些内容, 所以有时候想, 在其它的地方使用Django的 ORM呢? 它有这么丰富的 QuerySet API.

示例代码: [Django\\_DB.zip](#)

settings.py

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
SECRET_KEY = 'at8j8i9%=+m@topzgjzvhs#64^0&qlr6m5yc(_&me%!@jp-7y+'

INSTALLED_APPS = (
    'test',
)
# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

在这个文件中写上 SQLite, MySQL或PostgreSQL的信息, 这样就可以运用这个数据库了。

新建确保每个app下有一个 models.py 和 \_\_init\_\_.py 文件, 就可以享受 Django 的 ORM 带来的便利!

可以用 Django QuerySet API 来创建, 查询, 删除, 修改, 不用写SQL语句。

更详细的请查看本文提供的源代码。

[小额赞助, 支持作者!](#)  
[« Django Sitemap 站点地图](#)  
[Django 通用视图 »](#)

↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)  
[打印页面](#)

[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django ORM

[« Django Sitemap 站点地图](#)

[Django 通用视图 »](#)

本文介绍如何只使用Django的数据库。

Django 的数据库接口非常好用，我们甚至不需要知道SQL语句如何书写，就可以轻松地查询，创建一些内容，所以有时候想，在其它的地方使用Django的 ORM呢？它有这么丰富的 QuerySet API.

示例代码：[Django\\_DB.zip](#)

settings.py

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
SECRET_KEY = 'at8j8i9%+=m@topzgjzvhs#64^0&q1r6m5yc(_&me%!@jp-7y+'

INSTALLED_APPS = (
    'test',
)
# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

在这个文件中写上 SQLite, MySQL或PostgreSQL的信息，这样就可以运用这个数据库了。

新建确保每个app下有一个 models.py 和 \_\_init\_\_.py 文件，就可以享受 Django 的 ORM 带来的便利！

可以用 Django QuerySet API 来创建，查询，删除，修改，不用写SQL语句。

更详细的请查看本文提供的源代码。

[小额赞助，支持作者！](#)

[« Django Sitemap 站点地图](#)

[Django 通用视图](#) »



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。



CODING

零成本开启敏捷开发 五人以下团队免费

免费体验

推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 通用视图

[«Django ORM](#)  
[Django 上下文渲染器»](#)

我们用Django开发，比如做一个博客，我们需要做一个文章列表，文章详情页，这种需求是比较普遍的，所以Django中提供了Class-Based Views。

有时候我们想直接渲染一个模板，不得不写一个视图函数

```
def render_template_view(request):
    return render(request, '/path/to/template.html')
```

其实可以用 `TemplateView` 可以直接写在 `urls.py` 中，不需要定义一个这样的函数。

这样的例子还有很多，下面一一介绍：

在`urls.py`中使用类视图的时候都是调用它的 `.as_view()` 函数

### 一，Base Views

#### 1. `django.views.generic.base.View`

这个类是通用类的基类，其它类都是继承自这个类，一般不会用到这个类，个人感觉用函数更简单些。

```
# views.py
from django.http import HttpResponse
from django.views.generic import View

class MyView(View):
    def get(self, request, *args, **kwargs):
        return HttpResponse('Hello, World!')

# urls.py
from django.conf.urls import patterns, url

from myapp.views import MyView

urlpatterns = patterns('',
    url(r'^mine/$', MyView.as_view(), name='my-view'),
)
```

#### 2. `django.views.generic.base.TemplateView`

在 `get_context_data()` 函数中，可以传一些 额外内容 到 模板

```
# views.py

from django.views.generic.base import TemplateView

from articles.models import Article

class HomePageView(TemplateView):
    template_name = "home.html"

    def get_context_data(self, **kwargs):
        context = super(HomePageView, self).get_context_data(**kwargs)
        context['latest_articles'] = Article.objects.all()[:5]
        return context

# urls.py

from django.conf.urls import patterns, url

from myapp.views import HomePageView

urlpatterns = patterns('',
    url(r'^$', HomePageView.as_view(), name='home'),
)
```

#### 3. `django.views.generic.base.RedirectView`

用来进行跳转，默认是永久重定向（301），可以直接在`urls.py`中使用，非常方便：

```
from django.conf.urls import patterns, url
from django.views.generic.base import RedirectView

urlpatterns = patterns('',
    url(r'^go-to-django/$', RedirectView.as_view(url='http://djangoproject.com/'), name='go-to-django'),
    url(r'^go-to-ziqiangxuetang/$', RedirectView.as_view(url='http://www.ziqiangxuetang.com', permant=False), name='go-to-zqxt'),
)
```

其它的使用方式：(new in Django 1.6)

```
# views.py
from django.shortcuts import get_object_or_404
from django.views.generic.base import RedirectView

from articles.models import Article

class ArticleCounterRedirectView(RedirectView):
```

```

url = ' # 要跳转的网址,
# url 可以不给, 用 pattern_name 和 get_redirect_url() 函数 来解析到要跳转的网址

permanent = False # 是否为永久重定向, 默认为 True
query_string = True # 是否传递GET的参数到跳转网址. True时会传递, 默认为 False
pattern_name = 'article-detail' # 用来跳转的 URL, 看下面的 get_redirect_url() 函数

# 如果url没有设定, 此函数就会尝试用pattern_name和从网址中捕捉的参数来获取对应网址
# 即 reverse(pattern_name, args) 得到相应的网址,
# 在这个例子中是一个文章的点击数链接, 点击后文章浏览次数加1, 再跳转到真正的文章页面
def get_redirect_url(self, *args, **kwargs):
    If url is not set, get_redirect_url() tries to reverse the pattern_name using what was captured in the URL (both named and unnamed groups are used).
    article = get_object_or_404(Article, pk=kwargs['pk'])
    article.update_counter() # 更新文章点击数, 在models.py中实现
    return super(ArticleCounterRedirectView, self).get_redirect_url(*args, **kwargs)

# urls.py
from django.conf.urls import patterns, url
from django.views.generic.base import RedirectView

from article.views import ArticleCounterRedirectView, ArticleDetail

urlpatterns = patterns('',

    url(r'^counter/(?P<pk>\d+)/$', ArticleCounterRedirectView.as_view(), name='article-counter'),
    url(r'^details/(?P<pk>\d+)/$', ArticleDetail.as_view(), name='article-detail'),
)

```

## 二, Generic Display View (通用显示视图)

### 1. `django.views.generic.detail.DetailView`

**DetailView**有以下方法:

1. [`dispatch\(\)`](#)
2. [`http\_method\_not\_allowed\(\)`](#)
3. [`get\_template\_names\(\)`](#)
4. [`get\_slug\_field\(\)`](#)
5. [`get\_queryset\(\)`](#)
6. [`get\_object\(\)`](#)
7. [`get\_context\_object\_name\(\)`](#)
8. [`get\_context\_data\(\)`](#)
9. [`get\(\)`](#)
10. [`render\_to\_response\(\)`](#)

```

# views.py
from django.views.generic.detail import DetailView
from django.utils import timezone

from articles.models import Article

class ArticleDetailView(DetailView):

    model = Article # 要显示详情内容的类

    template_name = 'article_detail.html'
    # 模板名称, 默认为 应用名/类名_detail.html (即 app/modelname_detail.html)

    # 在 get_context_data() 函数中可以用于传递一些额外的内容到网页
    def get_context_data(self, **kwargs):
        context = super(ArticleDetailView, self).get_context_data(**kwargs)
        context['now'] = timezone.now()
        return context

# urls.py
from django.conf.urls import url

from article.views import ArticleDetailView

urlpatterns = [
    url(r'^(?P<slug>[-_\w]+)/$', ArticleDetailView.as_view(), name='article-detail'),
]

```

#### `article_detail.html`

```

<h1>标题: {{ object.title }}</h1>
<p>内容: {{ object.content }}</p>
<p>发表人: {{ object.reporter }}</p>
<p>发表于: {{ object.pub_date|date }}</p>

<p>日期: {{ now|date }}</p>

```

### 2. `django.views.generic.list.ListView`

**ListView**有以下方法:

1. [`dispatch\(\)`](#)



2. [http\\_method\\_not\\_allowed\(\)](#)
3. [get\\_template\\_names\(\)](#)
4. [get\\_queryset\(\)](#)
5. [get\\_context\\_object\\_name\(\)](#)
6. [get\\_context\\_data\(\)](#)
7. [get\(\)](#)
8. [render\\_to\\_response\(\)](#)

```
# views.py
from django.views.generic.list import ListView
from django.utils import timezone

from articles.models import Article

class ArticleListView(ListView):

    model = Article

    def get_context_data(self, **kwargs):
        context = super(ArticleListView, self).get_context_data(**kwargs)
        context['now'] = timezone.now()
        return context
```

```
# urls.py:

from django.conf.urls import url

from article.views import ArticleListView

urlpatterns = [
    url(r'^$', ArticleListView.as_view(), name='article-list'),
]
```

#### article\_list.html

```
<h1>文章列表</h1>
<ul>
{% for article in object_list %}
    <li>{{ article.pub_date|date }} - {{ article.headline }}</li>
{% empty %}
    <li>抱歉，目前还没有文章。</li>
{% endfor %}
</ul>
```

未完待续

Class-based views 官方文档:

<https://docs.djangoproject.com/en/dev/ref/class-based-views/#built-in-class-based-views-api>

[小额赞助，支持作者！](#)

[« Django ORM](#)

[Django 上下文渲染器 »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】](#)拼团上云更优惠, 高性能服务器2折起

[基于Django开发的，自强学堂网站源代码免费下载](#)

# Django 通用视图

«[Django ORM](#)  
[Django 上下文渲染器](#)»

我们用Django开发，比如做一个博客，我们需要做一个文章列表，文章详情页，这种需求是比较普遍的，所以Django中提供了Class-Based Views。

有时候我们想直接渲染一个模板，不得不写一个视图函数

```
def render_template_view(request):
    return render(request, '/path/to/template.html')
```

其实可以用 `TemplateView` 可以直接写在 `urls.py` 中，不需要定义一个这样的函数。

这样的例子还有很多，下面一一介绍：

在`urls.py`中使用类视图的时候都是调用它的 `.as_view()` 函数

## 一，Base Views

### 1. `django.views.generic.base.View`

这个类是通用类的基类，其它类都是继承自这个类，一般不会用到这个类，个人感觉用函数更简单些。

```
# views.py
from django.http import HttpResponse
from django.views.generic import View

class MyView(View):

    def get(self, request, *args, **kwargs):
        return HttpResponse('Hello, World!')

# urls.py
from django.conf.urls import patterns, url

from myapp.views import MyView

urlpatterns = patterns('',
    url(r'^mine/$', MyView.as_view(), name='my-view'),
)
```

### 2. `django.views.generic.base.TemplateView`

在 `get_context_data()` 函数中，可以传一些 额外内容 到 模板

```
# views.py

from django.views.generic.base import TemplateView

from articles.models import Article

class HomePageView(TemplateView):

    template_name = "home.html"

    def get_context_data(self, **kwargs):
        context = super(HomePageView, self).get_context_data(**kwargs)
        context['latest_articles'] = Article.objects.all()[:5]
        return context

# urls.py

from django.conf.urls import patterns, url

from myapp.views import HomePageView

urlpatterns = patterns('',
    url(r'^$', HomePageView.as_view(), name='home'),
)
```

### 3. `django.views.generic.base.RedirectView`

用来进行跳转，默认是永久重定向（301），可以直接在`urls.py`中使用，非常方便：

```
from django.conf.urls import patterns, url
from django.views.generic.base import RedirectView

urlpatterns = patterns('',
    url(r'^go-to-django/$', RedirectView.as_view(url='http://djangoproject.com/'), name='go-to-django'),
    url(r'^go-to-ziqiangxuetang/$', RedirectView.as_view(url='http://www.ziqiangxuetang.com', permant=False), name='go-to-zqxt'),
)
```

其它的使用方式：(new in Django1.6)

```
# views.py
from django.shortcuts import get_object_or_404
from django.views.generic.base import RedirectView

from articles.models import Article

class ArticleCounterRedirectView(RedirectView):

    url = ' ' # 要跳转的网址，
    # url 可以不给，用 pattern_name 和 get_redirect_url() 函数 来解析到要跳转的网址

    permanent = False # 是否为永久重定向，默认为 True
```

```

query_string = True # 是否传递GET的参数到跳转网址, True时会传递, 默认为 False
pattern_name = 'article-detail' # 用来跳转的 URL, 看下面的 get_redirect_url() 函数

# 如果url没有设定, 此函数就会尝试用pattern_name和从网址中捕捉的参数来获取对应网址
# 即 reverse(pattern_name, args) 得到相应的网址,
# 在这个例子中是一个文章的点击数链接: 点击后文章浏览次数加1, 再跳转到真正的文章页面
def get_redirect_url(self, *args, **kwargs):
    If url is not set, get_redirect_url() tries to reverse the pattern_name using what was captured in the URL (both named and unnamed groups are used).
    article = get_object_or_404(Article, pk=kwargs['pk'])
    article.update_counter() # 更新文章点击数, 在models.py中实现
    return super(ArticleCounterRedirectView, self).get_redirect_url(*args, **kwargs)

# urls.py
from django.conf.urls import patterns, url
from django.views.generic.base import RedirectView

from article.views import ArticleCounterRedirectView, ArticleDetail

urlpatterns = patterns('',

    url(r'^counter/(?P<pk>\d+)/$', ArticleCounterRedirectView.as_view(), name='article-counter'),
    url(r'^details/(?P<pk>\d+)/$', ArticleDetail.as_view(), name='article-detail'),
)

```

## 二, Generic Display View (通用显示视图)

### 1. django.views.generic.detail.DetailView

DetailView有以下方法:

1. [dispatch\(\)](#)
2. [http\\_method\\_not\\_allowed\(\)](#)
3. [get\\_template\\_names\(\)](#)
4. [get\\_slug\\_field\(\)](#)
5. [get\\_queryset\(\)](#)
6. [get\\_object\(\)](#)
7. [get\\_context\\_object\\_name\(\)](#)
8. [get\\_context\\_data\(\)](#)
9. [get\(\)](#)
10. [render\\_to\\_response\(\)](#)

```

# views.py
from django.views.generic.detail import DetailView
from django.utils import timezone

from articles.models import Article

class ArticleDetailView(DetailView):

    model = Article # 要显示详情内容的类

    template_name = 'article_detail.html'
    # 模板名称, 默认为 应用名/类名_detail.html (即 app/modelname_detail.html)

    # 在 get_context_data() 函数中可以用于传递一些额外的内容到网页
    def get_context_data(self, **kwargs):
        context = super(ArticleDetailView, self).get_context_data(**kwargs)
        context['now'] = timezone.now()
        return context

# urls.py
from django.conf.urls import url

from article.views import ArticleDetailView

urlpatterns = [
    url(r'^(?P<slug>[_\w]+)/$', ArticleDetailView.as_view(), name='article-detail'),
]

```

#### article\_detail.html

```

<h1>标题: {{ object.title }}</h1>
<p>内容: {{ object.content }}</p>
<p>发表人: {{ object.reporter }}</p>
<p>发表于: {{ object.pub_date|date }}</p>

<p>日期: {{ now|date }}</p>

```

### 2. django.views.generic.list.ListView

ListView有以下方法:

1. [dispatch\(\)](#)
2. [http\\_method\\_not\\_allowed\(\)](#)
3. [get\\_template\\_names\(\)](#)

```
4. get\_queryset\(\)
5. get\_context\_object\_name\(\)
6. get\_context\_data\(\)
7. get\(\)
8. render\_to\_response\(\)

# views.py
from django.views.generic.list import ListView
from django.utils import timezone

from articles.models import Article

class ArticleListView(ListView):

    model = Article

    def get_context_data(self, **kwargs):
        context = super(ArticleListView, self).get_context_data(**kwargs)
        context['now'] = timezone.now()
        return context
```

```
# urls.py:

from django.conf.urls import url

from article.views import ArticleListView

urlpatterns = [
    url(r'^$', ArticleListView.as_view(), name='article-list'),
]
```

#### article\_list.html

```
<h1>文章列表</h1>
<ul>
{% for article in object_list %}
    <li>{{ article.pub_date|date }} - {{ article.headline }}</li>
{% empty %}
    <li>抱歉，目前还没有文章。</li>
{% endfor %}
</ul>
```

未完待续

Class-based views 官方文档:

<https://docs.djangoproject.com/en/dev/ref/class-based-views/#built-in-class-based-views-api>

[小额赞助，支持作者！](#)

[« Django ORM](#)

[Django 上下文渲染器 »](#)

 CODING

零成本开启敏捷开发 五人以下团队免费

免费体验

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 上下文渲染器

[« Django 通用视图](#)

[Django 中间件 »](#)

有时候我们想让一些内容在多个模板中都要有, 比如导航内容, 我们又不想每个视图函数都写一次这些变量内容, 怎么办呢?

