# 1.4 上传文件、异步、初始化脚本

## 项目中的静态文件处理

1. Nginx

   Nginx 处理静态资源速度非常快，并且自身还带有缓存.

   但需要注意，分布式部署的多台 Nginx 服务器上，静态资源需要互相同步

   80: Nginx -> {server config} -> django:8080

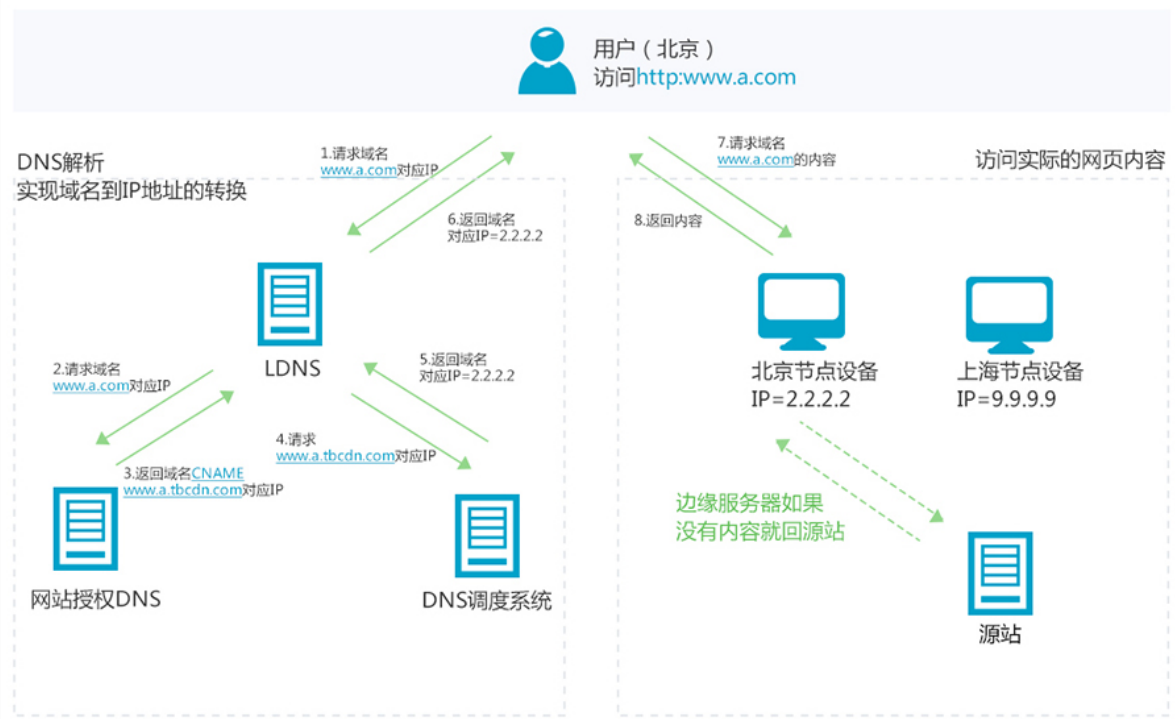
   -> Static : folders

   -> 云存储

   独立域名：static.baidu.com

   1. 过去：一个浏览器只能向同一个域名发起两个并发的连接
      1. 一个浏览器向同一个域名能发起的并发请求4-10
   2. 提高网页端的加载速度，改进用户体验
   3. 图片域名会用很多个img0...img7
   4. 更深的一点：独立域名就可以省掉传cookie了
2. 第三方的 CDN 服务：阿里云、51CDN(网宿)、蓝汛
   1. 分发：
      1. 分发的时候，我们自己只需要发到自己的一台服务器上
      2. CDN服务商负责把这个资源分发到全网的各个节点
   2. 用户在访问时：
      1. 请求域名
         1. CNAME -> GeoDNS
         2. 根据地理位置返回离用户最近的 IP
         3. GeoDNS
            1. static.somedomain.com - > china - > u.v.w.x
            2. static.somedomain.com - > amarica - > a.b.c.d
            3. static.somedomain.com - > canada - > 1.2.3.4
      4. 分发
         1. 先把资源传到一个服务器上：此服务器称为**源服务器**
         2. 主动分发

