

一杯茶的时间，上手 Django 框架开发



关注

857 人赞同了该文章

Django 是 Python 社区的两大最受欢迎的 Web 框架之一（另一个是 Flask）。凭借功能强大的脚手架和诸多开箱即用的组件，用 Django 搭建 Web 应用快速而又省力。然而，也正是因为过于强大，想要驾驭它需要花费不少的力气。本文将通过实现一个新闻发布网站带你快速熟悉 Django 框架，让你能够骑上这匹快马，在 Web 开发的战场上尽情驰骋。

提示

这篇文章写作时用的是 Django 2.x 版本，发表时已经推出了 3.x 版本。经过笔者测试，在 Python 3.7 的环境下运行 Django 3.x 会出现 admin 无法登录的情形，可以选择降到 Python 3.6 及以下的版本，或安装 Django 2.x。

起步

Django 由 Adrian Holovaty 和 Simon Willison 在 2003 年的秋天写成，并在 2005 年正式发布。他们俩当时为一个新闻报社制作网站，对**快速开发**有着比较高的需求，并且希望能够在开发的同时也能够**让非技术人员为网站添加内容**。于是这也使得 Django 具备了两项鲜明的特点：

- 高度强调**可复用性**和**可插拔性**，内置大量现成的成熟组件，开发效率极高
- 自带与数据库联动的**后台管理系统**，能够在开发的同时创建内容

Django 的名字取自吉他手 Django Reinhardt，发音为 JANG-goh（谐音“尖狗”），但实际上 Django 的吉祥物是一只长着翅膀的小马。



在这篇教程中，我们也将向 Django 的起源致敬——手把手带你开发一个新闻发布网站，并且可以从后台管理系统中添加新闻，展示到网站首页上。

预备知识

本教程假定你已经知道了：

- 基本的 Python 3 语言知识，包括使用 pip 安装包
- 了解 HTTP 协议基础知识，浏览器和服务器之间如何工作的

读完这篇教程后，你将掌握 Django MTV 框架的精髓：

- M (Model)：创建数据模型，并执行数据库迁移
- T (Template)：写出基本的 Django 模板，并从视图传入数据
- V (View)：在视图中访问数据库，实现业务逻辑，渲染模板，并接入路由表

虽然 Django 还有很多知识点，但是理解了 MTV，后面的知识点学习起来也就轻松多啦。

安装 Django 并启用脚手架

本文假定你已经安装好了 Python 3 和 pip，那么可以直接用 pip 安装 Django：

```
pip install django
# 如果你用的是 Python 3.7，请安装 Django 2.2:
# pip install django==2.2
```

直接用 pip 在全局安装 Django 的确不是一个很好的做法，用虚拟环境更符合最佳实践。为了减少初学者们的认知负担，在这里就简化了安装过程。熟悉 [pipenv](#) 等虚拟环境工具的老司机当然可以自行使用哈。

安装好 Django 后，我们用 Django 自带的脚手架工具 django-admin 创建项目：

```
django-admin startproject django_news
cd django_news
```

生成的项目骨架及每个文件的作用如下所示：

```
django_news
├── django_news           // 项目全局文件目录
│   ├── __init__.py
│   ├── settings.py      // 全局配置
│   ├── urls.py          // 全局路由
│   └── wsgi.py          // WSGI 服务接口（暂时不用纠结这个是神马）
└── manage.py            // 项目管理脚本
```

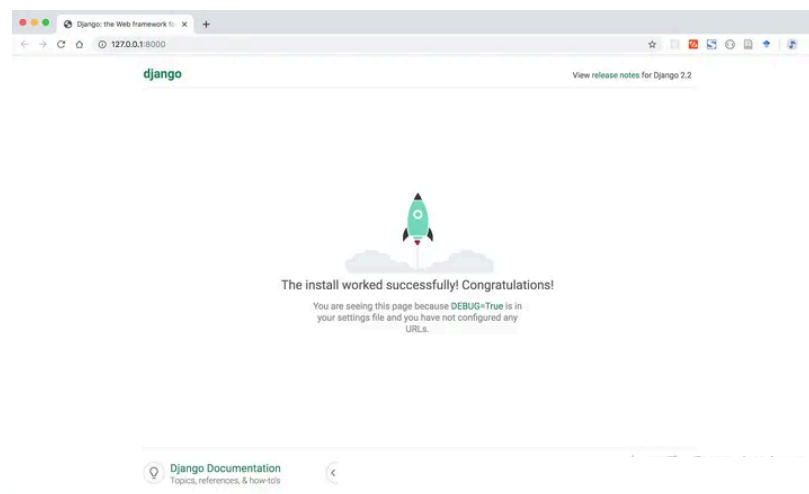
我们使用 manage.py 来运行开发服务器（Development Server）：

```
python manage.py runserver
```