这时候就可以用 Django 上下文渲染器来解决。

## 一, 初识上下文渲染器

我们从视图函数说起, 在 `views.py` 中返回字典在模板中使用:

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html', {'info': 'Welcome to ziqiangxuetang.com !'})
```

这样我们就可以在模板中使用 `info` 这个变量了。

```
{{ info }}
```

模板对应的地方就会显示: `Welcome to ziqiangxuetang.com !`

但是如果我们有一个变量, 比如用户的IP, 想显示在网站的每个网页上。再比如显示一些导航信息在每个网页上, 该怎么做呢?

一种方法是用死代码, 直接把栏目固定写在 模块中, 这个对于不经常变动的来说也是一个办法, 简单高效。

但是像用户IP这样的因人而异的, 或者经常变动的, 就不得不想一个更好的解决办法了。

由于上下文渲染器API在Django 1.8时发生了变化, 被移动到了 `templete` 文件夹下, 所以讲解的时候分两种, 一种是 Django 1.8 及以后的, 和Django 1.7及以前的。

我们来看Django官方自带的小例子:

Django 1.8 版本:

[django.template.context\\_processors](#) 中有这样一个函数

```
def request(request):
    return {'request': request}
```

Django 1.7 及以前的代码在这里: [django.core.context\\_processors.request](#) 函数是一样的。

在`settings.py`中:

Django 1.8 版本 `settings.py`:

```
TEMPLATES = [
```

```
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]
```

Django 1.7 版本 `settings.py` 默认是这样的：

```
TEMPLATE_CONTEXT_PROCESSORS = (
    "django.contrib.auth.context_processors.auth",
    "django.core.context_processors.debug",
    "django.core.context_processors.i18n",
    "django.core.context_processors.media",
    "django.core.context_processors.static",
    "django.core.context_processors.tz",
    "django.contrib.messages.context_processors.messages"
)
```

我们可以手动添加 `request` 的渲染器

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    "django.core.context_processors.request",
    ...
)
```

这里的 `context_processors` 中放了一系列的 渲染器，**上下文渲染器 其实就是函数返回字典，字典的 `keys` 可以用在模板中。**

**`request` 函数就是在返回一个字典，每一个模板中都可以使用这个字典中提供的 `request` 变量。**

比如 在 `template` 中 获取当前访问的用户的用户名：

```
User Name: {{ request.user.username }}
```

## 二，动手写个上下文渲染器

2.1 新建一个项目，基于 Django 1.8，低版本的请自行修改对应地方：

```
django-admin.py startproject zqxt
cd zqxt
python manage.py startapp blog
```

我们新建了 `zqxt` 项目和 `blog` 这个应用。

把 `blog` 这个app 加入到 `settings.py` 中

```
INSTALLED_APPS = (
    'django.contrib.admin',
    ...

    'blog',
)
```

整个项目当前目录结构如下：

```
zqxt
├── blog
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── manage.py
└── zqxt
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

2.2 我们在 `zqxt/zqxt/` 这个目录下（与 `settings.py` 在一起）新建一个 `context_processor.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Date      : 2015-10-31 14:26:26
# @Author    : Weizhong Tu (mail@tuweizhong.com)

from django.conf import settings as original_settings

def settings(request):
    return {'settings': original_settings}

def ip_address(request):
    return {'ip_address': request.META['REMOTE_ADDR']}
```

2.3 我们把新建的两个 上下文渲染器 加入到 `settings.py` 中：

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',

                'zqxt.context_processor.settings',
                'zqxt.context_processor.ip_address',
            ],
        },
    },
]
```

最后面两个是我们新加入的，我们稍后在模板中测试它。

2.4 修改 `blog/views.py`

```
from django.shortcuts import render

def index(request):
    return render(request, 'blog/index.html')

def columns(request):
    return render(request, 'blog/columns.html')
```

## 2.5 新建两个模板文件，放在 `zqxt/blog/templates/blog/` 中

### `index.html`

```
<h1>Blog Home Page</h1>

DEBUG: {{ settings.DEBUG }}

ip: {{ ip_address }}
```

### `columns.html`

```
<h1>Blog Columns</h1>

DEBUG: {{ settings.DEBUG }}

ip: {{ ip_address }}
```

## 2.6 修改 `zqxt/urls.py`

```
from django.conf.urls import include, url
from django.contrib import admin
from blog import views as blog_views

urlpatterns = [
    url(r'^blog_home/$', blog_views.index),
    url(r'^blog_columns/$', blog_views.columns),
    url(r'^admin/', include(admin.site.urls)),
]
```

## 2.7 打开开发服务器并访问，进行测试吧：

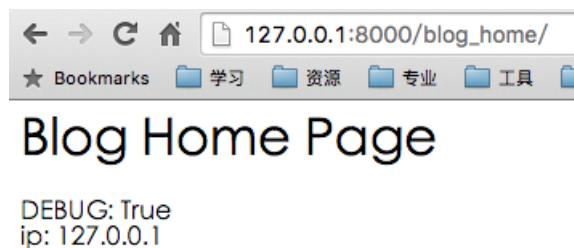
```
python manage.py runserver
```

就会看到所有的模板都可以使用 `settings` 和 `ip_address` 变量：

[http://127.0.0.1:8000/blog\\_home/](http://127.0.0.1:8000/blog_home/)

[http://127.0.0.1:8000/blog\\_columns/](http://127.0.0.1:8000/blog_columns/)

效果图：



最后，附上源代码下载：[zqxt\\_context\\_processor.zip](#)

[小额赞助，支持作者！](#)

[« Django 通用视图](#)

[Django 中间件 »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。



关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。  
自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于[Django](#)开发的, [自强学堂网站源代码免费下载](#)

## Django 上下文渲染器

[« Django 通用视图](#)  
[Django 中间件 »](#)

有时候我们想让一些内容在多个模板中都要有, 比如导航内容, 我们又不想每个视图函数都写一次这些变量内容, 怎么办呢?

这时候就可以用 Django 上下文渲染器来解决。

### 一, 初识上下文渲染器

我们从视图函数说起, 在 `views.py` 中返回字典在模板中使用:

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html', {'info': 'Welcome to ziqiangxuetang.com !'})
```

这样我们就可以在模板中使用 `info` 这个变量了。

```
{{ info }}
```

模板对应的地方就会显示: `Welcome to ziqiangxuetang.com !`

但是如果我们有一个变量, 比如用户的IP, 想显示在网站的每个网页上。再比如显示一些导航信息在每个网页上, 该怎么做呢?

一种方法是用死代码，直接把栏目固定写在 模块中，这个对于不经常变动的来说也是一个办法，简单高效。

但是像用户IP这样的因人而异的，或者经常变动的，就不得不想一个更好的解决办法了。

由于上下文渲染器API在Django 1.8时发生了变化，被移动到了 `templete` 文件夹下，所以讲解的时候分两种，一种是 Django 1.8 及以后的，和Django 1.7及以前的。

我们来看Django官方自带的小例子：

Django 1.8 版本：

[django.template.context\\_processors](#) 中有这样一个函数

```
def request(request):  
    return {'request': request}
```

Django 1.7 及以前的代码在这里：[django.core.context\\_processors.request](#) 函数是一样的。

在`settings.py` 中：

Django 1.8 版本 `settings.py`:

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    ],  
]
```

Django 1.7 版本 `settings.py` 默认是这样的：

```
TEMPLATE_CONTEXT_PROCESSORS = (  
    "django.contrib.auth.context_processors.auth",  
    "django.core.context_processors.debug",  
    "django.core.context_processors.i18n",  
    "django.core.context_processors.media",  
    "django.core.context_processors.static",  
    "django.core.context_processors.tz",  
    "django.contrib.messages.context_processors.messages"  
)
```

我们可以手动添加 `request` 的渲染器

```
TEMPLATE_CONTEXT_PROCESSORS = (  
    ...  
    "django.core.context_processors.request",  
    ...  
)
```

这里的 `context_processors` 中放了一系列的 渲染器，**上下文渲染器 其实就是函数返回字典，字典的 `keys` 可以用在模**

板中。

`request` 函数就是在返回一个字典，每一个模板中都可以使用这个字典中提供的 `request` 变量。

比如在 `template` 中 获取当前访问的用户的用户名：

```
User Name: {{ request.user.username }}
```

## 二，动手写个上下文渲染器

2.1 新建一个项目，基于 Django 1.8，低版本的请自行修改对应地方：

```
django-admin.py startproject zqxt
cd zqxt
python manage.py startapp blog
```

我们新建了 `zqxt` 项目和 `blog` 这个应用。

把 `blog` 这个app 加入到 `settings.py` 中

```
INSTALLED_APPS = (
    'django.contrib.admin',
    ...

    'blog',
)
```

整个项目当前目录结构如下：

```
zqxt
├── blog
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── manage.py
└── zqxt
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

2.2 我们在 `zqxt/zqxt/` 这个目录下（与`settings.py`在一起）新建一个 `context_processor.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Date      : 2015-10-31 14:26:26
# @Author    : Weizhong Tu (mail@tuweizhong.com)

from django.conf import settings as original_settings

def settings(request):
    return {'settings': original_settings}

def ip_address(request):
    return {'ip_address': request.META['REMOTE_ADDR']}
```

2.3 我们把新建的两个 上下文渲染器 加入到 `settings.py` 中：

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
```

```

'DIRS': [],
'APP_DIRS': True,
'OPTIONS': {
    'context_processors': [
        'django.template.context_processors.debug',
        'django.template.context_processors.request',
        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',

        'zqxt.context_processor.settings',
        'zqxt.context_processor.ip_address',
    ],
},
],
}

```

最后面两个是我们新加入的，我们稍后在模板中测试它。

## 2.4 修改 blog/views.py

```

from django.shortcuts import render

def index(request):
    return render(request, 'blog/index.html')

def columns(request):
    return render(request, 'blog/columns.html')

```

## 2.5 新建两个模板文件，放在 zqxt/blog/templates/blog/ 中

### index.html

```

<h1>Blog Home Page</h1>

DEBUG: {{ settings.DEBUG }}

ip: {{ ip_address }}

```

### columns.html

```

<h1>Blog Columns</h1>

DEBUG: {{ settings.DEBUG }}

ip: {{ ip_address }}

```

## 2.6 修改 zqxt/urls.py

```

from django.conf.urls import include, url
from django.contrib import admin
from blog import views as blog_views

urlpatterns = [
    url(r'^blog_home/$', blog_views.index),
    url(r'^blog_columns/$', blog_views.columns),
    url(r'^admin/', include(admin.site.urls)),
]

```

## 2.7 打开开发服务器并访问，进行测试吧：

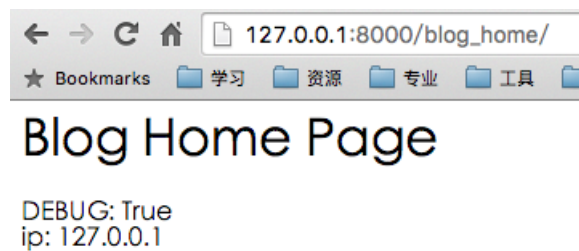
```
python manage.py runserver
```

就会看到所有的模板都可以使用 `settings` 和 `ip_address` 变量：

[http://127.0.0.1:8000/blog\\_home/](http://127.0.0.1:8000/blog_home/)

[http://127.0.0.1:8000/blog\\_columns/](http://127.0.0.1:8000/blog_columns/)

效果图:



最后，附上源代码下载: [zqxt\\_context\\_processor.zip](#)

[小额赞助，支持作者！](#)

[« Django 通用视图](#)

[Django 中间件 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

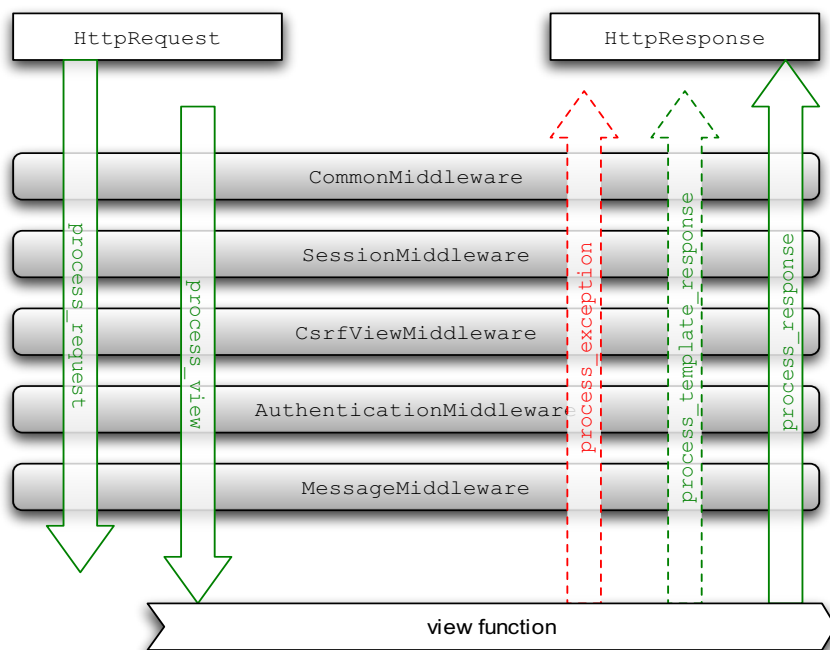
推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
基于Django开发的，[自强学堂网站源代码免费下载](#)

## Django 中间件

«[Django 上下文渲染器](#)  
[Python/Django 微信接口](#)»

提示：关于 Django 1.10 的变化在本文的最后面有详细的说明。

我们从浏览器发出一个请求 Request，得到一个响应后的内容 HttpResponse，这个请求传递到 Django 的过程如下：



也就是说，每一个请求都是先通过中间件中的 `process_request` 函数，这个函数返回 `None` 或者 `HttpResponse` 对象，如果返回前者，继续处理其它中间件，如果返回一个 `HttpResponse`，就处理中止，返回到网页上。

中间件不用继承自任何类（可以继承 `object`），下面一个中间件大概的样子：

```
class CommonMiddleware(object):
    def process_request(self, request):
        return None

    def process_response(self, request, response):
        return response
```

还有 `process_view`, `process_exception` 和 `process_template_response` 函数。

一，比如我们要做一个拦截器，发现有恶意访问网站的人，就拦截他！

假如我们通过一种技术，比如统计一分钟访问页面数，太多就把他的 IP 加入到黑名单 `BLOCKED_IPS`（这部分没有提供代码，主要讲中间件部分）

```
#项目 zqxt 文件名 zqxt/middleware.py

class BlockedIpMiddleware(object):
    def process_request(self, request):
        if request.META['REMOTE_ADDR'] in getattr(settings, "BLOCKED_IPS", []):
            return http.HttpResponseForbidden('<h1>Forbidden</h1>')
```

这里的代码的功能就是 获取当前访问者的 IP (`request.META['REMOTE_ADDR']`), 如果这个 IP 在黑名单中就拦截, 如果不在就返回 `None` (函数中没有返回值其实就是默认为 `None`), 把这个中间件的 Python 路径写到 `settings.py` 中

### 1.1 Django 1.9 和以前的版本:

```
MIDDLEWARE_CLASSES = (
    'zqxt.middleware.BlockedIpMiddleware',
    ...其它的中间件
)
```

### 1.2 Django 1.10 版本 更名为 **MIDDLEWARE** (单复同形), 写法也有变化, 详见 第四部分。

如果用 **Django 1.10** 版本开发, 部署时用 **Django 1.9** 版本或更低版本, 要特别小心此处。

```
MIDDLEWARE = (
    'zqxt.middleware.BlockedIpMiddleware',
    ...其它的中间件
)
```

Django 会从 `MIDDLEWARE_CLASSES` 或 `MIDDLEWARE` 中按照从上到下的顺序一个个执行中间件中的 `process_request` 函数, 而其中 `process_response` 函数则是最前面的最后执行。

二, 再比如, 我们在网站放到服务器上正式运行后, **DEBUG** 改为了 **False**, 这样更安全, 但是有时候发生错误我们不能看到错误详情, 调试不方便, 有没有办法处理好这两个事情呢?

1. 普通访问者看到的是友好的报错信息
2. 管理员看到的是错误详情, 以便于修复 BUG

当然可以有, 利用中间件就可以做到! 代码如下:

```
import sys
from django.views.debug import technical_500_response
from django.conf import settings

class UserBasedExceptionMiddleware(object):
    def process_exception(self, request, exception):
        if request.user.is_superuser or request.META.get('REMOTE_ADDR') in settings.INTERNAL_IPS:
            return technical_500_response(request, *sys.exc_info())
```

把这个中间件像上面一样, 加到你的 `settings.py` 中的 `MIDDLEWARE_CLASSES` 中, 可以放到最后, 这样可以看到其它中间件的 `process_request` 的错误。

当访问者为管理员时, 就给出错误详情, 比如访问本站的不存在的页面: <http://www.ziqiangxuetang.com/admin/>

普通人看到的是普通的 404 (自己点开看看), 而我可以看到:



三, 分享一个简单的识别手机的中间件, 更详细的可以参考这个: [django-mobi](#) 或 [django-mobile](#)

```
MOBILE_USERAGENTS = ("2.0 MMP", "240x320", "400X240", "AvantGo", "BlackBerry",
```

```
"Blazer", "Cellphone", "Danger", "DoCoMo", "Elaine/3.0", "EudoraWeb",
"Googlebot-Mobile", "hiptop", "IEMobile", "KYOCERA/WX310K", "LG/U990",
"MIDP-2.", "MMEF20", "MOT-V", "NetFront", "Newt", "Nintendo Wii", "Nitro",
"Nokia", "Opera Mini", "Palm", "PlayStation Portable", "portalmmm", "Proxinet",
"ProxiNet", "SHARP-TQ-GX10", "SHG-i900", "Small", "SonyEricsson", "Symbian OS",
"SymbianOS", "TS21i-10", "UP.Browser", "UP.Link", "webOS", "Windows CE",
"WinWAP", "YahooSeeker/M1A1-R2D2", "iPhone", "iPod", "Android",
"BlackBerry9530", "LG-TU915 Obigo", "LGE VX", "webOS", "Nokia5800")
```

```
class MobileTemplate(object):
    """
    If a mobile user agent is detected, inspect the default args for the view
    func, and if a template name is found assume it is the template arg and
    attempt to load a mobile template based on the original template name.
    """

    def process_view(self, request, view_func, view_args, view_kwargs):
        if any(ua for ua in MOBILE_USERAGENTS if ua in
            request.META["HTTP_USER_AGENT"]):
            template = view_kwargs.get("template")
            if template is None:
                for default in view_func.func_defaults:
                    if str(default).endswith(".html"):
                        template = default
            if template is not None:
                template = template.rsplit(".html", 1)[0] + ".mobile.html"
            try:
                get_template(template)
            except TemplateDoesNotExist:
                pass
            else:
                view_kwargs["template"] = template
                return view_func(request, *view_args, **view_kwargs)

        return None
```