        1. 主动从 源服务器上 copy 到 其他服务器上
    3. 回源请求
        1. 用户向 static.somedomain.com 发请求时
            1. 命中 某个服务器
            2. 该服务器有此文件，直接还给用户
            3. 该服务器无此文件，此服务器向源服务器发请求，拿到文件后，再还给用户，缺点：第一个用户需要等。
            4. 模拟用户发一次请求，服务器上文件就有了
2. 用什么指标来分析和衡量一个CDN性能好坏呢?
    1. 命中数
    2. 回源数
    3. 边缘节点的延迟高
        1. 5G低延迟：10ms以内
        2. 4G延迟：50-100ms
    4. 边缘节点的流量
    5. 配置错误
3. 用户请求自然是从离他最近的服务器下载




CDN 的全称是 Content Delivery Network, 即内容分发网络.

它依靠部署在各地的边缘服务器, 通过中心平台的负载均衡、内容分发、调度等功能模块, 使用户就近获取所需内容, 降低网络拥塞, 提高用户访问响应速度和命中率. CDN 的关键技术主要有内容存储和分发技术.

3. 云存储
    - 常见的云存储有：亚马逊 S3 服务、阿里云的 OSS 、七牛云 等
4. 面试题：
    1. 你了解CDN吗？能讲讲原理么?

# 七牛云接入

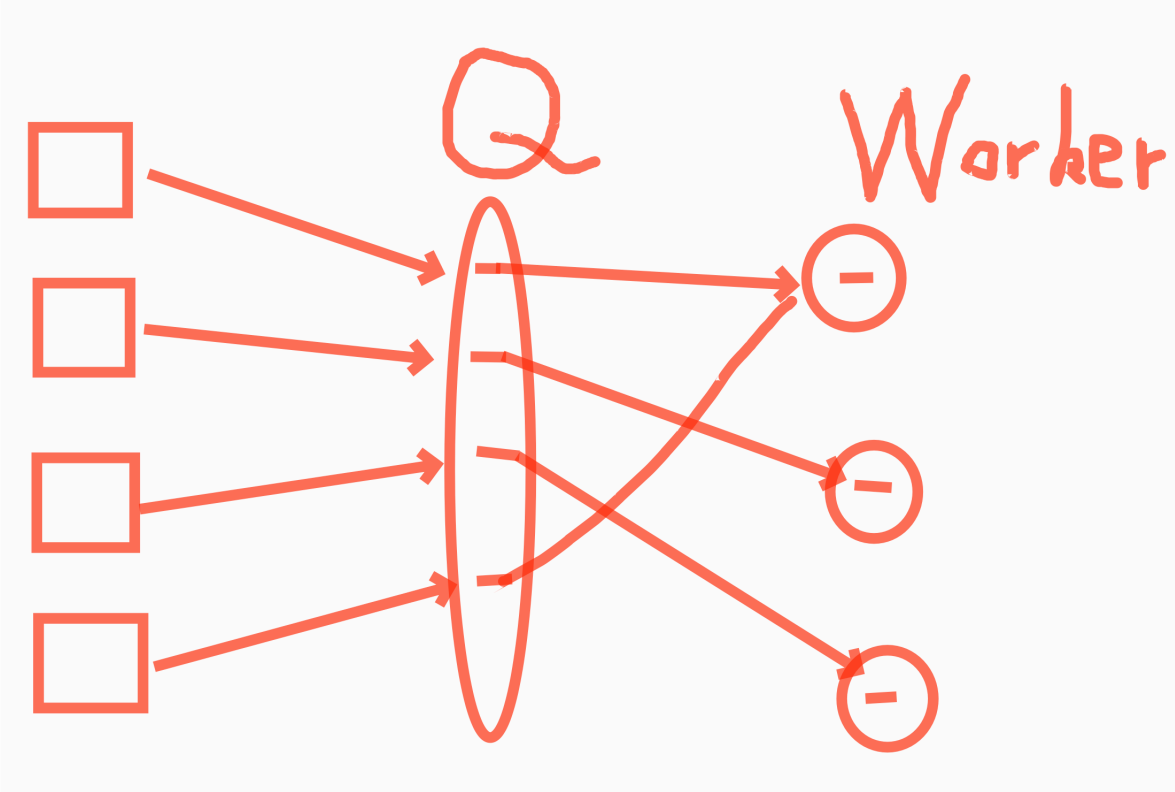
1. 注册七牛云账号
2. 创建存储空间
3. 获取相关配置
    - AccessKey
    - SecretKey
    - Bucket_name
    - Bucket_URL
4. 安装 qiniu SDK：`pip install qiniu`
5. 根据接口文档进行接口封装
6. 按照需要将上传、下载接口封装成异步任务
7. 程序处理流程
    1. **服务端上传方式**
        1. 用户图片上传服务器
        2. 服务器将图片上传到七牛云
        3. 上传成功后，拼接 avatar 的图片 url 地址：七牛云的 Bucket_URL/filename，将 avatar 的图片 url 存入数据库
    2. 客户端上传
        1. 客户端图片直接上传到云服务
        2. 客户端将图片地址告诉服务端，服务端更新数据库
        3. 其实存在一个安全隐患

