1.3 缓存、验证码、登录、表单验证

DJANGO 中的缓存

1. 接口及用法

```
from django.core.cache import cache

# 在缓存中设置 age = 123, 10秒过期
cache.set('age', 123, 10)

# 获取 age
a = cache.get('age')
print(a)

# 自增
x = cache.incr('age')
print(x)
```

2. 使用 Redis 做缓存后端
    - 安装 `pip install django-redis`
    - settings 配置

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/0",
        "OPTIONS": {
            "CLIENT_CLASS":
"django_redis.client.DefaultClient",
            "PICKLE_VERSION": -1,
        }
    }
}
```

3. 利用过期时间可以处理一些定时失效的临时数据，比如手机验证码
4. 面试题：
    1. 如何用Redis做缓存?

COOKIE、 SESSION 机制剖析

1. 回顾 HTTP 协议
    1. host
        1. linux: /etc/hosts
        2. windows: C:/Windows/System32/drivers/etc/hosts
2. 产生过程
    1. 浏览器: 向服务器发送请求
    2. 服务器: 接受并创建 session 对象 (该对象包含一个 session_id)
    3. 服务器: 执行 views 函数, 并得到一个 response 对象
    4. 服务器: 执行 response.set_cookie('sessionid', session_id) 将 session_id 写入 cookie
    5. 服务器: 将 response 传回浏览器
    6. 浏览器: 读取 response 报文, 从 Cookies 取出 session_id 并保存
3. 后续请求
    1. 浏览器: 向服务器发送请求, session_id 随 Cookies 一同发给 Server
    2. 服务器: 从 Headers 的 Cookies 中取出 session_id
    3. 服务器: 根据 session_id 找出对应的数据, 确认客户端身份
4. Django 中的代码实现

```
class SessionMiddleware(MiddlewareMixin):
    def __init__(self, get_response=None):
        self.get_response = get_response
        engine = import_module(settings.SESSION_ENGINE)
        self.SessionStore = engine.SessionStore  # 设置 Session 存
储类

    def process_request(self, request):
        # 从 Cookie 获取 sessionid
        session_key = request.COOKIES.get('session_id')

        # 通过 session_key 获取之前保存的数据
        request.session = self.SessionStore(session_key)

    def process_response(self, request, response):
        try:
            # View 函数结束后, 获取 session 状态
            accessed = request.session.accessed
            modified = request.session.modified
            empty = request.session.is_empty()
        except AttributeError:
            pass
        else:
            # 如果 Cookie 中有 sessionid, 但 session 为空,
            # 说明 view 中执行过 session.flush 等操作,
            # 直接删除 Cookie 中的 session
            if 'session_id' in request.COOKIES and empty:
                response.delete_cookie(
                    settings.SESSION_COOKIE_NAME,
```

```
                    path=settings.SESSION_COOKIE_PATH,
                    domain=settings.SESSION_COOKIE_DOMAIN,
                )
            else:
                if accessed:
                    patch_vary_headers(response, ('Cookie',))
                if (modified or
settings.SESSION_SAVE_EVERY_REQUEST) and not empty:
                    # 设置过期时间
                    if
request.session.get_expire_at_browser_close():
                        max_age = None
                        expires = None
                    else:
                        max_age =
request.session.get_expiry_age()
                        expires_time = time.time() + max_age
                        expires = cookie_date(expires_time)

                    # 保存会话数据，并刷新客户端 Cookie
                    if response.status_code != 500:
                        try:
                            request.session.save()
                        except UpdateError:
                            raise SuspiciousOperation(
                                "The request's session was
deleted before the "
                                "request completed. The user may
have logged "
                                "out in a concurrent request, for
example."
                            )