参考文档: <https://docs.djangoproject.com/en/1.8/topics/http/middleware/>

#### 四, 补充: Django 1.10 接口发生变化, 变得更加简洁

```
class SimpleMiddleware(object):
    def __init__(self, get_response):
        self.get_response = get_response
        # One-time configuration and initialization.

    def __call__(self, request):
        # Code to be executed for each request before
        # the view (and later middleware) are called.
        # 调用 view 之前的代码

        response = self.get_response(request)

        # Code to be executed for each request/response after
        # the view is called.
        # 调用 view 之后的代码

        return response
```

Django 1.10.x 也可以用函数来实现中间件, 详见[官方文档](#)。

#### 五, 让你写的中间件 兼容 Django 新版本和旧版本

```
try:
    from django.utils.deprecation import MiddlewareMixin # Django 1.10.x
except ImportError:
    MiddlewareMixin = object # Django 1.4.x - Django 1.9.x
```



```
class SimpleMiddleware(MiddlewareMixin):
    def process_request(self, request):
        pass

    def process_response(request, response):
        pass
```

新版本中 `django.utils.deprecation.MiddlewareMixin` 的 [源代码](#) 如下:

```
class MiddlewareMixin(object):
    def __init__(self, get_response=None):
        self.get_response = get_response
        super(MiddlewareMixin, self).__init__()

    def __call__(self, request):
        response = None
        if hasattr(self, 'process_request'):
            response = self.process_request(request)
        if not response:
            response = self.get_response(request)
        if hasattr(self, 'process_response'):
            response = self.process_response(request, response)
        return response
```

`__call__` 方法会先调用 `self.process_request(request)`, 接着执行 `self.get_response(request)` 然后调用 `self.process_response(request, response)`

旧版本(Django 1.4.x-Django 1.9.x)的话, 和原来一样。

[小额赞助, 支持作者!](#)  
[« Django 上下文渲染器](#)  
[Python/Django 微信接口 »](#)



↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

---

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云](#) ECS



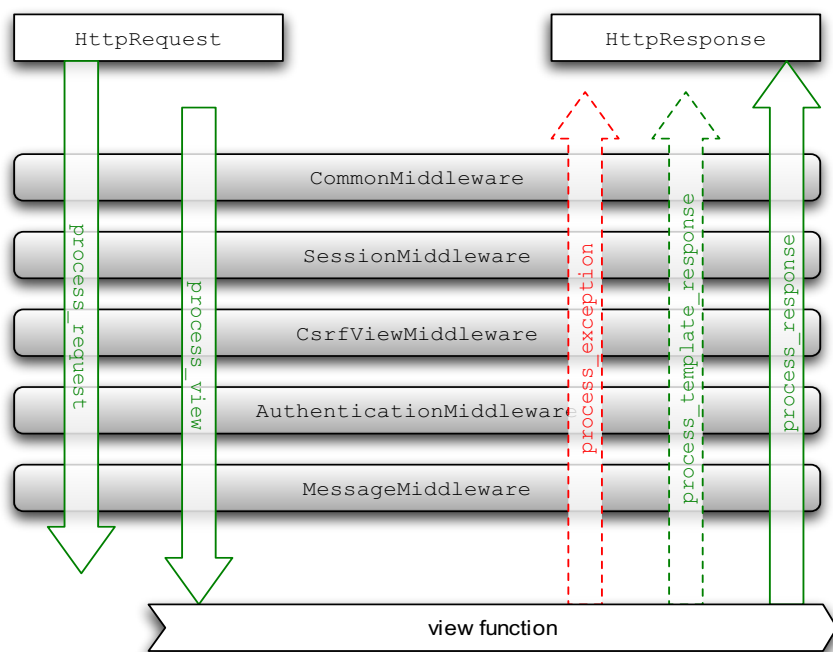
推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 中间件

[« Django 上下文渲染器](#)  
[Python/Django 微信接口 »](#)

提示: 关于 Django 1.10 的变化在本文的最后面有详细的说明。

我们从浏览器发出一个请求 Request, 得到一个响应后的内容 HttpResponse, 这个请求传递到 Django 的过程如下:



也就是说, 每一个请求都是先通过中间件中的 `process_request` 函数, 这个函数返回 `None` 或者 `HttpResponse` 对象, 如果返回前者, 继续处理其它中间件, 如果返回一个 `HttpResponse`, 就处理中止, 返回到网页上。

中间件不用继承自任何类 (可以继承 `object`), 下面一个中间件大概的样子:

```
class CommonMiddleware(object):
    def process_request(self, request):
        return None

    def process_response(self, request, response):
        return response
```

还有 `process_view`, `process_exception` 和 `process_template_response` 函数。

一, 比如我们要做一个拦截器, 发现有恶意访问网站的人, 就拦截他!

假如我们通过一种技术，比如统计一分钟访问页面数，太多就把他的 IP 加入到黑名单 **BLOCKED\_IPS**（这部分没有提供代码，主要讲中间件部分）

#项目 zqxt 文件名 zqxt/middleware.py

```
class BlockedIpMiddleware(object):
    def process_request(self, request):
        if request.META['REMOTE_ADDR'] in getattr(settings, "BLOCKED_IPS", []):
            return http.HttpResponseForbidden('<h1>Forbidden</h1>')
```

这里的代码的功能就是 获取当前访问者的 IP (`request.META['REMOTE_ADDR']`)，如果这个 IP 在黑名单中就拦截，如果不在就返回 `None` (函数中没有返回值其实就是默认为 `None`)，把这个中间件的 Python 路径写到 `settings.py` 中

### 1.1 Django 1.9 和以前的版本：

```
MIDDLEWARE_CLASSES = (
    'zqxt.middleware.BlockedIpMiddleware',
    ...其它的中间件
)
```

### 1.2 Django 1.10 版本 更名为 **MIDDLEWARE**（单复同形），写法也有变化，详见 第四部分。

如果用 **Django 1.10**版本开发，部署时用 **Django 1.9**版本或更低版本，要特别小心此处。

```
MIDDLEWARE = (
    'zqxt.middleware.BlockedIpMiddleware',
    ...其它的中间件
)
```

Django 会从 `MIDDLEWARE_CLASSES` 或 `MIDDLEWARE` 中按照从上到下的顺序一个个执行中间件中的 `process_request` 函数，而其中 `process_response` 函数则是最前面的最后执行。

二，再比如，我们在网站放到服务器上正式运行后，**DEBUG**改为了 **False**，这样更安全，但是有时候发生错误我们不能看到错误详情，调试不方便，有没有办法处理好这两个事情呢？

1. 普通访问者看到的是友好的报错信息
2. 管理员看到的是错误详情，以便于修复 BUG

当然可以有，利用中间件就可以做到！代码如下：

```
import sys
from django.views.debug import technical_500_response
from django.conf import settings

class UserBasedExceptionMiddleware(object):
    def process_exception(self, request, exception):
        if request.user.is_superuser or request.META.get('REMOTE_ADDR') in settings.INTERNAL_IPS:
            return technical_500_response(request, *sys.exc_info())
```

把这个中间件像上面一样，加到你的 `settings.py` 中的 `MIDDLEWARE_CLASSES` 中，可以放到最后，这样可以看到其它中间件的 `process_request` 的错误。

当访问者为管理员时，就给出错误详情，比如访问本站的不存在的页面：<http://www.ziqiangxuetang.com/admin/>

普通人看到的是普通的 404（自己点开看看），而我可以看到：



### 三，分享一个简单的识别手机的中间件，更详细的可以参考这个：[django-mobi](#) 或 [django-mobile](#)

```
MOBILE_USERAGENTS = ("2.0 MMP", "240x320", "400X240", "AvantGo", "BlackBerry",
    "Blazer", "Cellphone", "Danger", "DoCoMo", "Elaine/3.0", "EudoraWeb",
    "Googlebot-Mobile", "hiptop", "IEMobile", "KYOCERA/WX310K", "LG/U990",
    "MIDP-2.", "MMEF20", "MOT-V", "NetFront", "Newt", "Nintendo Wii", "Nitro",
    "Nokia", "Opera Mini", "Palm", "PlayStation Portable", "portalmmm", "Proxinet",
    "ProxiNet", "SHARP-TQ-GX10", "SHG-i900", "Small", "SonyEricsson", "Symbian OS",
    "SymbianOS", "TS21i-10", "UP.Browser", "UP.Link", "webOS", "Windows CE",
    "WinWAP", "YahooSeeker/M1A1-R2D2", "iPhone", "iPod", "Android",
    "BlackBerry9530", "LG-TU915 Obigo", "LGE VX", "webOS", "Nokia5800")

class MobileTemplate(object):
    """
    If a mobile user agent is detected, inspect the default args for the view
    func, and if a template name is found assume it is the template arg and
    attempt to load a mobile template based on the original template name.
    """

    def process_view(self, request, view_func, view_args, view_kwargs):
        if any(ua for ua in MOBILE_USERAGENTS if ua in
            request.META["HTTP_USER_AGENT"]):
            template = view_kwargs.get("template")
            if template is None:
                for default in view_func.func_defaults:
                    if str(default).endswith(".html"):
                        template = default
            if template is not None:
                template = template.rsplit(".html", 1)[0] + ".mobile.html"
            try:
                get_template(template)
            except TemplateDoesNotExist:
                pass
            else:
                view_kwargs["template"] = template
            return view_func(request, *view_args, **view_kwargs)
        return None
```

参考文档：<https://docs.djangoproject.com/en/1.8/topics/http/middleware/>

### 四，补充：Django 1.10 接口发生变化，变得更加简洁

```
class SimpleMiddleware(object):
    def __init__(self, get_response):
        self.get_response = get_response
        # One-time configuration and initialization.

    def __call__(self, request):
        # Code to be executed for each request before
        # the view (and later middleware) are called.
        # 调用 view 之前的代码

        response = self.get_response(request)

        # Code to be executed for each request/response after
        # the view is called.
        # 调用 view 之后的代码

        return response
```

Django 1.10.x 也可以用函数来实现中间件，详见[官方文档](#)。

### 五，让你写的中间件 兼容 Django 新版本和旧版本

try:

```
from django.utils.deprecation import MiddlewareMixin # Django 1.10.x
except ImportError:
    MiddlewareMixin = object # Django 1.4.x - Django 1.9.x

class SimpleMiddleware(MiddlewareMixin):
    def process_request(self, request):
        pass

    def process_response(request, response):
        pass
```

新版本中 `django.utils.deprecation.MiddlewareMixin` 的 [源代码](#) 如下:

```
class MiddlewareMixin(object):
    def __init__(self, get_response=None):
        self.get_response = get_response
        super(MiddlewareMixin, self).__init__()

    def __call__(self, request):
        response = None
        if hasattr(self, 'process_request'):
            response = self.process_request(request)
        if not response:
            response = self.get_response(request)
        if hasattr(self, 'process_response'):
            response = self.process_response(request, response)
        return response
```

`__call__` 方法会先调用 `self.process_request(request)`, 接着执行 `self.get_response(request)` 然后调用 `self.process_response(request, response)`

旧版本(Django 1.4.x-Django 1.9.x)的话, 和原来一样。

[小额赞助, 支持作者!](#)

[« Django 上下文渲染器](#)

[Python/Django 微信接口 »](#)



↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
基于Django开发的，[自强学堂网站源代码免费下载](#)

## Python/Django 微信接口

«[Django 中间件](#)  
[Django 单元测试](#)»

注册或登录 [微信公众平台](#) 点击左侧的 [开发者中心](#)

填写相应的网址，Token(令牌)是随便写的，你自己想写什么就写什么，微信验证时检验是否写的和服务器的TOKEN一样，一样则通过。

关注一下自强学堂的微信号吧，可以[随时随地查阅教程](#)哦，体验一下自强学堂的微信的各种功能再阅读效果更佳！



自己动手写微信的验证：[views.py](#)

```
#coding=utf-8
import hashlib
import json
from lxml import etree
from django.utils.encoding import smart_str
from django.views.decorators.csrf import csrf_exempt
from django.http import HttpResponseRedirect
from auto_reply.views import auto_reply_main # 修改这里

WEIXIN_TOKEN = 'write-a-value'

@csrf_exempt
def weixin_main(request):
    """
    所有的消息都会先进入这个函数进行处理，函数包含两个功能，
    微信接入验证是GET方法，
    微信正常的收发消息是用POST方法。
    """
    if request.method == "GET":
        signature = request.GET.get("signature", None)
        timestamp = request.GET.get("timestamp", None)
        nonce = request.GET.get("nonce", None)
        echostr = request.GET.get("echostr", None)
        token = WEIXIN_TOKEN
        tmp_list = [token, timestamp, nonce]
        tmp_list.sort()
        tmp_str = "%s%s%s" % tuple(tmp_list)
        tmp_str = hashlib.shal(tmp_str).hexdigest()
        if tmp_str == signature:
            return HttpResponseRedirect(echostr)
        else:
            return HttpResponseRedirect("weixin index")
    else:
        xml_str = smart_str(request.body)
        request_xml = etree.fromstring(xml_str)
        response_xml = auto_reply_main(request_xml) # 修改这里
        return HttpResponseRedirect(response_xml)
```

auto\_reply\_main是用来处理消息，回复消息的，需要自己进一步完善。

使用第三方包实现：

关于Django开发微信，有已经做好的现在的包可以使用 [wechat\\_sdk](#) 这个包，[使用文档](#) 也比较完善，但是在处理加密一部分没有做，在微信公众平台上，需要用[明文验证](#)，如果要加密，自己参照微信官网的加密算法。

使用 wechat\_sdk 的例子（自强学堂微信号简化后的例子）：

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.http.response import HttpResponseRedirect, HttpResponseRedirectBadRequest
from django.views.decorators.csrf import csrf_exempt

from wechat_sdk import WechatBasic
from wechat_sdk.exceptions import ParseError
from wechat_sdk.messages import TextMessage

WECHAT_TOKEN = 'zqxt'
AppID = ''
AppSecret = ''

# 实例化 WechatBasic
wechat_instance = WechatBasic(
    token=WECHAT_TOKEN,
    appid=AppID,
    appsecret=AppSecret
)

@csrf_exempt
def index(request):
    if request.method == 'GET':
        # 检验合法性
        # 从 request 中提取基本信息 (signature, timestamp, nonce, xml)
        signature = request.GET.get('signature')
        timestamp = request.GET.get('timestamp')
```

```

nonce = request.GET.get('nonce')

if not wechat_instance.check_signature(
    signature=signature, timestamp=timestamp, nonce=nonce):
    return HttpResponseBadRequest('Verify Failed')

return HttpResponse(
    request.GET.get('echostr', ''), content_type="text/plain")

# 解析本次请求的 XML 数据
try:
    wechat_instance.parse_data(data=request.body)
except ParseError:
    return HttpResponseBadRequest('Invalid XML Data')

# 获取解析好的微信请求信息
message = wechat_instance.get_message()

# 关注事件以及不匹配时的默认回复
response = wechat_instance.response_text(
    content = (
        '感谢您的关注! \n回复【功能】两个字查看支持的功能，还可以回复任意内容开始聊天'
        '\n【<a href="http://www.ziqiangxuetang.com">自强学堂手机版</a>】'
    ))
if isinstance(message, TextMessage):
    # 当前会话内容
    content = message.content.strip()
    if content == '功能':
        reply_text = (
            '目前支持的功能: \n1. 关键词后面加上【教程】两个字可以搜索教程，'
            '比如回复 "Django 后台教程"\n'
            '2. 回复任意词语，查天气，陪聊天，讲故事，无所不能! \n'
            '还有更多功能正在开发中哦 ^_^ \n'
            '【<a href="http://www.ziqiangxuetang.com">自强学堂手机版</a>】'
        )
    elif content.endswith('教程'):
        reply_text = '您要找的课程如下: '

    response = wechat_instance.response_text(content=reply_text)

return HttpResponse(response, content_type="application/xml")

```

下面是一个更详细复杂的使用例子:

#### models.py

```

# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models

class KeyWord(models.Model):
    keyword = models.CharField(
        '关键词', max_length=256, primary_key=True, help_text='用户发出的关键词')
    content = models.TextField(
        '内容', null=True, blank=True, help_text='回复给用户的内容')

    pub_date = models.DateTimeField('发表时间', auto_now_add=True)
    update_time = models.DateTimeField('更新时间', auto_now=True, null=True)
    published = models.BooleanField('发布状态', default=True)

    def __unicode__(self):
        return self.keyword

class Meta:
    verbose_name='关键词'
    verbose_name_plural=verbose_name

```

#### views.py

```

# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.http.response import HttpResponse, HttpResponseBadRequest
from django.views.decorators.csrf import csrf_exempt

from wechat_sdk import WechatBasic
from wechat_sdk.exceptions import ParseError
from wechat_sdk.messages import (TextMessage, VoiceMessage, ImageMessage,
    VideoMessage, LinkMessage, LocationMessage, EventMessage)

from wechat_sdk.context.framework.django import DatabaseContextStore
from .models import KeyWord as KeyWordModel

# 实例化 WechatBasic
wechat_instance = WechatBasic(
    token='zqxt',
    appid='xx',
    appsecret='xx'
)

@csrf_exempt
def index(request):
    if request.method == 'GET':
        # 检验合法性
        # 从 request 中提取基本信息 (signature, timestamp, nonce, xml)
        signature = request.GET.get('signature')
        timestamp = request.GET.get('timestamp')
        nonce = request.GET.get('nonce')

        if not wechat_instance.check_signature(
            signature=signature, timestamp=timestamp, nonce=nonce):
            return HttpResponseBadRequest('Verify Failed')

```

```

        return HttpResponse(
            request.GET.get('echostr', ''), content_type="text/plain")

# POST
# 解析本次请求的 XML 数据
try:
    wechat_instance.parse_data(data=request.body)
except ParseError:
    return HttpResponseBadRequest('Invalid XML Data')

