day04 订单处理

今日概要

- 拍卖结束时创建的定时任务【后台管理】
- 处理订单，设计思路【小程序+API】

今日详细

1.生成订单

1.1 处理逻辑

```
auction_object = 获取专场对象
item_object_list = 获取对应专场下的所有拍品(专场=auction_object)

1.专场状态的更新
    auction_object.status = "拍卖结束"
    auction_object.save()
2.拍品 状态 更新
    total_成交价 = 0
    for item_object in item_object_list:
      2.1 检查拍品没有人出价，则更新 item_object.status = "流派"

      2.2 有人出价则找到出价最高者
         lk=models.出价记录表.objects.filter(拍品=item_object).order_by('-出
价').first()

          item_object.成交价 = lk.出价
          item_object.状态 = 成交
          total_成交价 += lk.出价
      2.3 为出价高者创建订单
        order_object = models.订单.objects.create(
            订单号=随机字符串
            状态=待支付，
            拍品=item_object,
            用户=lk.用户，
            价格=lk.出价，
            保证金=当前这个人为这件商品付的保证金对象(专场/单品)
        )
      2.4 找到未拍到商品的人退还保证金(人去重)
        单品保证金：
            立即退换保证金
        全场的保证金：
            他没有拍到当前专场的其他拍品，立即退还保证金。
            拍到其他拍品，保证金冻结。
```

```
      2.5 调用一个定时任务 24小时之后执行。
        24小时任务.apply_async(arg=[order_object.id],eta=当前时间+24小时)
    auction_object.总成交额 = total_成交价
    auction_object.save()


@shared_task
def 24小时任务(订单id):
    如果订单已经支付，则不再处理。
    订单未支付，则需要处理(逾期未付款)
    order_object = models.订单.objects.filter(id=订单id).first()
    1. 订单状态更新
        order_object.status = "逾期未付款"
        order_object.save()
    2. 拍品状态更新
        order_object.拍品.status = "逾期未付款"
        order_object.拍品.save()
    3. 扣除保证金
        # 200
        if order_object.保证金.类型 == '单品':
            直接扣除保证金
            order_object.保证金.balance余额 = 0
            order_object.保证金.save()
        else: order_object.保证金.类型 == '专场保证金':
            如果 order_object.保证金.balance余额 <= order_object.拍品.保证金:
                order_object.保证金.balance余额 = 0
                order_object.保证金.save()
            else: 余额还多
                order_object.保证金.balance余额 -= order_object.拍品.保证金
                order_object.保证金.save()

                如果 当前用户在此专场中没有其他拍到的拍品，退还余额:
                    order_object.保证金.balance余额 按照原路返回。
```

1.2 表结构

- 拍到，订单
- 没拍到，退款
- 逾期未付款，扣除保证金

```
class Order(models.Model):
    """
    订单
    """
    status_choices = (
        (1, '未支付'),
        (2, '待收货'),
        (3, '已完成'),
```

```
        (4, '逾期未支付'),
    )
    status = models.PositiveSmallIntegerField(verbose_name='状态',
choices=status_choices)

    uid = models.CharField(verbose_name='流水号', max_length=64)
    user = models.ForeignKey(verbose_name='用户', to='UserInfo')
    item = models.ForeignKey(verbose_name='拍品', to='AuctionItem')
    deposit = models.ForeignKey(verbose_name='保证金', to='DepositRecord')
    price = models.PositiveIntegerField(verbose_name='出价')
    create_date = models.DateTimeField(verbose_name='创建时间',
auto_now_add=True)

    twenty_four_task_id = models.CharField(verbose_name='24小时后定时任务',
max_length=32, null=True, blank=True)


class DepositRefundRecord(models.Model):
    """ 保证金退款记录 """
    uid = models.CharField(verbose_name='流水号', max_length=64)
    status_choices = (
        (1, "待退款"),
        (2, '退款成功'),
    )
    status = models.PositiveSmallIntegerField(verbose_name='状态',
choices=status_choices)
    deposit = models.ForeignKey(verbose_name='保证金', to='DepositRecord')
    amount = models.PositiveIntegerField(verbose_name='退款金额')


class DepositDeduct(models.Model):
    """ 扣除保证金 """
    order = models.ForeignKey(verbose_name='订单', to='Order')
    amount = models.PositiveIntegerField(verbose_name='金额')
```