                        # 让客户端将 sessionid 添加到 Cookie 中
                        response.set_cookie(
                            'session_id',
                            request.session.session_key,
                            max_age=max_age,
                            expires=expires,

 domain=settings.SESSION_COOKIE_DOMAIN,
                            path=settings.SESSION_COOKIE_PATH,
                            secure=settings.SESSION_COOKIE_SECURE
or None,

 httponly=settings.SESSION_COOKIE_HTTPONLY or None,
                        )
        return response
```

5. 面试题：

    1. 如何区分 Cookie、Session?

2. 请描述 HTTP 协议

验证短信

1. urls.py 增加新的url路由映射
    1. url(r'api/user/submit/vcode', user_api.submit_vcode)
2. 实现路由映射：
    1. submit_vcode
        1. 获取手机号、验证码
        2. 取出缓存里的对应这个手机号的验证码
        3. 检查验证码一致，登录或自动注册
        4. 不一致返回错误

个人资料接口规划

1. 获取个人资料接口
2. 修改个人资料接口
3. 上传个人头像接口

PROFILE 模型设计 (仅作参考)

Field	Description
location	目标城市
min_distance	最小查找范围
max_distance	最大查找范围
min_dating_age	最小交友年龄
max_dating_age	最大交友年龄
dating_sex	匹配的性别
vibration	开启震动
only_matche	不让为匹配的人看我的相册

开发中的难点

1. Profile 与 User 两个模型是什么关系？
2. 企业中不使用外键如何构建 "表关联" ？
3. 接口中有太多字段批量提交时应如何验证？
4. 如何上传头像？ https://docs.djangoproject.com/zh-hans/2.0/topics/http/file-uploads/
5. 大型项目中如何保存大量的静态文件？
6. 上传文件、发送验证码、图像处理等较慢操作应如何处理才能让用户等待时间更短？

数据库表关系的构建

1. 关系分类
    - 一对一关系
    - 一对多关系
    - 多对多关系
2. 外键的优缺点
    - 优点:
        - 由数据库自身保证数据一致性和完整性, 数据更可靠
        - 可以增加 ER 图的可读性
        - 外键可节省开发量
    - 缺点:
        - 性能缺陷, 有额外开销
        - 主键表被锁定时, 会引发外键对应的表也被锁
        - 删除主键表的数据时, 需先删除外键表的数据
        - 修改外键表字段时, 需重建外键约束
        - 不能用于分布式环境
        - 不容易做到数据解耦
3. 应用场景
    - 适用场景: 内部系统、传统企业级应用可以使用 (需要数据量可控, 数据库服务器数量可控)
    - 不适用场景: 互联网行业不建议使用
4. 手动构建关联
    1. 一对一: 主表 id 与 子表 id 完全一一对应
    2. 一对多: 在 "多" 的表内添加 "唯一" 表 id 字段
    3. 多对多: 创建关系表, 关系表中一般只存放两个相关联的条目的 id
    4. 博客案例思考
        1. 用户和文字的关系：一对多
        2. 用户和收藏关系：多对多

3. 用户-角色-权限关系
4. 用户-角色：一对多
    2. 角色-权限：多对多
5. 可通过 `property` 的方式对子表进行关联操作
    - property 用法

```
class Box:
    def __init__(self):
        self.l = 123
        self.w = 10
        self.h = 80

    @property
    def V(self):
        return self.l * self.w * self.h

b = Box()
print(b.V)
```

    - 对子表关联操作

```
class User(models.Model):
    ...
    user_id = models.IntergerField()
    ...

    @property
    def user_profile(self):
        if not hasattr(self, '_profile'):
            self._profile =
Profile.objects.get(user_id=self.user_id)
        return self._profile



class User(models.Model):
...
    demo_id = models.IntegerField()
    ...