# 获取解析好的微信请求信息
message = wechat_instance.get_message()
# 利用本次请求中的用户OpenID来初始化上下文对话
context = DatabaseContextStore(openid=message.source)

response = None

if isinstance(message, TextMessage):
    step = context.get('step', 1) # 当前对话次数, 如果没有则返回 1
    # last_text = context.get('last_text') # 上次对话内容
    content = message.content.strip() # 当前会话内容

    if message.content == '新闻':
        response = wechat_instance.response_news([
            {
                'title': '自强学堂',
                'picurl': 'http://www.ziqiangxuetang.com/static/images/newlogo.png',
                'description': '自强学堂致力于提供优质的IT技术教程, 网页制作, 服务器后台编写, 以及编程语言, 如HTML, JS, Bootstrap, Python, Django. 同时也提供大量在线实例, 通过实例, 学习更容',
                'url': 'http://www.ziqiangxuetang.com',
            }, {
                'title': '百度',
                'picurl': 'http://doraemonext.oss-cn-hangzhou.aliyuncs.com/test/wechat-test.jpg',
                'url': 'http://www.baidu.com',
            }, {
                'title': 'Django 教程',
                'picurl': 'http://www.ziqiangxuetang.com/media/uploads/images/django_logo_20140508_061519_35.jpg',
                'url': 'http://www.ziqiangxuetang.com/django/django-tutorial.html',
            }
        ])
        return HttpResponse(response, content_type="application/xml")

    else:
        try:
            keyword_object = KeywordModel.objects.get(keyword=content)
            reply_text = keyword_object.content
        except KeywordModel.DoesNotExist:
            try:
                reply_text = KeywordModel.objects.get(keyword='提示').content
            except KeywordModel.DoesNotExist:
                reply_text = ('/:P- (好委屈, 数据库翻个遍也没找到你输入的关键词! \n'
                    '试试下面这些关键词吧: \nKEYWORD_LIST\n'
                    '<a href="http://www.rxnfinder.org">RxnFinder</a>'
                    '感谢您的支持! /:rose'
                )

# 将新的数据存入上下文对话中
context['step'] = step + 1
context['last_text'] = content
context.save() # 非常重要! 请勿忘记! 临时数据保存入数据库!

if 'KEYWORD_LIST' in reply_text:
    keyword_objects = KeywordModel.objects.exclude(keyword_in=[
        '关注事件', '测试', 'test', '提示']).filter(published=True)
    keywords = '{}. {}'.format(str(i), k.keyword)
    for i, k in enumerate(keyword_objects, 1)
    reply_text = reply_text.replace(
        'KEYWORD_LIST', '\n'.join(keywords))

# 文本消息结束

elif isinstance(message, VoiceMessage):
    reply_text = '语音信息我听不懂/:P- (/:P- (/:P- ('
elif isinstance(message, ImageMessage):
    reply_text = '图片信息我也看不懂/:P- (/:P- (/:P- ('
elif isinstance(message, VideoMessage):
    reply_text = '视频我不会看/:P- ('
elif isinstance(message, LinkMessage):
    reply_text = '链接信息'
elif isinstance(message, LocationMessage):
    reply_text = '地理位置信息'
elif isinstance(message, EventMessage): # 事件信息
    if message.type == 'subscribe': # 关注事件 (包括普通关注事件和扫描二维码造成的关注事件)
        follow_event = KeywordModel.objects.get(keyword='关注事件')
        reply_text = follow_event.content

    # 如果 key 和 ticket 均不为空, 则是扫描二维码造成的关注事件
    if message.key and message.ticket:
        reply_text += '\n来源: 扫描二维码关注'
    else:
        reply_text += '\n来源: 搜索名称关注'
elif message.type == 'unsubscribe':
    reply_text = '取消关注事件'
elif message.type == 'scan':
    reply_text = '已关注用户扫描二维码!'
elif message.type == 'location':
    reply_text = '上报地理位置'
elif message.type == 'click':
    reply_text = '自定义菜单点击'
elif message.type == 'view':
    reply_text = '自定义菜单跳转链接'
elif message.type == 'templatesendjobfinish':
    reply_text = '模板消息'

response = wechat_instance.response_text(content=reply_text)
return HttpResponse(response, content_type="application/xml")

```



[Django 单元测试](#) »



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!   
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Python/Django 微信接口

[« Django 中间件](#)  
[Django 单元测试 »](#)

注册或登陆 [微信公众平台](#) 点击左侧的 [开发者中心](#)

填写相应的网址, Token(令牌)是随便写的, 你自己想写什么就写什么, 微信验证时检验是否写的和服务器上的TOKEN一样, 一样则通过。

关注一下自强学堂的微信号吧, 可以[随时随地查阅教程](#)哦, 体验一下自强学堂的微信的各种功能再阅读效果更佳!



自己动手写微信的验证: [views.py](#)

```
#coding=utf-8
import hashlib
import json
from lxml import etree
from django.utils.encoding import smart_str
from django.views.decorators.csrf import csrf_exempt
from django.http import HttpResponse
from auto_reply.views import auto_reply_main # 修改这里

WEIXIN_TOKEN = 'write-a-value'

@csrf_exempt
def weixin_main(request):
    """
    所有的消息都会先进入这个函数进行处理, 函数包含两个功能,
    微信接入验证是GET方法,
    微信正常的收发消息是用POST方法。
    """
    if request.method == "GET":
        signature = request.GET.get("signature", None)
        timestamp = request.GET.get("timestamp", None)
        nonce = request.GET.get("nonce", None)
        echostr = request.GET.get("echostr", None)
        token = WEIXIN_TOKEN
        tmp_list = [token, timestamp, nonce]
        tmp_list.sort()
        tmp_str = "%s%s%s" % tuple(tmp_list)
        tmp_str = hashlib.sh1(tmp_str).hexdigest()
        if tmp_str == signature:
            return HttpResponse(echostr)
        else:
            return HttpResponse("weixin index")
    else:
        xml_str = smart_str(request.body)
        request_xml = etree.fromstring(xml_str)
        response_xml = auto_reply_main(request_xml) # 修改这里
        return HttpResponse(response_xml)
```

[auto\\_reply\\_main](#) 是用来处理消息, 回复消息的, 需要自己进一步完善。

### 使用第三方包实现：

关于Django开发微信，有已经做好的现在的包可以使用 [wechat\\_sdk](#) 这个包，[使用文档](#) 也比较完善，但是在处理加密一部分没有做，在微信公众平台上，需要用[明文验证](#)，如果要加密，自己参照微信官网的加密算法。

### 使用 wechat\_sdk 的例子（自强学堂微信号简化后的例子）：

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.http.response import HttpResponseRedirect, HttpResponseRedirect
from django.views.decorators.csrf import csrf_exempt

from wechat_sdk import WechatBasic
from wechat_sdk.exceptions import ParseError
from wechat_sdk.messages import TextMessage

WECHAT_TOKEN = 'zqxxt'
AppID = ''
AppSecret = ''

# 实例化 WechatBasic
wechat_instance = WechatBasic(
    token=WECHAT_TOKEN,
    appid=AppID,
    appsecret=AppSecret
)

@csrf_exempt
def index(request):
    if request.method == 'GET':
        # 检验合法性
        # 从 request 中提取基本信息 (signature, timestamp, nonce, xml)
        signature = request.GET.get('signature')
        timestamp = request.GET.get('timestamp')
        nonce = request.GET.get('nonce')

        if not wechat_instance.check_signature(
            signature=signature, timestamp=timestamp, nonce=nonce):
            return HttpResponseRedirect('Verify Failed')

        return HttpResponseRedirect(
            request.GET.get('echostr', ''), content_type="text/plain")

    # 解析本次请求的 XML 数据
    try:
        wechat_instance.parse_data(data=request.body)
    except ParseError:
        return HttpResponseRedirect('Invalid XML Data')

    # 获取解析好的微信请求信息
    message = wechat_instance.get_message()

    # 关注事件以及不匹配时的默认回复
    response = wechat_instance.response_text(
        content = (
            '感谢您的关注！\n回复【功能】两个字查看支持的功能，还可以回复任意内容开始聊天'
            '\n【<a href="http://www.ziqiangxuetang.com">自强学堂手机版</a>】'
        ))

    if isinstance(message, TextMessage):
        # 当前会话内容
        content = message.content.strip()
        if content == '功能':
            reply_text = (
                '目前支持的功能：\n1. 关键词后面加上【教程】两个字可以搜索教程，'
                '比如回复 "Django 后台教程"\n'
                '2. 回复任意词语，查天气，陪聊天，讲故事，无所不能！\n'
                '还有更多功能正在开发中哦 ^ _ ^\n'
                '【<a href="http://www.ziqiangxuetang.com">自强学堂手机版</a>】'
            )
        elif content.endswith('教程'):
            reply_text = '您要找的教程如下：'

        response = wechat_instance.response_text(content=reply_text)

    return HttpResponseRedirect(response, content_type="application/xml")
```

下面是一个更详细复杂的使用例子：

### models.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models

class Keyword(models.Model):
    keyword = models.CharField(
        '关键词', max_length=256, primary_key=True, help_text='用户发出的关键词')
    content = models.TextField(
        '内容', null=True, blank=True, help_text='回复给用户的内容')

    pub_date = models.DateTimeField('发表时间', auto_now_add=True)
    update_time = models.DateTimeField('更新时间', auto_now=True, null=True)
    published = models.BooleanField('发布状态', default=True)

    def __unicode__(self):
        return self.keyword

    class Meta:
        verbose_name='关键词'
        verbose_name_plural=verbose_name
```

### views.py

```

# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.http.response import HttpResponseRedirect
from django.views.decorators.csrf import csrf_exempt

from wechat_sdk import WechatBasic
from wechat_sdk.exceptions import ParseError
from wechat_sdk.messages import (TextMessage, VoiceMessage, ImageMessage,
                                VideoMessage, LinkMessage, LocationMessage, EventMessage)
)

from wechat_sdk.context.framework.django import DatabaseContextStore
from .models import KeyWord as KeyWordModel

# 实例化 WechatBasic
wechat_instance = WechatBasic(
    token='zqxt',
    appid='xx',
    appsecret='xx'
)

@csrf_exempt
def index(request):
    if request.method == 'GET':
        # 检验合法性
        # 从 request 中提取基本信息 (signature, timestamp, nonce, xml)
        signature = request.GET.get('signature')
        timestamp = request.GET.get('timestamp')
        nonce = request.GET.get('nonce')

        if not wechat_instance.check_signature(
            signature=signature, timestamp=timestamp, nonce=nonce):
            return HttpResponseRedirect('Verify Failed')

        return HttpResponseRedirect(
            request.GET.get('echostr', ''), content_type="text/plain")

    # POST
    # 解析本次请求的 XML 数据
    try:
        wechat_instance.parse_data(data=request.body)
    except ParseError:
        return HttpResponseRedirect('Invalid XML Data')

    # 获取解析好的微信请求信息
    message = wechat_instance.get_message()
    # 利用本次请求中的用户OpenID来初始化上下文对话
    context = DatabaseContextStore(openid=message.source)

    response = None

    if isinstance(message, TextMessage):
        step = context.get('step', 1) # 当前对话次数, 如果没有则返回 1
        # last_text = context.get('last_text') # 上次对话内容
        content = message.content.strip() # 当前会话内容

        if message.content == '新闻':
            response = wechat_instance.response_news([
                {
                    'title': '自强学堂',
                    'picurl': 'http://www.ziqiangxuetang.com/static/images/newlogo.png',
                    'description': '自强学堂致力于提供优质的IT技术教程, 网页制作, 服务器后台编写, 以及编程语言, 如HTML, JS, Bootstrap, Python, Django. 同时也提供大量在线实例, 通过实例, 学习更容',
                    'url': 'http://www.ziqiangxuetang.com',
                }, {
                    'title': '百度',
                    'picurl': 'http://doraemonext.oss-cn-hangzhou.aliyuncs.com/test/wechat-test.jpg',
                    'url': 'http://www.baidu.com',
                }, {
                    'title': 'Django 教程',
                    'picurl': 'http://www.ziqiangxuetang.com/media/uploads/images/django_logo_20140508_061519_35.jpg',
                    'url': 'http://www.ziqiangxuetang.com/django/django-tutorial.html',
                }
            ])
            return HttpResponseRedirect(response, content_type="application/xml")

        else:
            try:
                keyword_object = KeyWordModel.objects.get(keyword=content)
                reply_text = keyword_object.content
            except KeyWordModel.DoesNotExist:
                try:
                    reply_text = KeyWordModel.objects.get(keyword='提示').content
                except KeyWordModel.DoesNotExist:
                    reply_text = ('/:P- (好委屈, 数据库翻个遍也没找到你输入的关键词! \n'
                                '试试下面这些关键词吧: \nKEYWORD_LIST\n'
                                '<a href="http://www.rxnfinder.org">RxnFinder</a>'
                                '感谢您的支持! /:rose')
                )

            # 将新的数据存入上下文对话中
            context['step'] = step + 1
            context['last_text'] = content
            context.save() # 非常重要! 请勿忘记! 临时数据保存入数据库!

            if 'KEYWORD_LIST' in reply_text:
                keyword_objects = KeyWordModel.objects.exclude(keyword_in=[
                    '关注事件', '测试', 'test', '提示']).filter(published=True)
                keywords = '{}. {}'.format(str(i), k.keyword)
                for i, k in enumerate(keyword_objects, 1))
                reply_text = reply_text.replace(
                    'KEYWORD_LIST', '\n'.join(keywords))

            # 文本消息结束

    elif isinstance(message, VoiceMessage):
        reply_text = '语音信息我听不懂/:P- (/:P- (/:P- ('
    elif isinstance(message, ImageMessage):
        reply_text = '图片信息我也看不懂/:P- (/:P- (/:P- ('
    elif isinstance(message, VideoMessage):
        reply_text = '视频我不会看/:P- ('
    elif isinstance(message, LinkMessage):

```

```
        reply_text = '链接信息'
    elif isinstance(message, LocationMessage):
        reply_text = '地理位置信息'
    elif isinstance(message, EventMessage): # 事件信息
        if message.type == 'subscribe': # 关注事件 (包括普通关注事件和扫描二维码造成的关注事件)
            follow_event = KeywordModel.objects.get(keyword='关注事件')
            reply_text = follow_event.content

            # 如果 key 和 ticket 均不为空, 则是扫描二维码造成的关注事件
            if message.key and message.ticket:
                reply_text += '\n来源: 扫描二维码关注'
            else:
                reply_text += '\n来源: 搜索名称关注'
        elif message.type == 'unsubscribe':
            reply_text = '取消关注事件'
        elif message.type == 'scan':
            reply_text = '已关注用户扫描二维码!'
        elif message.type == 'location':
            reply_text = '上报地理位置'
        elif message.type == 'click':
            reply_text = '自定义菜单点击'
        elif message.type == 'view':
            reply_text = '自定义菜单跳转链接'
        elif message.type == 'templatesendjobfinish':
            reply_text = '模板消息'

    response = wechat_instance.response_text(content=reply_text)
    return HttpResponse(response, content_type="application/xml")
```

[小额赞助, 支持作者!](#)

[« Django 中间件](#)

[Django 单元测试 »](#)

 CODING  
SIMPLY EASY POWERFUL

零成本开启敏捷开发 五人以下团队免费

免费体验

↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 单元测试

[« Python/Django 微信接口 Django CMS »](#)

自强学堂Django一系列教程,前面的例子都是我们写好代码后,运行开发服务器,在浏览器上自己点击测试,看写的代码是否正常,但是这样做很麻烦,因为以后如果有改动,可能会影响以前本来正常的功能,这样以前的功能又得测试一遍,非常不方便,Django中有完善的单元测试,我们可以对开发的每一个功能进行单元测试,这样只要运行一个命令 `python manage.py test`,就可以测试功能是否正常。

一言以蔽之,测试就是检查代码是否按照自己的预期那样运行。

**测试驱动开发:**有时候,我们知道自己需要的功能(结果),并不知道代码如何书写,这时候就可以利用测试驱动开发(Test Driven Development),先写出我们期待得到的结果(把测试代码先写出来),再去完善代码,直到不报错,我们就完成了。

《改善Python的91个建议》一书中说:单元测试绝不是浪费时间的无用功,它是高质量代码的保障之一,在软件开发的一节中值得投入精力和时间去把好这一关。

## 1. Python 中 单元测试简介:

下面是一个Python的单元测试简单的例子:

假如我们开发一个除法的功能,有的同学可能觉得很简单,代码是这样的:

```
def division_funtion(x, y):  
    return x / y
```

但是这样写究竟对还是不对呢,有些同学可以在代码下面这样测试:

```
def division_funtion(x, y):  
    return x / y  
  
if __name__ == '__main__':  
    print division_funtion(2, 1)  
    print division_funtion(2, 4)  
    print division_funtion(8, 3)
```

但是这样运行后得到的结果,自己每次都得算一下去核对一遍,很不方便,Python中有 `unittest` 模块,可以很方便地进行测试,详情可以文章最后的链接,看官网文档的详细介绍。

下面是一个简单的示例:

```
import unittest  
  
def division_funtion(x, y):  
    return x / y  
  
class TestDivision(unittest.TestCase):
```

```

def test_int(self):
    self.assertEqual(division_funtion(9, 3), 3)

def test_int2(self):
    self.assertEqual(division_funtion(9, 4), 2.25)

def test_float(self):
    self.assertEqual(division_funtion(4.2, 3), 1.4)

if __name__ == '__main__':
    unittest.main()

```

我简单地写了三个测试示例（不一定全面，只是示范，比如没有考虑除数是0的情况），运行后发现：

```

F.F
=====
FAIL: test_float (__main__.TestDivision)
-----
Traceback (most recent call last):
  File "/Users/tu/YunPan/mydivision.py", line 16, in test_float
    self.assertEqual(division_funtion(4.2, 3), 1.4)
AssertionError: 1.4000000000000001 != 1.4

