# Django创建网站

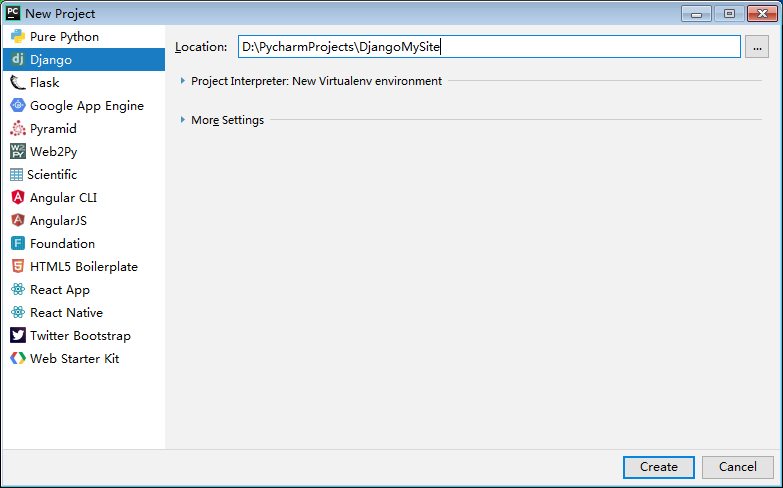
Django是Python下的重量级Web 框架，许多成功的网站和APP都基于Django。

Django遵守BSD版权，初次发布于2005年7月, 针对python3，Django的版本是2.0和2,1。

Django采用了MVC的软件设计模式，即模型M（操作数据库），视图V（负责界面）和控制器C（负责业务逻辑）。

## 在pycharm中创建django工程

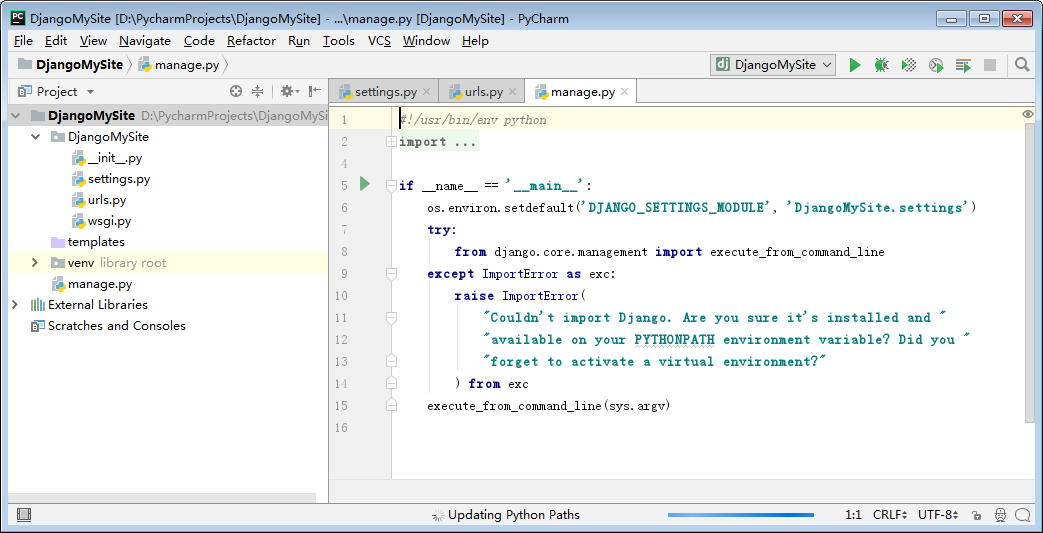
选择Django项目类型。



单击“Create”按钮，会自动安装Django框架。

![C:\Users\Administrator\AppData\Roaming\Tencent\Users\395702361\QQ\WinTemp\RichOle\R8SH](OWOPPJUHP@HA2H`GL.png](data:image/png;base64,)