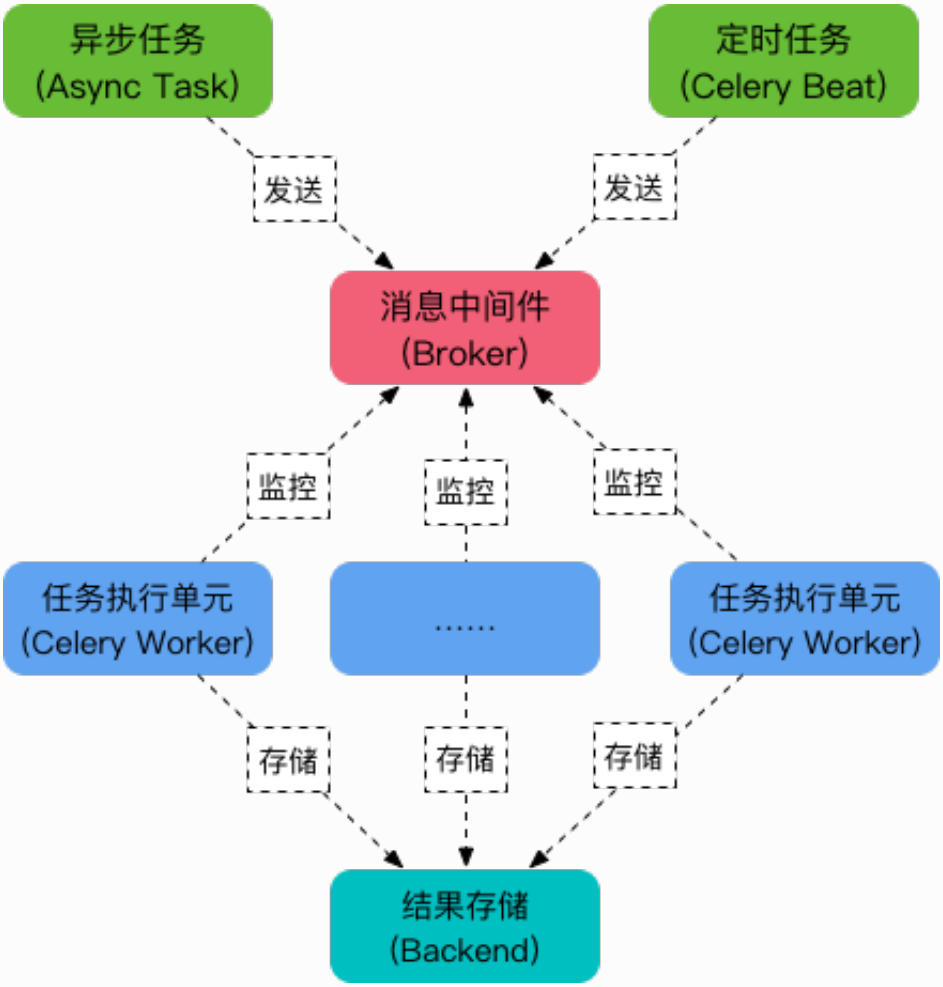
# 异步任务

- 同步：主线顺序执行，遇到中间有费时的操作，会卡住等待
    - 存钱，必须等数据都写好没问题了才返回
    - 游戏动作，必须按顺序完成的，Boss的最后一血，到底算谁砍的，算错了麻烦就大了
- 异步：主线顺序执行，遇到中间有费时的操作，就丢出一个分支去单独执行，主线不会等待，会继续往下执行，给用户更好的体验
    - 发短信发邮件，单独开一个分支去发就好了，不需要用户等到结果
    - 上传视频，传完后要编码，这个不需要用户等，单独开分支去编码就可以了
    - 运行一个费时的报告，不需要用户等，可以在报告就绪的时候给用户发一个消息提醒
    - 网页加载：图片就是异步后加载的，图片下载完之前，先出一个占位符，没必要非等头部的图片都出来了才往下加载

# CELERY 及异步任务的处理

1. 模 块 组 成




- 任务模块 Task

  包含异步任务和定时任务. 其中, 异步任务通常在业务逻辑中被触发并发往任务队列, 而定时任务由 Celery Beat 进程周期性地将任务发往任务队列.

- 消息中间件 Broker

  Broker，即为任务调度队列，接收任务生产者发来的消息（即任务），将任务存入队列。Celery 本身不提供队列服务，官方推荐使用 RabbitMQ 和 Redis 等。

- 任务执行单元 Worker

  Worker 是执行任务的处理单元，它实时监控消息队列，获取队列中调度的任务，并执行它。

- 任务结果存储 Backend

  Backend 用于存储任务的执行结果，以供查询。同消息中间件一样，存储也可使用 RabbitMQ，Redis 和 MongoDB 等。

2. 安装

```
pip install 'celery[redis]'
```

3. 创建实例

```
import time
from celery import Celery

broker = 'redis://127.0.0.1:6379'
backend = 'redis://127.0.0.1:6379/0'
app = Celery('my_task', broker=broker, backend=backend)

@app.task
def add(x, y):
    time.sleep(5)      # 模拟耗时操作
    return x + y
```

4. 启动 Worker

```
celery worker -A tasks --loglevel=info
```