=====
FAIL: test_int2 (__main__.TestDivision)
-----
Traceback (most recent call last):
  File "/Users/tu/YunPan/1.py", line 13, in test_int2
    self.assertEqual(division_funtion(9, 4), 2.25)
AssertionError: 2 != 2.25

-----
Ran 3 tests in 0.001s

FAILED (failures=2)

```

汗！发现了没，竟然两个都失败了，测试发现：

4.2除以3 等于 1.4000000000000001 不等于期望值 1.4

9除以4等于2，不等于期望的 2.25

**下面我们就是要修复这些问题，再次运行测试，直到运行不报错为止。**

譬如根据实际情况，假设我们只需要保留到小数点后6位，可以这样改：

```

def division_funtion(x, y):
    return round(float(x) / y, 6)

```

再次运行就不报错了：

```

...
-----
Ran 3 tests in 0.000s

OK

```

Python 单元测试 官方文档：

Python 2 (<https://docs.python.org/2/library/unittest.html>)

Python 3 (<https://docs.python.org/3/library/unittest.html>)

## 2. Django 中 单元测试: (不断完善中, 后期会增加对前面讲解的内容的测试)

### 2.1 简单测试例子:

```
from django.test import TestCase
from myapp.models import Animal

class AnimalTestCase(TestCase):
    def setUp(self):
        Animal.objects.create(name="lion", sound="roar")
        Animal.objects.create(name="cat", sound="meow")

    def test_animals_can_speak(self):
        """Animals that can speak are correctly identified"""
        lion = Animal.objects.get(name="lion")
        cat = Animal.objects.get(name="cat")
        self.assertEqual(lion.speak(), 'The lion says "roar"')
        self.assertEqual(cat.speak(), 'The cat says "meow"')
```

这个例子是测试myapp.models 中的 Animal 类相关的方法功能。

### 2.2 用代码访问网址的方法:

```
>>> from django.test import Client
>>> c = Client()
>>> response = c.post('/login/', {'username': 'john', 'password': 'smith'})
>>> response.status_code
200
>>> response = c.get('/customer/details/')
>>> response.content
'<!DOCTYPE html...'
```

我们可以用 django.test.Client 的实例来实现 get 或 post 内容, 检查一个网址返回的网页源代码。

默认情况下CSRF检查是被禁用的, 如果测试需要, 可以用下面的方法:

```
>>> from django.test import Client
>>> csrf_client = Client(enforce_csrf_checks=True)
```

使用 csrf\_client 这个实例进行请求即可。

### 指定浏览USER-AGENT:

```
>>> c = Client(HTTP_USER_AGENT='Mozilla/5.0')
```

### 模拟post上传附件:

```
from django.test import Client
c = Client()

with open('wishlist.doc') as fp:
    c.post('/customers/wishes/', {'name': 'fred', 'attachment': fp})
```

### 测试网页返回状态:

```
from django.test import TestCase

class SimpleTest(TestCase):
    def test_details(self):
        response = self.client.get('/customer/details/')
        self.assertEqual(response.status_code, 200)

    def test_index(self):
        response = self.client.get('/customer/index/')
        self.assertEqual(response.status_code, 200)
```

我们用 `self.client` 即可，不用 `client = Client()` 这样实例化，更方便，我们还可以继承 `Client`，添加一些其它方法：

```
from django.test import TestCase, Client

class MyTestClient(Client):
    # Specialized methods for your environment
    ...

class MyTest(TestCase):
    client_class = MyTestClient

    def test_my_stuff(self):
        # Here self.client is an instance of MyTestClient...
        call_some_test_code()
```

定制 `self.client` 的方法：

```
from django.test import Client, TestCase

class MyAppTests(TestCase):
    def setUp(self):
        super(MyAppTests, self).setUp()
        self.client = Client(enforce_csrf_checks=True)

    def test_home(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)
```

[小额赞助，支持作者！](#)  
[«Python/Django 微信接口](#)  
[Django CMS »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 单元测试

[« Python/Django 微信接口 Django CMS »](#)

自强学堂Django一系列教程, 前面的例子都是我们写好代码后, 运行开发服务器, 在浏览器上自己点击测试, 看写的代码是否正常, 但是这样做很麻烦, 因为以后如果有改动, 可能会影响以前本来正常的功能, 这样以前的功能又得测试一遍, 非常不方便, Django中有完善的单元测试, 我们可以对开发的每一个功能进行单元测试, 这样只要运行一个命令 `python manage.py test`, 就可以测试功能是否正常。

一言以蔽之, 测试就是检查代码是否按照自己的预期那样运行。

测试驱动开发: 有时候, 我们知道自己需要的功能(结果), 并不知道代码如何书写, 这时候就可以利用测试驱动开发(Test Driven Development), 先写出我们期待得到的结果(把测试代码先写出来), 再去完善代码, 直到不报错, 我们就完成了。

《改善Python的91个建议》一书中说: 单元测试绝不是浪费时间的无用功, 它是高质量代码的保障之一, 在软件开发的一节中值得投入精力和时间去把好这一关。

## 1. Python 中 单元测试简介:

下面是一个 Python 的单元测试简单的例子:

假如我们开发一个除法的功能, 有的同学可能觉得很简单, 代码是这样的:

```
def division_funtion(x, y):  
    return x / y
```

但是这样写究竟对还是不对呢, 有些同学可以在代码下面这样测试:

```
def division_funtion(x, y):  
    return x / y  
  
if __name__ == '__main__':  
    print division_funtion(2, 1)  
    print division_funtion(2, 4)  
    print division_funtion(8, 3)
```

但是这样运行后得到的结果, 自己每次都得算一下去核对一遍, 很不方便, Python中有 `unittest` 模块, 可以很方便地进行测试, 详情可以文章最后的链接, 看官网文档的详细介绍。

下面是一个简单的示例:

```
import unittest  
  
def division_funtion(x, y):  
    return x / y
```

```

class TestDivision(unittest.TestCase):
    def test_int(self):
        self.assertEqual(division_funtion(9, 3), 3)

    def test_int2(self):
        self.assertEqual(division_funtion(9, 4), 2.25)

    def test_float(self):
        self.assertEqual(division_funtion(4.2, 3), 1.4)

if __name__ == '__main__':
    unittest.main()

```

我简单地写了三个测试示例（不一定全面，只是示范，比如没有考虑除数是0的情况），运行后发现：

```

F.F
=====
FAIL: test_float (__main__.TestDivision)
-----
Traceback (most recent call last):
  File "/Users/tu/YunPan/mydivision.py", line 16, in test_float
    self.assertEqual(division_funtion(4.2, 3), 1.4)
AssertionError: 1.4000000000000001 != 1.4

=====
FAIL: test_int2 (__main__.TestDivision)
-----
Traceback (most recent call last):
  File "/Users/tu/YunPan/1.py", line 13, in test_int2
    self.assertEqual(division_funtion(9, 4), 2.25)
AssertionError: 2 != 2.25

-----
Ran 3 tests in 0.001s

FAILED (failures=2)

```

汗！发现了没，竟然两个都失败了，测试发现：

4.2除以3 等于 1.4000000000000001 不等于期望值 1.4

9除以4等于2，不等于期望的 2.25

**下面我们就是要修复这些问题，再次运行测试，直到运行不报错为止。**

譬如根据实际情况，假设我们只需要保留到小数点后6位，可以这样改：

```

def division_funtion(x, y):
    return round(float(x) / y, 6)

```

再次运行就不报错了：

```

...
-----
Ran 3 tests in 0.000s

OK

```

Python 单元测试 官方文档：

Python 2 (<https://docs.python.org/2/library/unittest.html>)

Python 3 (<https://docs.python.org/3/library/unittest.html>)

## 2. Django 中 单元测试: (不断完善中, 后期会增加对前面讲解的内容的测试)

### 2.1 简单测试例子:

```
from django.test import TestCase
from myapp.models import Animal

class AnimalTestCase(TestCase):
    def setUp(self):
        Animal.objects.create(name="lion", sound="roar")
        Animal.objects.create(name="cat", sound="meow")

    def test_animals_can_speak(self):
        """Animals that can speak are correctly identified"""
        lion = Animal.objects.get(name="lion")
        cat = Animal.objects.get(name="cat")
        self.assertEqual(lion.speak(), 'The lion says "roar"')
        self.assertEqual(cat.speak(), 'The cat says "meow"')
```

这个例子是测试myapp.models 中的 Animal类相关的方法功能。

### 2.2 用代码访问网址的方法:

```
>>> from django.test import Client
>>> c = Client()
>>> response = c.post('/login/', {'username': 'john', 'password': 'smith'})
>>> response.status_code
200
>>> response = c.get('/customer/details/')
>>> response.content
'<!DOCTYPE html...'
```

我们可以用 django.test.Client 的实例来实现 get 或 post 内容, 检查一个网址返回的网页源代码。

默认情况下CSRF检查是被禁用的, 如果测试需要, 可以用下面的方法:

```
>>> from django.test import Client
>>> csrf_client = Client(enforce_csrf_checks=True)
```

使用 csrf\_client 这个实例进行请求即可。

指定浏览USER-AGENT:

```
>>> c = Client(HTTP_USER_AGENT='Mozilla/5.0')
```

模拟post上传附件:

```
from django.test import Client
c = Client()

with open('wishlist.doc') as fp:
    c.post('/customers/wishes/', {'name': 'fred', 'attachment': fp})
```

测试网页返回状态:

```
from django.test import TestCase

class SimpleTest(TestCase):
    def test_details(self):
        response = self.client.get('/customer/details/')
        self.assertEqual(response.status_code, 200)

    def test_index(self):
```

```
response = self.client.get('/customer/index/')
self.assertEqual(response.status_code, 200)
```

我们用 `self.client` 即可，不用 `client = Client()` 这样实例化，更方便，我们还可以继承 `Client`，添加一些其它方法：

```
from django.test import TestCase, Client

class MyTestClient(Client):
    # Specialized methods for your environment
    ...

class MyTest(TestCase):
    client_class = MyTestClient

    def test_my_stuff(self):
        # Here self.client is an instance of MyTestClient...
        call_some_test_code()
```

定制 `self.client` 的方法：

```
from django.test import Client, TestCase

class MyAppTests(TestCase):
    def setUp(self):
        super(MyAppTests, self).setUp()
        self.client = Client(enforce_csrf_checks=True)

    def test_home(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)
```

[小额赞助，支持作者！](#)  
[«Python/Django 微信接口](#)  
[Django CMS »](#)



↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

# Django 开发内容管理系统

[«Python/Django 生成二维码](#)  
[Django 开发内容管理系统（第二天）»](#)

用Django开发一个简易的内容管理系统，比如显示新闻的列表，点击进去可以看内容详情等，新闻发布网站。

## 一，搭建互不干扰的 Python 包开发环境

我们有的时候会发现，一个电脑上有多个项目，一个依赖 Django 1.8，另一个比较旧的项目又要用 Django 1.5，这时候怎么办呢？

我们需要一个依赖包管理的工具来处理不同的环境。**如果不想搭建这个环境，可以直接去看 2.2**

### 1.1 环境搭建

开发会用 virtualenv 来管理多个开发环境，virtualenvwrapper 使得virtualenv变得更好用

```
# 安装：  
(sudo) pip install virtualenv virtualenvwrapper
```

Linux/Mac OSX 下：

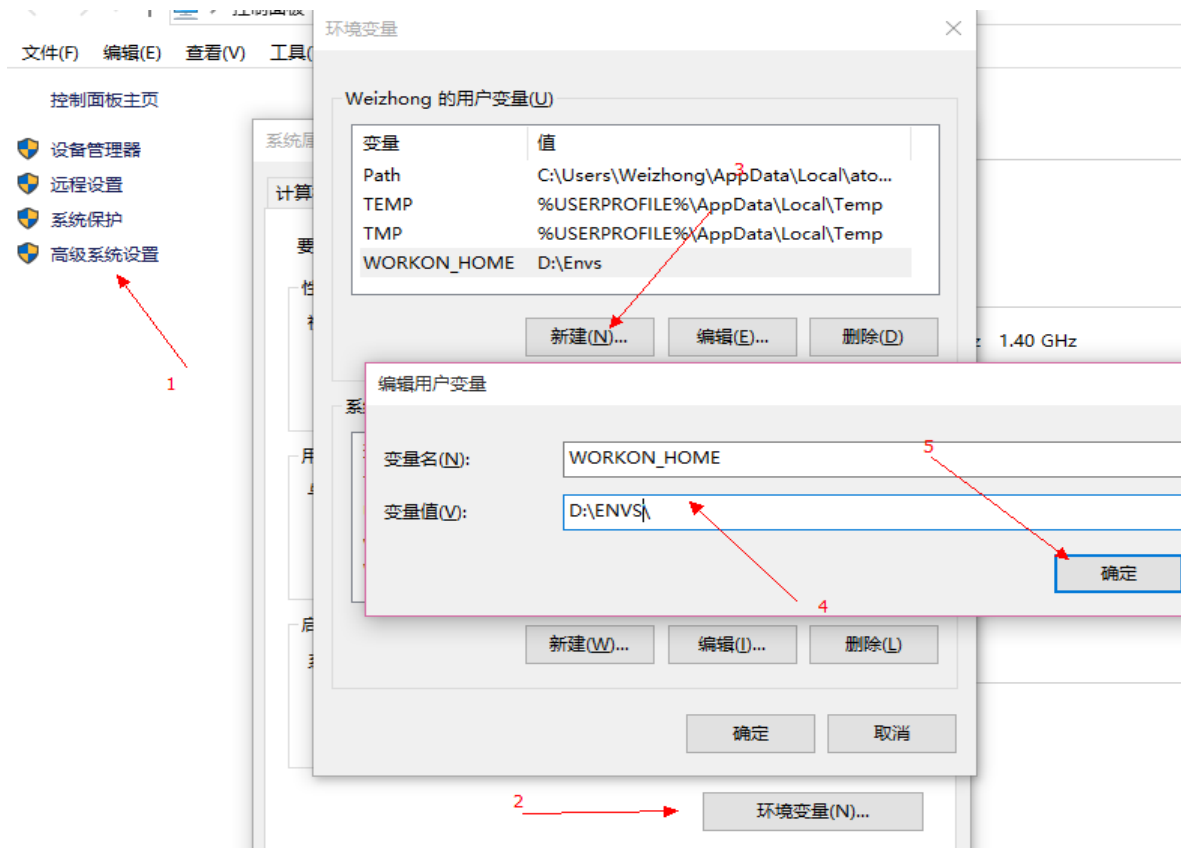
修改~/.bash\_profile或其它环境变量相关文件，添加以下语句

```
export WORKON_HOME=$HOME/.virtualenvs  
export PROJECT_HOME=$HOME/workspace  
source /usr/local/bin/virtualenvwrapper.sh
```

Windows 下：

```
pip install virtualenvwrapper-win
```

**【可选】**Windows下默认虚拟环境是放在用户名下面的Envs中的，与桌面，我的文档，下载等文件夹在一块的。更改方法：计算机，属性，高级系统设置，环境变量，添加WORKON\_HOME，如图（windows 10 环境变量设置截图）：



## 1.2 使用方法:

**mkvirtualenv zqxt:** 创建运行环境zqxt

**workon zqxt:** 工作在 zqxt 环境

其它的:

**rmvirtualenv ENV:** 删除运行环境ENV

**mkproject mic:** 创建mic项目和运行环境mic

**mktmpenv:** 创建临时运行环境

**lsvirtualenv:** 列出可用的运行环境

**lssitepackages:** 列出当前环境安装了的包

创建的环境是独立的，互不干扰，无需sudo权限即可使用 pip 来进行包的管理。

## 二，安装软件，开发 minicms 项目

### 2.1 创建一个开发环境 minicms

```
# Linux 或 Mac OSX
mkproject minicms
```

```
# windows
mkvirtualenv minicms
```

**Mac OSX 下的输出示例:**

```
mkproject minicms
```

```
New python executable in minicms/bin/python
Installing setuptools, pip...done.
Creating /Users/tu/YunPan/workspace//minicms
Setting project for minicms to /Users/tu/YunPan/workspace/minicms
(minicms)tu@mac ~/YunPan/workspace/minicms $
```

## 2.2 安装 Django

```
pip install Django==1.8.3
```

## 2.3 创建项目 minicms 和应用 news

```
django-admin.py startproject minicms
cd minicms
python manage.py startapp news
```

添加 `news` 到 `settings.py` 中的 `INSTALLED_APPS` 中。

## 2.4 规划 news 中的栏目和每篇文章相关的字段

栏目：名称，网址，简介等

文章：标题，作者，网址，内容等

我们假设一篇文章只有一个作者（文章和作者是多对一的关系），一篇文章可以属于多个栏目（栏目和文章是多对多的关系）

为了用到更多的情况，我们假设作者可以为空，栏目不能为空。

开写 `models.py`

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible
class Column(models.Model):
    name = models.CharField('栏目名称', max_length=256)
    slug = models.CharField('栏目网址', max_length=256, db_index=True)
    intro = models.TextField('栏目简介', default='')

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = '栏目'
        verbose_name_plural = '栏目'
        ordering = ['name']  # 按照哪个栏目排序

@python_2_unicode_compatible
class Article(models.Model):
```

```

column = models.ManyToManyField(Column, verbose_name='归属栏目')

title = models.CharField('标题', max_length=256)
slug = models.CharField('网址', max_length=256, db_index=True)

author = models.ForeignKey('auth.User', blank=True, null=True, verbose_name='作者')
content = models.TextField('内容', default='', blank=True)

published = models.BooleanField('正式发布', default=True)

def __str__(self):
    return self.title

class Meta:
    verbose_name = '教程'
    verbose_name_plural = '教程'

```

## 2.5 创建数据库

```

python manage.py makemigrations news
python manage.py migrate

```

## 2.6 创建完数据库后，用了一段时间，我们发现以前的文章的字段不合理

比如我们想记录文章添加的日期，修改的日期，我们更改 `models.py`（不变动的大部分省去了，添加两个字段）

```

...省略
class Article(models.Model):
    ...原来的字段省去

    pub_date = models.DateTimeField('发表时间', auto_now_add=True, editable=True)
    update_time = models.DateTimeField('更新时间', auto_now=True, null=True)

...省略

```

这时候，我们对 `models.py` 进行了更改，这些字段数据库中还没有，我们要同步更改到数据库中去：

```
python manage.py makemigrations news
```

You are trying to add a non-nullable field 'pub\_date' to article without a default; we can't do that (the database needs something to populate existing rows).