创建完成后，可以看到系统自动生成了一些文件和文件夹。



manage.py：Django管理主程序。

templates：放置模板文件的目录。

wsgi.py：网络通信接口。

urls.py：路由配置程序。

settings.py：主配置程序。

## 创建模块

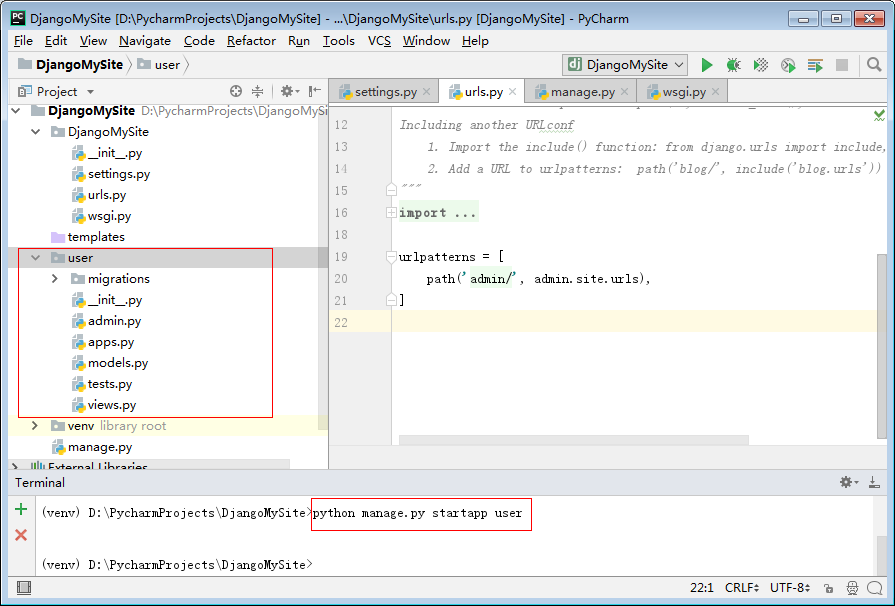
每个django项目中可以包含多个模块。比如一个项目里可以包含用户模块，产品模块，订单管理模块，等等，这样一个Django项目就可以建立多个模块。

项目里所有的模块共享项目资源，比如模板。

下面是在pycharm的Terminal终端窗口里创建了user模块。创建指令：

python manage.py startapp user

回车执行创建指令，系统会在项目下创建user目录，并在里面创建一些文件。



## 编写路由

路由是在urls文件里编写，它将浏览器输入的url映射到相应的业务处理逻辑，即某个访问网址交给哪个后台程序来处理。

**from** user **import** views  
  
urlpatterns = [  
 path(**'admin/'**, admin.site.urls),  
 path(**'index/'**, views.index),  
]

## 编写业务处理逻辑

**from** django.shortcuts **import** HttpResponse  
  
*# Create your views here.***def** index(request):  
 **return** HttpResponse(**'hello django!'**)

## 运行web服务

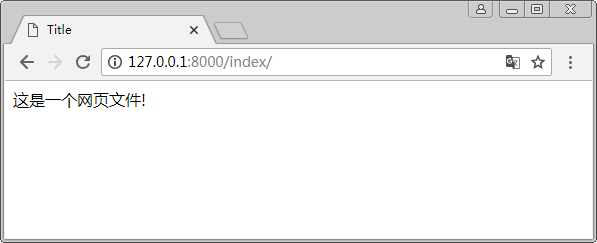


## 返回html文件

在templates文件夹里建立一个模板文件index.html。

修改index函数，直接渲染模板文件。

**def** index(request):  
 *#return HttpResponse('hello django!')* **return** render(request, **'index.html'**)

再次运行： 

## 返回动态页面

**def** index(request):  
 *#return HttpResponse('hello django!')  
 #return render(request, 'index.html')* userList = [  
 {**"userName"**:**"张三"**, **"age"**:10},  
 {**"userName"**:**"李四"**, **"age"**:15},  
 {**"userName"**:**"王五"**, **"age"**:20},  
 {**"userName"**:**"麻子"**, **"age"**:22}  
 ]  
 **return** render(request, **'index2.html'**, {**"data"**:userList})

django采用自己的模板语言，类似jinja2，根据提供的数据，替换掉html中的相应部分。

index2.html：

<**body**>  
 {% **for item in data** %}  
 <**div**>姓名：{{ **item**.**userName** }}, 年龄：{{ **item**.**age** }}</**div**>  
 {% **endfor** %}  
</**body**>

提醒，改变index函数或者index2.html的内容，不用重启服务器，刷新页面就可以看到结果。这是因为Django能自动感知文件内容发生变化。

## 获得用户提交的数据

登陆页面login:html：

<body>  
 <div>{{ ret }}</div>  
 <form action="/login" method="post">  
 <div>用户名：<input type="text" name="userName" /></div>  
 <div>密码：<input type="password" name="password" /></div>  
 <div><input type="submit" value="提交"></div>  
 </form>  
</body>

后台接口业务处理：

**def** login(request):  
 **if** request.method == **'POST'**:  
 userName = request.POST.get(**"userName"**, **None**)  
 password= request.POST.get(**"password"**, **None**)  
  
 **if** userName == **"sg" and** password == **"111"**:   
 **return** render(request, **'success.html'**, {**"userName"**: userName})  
 **else**:  
 **return** render(request, **'login.html'**, {**"ret"**:**"用户名和密码不正确，登陆失败"**})  
 **return** render(request, **'login.html'**)

request.method对提交进行判断，如果是POST，表明是执行提交动作。request.POST.get是获取提交表单的字段值。

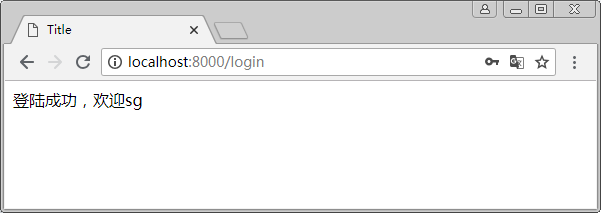
现在重启服务器，会得到下面的错误：

CSRF verification failed. Request aborted.