5. 调用任务

```
from tasks import add

add.delay(2, 8)
```

6. 常规配置

```
broker_url = 'redis://127.0.0.1:6379/0'
broker_pool_limit = 1000  # Borker 连接池，默认是10

timezone = 'Asia/Shanghai'
accept_content = ['pickle', 'json']

task_serializer = 'pickle'
result_expires = 3600  # 任务过期时间
```

```
result_backend = 'redis://127.0.0.1:6379/0'
result_serializer = 'pickle'
result_cache_max = 10000  # 任务结果最大缓存数量

worker_redirect_stdouts_level = 'INFO'
```

# CELERY 工作的流程

1. 一次性的配置工作
    1. 安装：pip install 'celery(redis)'
    2. 常规配置：
        1. ...
2. 创建实例
    1. worker.__init__.py

```
import os

from celery import Celery

from worker import config

os.environ.setdefault("DJANGO_SETTINGS_MODULE",
"swiper.settings")

# 创建 celery 的实例
celery_app = Celery('swiper')

# 配置来自 worker.config
celery_app.config_from_object(config)

# 自动发现任务
celery_app.autodiscover_tasks()
```

3. 给需要异步的方法，添加装饰器

```
from worker import celery_app

# 给需要异步的方法，添加装饰器
@celery_app.task
def function_name():
```

4. 发布异步任务(并未真正执行)：
    1. function_name.delay(params)
5. 后台启动异步任务工作(真正的执行)：
    1. **celery worker -A worker --loglevel=info**

6. 面试题：
    1. Celery任务执行出错如何处理?