1.3 具体逻辑实现

```
@shared_task
def end_auction_task(auction_id):
    # ########## 状态更新 ############
    models.Auction.objects.filter(id=auction_id).update(status=4)
    models.AuctionItem.objects.filter(auction_id=auction_id).update(status=4)

    total = 0
    total_unfortunate_list = []
    lucky_auction_deposit_id = set()

    auction_object = models.Auction.objects.filter(id=auction_id).first()
```

```
    item_object_list =
models.AuctionItem.objects.filter(auction=auction_object)

    # 循环所有的拍品
    for item_object in item_object_list:

        # 获取当前拍品出价最高者
        lucky_object =
models.BidRecord.objects.filter(item=item_object).order_by('-price').first()

        # 无出价，则流派
        if not lucky_object:
            item_object.status = 5
            item_object.save()
            continue

        lucky_object.status = 2
        lucky_object.save()

        # 拍品：设置成交价
        item_object.deal_price = lucky_object.price
        item_object.save()

        # 专场：总成交额
        total += lucky_object.price

        # 获取当前用户为此 拍品/专场 支付的保证金对象
        deposit_object = models.DepositRecord.objects.filter(
            user=lucky_object.user,
            item=item_object,
            deposit_type=1).first()

        if not deposit_object:
            deposit_object =
models.DepositRecord.objects.filter(user=lucky_object.user,
auction=auction_object,

deposit_type=2, item__isnull=True).first()
            # 所有已经拍到商品的人缴纳的保证金id
            lucky_auction_deposit_id.add(deposit_object.id)

        # 生成订单（待支付）
        order_object = models.Order.objects.create(
            uid=md5(uuid.uuid4()),
            user=lucky_object.user,
            item=item_object,
            deposit=deposit_object,  #  （单品、专场）
            price=lucky_object.price,
        )
```

```
        # 单品保证金：所有没有拍到商品 & 缴纳的是单品保证金记录。
        item_unfortunate_list =
models.DepositRecord.objects.filter(item=item_object, deposit_type=1).exclude(
            user=lucky_object.user)
        total_unfortunate_list.extend(item_unfortunate_list)

        # 调用定时任务：24小时内要支付，否则流拍扣除保证金。
        date = datetime.datetime.utcnow() + datetime.timedelta(hours=24)
        task_id = twenty_four_hour.apply_async(args=[order_object.id],
eta=date).id
        order_object.twenty_four_task_id = task_id
        order_object.save()

    # 专场：更新成交额
    auction_object.total_price = total
    auction_object.save()

    # 未拍到任何商品的用户的全场保证金
    auction_unfortunate_list = models.DepositRecord.objects.filter(
        deposit_type=2,
        auction=auction_object,
        item__isnull=True).exclude(id__in=lucky_auction_deposit_id)

    # 退保证金（原路退还）
    for deposit in itertools.chain(total_unfortunate_list,
auction_unfortunate_list):
        uid = md5(uuid.uuid4())
        if deposit.pay_type == 1:  # 微信
            # res = refund(uid, deposit.uid, deposit.amount, deposit.amount)
            res = True
            models.DepositRefundRecord.objects.create(
                uid=uid,
                status=2 if res else 1,
                amount=deposit.amount,
                deposit=deposit
            )
            if res:
                deposit.balance = 0
                deposit.save()
        else:  # 余额
            deposit.user.balance += deposit.amount
            deposit.user.save()
            models.DepositRefundRecord.objects.create(
                uid=uid,
                status=2,
                amount=deposit.amount,
                deposit=deposit
            )
```

```
            deposit.balance = 0
            deposit.save()
```

```
@shared_task
def twenty_four_hour(order_id):
    """ 24小时不支付订单，则直接扣除保证金 """

    # 订单已支付
    order_object = models.Order.objects.filter(id=order_id).first()
    if order_object.status != 1:
        return

    # 订单状态为 预期未支付
    order_object.status = 4
    order_object.save()

    # 拍品状态为 预期未支付
    order_object.item.status = 6
    order_object.item.save()

    # 单品保证金，直接扣除
    if order_object.deposit.deposit_type == 1:
        order_object.deposit.balance = 0
        order_object.deposit.save()
        models.DepositDeduct.objects.create(order=order_object,
amount=order_object.deposit.amount)
        return

    # 全场保证金，扣除部分保证金（如果有剩余，则检查是否还有其他订单了，没了则剩余保证金直接
退回到原账户）
    """
        情景一：
            全场保证金：1000
                A  9000  200     扣除200 退还800

            全场保证金：1000
                A  9000  200     扣除200
                B  800   400     扣除400 退还400


            全场保证金：1000
                A  9000  200     扣除200
                B  9000  900     扣除800 退还0

    """
    if order_object.deposit.balance <= order_object.item.deposit:
        order_object.deposit.balance = 0
        order_object.deposit.save()
```

```
        models.DepositDeduct.objects.create(order=order_object,
amount=order_object.deposit.balance)
        return

    order_object.deposit.balance -= order_object.item.deposit
    order_object.deposit.save()
    models.DepositDeduct.objects.create(order=order_object,
amount=order_object.item.deposit)

    # 检查此专场保证金下是否还有其他订单未支付
    exists = models.Order.objects.filter(user=order_object.user,
                                         status=1,

item__auction_id=order_object.deposit.auction).exclude(id=order_id).exists()
    if exists:
        return

    uid = md5(uuid.uuid4())
    if order_object.deposit.pay_type == 1:  # 微信
        # res = refund(uid, deposit.uid, deposit.amount, deposit.amount)
        res = True
        models.DepositRefundRecord.objects.create(
            uid=uid,
            status=2 if res else 1,
            amount=order_object.deposit.balance,
            deposit=order_object.deposit
        )
        if res:
            order_object.deposit.balance = 0
            order_object.deposit.save()
    else:  # 余额
        order_object.deposit.user.balance += order_object.deposit.balance
        order_object.deposit.user.save()
        models.DepositRefundRecord.objects.create(
            uid=uid,
            status=2,
            amount=order_object.deposit.balance,
            deposit=order_object.deposit
        )
        order_object.deposit.balance = 0
        order_object.deposit.save()
```