这是Django的一个跨域保护机制，先在配置文件settings.py里取消：

MIDDLEWARE = [  
 'django.middleware.security.SecurityMiddleware',  
 'django.contrib.sessions.middleware.SessionMiddleware',  
 'django.middleware.common.CommonMiddleware',  
 *#'django.middleware.csrf.CsrfViewMiddleware',* 'django.contrib.auth.middleware.AuthenticationMiddleware',  
 'django.contrib.messages.middleware.MessageMiddleware',  
 'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]

重新运行：



## 动态路由

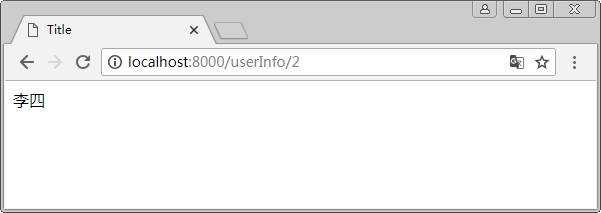
在urls.py里添加：

path(**'userInfo/<userId>'**, views.userInfo),

处理接口的业务逻辑：

**def** userInfo(request, userId):  
 **if** userId == **'1'**:  
 userName = **'张三'  
 elif** userId == **'2'**:  
 userName = **'李四'  
 else**:  
 userName = **'未知'  
 return** render(request, **'userInfo.html'**, {**"userName"**: userName})

运行：



## Django使用mysql数据库

（1）、安装pymysql模块

可以通过pycharm的settings向导窗口安装，也可以通过pycharm的Terminal终端窗口执行下面的pip3命令来安装。

pip3 install pymysql

还需要安装mysqlclient。

pip3 install mysqlclient

如果直接安装失败，可以下载whl包：

<https://www.lfd.uci.edu/~gohlke/pythonlibs/#Mysqlclient>

然后安装：pip3 install C:\Users\Administrator\Downloads\mysqlclient-1.3.13-cp36-cp36m-win\_amd64.whl

（2）、在settings.py里配置数据库连接信息

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'django\_test\_db',  
 'HOST': '120.76.243.109',  
 'PORT': 4776,  
 'USER': 'django\_user\_123',  
 'PASSWORD': '123456',  
 }  
}

将默认的sqlite3配置注释掉。

（3）、创建模型

python manage.py inspectdb > user\models.py

将已经存在的产品表t\_product相关信息导入到models.py文件。

**from** django.db **import** models  
  
  
**class** TProduct(models.Model):  
 id = models.CharField(primary\_key=**True**, max\_length=50)  
 product\_name = models.CharField(max\_length=200, blank=**True**, null=**True**)  
 price = models.FloatField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_product'**

（4）、查询数据：

先汇入模型模块：

**from** user.models **import** TProduct

查询接口：

**def** getProductList(request):  
 data = TProduct.objects.all()  
 **return** render(request, **'product.html'**, {**"productList"**: data})

也可以用原生查询语句：

data = TProduct.objects.raw(**'select** *\** **from t\_product'**)

（5）、增加数据：

调用create方法

**def** addProduct(request):  
 TProduct.objects.create(id=100, product\_name=**'格力空调'**, price=1200)  
 data = TProduct.objects.all()  
 **return** render(request, **'product.html'**, {**"productList"**: data})

（6）、修改数据：

调用update方法

**def** updateProduct(request):  
 TProduct.objects.filter(id=1).update(price=3)  
 data = TProduct.objects.all()  
 **return** render(request, **'product.html'**, {**"productList"**: data})

（7）、删除数据：

**def** deleteProduct(request):  
 TProduct.objects.filter(id=100).delete()  
 data = TProduct.objects.all()  
 **return** render(request, **'product.html'**, {**"productList"**: data})

（8）、执行原生sql

**def** productSql(request):  
 **from** django.db **import** connection, transaction  
 **import** MySQLdb  
 cursor = connection.cursor()  
  
 *# 数据修改操作——提交要求* cursor.execute(**"UPDATE t\_product SET price = 1 WHERE id = '100'"**)  
 transaction.commit()  
  
 *# 数据检索操作,不需要提交* cursor.execute(**"SELECT** *\** **FROM t\_product WHERE id = '100'"**)  
 data = dictfetchall(cursor)**return** render(request, **'product.html'**, {**"productList"**: data})  
  
  
**def** dictfetchall(cursor):  
 *"将游标返回的结果保存到一个字典对象中"* desc = cursor.description  
 **return** [  
 dict(zip([col[0] **for** col **in** desc], row))  
 **for** row **in** cursor.fetchall()  
 ]

fetchall返回的是元组，所以要转换成字典对象，便于模板读取数据。