上午课程核心内容

1. SKU 商品图片数据获取

图片对象 sku 和 sku_id 属性的区分：

sku_image.sku：图片对象关联的 sku 对象

sku_image.sku_id：图片所属的 sku_id，即图片表中的外键的值

```
>>> from goods.models import SKUImage
>>> sku_image = SKUImage.objects.get(id=4)
>>> sku_image
<SKUImage: Apple MacBook Pro 13.3英寸笔记本 银色 4>
>>> sku_image.sku
<SKU: 1: Apple MacBook Pro 13.3英寸笔记本 银色>
>>> sku_image.sku_id
1
```

图片序列化器类的定义：

```
class SKUImageSerializer(serializers.ModelSerializer):
    """图片序列化器类"""
    # 自己定义 sku_id 字段
    sku_id = serializers.IntegerField(label='SKU商品ID')
    # 关联对象的嵌套序列化
    # StringRelatedField 默认也是 read_only=True
    sku = serializers.StringRelatedField(label='SKU商品')

    class Meta:
        model = SKUImage
        exclude = ('create_time', 'update_time')
```

模型类中没有，无法自动生成，需要自己添加

sku 默认序列化成关联对象的主键，此处需要重定义

2. SKU 商品简单数据获取

```
================================================================================
======
```

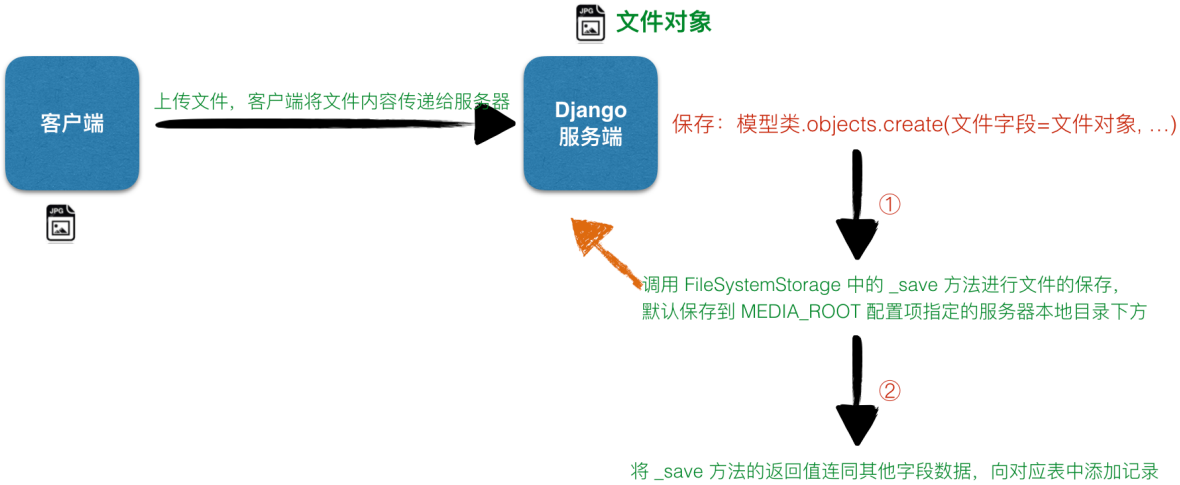
```
API: GET /meiduo_admin/skus/simple/
================================================================================
=====
请求参数：
    1）通过请求头传递 JWT Token，格式：Authorization: "JWT Token的值"
================================================================================
=====
响应数据：
    [
        {
            "id": "SKU商品ID",
            "name": "SKU商品名称"
        },
        ...
    ]
================================================================================
=====
```

```python
# GET /meiduo_admin/skus/simple/
class SKUSimpleView(ListAPIView):
    # 指定权限：只有管理员用户才能进行访问
    permission_classes = [IsAdminUser]

    # 指定视图所使用的查询集
    queryset = SKU.objects.all()

    # 指定视图所使用的序列化器类
    serializer_class = SKUSimpleSerializer

    # 注：关闭分页
    pagination_class = None  注意一定要关闭分页
```