Please select a fix:

- 1) Provide a one-off default now (will be set on all existing rows)
- 2) Quit, and let me add a default in models.py

这段话的意思是 `pub_date` 字段没有默认值，而且非Null 那么

- 1) 指定一个一次性的值供更改数据库时使用。
- 2) 停止当前操作，在 `models.py` 中给定默认值，然后再来 `migrate`。

我们选择第一个，输入 1

Select an option: 1

Please enter the default value now, as valid Python

The datetime and django.utils.timezone modules are available, so you can do e.g. `timezone.now()`

```
>>> timezone.now()
```

Migrations for 'news':



0002\_auto\_20150728\_1232.py:

- Add field pub\_date to article
- Add field update\_time to article

这样是生成了一个对表进行更改的 py 文件在 news/migrations 文件夹中，我们要执行更改

python manage.py migrate 或 python manage.py migrate news

2.7 创建一个脚本，导入一些数据到数据库中

我们导入一些演示数据：

栏目： [<Column: 体育新闻>, <Column: 社会新闻>, <Column: 科技新闻>]

文章： [<Article: 体育新闻\_1>, <Article: 体育新闻\_2>, <Article: 体育新闻\_3>, <Article: 体育新闻\_4>, <Article: 体育新闻\_5>, <Article: 体育新闻\_6>, <Article: 体育新闻\_7>, <Article: 体育新闻\_8>, <Article: 体育新闻\_9>, <Article: 体育新闻\_10>, <Article: 社会新闻\_1>, <Article: 社会新闻\_2>,...(remaining elements truncated)...]

create\_demo\_records.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Date      : 2015-07-28 20:38:38
# @Author    : Weizhong Tu (mail@tuweizhong.com)
# @Link      : http://www.tuweizhong.com

'''
create some records for demo database
'''

from minicms.wsgi import *
from news.models import Column, Article

def main():
    columns_urls = [
        ('体育新闻', 'sports'),
        ('社会新闻', 'society'),
        ('科技新闻', 'tech'),
    ]

    for column_name, url in columns_urls:
        c = Column.objects.get_or_create(name=column_name, slug=url)[0]

        # 创建 10 篇新闻
        for i in range(1, 11):
            article = Article.objects.get_or_create(
                title='{}_{}'.format(column_name, i),
                slug='article_{}'.format(i),
                content='新闻详细内容: {} {}'.format(column_name, i)
            )[0]

            article.column.add(c)

if __name__ == '__main__':
    main()
    print("Done!")
```

假设这个文件被保存为 create\_demo\_records.py（和 manage.py 放在一块，同一个文件夹下）

运行脚本 导入数据：

```
python create_demo_records.py
```

Done!

终端上显示一个 Done! 就这样 Duang 的一下，数据就导进去了！

上面所有操作的代码：[minicms1.zip](#)

第一次提交：github: <https://github.com/twz915/django-minicms/tree/dff31758173852344af5d8d5b4fad858a0b16907>

内容管理系统继续开发，[点此查看第二部分](#)

[小额赞助，支持作者！](#)

[«Python/Django 生成二维码](#)

[Django 开发内容管理系统（第二天）»](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 开发内容管理系统

[«Python/Django 生成二维码](#)

[Django 开发内容管理系统（第二天）»](#)

用Django开发一个简易的内容管理系统，比如显示新闻的列表，点击进去可以看内容详情等，新闻发布网站。

## 一，搭建互不干扰的 Python 包开发环境

我们有的时候会发现，一个电脑上有多个项目，一个依赖 Django 1.8，另一个比较旧的项目又要用 Django 1.5，这时候怎么办呢？

我们需要一个依赖包管理的工具来处理不同的环境。不想搭建这个环境，可以直接去看 2.2

### 1.1 环境搭建

开发会用 virtualenv 来管理多个开发环境，virtualenvwrapper 使得 virtualenv 变得更好用

# 安装：

```
(sudo) pip install virtualenv virtualenvwrapper
```

Linux/Mac OSX 下：

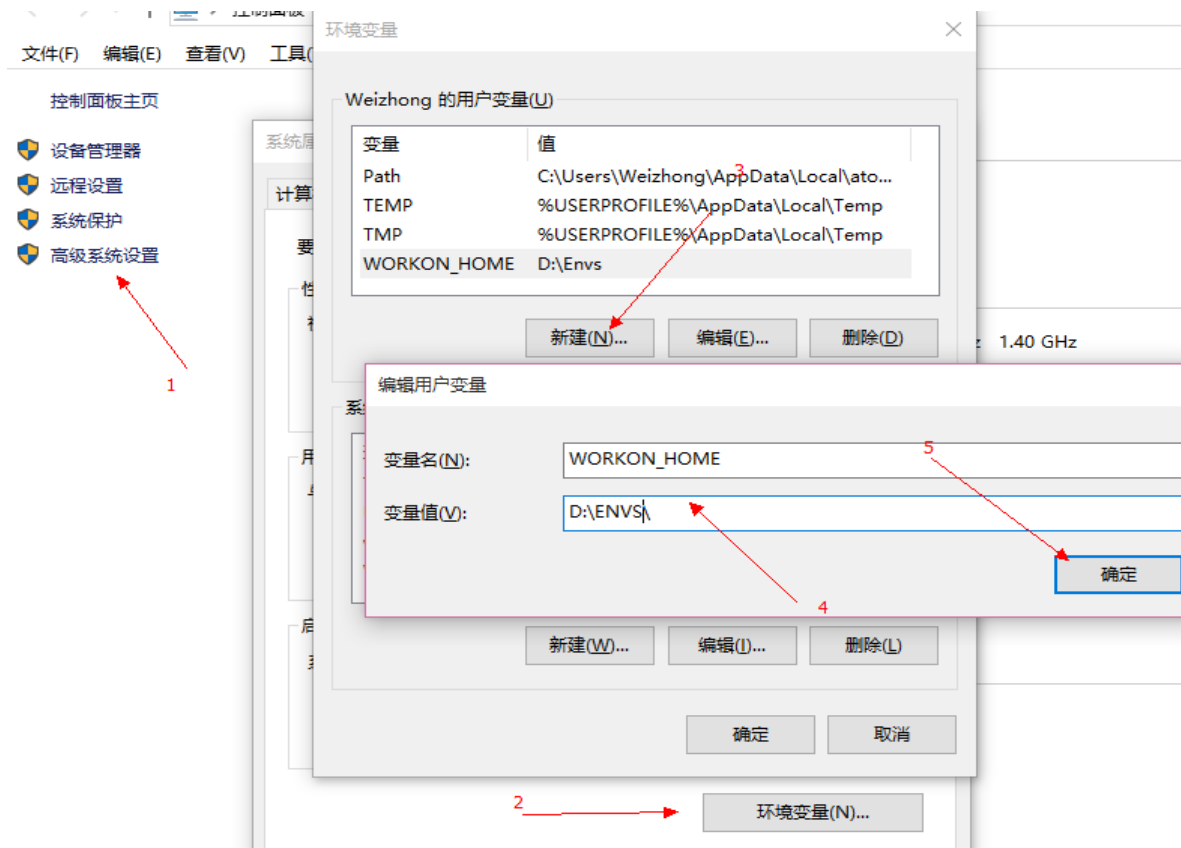
修改 ~/.bash\_profile 或其它环境变量相关文件，添加以下语句

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/workspace
source /usr/local/bin/virtualenvwrapper.sh
```

Windows 下：

```
pip install virtualenvwrapper-win
```

【可选】Windows 下默认虚拟环境是放在用户名下面的 Envs 中的，与桌面，我的文档，下载等文件夹在一块的。更改方法：计算机，属性，高级系统设置，环境变量，添加 WORKON\_HOME，如图（windows 10 环境变量设置截图）：



### 1.2 使用方法：

**mkvirtualenv zqxt:** 创建运行环境 zqxt

**workon zqxt:** 工作在 **zqxt** 环境

其它的:

**rmvirtualenv** ENV: 删除运行环境ENV

**mkproject mic:** 创建mic项目和运行环境mic

**mktmpenv:** 创建临时运行环境

**lsvirtualenv:** 列出可用的运行环境

**lsitepackages:** 列出当前环境安装了了的包

创建的环境是独立的，互不干扰，无需sudo权限即可使用 **pip** 来进行包的管理。

## 二，安装软件，开发 **minicms** 项目

### 2.1 创建一个开发环境 **minicms**

```
# Linux 或 Mac OSX
mkproject minicms

# windows
mkvirtualenv minicms
```

**Mac OSX** 下的输出示例:

```
mkproject minicms

New python executable in minicms/bin/python
Installing setuptools, pip...done.
Creating /Users/tu/YunPan/workspace//minicms
Setting project for minicms to /Users/tu/YunPan/workspace/minicms
(minicms)tu@mac ~/YunPan/workspace/minicms $
```

### 2.2 安装 Django

```
pip install Django==1.8.3
```

### 2.3 创建项目 **minicms** 和应用 **news**

```
django-admin.py startproject minicms
cd minicms
python manage.py startapp news
```

添加 **news** 到 **settings.py** 中的 **INSTALLED\_APPS** 中。

### 2.4 规划 **news** 中的栏目和每篇文章相关的字段

栏目: 名称, 网址, 简介等

文章: 标题, 作者, 网址, 内容等

我们假设一篇文章只有一个作者（文章和作者是多对一的关系），一篇文章可以属于多个栏目（栏目和文章是多对多的关系）

为了用到更多的情况，我们假设作者可以为空，栏目不能为空。

## 开写 models.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible
class Column(models.Model):
    name = models.CharField('栏目名称', max_length=256)
    slug = models.CharField('栏目网址', max_length=256, db_index=True)
    intro = models.TextField('栏目简介', default='')

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = '栏目'
        verbose_name_plural = '栏目'
        ordering = ['name'] # 按照哪个栏目排序

@python_2_unicode_compatible
class Article(models.Model):
    column = models.ManyToManyField(Column, verbose_name='归属栏目')

    title = models.CharField('标题', max_length=256)
    slug = models.CharField('网址', max_length=256, db_index=True)

    author = models.ForeignKey('auth.User', blank=True, null=True, verbose_name='作者')
    content = models.TextField('内容', default='', blank=True)

    published = models.BooleanField('正式发布', default=True)

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = '教程'
        verbose_name_plural = '教程'
```

## 2.5 创建数据库

```
python manage.py makemigrations news
python manage.py migrate
```

## 2.6 创建完数据库后，用了一段时间，我们发现以前的文章的字段不合理

比如我们想记录文章添加的日期，修改的日期，我们更改 **models.py**（不变动的大部分省去了，添加两个字段）

```
...省略
class Article(models.Model):
    ...原来的字段省去

    pub_date = models.DateTimeField('发表时间', auto_now_add=True, editable=True)
    update_time = models.DateTimeField('更新时间', auto_now=True, null=True)

    ...省略
```

这时候，我们对 **models.py** 进行了更改，这些字段数据库中还没有，我们要同步更改到数据库中去：

```
python manage.py makemigrations news
```

You are trying to add a non-nullable field 'pub\_date' to article without a default; we can't do that (the database needs something to populate existing rows).

Please select a fix:

1) Provide a one-off default now (will be set on all existing rows)

2) Quit, and let me add a default in models.py

这段话的意思是 `pub_date` 字段没有默认值，而且非Null 那么

1) 指定一个一次性的值供更改数据库时使用。

2) 停止当前操作，在 `models.py` 中给定默认值，然后再来migrate。

我们选择第一个，输入 1

Select an option: 1

Please enter the default value now, as valid Python

The datetime and django.utils.timezone modules are available, so you can do e.g. `timezone.now()`

```
>>> timezone.now()
```

Migrations for 'news':

**0002\_auto\_20150728\_1232.py:**

- Add field `pub_date` to article
- Add field `update_time` to article

这样是生成了一个对表进行更改的 `py` 文件在 `news/migrations` 文件夹中，我们要执行更改

`python manage.py migrate` 或 `python manage.py migrate news`

2.7 创建一个脚本，导入一些数据到数据库中

我们导入一些演示数据：

栏目： [`<Column: 体育新闻>`, `<Column: 社会新闻>`, `<Column: 科技新闻>`]

文章： [`<Article: 体育新闻_1>`, `<Article: 体育新闻_2>`, `<Article: 体育新闻_3>`, `<Article: 体育新闻_4>`, `<Article: 体育新闻_5>`, `<Article: 体育新闻_6>`, `<Article: 体育新闻_7>`, `<Article: 体育新闻_8>`, `<Article: 体育新闻_9>`, `<Article: 体育新闻_10>`, `<Article: 社会新闻_1>`, `<Article: 社会新闻_2>`, '...(remaining elements truncated)...']

`create_demo_records.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Date      : 2015-07-28 20:38:38
# @Author    : Weizhong Tu (mail@tuweizhong.com)
# @Link      : http://www.tuweizhong.com

'''
create some records for demo database
'''

from minicms.wsgi import *
from news.models import Column, Article
```

```
def main():
    columns_urls = [
        ('体育新闻', 'sports'),
        ('社会新闻', 'society'),
        ('科技新闻', 'tech'),
    ]

    for column_name, url in columns_urls:
        c = Column.objects.get_or_create(name=column_name, slug=url)[0]

        # 创建 10 篇新闻
        for i in range(1, 11):
            article = Article.objects.get_or_create(
                title='{}_{}'.format(column_name, i),
                slug='article_{}'.format(i),
                content='新闻详细内容: {} {}'.format(column_name, i)
            )[0]

            article.column.add(c)

if __name__ == '__main__':
    main()
    print("Done!")
```

假设这个文件被保存为 `create_demo_records.py`（和 `manage.py` 放在一块，同一个文件夹下）

运行脚本 导入数据：

```
python create_demo_records.py
```

Done!

终端上显示一个 Done! 就这样 Duang 的一下，数据就导进去了！

上面所有操作的代码：[minicms1.zip](#)

第一次提交：[github: https://github.com/twz915/django-minicms/tree/df31758173852344af5d8d5b4fad858a0b16907](https://github.com/twz915/django-minicms/tree/df31758173852344af5d8d5b4fad858a0b16907)

内容管理系统继续开发，[点此查看第二部分](#)

[小额赞助，支持作者！](#)

[«Python/Django 生成二维码](#)

[Django 开发内容管理系统（第二天）»](#)



**CODING**  
LOW-DEVELOPMENT

**零成本开启敏捷开发 五人以下团队免费**

**免费体验**

↑ 上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

# BVDN-1 这个教程要做什么

[« Django QuerySet 进阶](#)  
[BVDN-2 环境搭建 »](#)

作者：邵靖隆（自强学堂 Django 受益用户，回馈教程）

作者授权自强学堂发布。他的目标：全栈！技术主管兼产品经理！

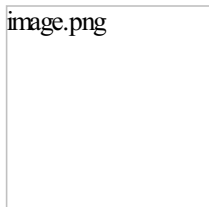
这个教程的目标是让没有任何基础的人，也可以按照本教程建设出一个**真正的网站**，期间不用苦苦的查资料，找资源，或者为一个莫名其妙的bug搞得焦头烂额，从而把精力集中在自己想做的事情上。

本教程的目标是建出网站，而不是让你学会某一个具体的工具，所以通过这个教程，你将学会搭建网站所需要用到的所有东西。

## 什么是BVDN

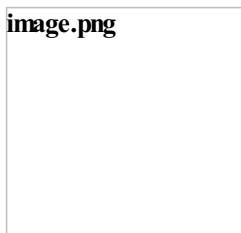
BVDN指的是Bootstrap Vue Django Nginx，其中：

**Bootstrap**负责界面，也就是用户直接看到的部分，**Bootstrap**是目前特别火的前端框架，许多网站的页面都是基于**Bootstrap**编写的，在使用它的过程中你一定不会感到陌生（换句话说，用了**Bootstrap**，你写出的网页也和那些网站的页面处在差不多的水平）。



Bootstrap

**Vue**负责前端的运作，也就是网页内的代码。可以这么说：没有**Bootstrap**的网页是简陋的，你甚至不能称之为网页；而只有**Bootstrap**的网页是死的，静止的，当**Bootstrap**遇到了**Vue**，网页就有了生命，活了！



Vue

**Django**是后端框架，负责网站后端的运作。没有**Django**，你写的东西只能叫网页，有了它，才能称之为网站，因为后端才是一个网站的根本！**Django**基于**Python**，而**Python**是当今最容易学的编程语言，而且**Python**近几年越来越普及，教程到处都是，出了什么bug立刻就能查到原因！如果你不了解**Python**，也没有关系，因为这是个傻瓜式的教程，所有的细节都会有所交代。**Django**是**Python**里最简单的后端框架，而**Python**又是最简单的编程语言，可想而知这个教程到底有多简单！





## Django

Nginx负责把你在电脑上用Django建出的后台程序部署在服务器上。没有它，你建的网站只能在自己的个人电脑上的Python环境里跑着玩，访问量一旦变多，网站就会崩溃，始终是个玩具。而有了Nginx，你的网站就可以进行强大的高并发处理，可以承受的起巨大的访问量和请求数量，从跑在Python环境下的小玩具变成一个运行在服务器上的真正的网站。



## Nginx

这四个东西本身也是比较成熟的框架，需要自己做的内容并不多，用上这个组合，我们可以只专注于自己想做的事情，而不在代码本身上面浪费时间和精力。

我个人的理解：把网站类比作剧场，用户看到的部分是舞台，是台上的演员和布景，这一块由Bootstrap负责。Vue负责的是幕后，比如舞台上的场景和配音，这些总得有人去操作，而Vue就是干这个的。参与表演的人员（包括台前和幕后）统称前端。

后端，则可以称作“幕后的幕后”，或者说剧场的办公区，以及售票厅，导游之类。至于数据库，则像是道具仓库。

回到一开始所说的：本教程的目标是建出网站，而不是让你学会某一个具体的工具，所以通过这个教程，你将学会搭建网站所需要用到的所有东西。相信读过这个教程之后，人人都可以立刻建出一个漂亮、功能强大、性能优越的网站。

BVDN，傻子也能建网站。

[小额赞助，支持作者！](#)

[« Django QuerySet 进阶](#)

[BVDN-2 环境搭建 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#) ECS



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## BVDN-1 这个教程要做什么

[« Django QuerySet 进阶](#)  
[BVDN-2 环境搭建 »](#)

作者: 邵靖隆 (自强学堂 Django受益用户, 回馈教程)

作者授权自强学堂发布。他的目标: 全栈! 技术主管兼产品经理!