    @property
def demo(self):
        if not hasattr(self, '_demo'):
        self._demo = Demo.objects.get(id=self.demo_id)
        return self._demo
```

class Demo(models.Model): xxx = models.CharField() yyy = models.CharField()

```
user = User.objects.get(id=123)
```

print(user.demo.xxx) print(user.demo.yyy) ```

- 也可以使用 cached_property 对属性值进行缓存

```
from django.utils.functional import cached_property

class User(models.Model):
    year = 1990
    month = 10
    day = 29

    @cached_property
    def age(self):
        today = datetime.date.today()
        birth_date = datetime.date(self.year, self.month,
self.day)
        times = today - birth_date
        return times.days // 365
```

- 面试题：
  - 复杂SQL查询：分组、连接
    - Student表有：id和name字段
    - Score表有：student_id, exam_id, object_id, score
    - 求：
      1. 某次考试成绩最好的学生
    - 进阶思考：每次考试成绩前三的学生?
      - 提示：窗口函数。

DJANGO 中的 FORM 表单验证

- Django Form 核心功能：数据验证
- 网页中 `<form>` 标签
  - `<form>` 标签的 method 只能是 POST 或 GET
  - method=POST 时，表单数据在请求的 body 部分
  - method=GET 时, 表单数据会出现在 URL 里
- Form 对象的属性和方法
  - `form.is_valid()` : 表单验证
  - `form.has_changed()` : 检查是否有修改
  - `form.clean_<field>()` : 针对某字段进行特殊清洗和验证
  - `form.cleaned_data['fieldname']` : 清洗后的数据存放于这个属性
- Form 的定义和使用

```
from django import forms

class TestForm(forms.Form):
    TAGS = (
        ('py', 'python'),
        ('ln', 'linux'),
        ('dj', 'django'),
    )
    fid = forms.IntegerField()
    name = forms.CharField(max_length=10)
    tag = forms.ChoiceField(choices=TAGS)
    date = forms.DateField()

POST = {'fid': 'bear',
        'name': 'hello-1234567890',
        'tag': 'django',
        'date': '2017/12/17'}
form = TestForm(POST)
print(form.is_valid())
print(form.cleaned_data)  # cleaned_data 属性是 is_valid 函数执行时动态添加的
print(form.errors)
```

- ModelForm 可以通过相应的 Model 创建出 Form

```
class UserForm(ModelForm):
    class Meta:
        model = User
        fields = ['name', 'birth']
```

云服务市场

1. **AWS**：亚马逊：Amazon Web Service，全球领先
    1. 云服务器：EC2：elastic cloud computer
    2. 云存储：S3：simple storage service
    3. 云数据库：RDS：relationship database service：MySQL
    4. 消息队列：SQS：Simple Queue Service
    5. 全文检索：
    6. 云数据仓库：redshift
2. Azure：微软：非常多的linux服务器，各种服务都有
3. Google Cloud
4. **阿里云**：国内第一
    1. 云服务器：ECS
    2. 云存储：OSS
    3. 云数据库：RDS
5. 腾讯云

1. ...

6. 不值一提的云：
    1. 金山云
    2. 百度云
    3. 华为云
7. 专用云：
    1. 存储：
        1. 七牛
            1. 图片缩放：
                1. 新建图片样式
                2. 选择自己想要的缩放方式
                3. 起名字
                4. 保存
                5. 使用方法：
                    1. http://pvhv7tfry.bkt.clouddn.com/space_city.jpg?imageView2/1/w/200/h/200/q/75|imageslim
            2. 图片的渐进式显示：
                1.
        2. 又拍
    2. 直播：
    3. 短信：云之讯
    4. 邮件：sendcloud
8. 上云之后服务的构建方法：
    1. 只用云服务器，其他都自己在云服务器里实现
        1. 不依赖云，换云或离开的成本极低
        2. 用的人多，开发周期长
    2. 充分利用各种云服务
        1. 节约人力成本，用的人少，开发周期短
        2. 依赖于云服务，换云或者离开云的成本高
        3. 在云上自己做高可用
9. 不上云，
    1. 传统业务
    2. 纯流量型业务：视频分发
10. 面试题：
    1. 你们用了什么云服务?
    2. 都用了哪些功能?
    3. 多少钱?