提示

细心的你会发现出现了一行鲜红色的提示：You have 17 unapplied migration(s)...（省略 n 个字符）。不用担心，我们会在接下来的步骤中详细讲解来龙去脉。

按照提示，我们通过浏览器访问 localhost:8000，可以看到欢迎界面：



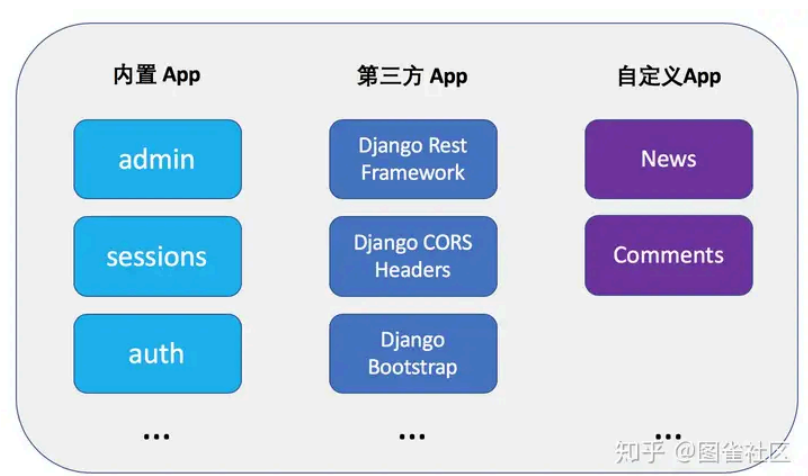


Django 开发服务器可以保持开启，并且后面修改代码会自动重新加载，非常方便。后面运行其他命令时，再打开一个终端（命令行）即可。

一切准备就绪，缰绳已在你手中！

创建第一个自定义 Django App

在上一节中我们讲到，Django 是一个高度模块化的框架。具体而言，一个 Django 应用由多个子应用组成，我们一般称之为 App（注意不是我们常说的移动应用 APP，而是 Application 的简写），如下图所示。



Django App 的类别

Django App 一般分为三大类（根据来源）：

- **内置**：即 Django 框架自带的应用，包括 admin（后台管理）、auth（身份鉴权）、sessions（会话管理）等等
- **自定义**：即用来实现我们自身业务逻辑的应用，这里我们将创建一个新闻展示应用
- **第三方**：即社区提供的应用，数量极其丰富，功能涵盖几乎所有方面，能够大大减少开发成本

所有的 Django 应用都在 django_news/settings.py 的 INSTALLED_APPS 列表中定义：

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

实现自定义 App

话不多说，让我们来创建第一个自定义 App，名称为 news：

```
python manage.py startapp news
```

生成的 news 应用文件夹结构如下所示：

```
news                                // news 应用目录  
├── __init__.py                     // 初始化模块  
├── admin.py                        // 后台管理  
└── apps.py                        // 应用配置
```



```
|— models.py      // 数据模型
|— tests.py       // 单元测试
|— views.py       // 视图
```

这个子目录里面乍一看好多文件啊！这是因为 Django 始终坚持**解耦**的原则——尽量减少代码之间的耦合，把不相关的代码拆成多个模块，让同一个模块具有**内聚性**。相信我，等到后面慢慢熟悉之后，你会对每一个模块都了如指掌的。

实际上，每个 Django App 的组织结构符合 Django 的 MTV 法则——Model（模型）+ Template（模板）+ View（视图）。MTV 与大家比较熟悉的 MVC 在思想上非常相似，但是命名有比较大的出入，如下表所示：