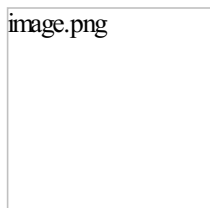
这个教程的目标是让没有任何基础的人, 也可以按照本教程建设出一个**真正的网站**, 期间不用苦苦的查资料, 找资源, 或者为一个莫名其妙的bug搞得焦头烂额, 从而把精力集中在自己想做的事情上。

本教程的目标是建出网站, 而不是让你学会某一个具体的工具, 所以通过这个教程, 你将学会搭建网站所需要用到的所有东西。

### 什么是BVDN

BVDN指的是Bootstrap Vue Django Nginx, 其中:

**Bootstrap**负责界面, 也就是用户直接看到的部分, **Bootstrap**是目前特别火的前端框架, 许多网站的页面都是基于**Bootstrap**编写的, 在使用它的过程中你一定不会感到陌生(换句话说, 用了**Bootstrap**, 你写出的网页也和那些网站的页面处在差不多的水平)。



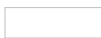
Bootstrap

**Vue**负责前端的运作, 也就是网页内的代码。可以这么说: 没有**Bootstrap**的网页是简陋的, 你甚至不能称之为网页; 而只有**Bootstrap**的网页是死的, 静止的, 当**Bootstrap**遇到了**Vue**, 网页就有了生命, 活了!



Vue

Django是后端框架，负责网站后端的运作。没有Django，你写的东西只能叫网页，有了它，才能称之为网站，因为后端才是一个网站的根本！Django基于Python，而Python是当今最容易学的编程语言，而且Python近几年越来越普及，教程到处都是，出了什么bug立刻就能查到原因！如果你不了解Python，也没有关系，因为这是个傻瓜式的教程，所有的细节都会有所交代。Django是Python里最简单的后端框架，而Python又是最简单的编程语言，可想而知这个教程到底有多简单！



Django

Nginx负责把你在电脑上用Django建出的后台程序部署在服务器上。没有它，你建的网站只能在自己的个人电脑上的Python环境里跑着玩，访问量一旦变多，网站就会崩溃，始终是个玩具。而有了Nginx，你的网站就可以进行强大的高并发处理，可以承受的起巨大的访问量和请求数量，从跑在Python环境下的小玩具变成一个运行在服务器上的真正的网站。



Nginx

这四个东西本身也是比较成熟的框架，需要自己做的内容并不多，用上这个组合，我们可以只专注于自己想做的事情，而不在代码本身上面浪费时间和精力。

我个人的理解：把网站类比作剧场，用户看到的部分是舞台，是台上的演员和布景，这一块由Bootstrap负责。Vue负责的是幕后，比如舞台上的场景和配音，这些总得有人去操作，而Vue就是干这个的。参与表演的人员（包括台前和幕后）统称前端。

后端，则可以称作“幕后的幕后”，或者说剧场的办公区，以及售票厅，导游之类。至于数据库，则像是道具仓库。

回到一开始所说的：本教程的目标是建出网站，而不是让你学会某一个具体的工具，所以通过这个教程，你将学会搭建网站所需要用到的所有东西。相信读过这个教程之后，人人都可以立刻建出一个漂亮、功能强大、性能优越的网站。

BVDN，傻子也能建网站。

[小额赞助，支持作者！](#)

[« Django QuerySet 进阶](#)

[BVDN-2 环境搭建 »](#)

 CODING  
CLOUD DEVELOPMENT

零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django CMS

[« Django 单元测试](#)  
[Python/Django 生成二维码 »](#)

CMS 的意思是 Content Management System 内容管理系统, 一般拿就可以使用, 不会编程也能做出网站来, 还可以在原来的基础上再次开发, 减少工作量, 这里列举了一些出名的 CMS:

- [Opps](#) - A content management platform built for large portals.
- [django-cms](#) - The easy-to-use and developer-friendly CMS.

这个不是为初学者开发的, 想在基础上开发需要对Django比较了解才行, 是一个基本的框架。

- [mezzanine](#) - A content management platform built using the Django framework.  
简单易用, 自带Blog和用户注册系统, 拿来就可以用
- [wagtail](#) - A new Django content management system.
- [django-fiber](#) - Django Fiber, a simple, user-friendly CMS for all your Django projects

[小额赞助, 支持作者!](#)  
[« Django 单元测试](#)  
[Python/Django 生成二维码 »](#)

↑ 上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习, 测试和培训。实例可能为了更容易理解而简化。我们一直对教程, 参考手册, 在线实例保持修订, 但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时, 代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的, 对任何法律问题及风险不承担任何责任。版权所有, 保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点, 托管在[阿里云ECS](#)

 CODING  
CLOUD DEPLOYMENT

零成本开启敏捷开发 五人以下团队免费

免费体验

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django CMS

[« Django 单元测试](#)  
[Python/Django 生成二维码 »](#)

CMS 的意思是 Content Management System 内容管理系统, 一般拿就可以使用, 不会编程也能做出网站来, 还可以在原来的基础上再次开发, 减少工作量, 这里列举了一些出名的 CMS:

- [Opps](#) - A content management platform built for large portals.
- [django-cms](#) - The easy-to-use and developer-friendly CMS.

这个不是为初学者开发的, 想在基础上开发需要对Django比较了解才行, 是一个基本的框架。

- [mezzanine](#) - A content management platform built using the Django framework.  
简单易用, 自带Blog和用户注册系统, 拿来就可以用
- [wagtail](#) - A new Django content management system.
- [django-fiber](#) - Django Fiber, a simple, user-friendly CMS for all your Django projects

[小额赞助, 支持作者!](#)  
[« Django 单元测试](#)  
[Python/Django 生成二维码 »](#)

 CODING  
CLOUD DEPLOYMENT

零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商, 非常荣幸有他们支持自强学堂的发展, 感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Python/Django 生成二维码

[« Django APP 推荐](#)  
[Django 开发内容管理系统 »](#)

## 一, 包的安装和简单使用

1.1 用Python来生成二维码很简单, 可以看 `qrcode` 这个包:

```
pip install qrcode
```

`qrcode` 依赖 `Image` 这个包:

```
pip install Image
```

如果这个包安装有困难, 可选纯Python的包来实现此功能, 见下文。

1.2 安装后就可以使用了, 这个程序带了一个 `qr` 命令:

```
qr 'http://www.ziqiangxuetang.com' > test.png
```

1.3 下面我们看一下如何在 代码 中使用

```
import qrcode

img = qrcode.make('http://www.tuweizhong.com')
# img <qrcode.image.pil.PilImage object at 0x1044ed9d0>

with open('test.png', 'wb') as f:
    img.save(f)
```

这样就可以生成一个带有网址的二维码, 但是这样得把文件保存到硬盘中。

**【备注】:** 纯Python的包的使用:

安装:

```
pip install git+git://github.com/ojii/pymaging.git#egg=pymaging
pip install git+git://github.com/ojii/pymaging-png.git#egg=pymaging-png
```

使用方法大致相同, 命令行上:

```
qr --factory=pymaging "Some text" > test.png
```

Python中调用:

```
import qrcode
from qrcode.image.pure import PymagingImage
img = qrcode.make('Some data here', image_factory=PymagingImage)
```

## 二, Django 中使用

我们可以用 Django 直接把生成的内容返回到网页, 以下是操作过程:

2.1 新建一个 `zqxtqrcode` 项目, `tools` 应用:

```
django-admin.py startproject zqxtqrcode
python manage.py startapp tools
```

## 2.2 将 tools 应用 添加到 项目 settings.py 中

```
INSTALLED_APPS = (
    ...
    'tools',
)
```

## 2.3 我们修改 tools/views.py

```
from django.http import HttpResponse
import qrcode
from django.utils.six import BytesIO

def generate_qrcode(request, data):
    img = qrcode.make(data)

    buf = BytesIO()
    img.save(buf)
    image_stream = buf.getvalue()

    response = HttpResponse(image_stream, content_type="image/png")
    return response
```

## 2.4 添加视图函数到 zqxtqrcode/urls.py

```
url(r'^qrcode/(.+)$', 'tools.views.generate_qrcode', name='qrcode'),
```

## 2.5 同步数据库，打开开发服务器：

```
python manage.py syncdb
python manage.py runserver
```


打开：<http://127.0.0.1:8000/qrcode/http://www.tuweizhong.com> 就可以看到如下效果：



这样生成二维码的接口就写好了^^，实例采用的是返回图片流的方式，这样不用写文件到硬盘，接口调用更方便，如果要加速，可以用[Django缓存](#)来实现。

源代码下载：

基于 Django 1.8, tools app 可以在 Django 1.4-Django1.8之间使用，更低版本的自测，应该也没什么问题，建议按教程步骤来一遍，这样学的更好。Django 1.8 以上版本按照教程来也可以使用。

 [zqxtqrcode.zip](#) （更新于 2017-06-10 23:01:49）

[小额赞助，支持作者！](#)

[« Django APP 推荐](#)

[Django 开发内容管理系统 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

---

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在



线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Python/Django 生成二维码

[« Django APP 推荐](#)  
[Django 开发内容管理系统 »](#)

### 一，包的安装和简单使用

1.1 用Python来生成二维码很简单，可以看 `qrcode` 这个包：

```
pip install qrcode
```

`qrcode` 依赖 `Image` 这个包：

```
pip install Image
```

如果这个包安装有困难，可选纯Python的包来实现此功能，见下文。

1.2 安装后就可以使用了，这个程序带了一个 `qr` 命令：

```
qr 'http://www.ziqiangxuetang.com' > test.png
```

1.3 下面我们看一下如何在 代码 中使用

```
import qrcode

img = qrcode.make('http://www.tuweizhong.com')
# img <qrcode.image.pil.PilImage object at 0x1044ed9d0>

with open('test.png', 'wb') as f:
    img.save(f)
```

这样就可以生成一个带有网址的二维码，但是这样得把文件保存到硬盘中。

**【备注】：**纯Python的包的使用：

安装：

```
pip install git+git://github.com/ojii/pymaging.git#egg=pymaging
pip install git+git://github.com/ojii/pymaging-png.git#egg=pymaging-png
```

使用方法大致相同，命令行上：

```
qr --factory=pymaging "Some text" > test.png
```

Python中调用：

```
import qrcode
```

```
from qrcode.image.pure import PymagingImage
img = qrcode.make('Some data here', image_factory=PymagingImage)
```

## 二， Django 中使用

我们可以用 Django 直接把生成的内容返回到网页，以下是操作过程：

### 2.1 新建一个 zqxtqrcode 项目， tools 应用：

```
django-admin.py startproject zqxtqrcode
python manage.py startapp tools
```

### 2.2 将 tools 应用 添加到 项目 settings.py 中

```
INSTALLED_APPS = (
    ...
    'tools',
)
```

### 2.3 我们修改 tools/views.py

```
from django.http import HttpResponse
import qrcode
from django.utils.six import BytesIO

def generate_qrcode(request, data):
    img = qrcode.make(data)

    buf = BytesIO()
    img.save(buf)
    image_stream = buf.getvalue()

    response = HttpResponse(image_stream, content_type="image/png")
    return response
```

### 2.4 添加视图函数到 zqxtqrcode/urls.py

```
url(r'^qrcode/(.+)$', 'tools.views.generate_qrcode', name='qrcode'),
```

### 2.5 同步数据库， 打开开发服务器：

```
python manage.py syncdb
python manage.py runserver
```


打开：<http://127.0.0.1:8000/qrcode/http://www.tuweizhong.com> 就可以看到如下效果：



这样生成二维码的接口就写好了^\_^，实例采用的是返回图片流的方式，这样不用写文件到硬盘，接口调用更方便，如果要加速，可以用[Django缓存](#)来实现。

源代码下载：

基于 Django 1.8，tools app 可以在 Django 1.4-Django1.8之间使用，更低版本的自测，应该也没什么问题，建议按教程步骤来一遍，这样学的更好。Django 1.8 以上版本按照教程来也可以使用。

 [zqxtqrcode.zip](#) （更新于 2017-06-10 23:01:49）

[小额赞助，支持作者！](#)

[« Django APP 推荐](#)

[Django 开发内容管理系统 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 简介

[« Django 2.2 基础教程](#)  
[Django 环境搭建 »](#)

自强学堂的django教程将节省你大量的时间, 并且使你的web开发充满乐趣。通过Django, 你可以建立一个高性能的web应用而只花费最少的时间和精力。

Django 中提供了开发网站经常用到的模块, 常见的代码都为你写好了, 通过减少重复的代码, Django 使你能够专注于web 应用上有趣的关键性的东西。为了达到这个目标, Django 提供了通用Web开发模式的高度抽象, 提供了频繁进行的编程作业的快速解决方法, 以及为“如何解决问题”提供了清晰明了的约定。Django的理念是DRY(Don't Repeat Yourself)来鼓励快速开发!

## 让我们一览 Django 全貌

### [urls.py](#)

网址入口, 关联到对应的views.py中的一个函数(或者generic类), 访问网址就对应一个函数。

### [views.py](#)

处理用户发出的请求, 从urls.py中对应过来, 通过渲染templates中的网页可以将显示内容, 比如登陆后的用户名, 用户请求的数据, 输出到网页。

### [models.py](#)

与数据库操作相关, 存入或读取数据时用到这个, 当然用不到数据库的时候 你可以不使用。

### [forms.py](#)

表单, 用户在浏览器上输入数据提交, 对数据的验证工作以及输入框的生成等工作, 当然你也可以不使用。

### templates 文件夹

views.py 中的函数渲染templates中的Html模板, 得到动态内容的网页, 当然可以用缓存来提高速度。

### [admin.py](#)

后台, 可以用很少量的代码就拥有一个强大的后台。

### [settings.py](#)

Django 的设置, 配置文件, 比如 DEBUG 的开关, 静态文件的位置等。

[小额赞助, 支持作者!](#)  
[« Django 2.2 基础教程](#)  
[Django 环境搭建 »](#)



零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



零成本开启敏捷开发 五人以下团队免费

免费体验

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 简介

[« Django 2.2 基础教程](#)

[Django 环境搭建 »](#)

自强学堂的django教程将节省你大量的时间，并且使你的web开发充满乐趣。通过Django，你可以建立一个高性能的web应用而只花费最少的时间和精力。

Django 中提供了开发网站经常用到的模块，常见的代码都为你写好了，通过减少重复的代码，Django 使你能够专注于web 应用上有趣的关键性的东西。为了达到这个目标，Django 提供了通用Web开发模式的高度抽象，提供了频繁进行的编程作业的快速解决方法，以及为“如何解决问题”提供了清晰明了的约定。Django的理念是DRY(Don't Repeat Yourself)来鼓励快速开发！

## 让我们一览 Django 全貌

[urls.py](#)

网址入口，关联到对应的views.py中的一个函数（或者generic类），访问网址就对应一个函数。

[views.py](#)

处理用户发出的请求，从`urls.py`中对应过来，通过渲染`templates`中的网页可以将显示内容，比如登陆后的用户名，用户请求的数据，输出到网页。

## [models.py](#)

与数据库操作相关，存入或读取数据时用到这个，当然用不到数据库的时候 你可以不使用。

## [forms.py](#)

表单，用户在浏览器上输入数据提交，对数据的验证工作以及输入框的生成等工作，当然你也可以不使用。

## `templates` 文件夹

`views.py` 中的函数渲染`templates`中的`Html`模板，得到动态内容的网页，当然可以用缓存来提高速度。

## [admin.py](#)

后台，可以用很少量的代码就拥有一个强大的后台。

## [settings.py](#)

Django 的设置，配置文件，比如 `DEBUG` 的开关，静态文件的位置等。

[小额赞助，支持作者！](#)

[« Django 2.2 基础教程](#)

[Django 环境搭建 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

# Django 基础教程

这是第一篇  
[Django 简介 »](#)

django

Django 是用Python开发的一个免费开源的Web框架, 可以用于快速搭建高性能, 优雅的网站!

你一定可以学会, Django 很简单! 本教程一直在维护, 更新, 提供免费答疑, 免费邮件咨询。

从2015年01月18日写第一篇Django教程, 一直到现在, 每一篇教程由于新版 Django 一些改变, 或用户的反馈, 随时可能会进行更新完善, 可以在网站首页看到最近更新的情况。截止2017年02月16日, 此Django教程已经有4400多条评论, 两年多以来, 几乎每天回复评论, 回复邮件。教程也根据评论进行了大量的改进, 包含大量的工程经验, 致力打造出中国最实用的 Django 教程。

我在2015年初时, 阅读了全部的 Django 英文的官方文档, 觉得国内比较好的Django学习资源不多, 所以决定写自己的教程。本教程开始写的时候是 Django 的版本是 1.6, Django的更新很快, 自强学堂的教程也随着更新了, 兼顾了后来的新版本, 从 Django 1.4 到最新的 Django 1.11 应该都没有问题。

自强学堂 就是用 Django 搭建的站点! 提供书籍下载, 可以在 kindle, ipad 等上阅读。

本教程作者: 涂伟忠(未经同意,禁止转载!)