# SOCIAL 功能概述

1. 需求
    1. 交友模块
        - 获取推荐列表
        - 喜欢 / 超级喜欢 / 不喜欢
        - 反悔 (每天允许反悔 3 次)
        - 查看喜欢过我的人
    2. 好友模块
        - 查看好友列表
        - 查看好友信息

# 开发中的难点

1. 滑动需有大量用户，如何初始化大量用户以供测试?
2. 推荐算法
3. 如何从推荐列表中去除已经滑过的用户
4. 滑动操作，如何避免重复滑动同一人
5. 如果双方互相喜欢，需如何处理
6. 好友关系如何记录，数据库表结构如何设计?
7. 反悔接口
    1. "反悔"都应该执行哪些操作
    2. 每日只允许"反悔" 3 次应如何处理
    3. 后期运营时，如何方便的修改反悔次数
8. 内部很深的逻辑错误如何比较方便的将错误码返回给最外层接口

# 关系分析

1. 滑动者与被滑动者
    - 一个人可以滑动很多人
    - 一个人可以被多人滑动
    - 结论: 同表之内构建起来的逻辑上的多对多关系，Swiper实际上是一个关系表
2. 用户与好友

- 一个用户有多个好友
- 一个用户也可以被多人加为好友
- 结论: 同表之内构建起来的逻辑上的多对多关系, Friend 表实际上就是一个关系表

# 模型设计参考

1. Swiped (划过的记录)

| Field | Description |
| --- | --- |
| uid | 用户自身 id |
| sid | 被滑的陌生人 id |
| mark | 滑动类型 |
| time | 滑动的时间 |

2. Friend (匹配到的好友)

| Field | Description |
| --- | --- |
| uid1 | 好友 ID |
| uid2 | 好友 ID |

# 类方法与静态方法

- `method`
- 通过实例调用
  - 可以引用类内部的 **任何属性和方法**
- `classmethod`
  - 无需实例化
  - 可以调用类属性和类方法
  - 无法取到普通的成员属性和方法
- `staticmethod`
  - 无需实例化
  - **无法**取到类内部的任何属性和方法, 完全独立的一个方法

# 利用 Q 对象进行复杂查询

```
from django.db.models import Q

# AND
Model.objects.filter(Q(x=1) & Q(y=2))

# OR
Model.objects.filter(Q(x=1) | Q(y=2))

# NOT
Model.objects.filter(~Q(name='kitty'))
```

# 1.5代码：

1. python manage.py makemigerations
2. python manage.py migerate
3. ./scripts/init.py
4. django shell，用django shell调试

```
from lib.http import render_json
from social import logics
from social.models import Swiped
import datetime

from django.core.cache import cache

from swiper import config
from common import errors
from user.models import User
from social.models import Swiped
from social.models import Friend

userid = 2218
swiped = Swiped.objects.filter(uid=userid).only('sid')
swiped_uid_list = [sw.sid for sw in swiped]
swiped_uid_list.append(userid)

user = User.objects.get(id=uid)

users = User.objects.filter(
        location=user.profile.location,
        sex=user.profile.dating_sex,
        birth_year__range=[1984, 1999]
    ).exclude(id__in=swiped_uid_list)[:20]
users
```

编程范式：

1. 结构化编程
    1. 顺序思维
2. 代码 + 数据结构
3. 面向对象编程
    1. 对象思维
4. OOP
5. 函数式编程
    1. 数学思维