大家熟知的 View，在 Django 里面代表的是业务逻辑，也就是 MVC 中的控制器哦！

将自定义 App 添加到全局配置

最后，我们在 settings.py 中将 news 应用加入 INSTALLED_APPS 中：

```
# ...

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

至此，我们已经创建了第一个 Django 应用！但是现在这个应用还没有任何内容，我们接下来将逐步完善这个应用。

理解视图：业务逻辑的编写

也许你已经注意到，通过访问 localhost:8000/admin 已经可以访问后台管理系统了（虽然会跳转到登录界面）。接下来，我们也希望能够访问到刚才创建的 news 应用。因此，这一步中我们将：

- 在视图（View）中写一点业务逻辑
- 接入路由，使其能够被访问

Django 的路由系统

Django 的路由系统是由**全局路由**和**子应用路由**组成。简单来说，根据用户输入的 URL，全局路由表进行匹配并选择正确的子应用路由，再由所选择的子应用路由匹配并选择正确的视图（View）。整个流程如下图所示：

例如，用户访问 `http://example.com/apple/buy`，然后全局路由根据 `/apple/buy` 先选择 `apple` 的路由表，再从 `apple` 路由表中根据 `/buy` 选择 `/buy` 路由，然后执行 `/buy` 对应的 `BuyView` 视图，返回给用户结果。

编写第一个视图

对视图访问的流程大致了解之后，我们就可以开始动手了。首先打开 `news/views.py`，写一个简单的视图函数，返回一串 `Hello World!`：

```
from django.http import HttpResponse

def index(request):
    return HttpResponse('Hello World!')
```

上面这个 `index` 函数可以说是一个最简单的视图函数了，实际大部分应用的视图要比这复杂得多。Django 同时支持**基于函数的视图**（FBV，Function-based View）和**基于类的视图**（CBV，Class-based View），这里显然是 FBV，接收一个 `request` 请求对象作为参数，返回了一个 `HttpResponse` 对象。

将视图接入路由

接着，我们要让路由系统能够访问到刚才写好的视图函数。因此先实现子应用 `news` 的路由表，创建 `news/urls.py` 文件如下：

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

每一个 Django 路由表模块（`urls.py`）中都约定必须包含一个 `urlpatterns` 列表用来存放路由映射表。列表中每个元素是一个用 `django.urls.path` 函数封装好的路由映射，通常接收以下三个参数：

- `route`：必须，即实际的访问路由，空字符串等于 `/`，即空路由
- `view`：必须，该路由将要访问的视图
- `name`：可选，该路由的名称，方便后续在模板中使用

我们将刚刚写好的 `news` 路由表接入全局路由表。由于我们希望新闻能够展示在首页（即通过 `/` 就能访问，无需 `/news`），因此 `news` 应用路由在全局路由中的 URL 是一个空字符串。在 `django_news/urls.py` 中修改如下：

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls
```

这里使用 `django.urls.include` 函数将 `news` 应用的路由表接入进来，并且 `include` 函数的参数是路由模块路径的字符串 `news.urls`，省去了手动 `import` 的麻烦。

注意

添加路由规则时顺序是很重要的，因为在尝试匹配时会按照从上到下的顺序进行，因此应该把最模糊的路由（即空路由）放在最下面。

如果你开发服务器还在运行（如果没有可以再打开），访问 `localhost:8000`，就可以看到那一串熟悉的字符了：

理解模板：网页前端的实现

上一步中，我们学会了如何实现视图，并将其接入路由配置中，使其能够被用户访问。接下来，我们将实现一个 Django 模板作为网页前端，从而给用户呈现更丰富的内容。

提示

如果你有过其他模板（或者类似技术）的开发经验，例如 Jinja、EJS 或是 JSP 等，对 Django 模板会有一种似曾相识的感觉。如果你不了解什么是模板引擎，也不用担心，简单的理解就是一个可以填充内容、甚至能够加入代码逻辑的**类似 HTML 文档**，最终会被转换成浏览器能够识别的 HTML 文档。