注意事项

- celery任务函数之间可以相互调用

```
@shared_task
def f1(order_id):
    result = f2.delay(...)
    result.id

@shared_task
def f2(order_id):
    pass
```

- 调用celery任务函数时，传入的参数需要可json

```
@shared_task
def twenty_four_hour(可json):
    pass
```

1.4 知识点补充

1.4.1 退款的API

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
"""
退款
根据订单号进行退款（需要使用证书才能操作）
文档：https://pay.weixin.qq.com/wiki/doc/api/H5.php?chapter=9_4&index=4
"""
import uuid
import random
import hashlib
import requests
from xml.etree import ElementTree as ET


def md5(string):
    ha = hashlib.md5()
    ha.update(string.encode('utf-8'))
    return ha.hexdigest()


def refund(trade_no, out_refund_no, total_fee, refund_fee):
    """
    订单号
    :param trade_no: 创建订单时自动生成的订单号
    :param out_refund_no: 商户退款单号
    :param total_fee: 订单金额
    :param refund_fee: 退款金额
    :return:
    """
```

```
    info = {
        'appid': 'wx55cca0b94f723dc7',
        'mch_id': '1526049051',
        'out_trade_no': trade_no,
        'nonce_str': "".join([chr(random.randint(65, 90)) for _ in
range(12)]),
        'sign_type': "MD5",
        'out_refund_no': out_refund_no,
        'total_fee': total_fee,
        'refund_fee': refund_fee
    }
    key = "2SzCvaKgYExuItWBfYAqJFs72uUleD14"
    string = "&".join(["{0}={1}".format(k, info[k]) for k in sorted(info)] +
["{0}={1}".format("key", key, ), ])
    info['sign'] = md5(string).upper()