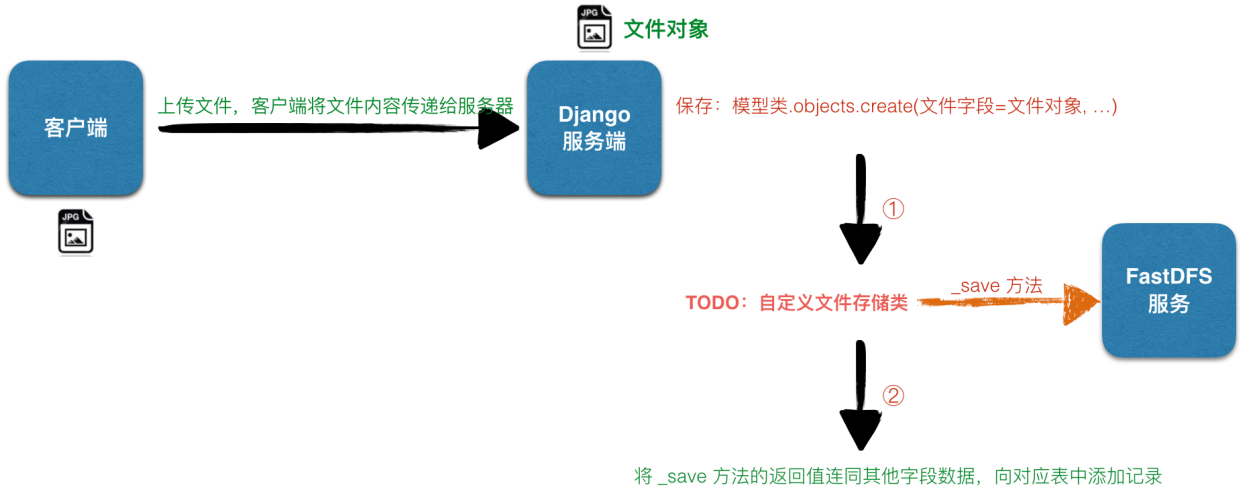
3. SKU 商品图片数据新增

FileField 和 ImageField

Django 默认保存文件过程：



SKU 商品图片数据上传需求：



自定义文件存储：

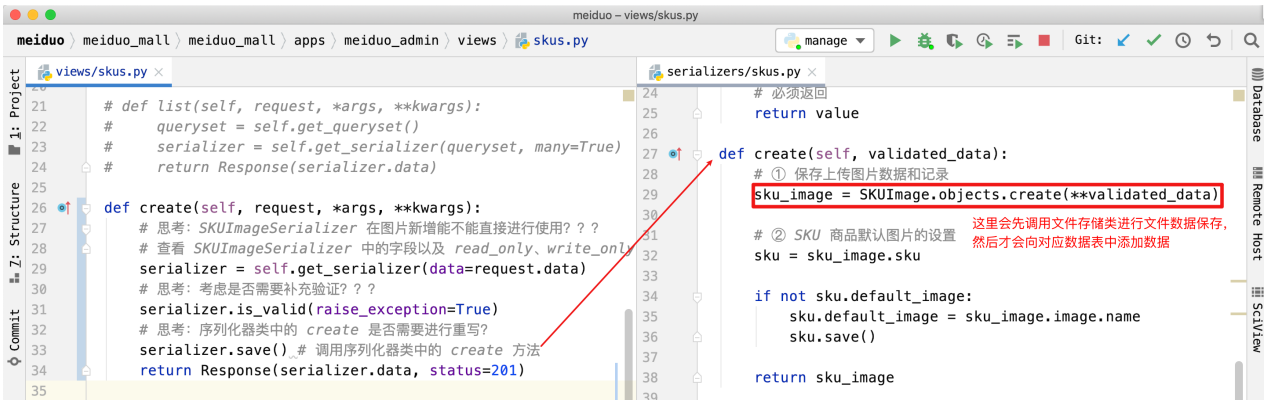
1）自定义文件存储类

```python
class FastDFSStorage(Storage):
    def _save(self, name, content):
        """
        决定上传的文件存储到哪里
        name：客户端上传文件的名字
        content：文件对象，通过 content.read() 可以获取上传文件的内容
        """
        # 将上传的文件存入 FastDFS 文件存储系统
        client = Fdfs_client(settings.FDFS_CLIENT_CONF)

        # 上传文件
        res = client.upload_by_buffer(content.read())

        # 判断上传是否成功
        if res.get('Status') != 'Upload successed.':
            raise APIException('上传文件到FastDFS失败!')

        file_id = res.get('Remote file_id')
        return file_id

    def exists(self, name):
        """判断上传的文件和文件系统中原来的文件是否同名"""
        return False

    def url(self, name):
        """
        name：数据表中文件字段存储的内容
        """
        # 返回可访问到文件系统中文件的一个完整的 URL 地址
        return settings.FDFS_URL + name
```