Django 模板语言基础

Django 模板本质上是一个 HTML 文档，只不过通过一些特殊的语法实现数据的填充。这里我们讲解一下最常用的三个语法：

表达式插值

最常用的语法，没有之一。通过在花括号 `{{ }}` 放入一个表达式，就能够在视图传入表达式中变量的内容，并最终渲染成包含变量具体内容的 HTML 代码。需要注意的是，所支持的表达式仅支持以下形式（可以自由组合）：

```
<!-- 单个变量 -->
{{ variable }}

<!-- 获取字典的键或对象的属性 -->
{{ dict.key }}
{{ object.attribute }}

<!-- 获取列表中的某个元素 -->
{{ list.0 }}
```

例如，模板这样写：

```
<h1>{{ name }}</h1>
<p>{{ news.title }}</p>
```

如果我们在视图传入以下上下文字典（Context Dictionary）：

```
{
    'name': 'Tuture',
    'news': {
        'title': 'Hello World',
        'visitors': ['Tom', 'Marc'],
    }
}
```

那么最终渲染成的 HTML 代码就是：

```
<h1>Tuture</h1>
<p>Hello World</p>
<p>Tom</p>
```

条件语句

条件语句的定义如下：

```
{% if is_true %}
    <h1>It is true!</h1>
{% else %}
    <h1>It is false!</h1>
{% endif %}
```

如果变量 `is_true` 为真，那么最终渲染出来的就是 `<h1>It is true!</h1>`，否则就是 `<h1>It is false!</h1>`。注意：整个条件语句必须以 `{% endif %}` 结束，并且 `{% else %}` 是可选的。

循环语句

循环语句用来在模板上展示任意长的列表内容。其语法如下：

```
{% for elem in some_list %}
    <p>{{ elem }}</p>
{% endfor %}
```

如果传入的 `some_list` 为 `['Apple', 'Banana', 'Orange']`，那么渲染出的 HTML 代码就是：

```
<p>Apple</p>
<p>Banana</p>
<p>Orange</p>
```

实现第一个 Django 模板

到了动手时间了，我们先实现第一个 Django 模板。在 `news` 目录中创建一个 `templates` 目录，再在 `templates` 目录中创建一个 `news` 目录，并在内层的 `news` 目录中创建 `index.html` 文件：

```
mkdir -p news/templates/news
touch news/templates/news/index.html
```

思考

听上去很麻烦，只创建 `news/templates`，然后把模板放里面不就好了，为什么还要再创建一个 `news` 目录？这是由于 Django 的模板查找机制会将所有应用里面的模板全部收集到一起，如果两个模板的名字冲突，就会导致其中一个模板不能被正确访问。如果放在 `news` 子文件夹里面，就能够通过 `news/index.html` 访问，通过命名空间的机制避免了冲突。

模板的代码如下：

```
{% if news_list %}
    <ul>
```

```
<h3>{{ elem.title }}</h3>
<p>{{ elem.content }}</p>
</li>
{% endfor %}
</ul>
{% else %}
<p>暂无新闻</p>
{% endif %}
```

这短短几行模板代码却很好地覆盖了我们刚刚讲述的三个模板语法：表达式插值、条件语句和循环语句。如果忘记其中某个地方是什么意思的话，翻上去看看吧！

完成模板的编写后，我们要在视图中对其进行渲染。打开 `news/views.py` 文件，修改代码如下：

```
from django.shortcuts import render

def index(request):
    context = {
        'news_list': [
            {
                "title": "图雀写作工具推出了新的版本",
                "content": "随随便便能写出一篇好教程，真的很神奇",
            },
            {
                "title": "图雀社区正式推出快速入门系列教程",
                "content": "一杯茶的功夫，让你快速上手，绝无担忧",
            },
        ]
    }

    return render(request, 'news/index.html', context=context)
```

这里我们调用 `django.shortcuts.render` 函数来渲染模板，这个函数通常接受三个参数（有其他参数，但是这里我们不关心）：

- `request`：请求对象，直接把视图的参数 `request` 传进来就可以
- `template_name`：模板名称，这里就是我们刚刚创建的 `news/index.html`
- `context`：传入模板的上下文对象，必须是一个字典，字典中的每个键对应模板中的变量。这里我们弄了些假数据，假装是从数据库里面取来的。