    xml = "<xml>{0}</xml>".format("".join(["<{0}>{1}</{0}>".format(k, v) for
k, v in info.items()]))

    key = "xx/xxx/client_key.pem"
    cert = "xxx/xxx/xx/client_cert.pem"

    res = requests.post(
        url='https://api.mch.weixin.qq.com/secapi/pay/refund',
        data=xml.encode('utf-8'),
        headers={
            'Accept-Language': 'zh-CN,zh;q=0.9'
        },
        cert=(cert, key),
        verify=True
    )

    root = ET.XML(res.content.decode('utf-8'))
    response = {child.tag: child.text for child in root}
    if response['return_code'] == 'SUCCESS':
        return True


if __name__ == '__main__':
    out_refund_no = md5(str(uuid.uuid4()))
    print(out_refund_no)
    refund('8ccdbbd652d9ad12b82cf2b021669cb9', out_refund_no, 0.1, 0.1)
```

1.4.2 数据库FileField字段影响rest API

```python
class AuctionModelSerializer(serializers.ModelSerializer):
    status = serializers.SerializerMethodField()
    items = serializers.SerializerMethodField()
    cover = serializers.CharField()
    ct = serializers.SerializerMethodField()

    class Meta:
        model = models.Auction
        fields = ['ct','id', 'title', 'status', 'cover', 'total_price', 'look_count', 'goods_count', 'items']

    def get_ct(self, obj):
        # return obj.cover.path
        return obj.cover.name

    def get_status(self, obj):
        status_class_mapping = {
            2: 'preview',
            3: 'auction',
            4: 'stop'
        }
        return {'text': obj.get_status_display(), 'class': status_class_mapping.get(obj.status)}

    def get_items(self, obj):
        queryset = models.AuctionItem.objects.filter(auction=obj)[0:5]
        return [row.cover.name for row in queryset]
```

```
1. 上传文件 x1.png    ->    项目目录/static/x1.png
2. 获取地址，路径凭借 http://www.xxxx.com/ + static/x1.png
    服务器地址 + static/x1.png
3. 获取文件，django读取服务器本地 open(项目目录/static/x1.png)
    对象.cover
    本地路径 + static/x1.png

对象.name
```

开发任务

1.创建相应表结构

```python
class DepositRecord(models.Model):
    """ 保证金 """
    status_choices = (
        (1, '未支付'),
        (2, '支付成功')
    )
    status = models.PositiveSmallIntegerField(verbose_name='状态', choices=status_choices, default=1)

    uid = models.CharField(verbose_name='流水号', max_length=64)
    deposit_type_choices = (
        (1, '单品保证金'),
        (2, '全场保证金')
    )
    deposit_type = models.SmallIntegerField(verbose_name='保证金类型', choices=deposit_type_choices)
    pay_type_choices = (
        (1, '微信'),
```

```
        (2, '余额')
    )
    pay_type = models.SmallIntegerField(verbose_name='支付方式',
choices=pay_type_choices)

    amount = models.PositiveIntegerField(verbose_name='金额')  # 200
    balance = models.PositiveIntegerField(verbose_name='余额') # 0

    user = models.ForeignKey(verbose_name='用户', to='UserInfo')

    # 单品保证金则设置值，全场保证金，则为空
    item = models.ForeignKey(verbose_name='拍品', to='AuctionItem', null=True,
blank=True)

    auction = models.ForeignKey(verbose_name='拍卖', to='Auction')


class Order(models.Model):
    """
    订单，拍卖结束时，执行定时任务处理：
        - 拍得，创建订单。
        - 未拍得，则退款到原账户
    """
    status_choices = (
        (1, '未支付'),
        (2, '待收货'),
        (3, '已完成'),
        (4, '逾期未支付'),
    )
    status = models.PositiveSmallIntegerField(verbose_name='状态',
choices=status_choices)

    uid = models.CharField(verbose_name='流水号', max_length=64)
    user = models.ForeignKey(verbose_name='用户', to='UserInfo')
    item = models.ForeignKey(verbose_name='拍品', to='AuctionItem')
    deposit = models.ForeignKey(verbose_name='保证金', to='DepositRecord')
    price = models.PositiveIntegerField(verbose_name='出价')
    create_date = models.DateTimeField(verbose_name='创建时间',
auto_now_add=True)

    twenty_four_task_id = models.CharField(verbose_name='24小时后定时任务',
max_length=32, null=True, blank=True)


class DepositRefundRecord(models.Model):
    """ 保证金退款记录 """
    uid = models.CharField(verbose_name='流水号', max_length=64)
    status_choices = (
        (1, "待退款"),
```