2）修改默认文件存储类配置项

```python
# 配置 Django 框架的默认文件存储类
DEFAULT_FILE_STORAGE = 'meiduo_mall.utils.fastdfs.storage.FastDFSStorage'
```

上传图片数据的保存：



下午课程核心内容

1. 不同身份用户权限控制数据表的设计

权限控制的层次：

① 用户登录还是未登录

比如：项目中某些API只允许登录过的用户进行访问

② 登录系统用户的权限控制

比如：登录系统之后，不同身份的用户也只能做指定的操作

举例：

以博学谷为例，其中有以下几个角色用户：

① 讲师：发布每日反馈、查看反馈结果...

② 助教：查看反馈结果...

③ 班主任：发布评分数据、查看学员信息...

④ 学员：提交每日反馈、参加阶段考试...

⑤ ......

思考：实现登录用户不同身份的权限控制，如何设计数据表？

数据表的设计：

1）用户表：保存每个用户的数据(用户名、密码、手机号...)

2）角色表(用户组表)：保存有哪些角色，哪些用户组

3）权限表：保存权限记录的数据，有哪些权限。

用户表和用户组表的关系：

一对多：一个用户只能属于一个组，而一个组可以包含多个用户。

多对多：一个用户可以属于多个组，同时一个组可以包含多个用户。

用户组表和权限表的关系：

多对多：一个组可以被分配很多权限，同一个权限也可以分配给不同的组

用户表和权限表的关系：

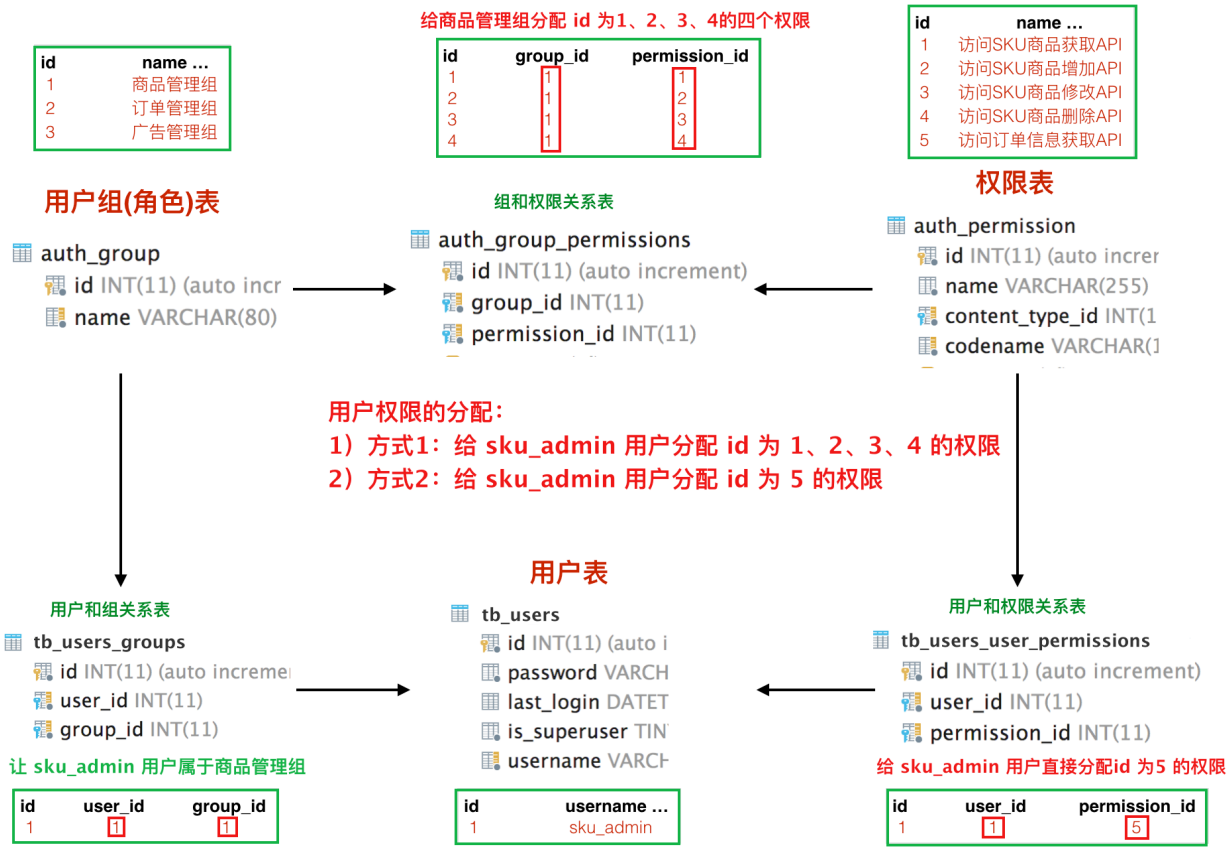
多对多：一个用户可以指定被分配很多权限，同一个权限也可以分配给不同的用户

怎么给一个用户进行权限的分配：

1）方式1：先给用户分配组，然后给组分配权限，这个用户就拥有了所属组的权限

2）方式2：直接给用户分配权限

2. Django 框架中权限相关的模型类和权限分配



3. 权限相关数据表对应的模型类

模型类	说明
User	保存用户的数据，在users.models中，继承AbstractUser
Group	保存用户组数据，在django.contrib.auth.models中
Permission	保存操作权限数据，在django.contrib.auth.models中
ContentType	保存权限类型数据，在django.contrib.contenttypes.models中

4. 模型之间的三种关系定义及关联查询

表之间的三种关系：

1）一对多：models.ForeignKey

一对多的关联属性需要定义在多的那个类中

2）多对多：models.ManyToManyField