再访问 `localhost:8000`，看一下我们的首页是不是有内容了：

完美！

Django 的 MTV，我们已经讲了 T (Template) 和 V (View)，现在来到了最后一关：M (Model) 了。数据模型是 Django 入门最大的难点，消化这一步的内容需要花点力气，但是相信我，当你迈过 M 这最后一关，你便能真正上手 Django 开发了！下面我们先介绍一下 Django 的数据模型设计。

Django 在数据模型方面的设计堪称典范，列举一些闪光点：

- 由于高度解耦的设计，可轻松切换各种关系型数据库（默认的 SQLite，可选 MySQL、PostgreSQL、Oracle 等等）
- 强大的 ORM (Object Relation Mapping, 对象关系映射) 模块，使得用 Python 操作数据库非常轻松，免去了使用 SQL 的麻烦
- 优秀的数据库迁移机制 (Migration)，修改数据模式 (Schema) 比较方便，能够适应不断变化的功能需求

对于初学者而言，我们暂且选择默认的 SQLite 数据库，省去了配置数据库的烦恼。在后面的进阶教程中，我们会切换到其他适合生产环境的数据库。

理解 ORM

简单来说，ORM 能够将面向对象的代码转换成相应的 SQL 语句，从而对数据库进行操作。SQL 是用于访问和处理数据库的标准的计算机语言，但是直接写在代码里面显然难以维护，而且对使用者的要求也非常高，写的糟糕的 SQL 代码查询效率非常低下。因此，使用设计良好的 ORM 不仅让代码可读性更好，也能帮助开发者进行查询优化，节省不少力气。

我们来看一些简单的 Django ORM 例子：

```
# 查询所有模型
# 等价于 SELECT * FROM Blog
Blog.objects.all()

# 查询单个模型
# 等价于 SELECT * FROM Blog WHERE ID=1
Blog.objects.get(id=1)

# 添加单个模型
# 等价于 INSERT INTO Blog (title, content) VALUES ('hello', 'world')
blog = Blog(title='hello', content='world')
blog.save()
```

有木有感觉操作起来比 SQL 方便很多呢？

理解数据库迁移

数据库迁移是指将用 Django 定义的模型转换成 SQL 代码（即迁移文件），并在数据库中进行建表操作（或更新表）。看下面这张图就知道了：

一般的开发流程就是这样：

3. 用 `migrate` 命令执行迁移
4. 在开发中发现第 1 步中定义的模型不完善，更新数据模型
5. 跳转到第 2 步，反复循环

实现第一个数据模型

终于到了动手的环节。我们首先定义数据模型 `Post`，包括标题 `title` 字段和 `content` 字段，代码如下：

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()

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

定义好之后，运行以下命令创建迁移文件：

```
python manage.py makemigrations
```

可以看到输出如下：

```
Migrations for 'news':
  news/migrations/0001_initial.py
    - Create model Post
```

并且成功地自动创建了 `news/migrations/0001_initial.py` 迁移脚本。接着我们进行数据库迁移：

```
python manage.py migrate
```

输出如下图所示：

数据库迁移完成后，我们就可以创建用于登录后台管理的超级用户：

```
python manage.py createsuperuser
```

按照提示填写用户名和密码即可。然后访问 `localhost:8000/admin`，进入后台系统的登录页面：

填入刚才设置的用户名和密码，进入后台管理页面：

咦，我们刚才创建的 news 应用还有 Post 模型去哪了？

配置后台管理接口

那是因为我们没有实现 news 应用的后台管理接口。在 news/admin.py 中填入代码如下：

```
from django.contrib import admin

from .models import Post

admin.site.register(Post)
```

再次进入后台管理系统，可以看到我们的 news 应用和 Post 模型了：

点击 Posts 一栏的 +Add 按钮，开始添加新闻（内容随意）：

大概添加个两三条新闻就差不多啦。你也可以进一步探索后台管理系统，包括修改新闻、添加用户等等，都可以。

在视图中添加数据查询

最后，我们在视图中加入从数据库中查询的代码：

```
from django.shortcuts import render

from .models import Post

def index(request):
    context = { 'news_list': Post.objects.all() }
    return render(request, 'news/index.html', context=context)
```