```
        (2, '退款成功'),
    )
    status = models.PositiveSmallIntegerField(verbose_name='状态',
choices=status_choices)
    deposit = models.ForeignKey(verbose_name='保证金', to='DepositRecord')
    amount = models.PositiveIntegerField(verbose_name='退款金额')


class DepositDeduct(models.Model):
    """ 扣除保证金 """
    order = models.ForeignKey(verbose_name='订单', to='Order')
    amount = models.PositiveIntegerField(verbose_name='金额')
```

2.定时任务代码集成到项目

```
@shared_task
def end_auction_task(auction_id):
    # ########## 状态更新 ############
    models.Auction.objects.filter(id=auction_id).update(status=4)
    models.AuctionItem.objects.filter(auction_id=auction_id).update(status=4)

    total = 0
    total_unfortunate_list = []
    lucky_auction_deposit_id = set()

    auction_object = models.Auction.objects.filter(id=auction_id).first()
    item_object_list =
models.AuctionItem.objects.filter(auction=auction_object)

    # 循环所有的拍品
    for item_object in item_object_list:

        # 获取当前拍品出价最高者
        lucky_object =
models.BidRecord.objects.filter(item=item_object).order_by('-price').first()

        # 无出价，则流派
        if not lucky_object:
            item_object.status = 5
            item_object.save()
            continue

        lucky_object.status = 2
        lucky_object.save()

        # 拍品：设置成交价
        item_object.deal_price = lucky_object.price
        item_object.save()
```

```
            # 专场：总成交额
            total += lucky_object.price

            # 获取当前用户为此 拍品/专场 支付的保证金对象
            deposit_object = models.DepositRecord.objects.filter(
                user=lucky_object.user,
                item=item_object,
                deposit_type=1).first()

            if not deposit_object:
                deposit_object =
models.DepositRecord.objects.filter(user=lucky_object.user,
auction=auction_object,

deposit_type=2, item__isnull=True).first()
                # 所有已经拍到商品的人缴纳的保证金id
                lucky_auction_deposit_id.add(deposit_object.id)

            # 生成订单（待支付）
            order_object = models.Order.objects.create(
                uid=md5(uuid.uuid4()),
                user=lucky_object.user,
                item=item_object,
                deposit=deposit_object,  #  （单品、专场）
                price=lucky_object.price,
            )

            # 单品保证金：所有没有拍到商品 & 缴纳的是单品保证金记录。
            item_unfortunate_list =
models.DepositRecord.objects.filter(item=item_object, deposit_type=1).exclude(
                user=lucky_object.user)
            total_unfortunate_list.extend(item_unfortunate_list)

            # 调用定时任务：24小时内要支付，否则流拍扣除保证金。
            date = datetime.datetime.utcnow() + datetime.timedelta(hours=24)
            task_id = twenty_four_hour.apply_async(args=[order_object.id],
eta=date).id
            order_object.twenty_four_task_id = task_id
            order_object.save()

        # 专场：更新成交额
        auction_object.total_price = total
        auction_object.save()

        # 未拍到任何商品的用户的全场保证金
        auction_unfortunate_list = models.DepositRecord.objects.filter(
            deposit_type=2,
            auction=auction_object,
            item__isnull=True).exclude(id__in=lucky_auction_deposit_id)
```