3）一对一：models.OneToOneField

多对多和一对一的关联属性定义在两个类中的哪一边都可以，因为两边是等价的。

举例：

1）一对多

```
class BookInfo(models.Model):
    """一端"""
    pass


class HeroInfo(models.Model):
    """多端"""
    # 一对多关联属性
    hbook = models.ForeignKey(BookInfo, related_name='heros', …)
```

```
# 1. 通过英雄查图书：hero.hbook
# 2. 通过图书查英雄：book.heroinfo_set.all()  ->  book.heros.all()
```

2）多对多

```
class User(models.Model):
    # 多对多关联属性
    groups = models.ManyToManyField(Group, related_name='users', on_delete=models.CASCADE)


class Group(models.Model):
    pass
```

```
# 1. 通过用户查询用户组：user.groups.all()
# 2. 通过用户组查询用户：group.user_set.all()  -> group.users.all()
```

3）一对一

```python
class UserBasic(models.Model):
    """用户基本信息模型类"""
    pass


class UserDetail(models.Model):
    """用户详细信息模型类"""
    # 一对一关联属性
    basic = models.OneToOneField(UserBasic, related_name='detail', ...)
```

```python
# 1. 通过详细用户查询基本用户：obj.basic
# 2. 通过基本用户查询详细用户：obj.userdetail  -> obj.detail
```

5. 这段话读 3 遍！！！ 这段话读 3 遍！！！ 这段话读 3 遍！！！

通过模型对象进行关联查询时，如果对象的模型类中有关联属性，就是点关联属性；如果对象的模型类中没有关联属性，那么默认就是点关联模型类名小写，至于后面加不加_set(一对多和多对多中需要加、一对一中不需要加)，另外还要看关联属性中有没有指定 related_name 参数，如果指定了 related_name，默认写法需要换成 related_name 指定的值