访问网站首页，可以看到刚才在后台管理系统添加的新闻了：


大功告成！在这篇教程中，我们完成了一个新闻发布网站，并且可以从后台管理系统中添加新闻，最终展示到我们的网站首页上。

希望这篇教程能够让你对 Django 最重要的一些概念和操作有了基本的了解。Django 还有很多很多的高级玩法，例如数据模型中的高级查询、字段索引、更换数据库等等，模板中的继承机制、内部标签等等，还有视图中如何处理各类请求（POST、PUT等），我们会在后续更多教程中逐一为大家讲解，不见不散！

想要学习更多精彩的实战技术教程？来[图雀社区](#)逛逛吧。


编辑于 2020-01-03 02:45

Django（框架） Web 开发框架 Python

 欢迎参与讨论


70 条评论

默认 最新

 良辰美景伴伊人

本人在知乎看到最好的django入门帖子，言简意赅，举一反三！强力支持作者大大，希望可以后续更新下django restframe work入门，毕竟现在前后端分离大趋势，需要restful api的情况太多了！最后感谢作者！祝福，岁月安好！🙏

2020-03-09 回复 37

 一只图雀

作者

知乎

首发于
图雀社区

2020-03-09

回复 7

 特青年

一杯茶的时间，我觉得你在骗我😂

2019-12-24

回复 16

 一只图雀 (作者)

哈哈😂，Django 这杯茶要细品哦~

2019-12-25

回复 4

 杨坤

跳了一步，在哪个文件下定义数据模

2020-02-20

回复 6

 一只图雀 (作者)

抱歉，忘记讲清楚了，是在 news/models.py文件中

2020-02-20

回复 9

 唐铭

是我今年的专业课程《web开发》讲的主要内容。
第一次在课外学习这个
深感老师讲课太繁琐太慢，阅读了五分钟已经比得上老师2节课的教学进度。
继续学习了，这玩意挺助眠

2021-05-05

回复 6

 悟实

我看了一周文档 看到这篇简直打通任督二脉

2021-08-28

回复 3

 OrthoPole

作者您好！我想问一下能不能把所有模板文件都放在大的template文件下，而不是每个app里放一个template呢？是按照django的设计习惯应该这么做，还是Django规定了必须这么做？

2023-10-07

回复 1

 吴扬

可以的哇，template路基是可以自定义的。docs.djangoproject.com/...

昨天 02:33

回复 喜欢

 当当

我都吃两个苹果了，还有点懵

2020-04-09

回复 3

 一只图雀 (作者)

喝茶喝茶🍵

2020-04-09

回复 喜欢

 mute.X

写得真棒👍👍👍

2020-03-27

回复 3

 一只图雀 (作者)

谢谢😁，可以扫描文章底部的二维码关注我们的公众号哦

2020-03-27

回复 喜欢

 唉哟哥哥

大神您好！您刚开始的index.html文件里的elem.context应该是elem.content，因为您在models.py里定义的属性是content。😂

2020-04-15

回复 2

 唉哟哥哥

一只图雀

😊

2020-04-15

回复 喜欢

 一只图雀 (作者)

🙏感谢指正

2020-04-15

回复 喜欢

 邓博文

知乎 首发于 图雀社区

2020-04-02

一只图雀

作者

👍👍👍

谢谢

2020-04-09

点击查看全部评论 >

回复

喜欢

🔗

欢迎参与讨论

文章被以下专栏收录

- 

图雀社区
汇集精彩的实战技术教程，公众号「图雀社区」
- 

python
专注于Python学习
- 

Python Web与Django开发
Python后台开发，爬虫与Django
- 

python开发
python小白进阶之路

推荐阅读

《Django企业开发实战》FAQ - 持续更新

问：Django 的文档这么全了，还需要买书看吗？the5fire答：单纯的学习 Django 提供的功能的话，看文档就够了，前提是你英文阅读还行，不过很多「残卷」的中文翻译看起来也不太省时间。当然...

胡阳 发表于Djang...



《Django企业开发实战》正式开售

胡阳



《Django企业开发实战》书评&抽奖

Manjusaka

开发app

开发一款A要组建一个产品定位、等问题都E整的产品B下角色成5

大胆的巨ノ