```
    # 退保证金（原路退还）
    for deposit in itertools.chain(total_unfortunate_list,
auction_unfortunate_list):
        uid = md5(uuid.uuid4())
        if deposit.pay_type == 1:  # 微信
            # res = refund(uid, deposit.uid, deposit.amount, deposit.amount)
            res = True
            models.DepositRefundRecord.objects.create(
                uid=uid,
                status=2 if res else 1,
                amount=deposit.amount,
                deposit=deposit
            )
            if res:
                deposit.balance = 0
                deposit.save()
        else:  # 余额
            deposit.user.balance += deposit.amount
            deposit.user.save()
            models.DepositRefundRecord.objects.create(
                uid=uid,
                status=2,
                amount=deposit.amount,
                deposit=deposit
            )
            deposit.balance = 0
            deposit.save()


@shared_task
def twenty_four_hour(order_id):
    """ 24小时不支付订单，则直接扣除保证金 """

    # 订单已支付
    order_object = models.Order.objects.filter(id=order_id).first()
    if order_object.status != 1:
        return

    # 订单状态为 预期未支付
    order_object.status = 4
    order_object.save()

    # 拍品状态为 预期未支付
    order_object.item.status = 6
    order_object.item.save()

    # 单品保证金，直接扣除
    if order_object.deposit.deposit_type == 1:
```

```
        order_object.deposit.balance = 0
        order_object.deposit.save()
        models.DepositDeduct.objects.create(order=order_object,
amount=order_object.deposit.amount)
        return

    # 全场保证金，扣除部分保证金（如果有剩余，则检查是否还有其他订单了，没了则剩余保证金直接
退回到原账户）
    """
        情景一：
            全场保证金：1000
                A  9000  200     扣除200 退还800

            全场保证金：1000
                A  9000  200     扣除200
                B  800   400     扣除400 退还400


            全场保证金：1000
                A  9000  200     扣除200
                B  9000  900     扣除800 退还0

    """
    if order_object.deposit.balance <= order_object.item.deposit:
        order_object.deposit.balance = 0
        order_object.deposit.save()
        models.DepositDeduct.objects.create(order=order_object,
amount=order_object.deposit.balance)
        return

    order_object.deposit.balance -= order_object.item.deposit
    order_object.deposit.save()
    models.DepositDeduct.objects.create(order=order_object,
amount=order_object.item.deposit)

    # 检查此专场保证金下是否还有其他订单未支付
    exists = models.Order.objects.filter(user=order_object.user,
                                         status=1,

item__auction_id=order_object.deposit.auction).exclude(id=order_id).exists()
    if exists:
        return

    uid = md5(uuid.uuid4())
    if order_object.deposit.pay_type == 1:  # 微信
        # res = refund(uid, deposit.uid, deposit.amount, deposit.amount)
        res = True
        models.DepositRefundRecord.objects.create(
            uid=uid,
```