遇到问题请直接回复对应的教程, 如果有截图, 附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong#163.com](mailto:tuweizhong#163.com) (请把#换成@)

目前不提供QQ群, 我觉得邮件可以集中处理, 更高效些。

本教程系列文章电子书下载: 链接: <https://pan.baidu.com/s/1rap4t20> 密码: uta2 (不含附件)

## 除了本教程外的其它教程

自强学堂 学习分享 的文章: [Python 学习资源](#) 和 [Django 学习资源](#), 如果有更好的教程也可以在文章下推荐哦!

Django 是基于 Python, 所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

## 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包, 所以你得知道一些 [Python](#) 基础知识。
2. 其次你最好有一些做网站的经验, 懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

### 优雅的网址

用正则匹配网址，传递到对应函数，随意定义，如你所想！

### 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

### 缓存系统

与Memcached, Redis等缓存系统联用，更出色的表现，更快的加载速度。

### 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

这是第一篇

[Django 简介 »](#)

关注自强学堂官方微信,随时查教程 提升自己!  
微信扫码关注

---

[报告错误](#)  
[打印页面](#)



[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 基础教程

这是第一篇  
[Django 简介 »](#)

|        |  |
|--------|--|
| django | Django 是用 <a href="#">Python</a> 开发的一个免费开源的Web框架，可以用于快速搭建高性能，优雅的网站！<br><br>你一定可以学会，Django 很简单！本教程一直在维护，更新，提供免费答疑，免费邮件咨询。 |
|--------|--|

从2015年01月18日写第一篇Django教程，一直到现在，每一篇教程由于新版 Django 一些改变，或用户的反馈，随时可能会进行更新完善，可以在网站首页看到最近更新的情况。截止2017年02月16日，此Django教程已经有4400多条评论，两年多以来，几乎每天回复评论，回复邮件。教程也根据评论进行了大量的改进，包含大量的工程经验，致力打造出中国最实用的 Django 教程。

我在2015年初时，阅读了全部的 Django 英文的官方文档，觉得国内比较好的Django学习资源不多，所以决定写自己的教程。本教程开始写的时候是 Django 的版本是 1.6，Django的更新很快，自强学堂的教程也随着更新了，兼顾了后来的新版本，从 Django 1.4 到最新的 Django 1.11 应该都没有问题。

自强学堂 就是用 Django 搭建的站点！提供书籍下载，可以在 kindle, ipad 等上阅读。

本教程作者: 涂伟忠(未经同意,禁止转载!)

遇到问题请直接回复对应的教程，如果有截图，附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong#163.com](mailto:tuweizhong#163.com) (请把#换成@)

目前不提供QQ群，我觉得邮件可以集中处理，更高效些。

本教程系列文章电子书下载：链接: <https://pan.baidu.com/s/1rap4t20> 密码: uta2 (不含附件)

除了本教程外的其它教程

自强学堂 学习分享 的文章：[Python 学习资源](#) 和 [Django 学习资源](#)，如果有更好的教程也可以在文章下推荐哦！

Django 是基于 Python，所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

## 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包，所以你得知道一些 [Python](#) 基础知识。

2. 其次你最好有一些做网站的经验，懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入

学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

### 优雅的网站

用正则匹配网址，传递到对应函数，随意定义，如你所想！

### 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

### 缓存系统

与Memcached, Redis等缓存系统联用，更出色的表现，更快的加载速度。

### 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

这是第一篇

[Django 简介 »](#)

推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
基于Django开发的，[自强学堂网站源代码免费下载](#)

## Django 学习资源

[« 如何用好 Google 等搜索引擎  
Python 学习资源 »](#)

入门书籍：

1. [Django book 2.0 中文版](#)（注：部分内容已经过时，但是入门还是可以的）
2. [自强学堂 Django教程](#)
3. [官网的 tutorial](#)
4. [Django搭建简易博客教程 - Andrew\\_liu](#)

相关的分享：

开发者头条：<http://toutiao.io/search?utf8=%E2%9C%93&q=django>

极客头条及Django资讯：<http://www.csdn.net/tag/django/news>

一些优秀的文章：

Django 常用测试方法：<https://messense.me/django-common-testcase-writing.html>

站点：

有用的Django代码片断：<https://djangosnippets.org/>

Django 开源包或者app以及框架：<https://www.djangopackages.com/>

### Books

- [Django by Example](#) (1.2)
- [Django by Example for Django 1.5](#) (1.5)
- [Djen of Django](#)
- [Effective Django](#) (1.5)
- [Getting started with Django](#) (video)
- [Tango With Django](#) (1.5)
- [Test-Driven Web Development with Python](#) (1.7)
- [The Django book](#)
- [Two Scoops of Django: Best Practices for Django 1.6](#) - Making Python and Django as fun as ice cream.

另外：自强学堂的 Django 教程都可以评论，有问题的话请直接评论即可，有时间就会回复（一般会在24小时之内回复）。

当入门后经常去参考官方的文档：<https://docs.djangoproject.com/en/dev/> 很多东西官方文档都说得很清楚。

[« 如何用好 Google 等搜索引擎  
Python 学习资源 »](#)

关注自强学堂官方微信, 随时查教程 提升自己!  
微信扫码关注

[报告错误](#)  
[打印页面](#)  
[百度口碑](#)  
[赞助我们](#)  
[免责声明](#)  
[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广：[阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠，高性能服务器2折起](#)  
[基于Django开发的，自强学堂网站源代码免费下载](#)

## Django 学习资源

[« 如何用好 Google 等搜索引擎  
Python 学习资源 »](#)

入门书籍：

1. [Django book 2.0 中文版](#)（注：部分内容已经过时，但是入门还是可以的）
2. [自强学堂 Django教程](#)
3. [官网的 tutorial](#)
4. [Django搭建简易博客教程 - Andrew\\_liu](#)

相关的分享：

开发者头条：<http://toutiao.io/search?utf8=%E2%9C%93&q=django>

极客头条及Django资讯: <http://www.csdn.net/tag/django/news>

一些优秀的文章:

Django 常用测试方法: <https://messense.me/django-common-testcase-writing.html>

站点:

有用的Django代码片断: <https://djangosnippets.org/>

Django 开源包或者app以及框架: <https://www.djangopackages.com/>

## Books

- [Django by Example](#) (1.2)
- [Django by Example for Django 1.5](#) (1.5)
- [Djen of Django](#)
- [Effective Django](#) (1.5)
- [Getting started with Django](#) (video)
- [Tango With Django](#) (1.5)
- [Test-Driven Web Development with Python](#) (1.7)
- [The Django book](#)
- [Two Scoops of Django: Best Practices for Django 1.6](#) - Making Python and Django as fun as ice cream.

另外: 自强学堂的 Django 教程都可以评论, 有问题的话请直接评论即可, 有时间就会回复 (一般会在24小时之内回复)。

当入门后经常去参考官方的文档: <https://docs.djangoproject.com/en/dev/> 很多东西官方文档都说得很清楚。

« [如何用好 Google 等搜索引擎](#)  
[Python 学习资源](#) »

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

# Django 简介

[« Django 2.2 基础教程](#)  
[Django 环境搭建 »](#)

自强学堂的django教程将节省你大量的时间, 并且使你的web开发充满乐趣。通过Django, 你可以建立一个高性能的web应用而只花费最少的时间和精力。

Django 中提供了开发网站经常用到的模块, 常见的代码都为你写好了, 通过减少重复的代码, Django 使你能够专注于web 应用上有趣的关键性的东西。为了达到这个目标, Django 提供了通用Web开发模式的高度抽象, 提供了频繁进行的编程作业的快速解决方法, 以及为“如何解决问题”提供了清晰明了的约定。Django的理念是DRY(Don't Repeat Yourself)来鼓励快速开发!

## 让我们一览 Django 全貌

### [urls.py](#)

网址入口, 关联到对应的views.py中的一个函数(或者generic类), 访问网址就对应一个函数。

### [views.py](#)

处理用户发出的请求, 从urls.py中对应过来, 通过渲染templates中的网页可以将显示内容, 比如登陆后的用户名, 用户请求的数据, 输出到网页。

### [models.py](#)

与数据库操作相关, 存入或读取数据时用到这个, 当然用不到数据库的时候 你可以不使用。

### [forms.py](#)

表单, 用户在浏览器上输入数据提交, 对数据的验证工作以及输入框的生成等工作, 当然你也可以不使用。

### **templates 文件夹**

views.py 中的函数渲染templates中的Html模板, 得到动态内容的网页, 当然可以用缓存来提高速度。

### [admin.py](#)

后台, 可以用很少量的代码就拥有一个强大的后台。

### [settings.py](#)

Django 的设置, 配置文件, 比如 DEBUG 的开关, 静态文件的位置等。

[小额赞助, 支持作者!](#)  
[« Django 2.2 基础教程](#)  
[Django 环境搭建 »](#)



零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信,随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#)ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



零成本开启敏捷开发 五人以下团队免费

免费体验

推广: [阿里云限时领红包 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 简介

[« Django 2.2 基础教程](#)

[Django 环境搭建 »](#)

自强学堂的django教程将节省你大量的时间，并且使你的web开发充满乐趣。通过Django，你可以建立一个高性能的web应用而只花费最少的时间和精力。

Django 中提供了开发网站经常用到的模块，常见的代码都为你写好了，通过减少重复的代码，Django 使你能够专注于web 应用上有趣的关键性的东西。为了达到这个目标，Django 提供了通用Web开发模式的高度抽象，提供了频繁进行的编程作业的快速解决方法，以及为“如何解决问题”提供了清晰明了的约定。Django的理念是DRY(Don't Repeat Yourself)来鼓励快速开发！

## 让我们一览 Django 全貌

[urls.py](#)

网址入口，关联到对应的views.py中的一个函数（或者generic类），访问网址就对应一个函数。

[views.py](#)



处理用户发出的请求，从`urls.py`中对应过来，通过渲染`templates`中的网页可以将显示内容，比如登陆后的用户名，用户请求的数据，输出到网页。

## [models.py](#)

与数据库操作相关，存入或读取数据时用到这个，当然用不到数据库的时候 你可以不使用。

## [forms.py](#)

表单，用户在浏览器上输入数据提交，对数据的验证工作以及输入框的生成等工作，当然你也可以不使用。

## `templates` 文件夹

`views.py` 中的函数渲染`templates`中的`Html`模板，得到动态内容的网页，当然可以用缓存来提高速度。

## [admin.py](#)

后台，可以用很少量的代码就拥有一个强大的后台。

## [settings.py](#)

Django 的设置，配置文件，比如 `DEBUG` 的开关，静态文件的位置等。

[小额赞助，支持作者！](#)

[« Django 2.2 基础教程](#)

[Django 环境搭建 »](#)



↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

推广: [阿里云限时领红包,助力一步上云](#) | [【全民云计算升级】拼团上云更优惠,高性能服务器2折起](#)  
基于Django开发的, [自强学堂网站源代码免费下载](#)

## Django 2.2 基础教程

这是第一篇  
[Django 简介 »](#)

django

Django 是用Python开发的一个免费开源的Web框架, 可以用于快速搭建高性能, 优雅的网站!

你一定可以学会, Django 很简单! 本教程一直在维护, 更新, 提供免费答疑, 免费邮件咨询。

从2015年01月18日写第一篇Django教程, 一直到现在, 每一篇教程由于新版 Django 一些改变, 或用户的反馈, 随时可能会进行更新完善, 可以在网站首页看到最近更新的情况。

截止2017年02月16日, 此Django教程已经有4400多条评论, 两年多以来, 几乎每天回复评论, 回复邮件。教程也根据评论进行了大量的改进, 包含大量的工程经验, 致力打造出中国最实用的 Django 教程。

我在2015年初时, 阅读了全部的 Django 英文的官方文档, 觉得国内比较好的Django学习资源不多, 所以决定写自己的教程。

Django 2.2 已经发布, 变动比较大, 不再兼容Python 2 版本, 本教程采用 **Python 3.7 + Django 2.2**。

如果想学习 Django 1.4 - Django 1.11 系列的 Django 1.x 的版本, 请[点击此处](#)。

本教程作者: 涂伟忠(未经同意,禁止转载!)

《Django开发从入门到实践》书籍出版啦! 基于 Python3 + Django 2.x

image.png

书籍购买: [淘宝](#)、[京东](#)、[当当](#)

这是一本写给初学者的书籍, 简介可以关注右侧的公众号了解。

遇到问题请直接回复对应的教程, 如果有截图, 附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong#163.com](mailto:tuweizhong#163.com) (请把#换成@)

书籍读者交流 QQ群: 1020663804

### 除了本教程外的其它教程

自强学堂 学习分享 的文章: [Python 学习资源](#) 和 [Django 学习资源](#), 如果有更好的教程也可以在文章下推荐哦!

Django 是基于 Python, 所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

# 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包，所以你得知道一些 [Python](#) 基础知识。

2. 其次你最好有一些做网站的经验，懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入

学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

### 优雅的网址

用正则匹配网址，传递到对应函数，随意定义，如你所想！

### 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

### 缓存系统

与Memcache, Redis等缓存系统联用，更出色的表现，更快的加载速度。

### 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

[小额赞助，支持作者！](#)

这是第一篇

[Django 简介](#) »



零成本开启敏捷开发 五人以下团队免费

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。

关注自强学堂官方微信, 随时查教程 提升自己!

微信扫码关注

[报告错误](#)

[打印页面](#)

[百度口碑](#)

[赞助我们](#)

[免责声明](#)

[关于我们](#)

自强学堂为提供的内容仅用于学习，测试和培训。实例可能为了更容易理解而简化。我们一直对教程，参考手册，在线实例保持修订，但是我们不能保证所有内容全部正确。通过使用本站进行学习随之而来的风险与本站无关。当使用本站时，代表您已接受了本站的使用条款和隐私条款。自强学堂是以学习和分享知识为目的，对任何法律问题及风险不承担任何责任。版权所有，保留一切权利。

自强学堂是用 [Django](#) 技术开发的站点，托管在[阿里云](#) ECS

Copyright © 2008-2020 Powered by 自强学堂 All Rights Reserved. [吉ICP备13002477号-1](#)



推广: [阿里云限时领红包, 助力一步上云](#) | [【全民云计算升级】拼团上云更优惠, 高性能服务器2折起](#)  
[基于Django开发的, 自强学堂网站源代码免费下载](#)

## Django 2.2 基础教程

这是第一篇

[Django 简介 »](#)

django

Django 是用[Python](#)开发的一个免费开源的Web框架，可以用于快速搭建高性能，优雅的网站！

你一定可以学会，Django 很简单！**本教程一直在维护，更新，提供免费答疑，免费邮件咨询。**

从2015年01月18日写第一篇Django教程，一直到现在，每一篇教程由于新版 Django 一些改变，或用户的反馈，随时可能会进行更新完善，可以在网站首页看到最近更新的情况。

截止2017年02月16日，此Django教程已经有4400多条评论，两年多以来，几乎每天回复评论，回复邮件。教程也根据评论进行了大量的改进，包含大量的工程经验，致力打造出中国最实用的 Django 教程。

我在2015年初时，阅读了全部的 Django 英文的官方文档，觉得国内比较好的Django学习资源不多，所以决定写自己的教程。

Django 2.2 已经发布，变动比较大，不再兼容Python 2 版本，本教程采用 **Python 3.7 + Django 2.2**。

如果想学习 **Django 1.4 - Django 1.11** 系列的 **Django 1.x** 的版本，[请点击此处](#)。

本教程作者: **涂伟忠(未经同意,禁止转载!)**

《Django开发从入门到实践》书籍出版啦！基于 Python3 + Django 2.x

image.png

书籍购买：[淘宝](#)、[京东](#)、[当当](#)

这是一本写给初学者的书籍，简介可以关注右侧的公众号了解。

遇到问题请直接回复对应的教程，如果有截图，附件等不方便可以发邮件到

自强学堂答疑邮箱 [tuweizhong@163.com](mailto:tuweizhong@163.com) (请把#换成@)

书籍读者交流 QQ群：1020663804

## 除了本教程外的其它教程

自强学堂 学习分享 的文章：[Python 学习资源](#) 和 [Django 学习资源](#)，如果有更好的教程也可以在文章下推荐哦！

Django 是基于 Python，所有的 Django 代码都是用Python写成的。

开发过程中遇到的各种问题可直接在相应的教程下回复该教程相关的问题。

## 学Django需要什么基础

1. Django是 [python](#) 语言写的一个Web框架包，所以你得知道一些 [Python](#) 基础知识。
2. 其次你最好有一些做网站的经验，懂一些网页 [HTML](#), [CSS](#), [JavaScript](#) 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

下面这些建议虽然在初学时Django似乎不那么重要，但会决定你能不能走的更远，更深入

学习 前端相关知识（看个人兴趣，了解一些也挺好的）

学习 Linux 基础命令

学习 数据库，缓存等相关知识

学习 HTTP协议相关内容

学习 网络相关知识（TCP/IP协议）

学习 算法，数据结构等知识（基本的数据结构和算法要会）

学习 操作系统原理等

## Django 特点

### 强大的数据库功能

拥有强大的数据库操作接口（QuerySet API），如需要也能执行原生SQL。

### 自带强大后台

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理内容！

## 优雅的网站

用正则匹配网址，传递到对应函数，随意定义，如你所想！

## 模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

注：前后端分离时，也可以用Django开发API，完全不用模板系统。

## 缓存系统

与Memcache, Redis等缓存系统联用，更出色的表现，更快的加载速度。

## 国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

[小额赞助，支持作者！](#)

这是第一篇

[Django 简介 »](#)

 CODING  
CLOUD DEVELOPMENT

**零成本开启敏捷开发 五人以下团队免费**

免费体验

↑上方为自强学堂赞助商，非常荣幸有他们支持自强学堂的发展，感兴趣的了解一下他们的产品。