定时任务

1. crontab

   Linux 本身自带的一个命令，由Linux操作系统维护定时任务

2. APScheduler

   Python实现的，定时不是由Linux操作系统维护，是单独开启进程的方式，在进程中管理定时

定时任务有两种：

- 定时任务进行页面静态化 在django运行起来之前，我们明确知道要有这个定任务

  不是动态添加的

  - 一般可以直接采用Linux crontab
  - APScheduler

- 在django 程序已经运行的情况下

  用户1下单 判断在30分钟内必须支付，否则取消订单恢复库存

  在用户下单的时刻起，创建一个30分钟的定时任务，任务的功能是到30分钟的时刻，判断订单状态，如果未支付，取消订单

  (动态添加定时任务)

  -> 支付

  - 如果是用crontab 不是很方便， 通常使用APScheduler

子系统对接

在首页中 获取特定用户的推荐文章列表 需要web系统和推荐系统配合

- 有web系统告知 推荐系统 用户id是谁
- 推荐系统 根据用户id 决定 推荐的文章id
- web系统 根据推荐的文章id 查询文章数据，返回给客户端

在Web系统中

构建客户端请求首页数据的接口

GET /articles?channel_id=10

```
class ArticleListResource(Resource):

    def get(self):
        channel_id
        user_id
        # 调用推荐系统的接口 获取推荐文章id
        ret = recommend_article(channel_id, user_id) 方式一
        ret = urllib.reqeust('http://192.168.10.4:8000/recomment') 方式二
        # 查询缓存或数据库 获取文章的具体信息( 通过我们自己分装的缓存工具类)
        。。
        return
```

- 方式一： X 本地调用
  - 把推荐系统封装成工具类或函数
  - 两个系统耦合性太高 在web代码中包含了推荐系统的具体实现，关联性太高
- 方式二： HTTP调用
  - 把推荐系统当做单独的一个小项目，独立运行部署，web直接调用
  - 耦合性很低

- 网络调用

  由推荐系统封装HTTP 接口，在web中发起http请求进行调用

  缺点 HTTP的效率低下

  ```
  GET http://192.168.10.4:8000/recomment HTTP/1.1
  ....
  body
  ```

- 方式三： RPC调用 远端过程调用
  - 将网络调用封装的如同本地函数调用一样
  - 网络通讯效率越高越好，网络上传传输的调用数据 以二进制数据为主
  - 可以调用的客户端 不像http一样 ， 不是标准，只要自己系统能调用即可。
    - Facebook -> Thrift
    - Google -> gRPC 方案 传输的数据 协议 protobuf

使用RPC方法

1. 声明RPC调用的接口形式 IDL

   ret = recommend_article(channel_id, user_id)

   接口的名字 recommend_article

   调用时传递的参数 int channel_id , int user_id

   接口返回数据 int list [article_id, article_id, ...]

2. 生成 调用的代码 (包含了 参数转换为二进制传输的方法、网络传输收发的方法)

   rpc框架会提供生成代码的工具 (编译器)

使用编译器生成不同语言的代码

调用方 python -> 使用编译器 根据上面的接口描述，生成python代码

被调用方 java -> 使用编译器 根据上面的接口，生成java代码

3. 需要补充代码

   在被调用的一端 服务端 补充被调用时执行的逻辑函数

   在调用的一方，需要在 调用的代码地方 补充上调用的代码

RPC业务实现

RPC接口分析

调用请求

- channel_id
- user_id
- article_num 推荐的文章数量 10
- timestamp 时间戳 明确 跟推荐系统索要历史推荐数据还是最新的推荐结果
  - 如果timestamp传递的是最新的当前时间，则表明索要新的推荐数据
  - 如果传递的是历史的时间，则索要历史推荐

调用返回值

[article_id 文章id [1,2,3,4,5.6] 埋点

```
 曝光参数   exposure
 请求上一页历史推荐的时间戳 time_stamp
[

    {
        article_id
```

```
            {
                click: 'click param'
                collect: 'collct param'
                share:
                read:
            }
    }
    {
        article_id
            [
                click: 'click param'
                collect: 'collct param'
                share:
                read:
            ]
    }
]
```

使用IDL 接口定义语言 将上述接口写到文件中

- gRPC -> IDL ProtoBuf protobuf proto

```
syntax = "proto3";

// 使用message定义数据类型
message UserRequest {
    int64 user_id=1;
    int32 channel_id=2;
    int32 article_num=3;
    int64 time_stamp=4;
}

message Track {
    reserved 3;
    reserved "share";
    string click=1;
    string collect=2;
    //string share=3;
```

```
    string liking=5;
    string read=4;

}

message Article {
    int64 article_id=1;
    Track track=2;
}

message ArticleResponse {
    string expousre=1;
    int64 time_stamp=2;
    repeated Article recommends=3;
}


// 使用service 定义一组服务
service UserRecommend {
    // 使用rcp 定义被调用的方法(函数)
    rpc user_recommend(UserRequest) returns (ArticleResponse) {}
    // rpc simpla_recommend() returns () {}
}
```

即时通讯 IM

http只有请求 才能响应

符合WSGI协议的服务器 工作模式时多进程加多线程

yield 生成器

```
def func():
    print(1)
    print(1)
    print(1)
    a = 100
    b = 200
    yield  # 暂定代码  保存现场
    print(1)
    print(1)


generate_obj = func()
generate_obj.next()
generate_obj.next()  # send()

```

切换的场景 在程序中发生阻塞的时候 (读磁盘 ， 读写文件，网络IO操作，收发hTTP请求)

协程库

- gevent
- eventlet

协程的原理

```python
# 原生python方法
f = open('', 'rb')
f.read()

# 协程提供改写的方法
def read():
    进行操作系统调用 read #操作系统的非阻塞调用
    yield


```