```
            status=2 if res else 1,
            amount=order_object.deposit.balance,
            deposit=order_object.deposit
        )
        if res:
            order_object.deposit.balance = 0
            order_object.deposit.save()
    else:  # 余额
        order_object.deposit.user.balance += order_object.deposit.balance
        order_object.deposit.user.save()
        models.DepositRefundRecord.objects.create(
            uid=uid,
            status=2,
            amount=order_object.deposit.balance,
            deposit=order_object.deposit
        )
        order_object.deposit.balance = 0
        order_object.deposit.save()
```

3.测试

- 在后台创建专场：预展、拍卖、结束，后台管理检查状态。
- 在后台创建专场+拍品：预展、拍卖、结束，后台管理检查状态。

2. 临时分享 django contenttypes组件

2.1 需求：课程&价格设计方案

大课	
ID	课程
1	python
2	linux
3	数据分析

价格			
ID	周期	价格	大课程ID
1	30	10000	1
2	60	15000	1
3	90	18000	1
4	30	8000	2
5	60	10000	2
6	30	20000	3
7	60	25000	3

小课	
ID	课程
1	crm项目
2	小程序开发

价格			
ID	周期	价格	小课程ID
1	30	10	1
2	60	20	1
3	90	30	1
4	30	10	2
5	60	20	2

软件服务	
ID	课程
1	运维
2	系统修复

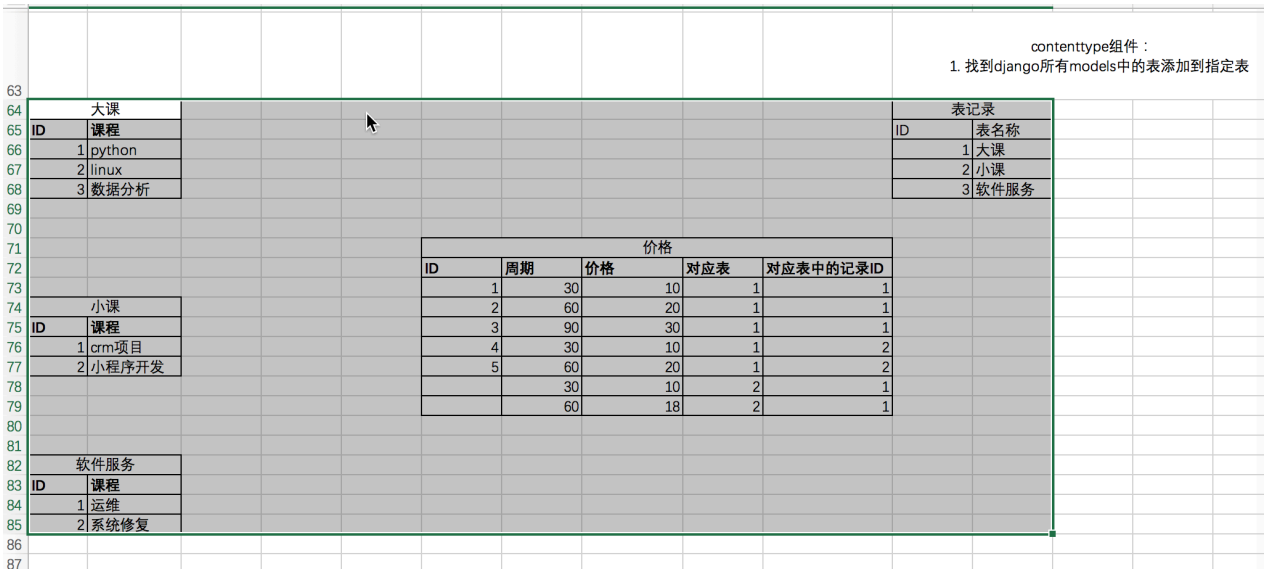
价格			
ID	周期	价格	软件服务ID
1	30	10	1
2	60	20	1
3	90	30	1
4	30	10	2
5	60	20	2

大课	
ID	课程
1	python
2	linux
3	数据分析

小课	
ID	课程
1	crm项目
2	小程序开发

软件服务	
ID	课程
1	运维
2	系统修复

价格					
ID	周期	价格	大课程ID	小课ID	软件服务ID
1	30	10000	1		
2	60	15000	1		
3	90	18000	1		
4	30	8000	2		
5	60	10000	2		
6	30	20000	3		
7	60	25000	3		
	30	10		1	
	60	20		1	
	90	30		1	
	30	10		2	
	60	20		2	
1	30	10			1
2	60	20			1
3	90	30			1
4	30	10			2
5	60	20			2



contenttype组件：
1. 找到django所有models中的表添加到指定表

	大课	
ID	课程	
1	python	
2	linux	
3	数据分析	

	小课	
ID	课程	
1	crm项目	
2	小程序开发	

	软件服务	
ID	课程	
1	运维	
2	系统修复	

表记录	
ID	表名称
1	大课
2	小课
3	软件服务

价格				
ID	周期	价格	对应表	对应表中的记录ID
1	30	10	1	1
2	60	20	1	1
3	90	30	1	1
4	30	10	1	2
5	60	20	1	2
	30	10	2	1
	60	18	2	1

```python
from django.db import models
from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes.fields import
GenericForeignKey,GenericRelation

# Create your models here.

class BigCourse(models.Model):
    """大课"""
    name = models.CharField(max_length=128)

    # 不会创建额外列，帮助你快速操作。
    price_policy = GenericRelation("PricePolicy")


class SmallCourse(models.Model):
    """小"""
    name = models.CharField(max_length=128)


class PricePolicy(models.Model):
    """价格策略"""
    period = models.IntegerField(verbose_name='周期')
    price = models.FloatField(verbose_name='价格')

    content_type = models.ForeignKey(ContentType)
    object_id = models.PositiveIntegerField()
    # 不会创建额外列，帮助你快速操作。
    content_object = GenericForeignKey('content_type', 'object_id')
```

2.2 需求：添加数据

- 创建一个大课 & 创建3个价格策略（笨方法）

```
big_object = models.BigCourse.objects.create(name='Python')

ct =
ContentType.objects.filter(app_label='app02',model='bigcourse').first()
models.PricePolicy.objects.create(
    period=30,
    price=10000,
    content_type=ct,
    object_id=big_object.id
)
models.PricePolicy.objects.create(
    period=60,
    price=15000,
    content_type=ct,
    object_id=big_object.id
)
models.PricePolicy.objects.create(
    period=90,
    price=18000,
    content_type=ct,
    object_id=big_object.id
)
```

- 创建一个大课 & 创建3个价格策略（简便方法）

```
big_object = models.BigCourse.objects.create(name='Linux')

models.PricePolicy.objects.create(
    period=30,
    price=10000,
    content_object=big_object
)
models.PricePolicy.objects.create(
    period=60,
    price=15000,
    content_object=big_object
)
models.PricePolicy.objects.create(
    period=90,
    price=18000,
    content_object=big_object
)
```

- 创建一个小课 & 创建3个价格策略（简便方法）

```
small_object = models.SmallCourse.objects.create(name='CRM')

models.PricePolicy.objects.create(
```

```
    period=30,
    price=10000,
    content_object=small_object
)
models.PricePolicy.objects.create(
    period=60,
    price=15000,
    content_object=small_object
)
models.PricePolicy.objects.create(
    period=90,
    price=18000,
    content_object=small_object
)
```

- 获取所有价格策略

```
data_list = models.PricePolicy.objects.all()
for item in data_list:
    item.id
    item.price
    # 字段找到与之关联的对象：BigCourse/SmallCourse
    item.content_object
```

- 获取大课 Python 的所有价格策略

```
course_object = models.BigCourse.objects.filter(name='Python').first()

price_object_list = course_object.price_policy.all()
```

3.临时分享：related_name/related_query_name的区别?

```
class Department(models.Model):
    title = models.CharField(verbose_name='名称',max_length=32)

class UserInfo(models.Model):
    depart = models.ForeignKey(verbose_name='部门',to='Department')
    user = models.CharField(verbose_name='用户',max_length=32)
    pwd = models.CharField(verbose_name='用户', max_length=32)
```

- 以前查询

```
正向查找:
    user_object = UserInfo.objects.get(id=1)
    user_object.depart
反向操作:
    depart_object = Department.objects.get(id=9)
    depart_object.userinfo_set.all()
```

- related_query_name

```
class Department(models.Model):
    title = models.CharField(verbose_name='名称',max_length=32)

class UserInfo(models.Model):
    depart=models.ForeignKey(verbose_name='部
门',to='Department',related_query_name="u")
    user = models.CharField(verbose_name='用户',max_length=32)
    pwd = models.CharField(verbose_name='用户', max_length=32)
```

```
正向查找:
    user_object = UserInfo.objects.get(id=1)
    user_object.depart
反向操作:
    depart_object = Department.objects.get(id=9)
    depart_object.u_set.all()
```

- related_name

```
class Department(models.Model):
    title = models.CharField(verbose_name='名称',max_length=32)

class UserInfo(models.Model):
    depart=models.ForeignKey(verbose_name='部
门',to='Department',related_name="u")
    user = models.CharField(verbose_name='用户',max_length=32)
    pwd = models.CharField(verbose_name='用户', max_length=32)
```

```
正向查找:
    user_object = UserInfo.objects.get(id=1)
    user_object.depart
反向操作:
    depart_object = Department.objects.get(id=9)
    depart_object.u.all()
```

https://www.cnblogs.com/wupeiqi/articles/6216618.html