一．时代复兴实习

概括介绍：

我在实习初期承担了检查商品异常情况和对异常商品进行处理的业务需求，首先学习了keystonejs和mongodb的相关内容，使用JavaScript语言在服务端新增了一些需要用到的例如获取商品信息更新商品状态的接口，然后用python开发了相应的多线程的中台服务，定时对商品进行检查和处理；在实习后期，我在公司前辈的帮助下，开始学习尝试用helm或者gitlab cicd对服务进行集成部署，然后也完成了一些对微信机器人一些功能的开发工作。

1. 服务端接口

KeystoneJS与MongoDB

keystoneJs是一个以MongoDB和Express为基础的开源的web应用平台，比较适合用来开发一些数据驱动的网站，可以自动生成一个管理后台，支持增删改查，上传文件等功能，省去了这部分前端的开发工作；也提供了丰富的在现实工作中比较实用的数据库域类型，比如email，name，password这些，因此比较高效。（域可以理解为sql中的列，即属性）比较适合作为运营后台或者数据管理系统，但对于特定的业务场景，可能定制性较差。

Mongodb是一种非关系型数据库，是一个基于分布式文件存储的数据库，其数据存储采用BSON的格式，BSON（）是一种类json的一种二进制形式的存储格式，简称Binary JSON.

相对于json多了date类型和二进制数组。每个表叫做一个collection，每一行为一个document，即文档。

MongoDB 中存储的文档必须有一个 _id 键。这个键的值可以是任何类型的，默认是个 ObjectId 对象。ObjectId 类似唯一主键，可以很快的去生成和排序，包含 12 bytes，前 4 个字节表示创建 **unix** 时间戳，比北京时间晚了 8 个小时。由于 ObjectId 中保存了创建的时间戳，所以你不需要为你的文档保存时间戳字段，你可以通过 getTimestamp 函数来获取文档的创建时间。

Mongodb分片

Express

Mvc

2. 中台服务

多线程

重要知识点 - 什么是进程(process)和线程(thread)

- 进程是操作系统分配资源的最小单元, 线程是操作系统调度的最小单元。
- 一个应用程序至少包括1个进程，而1个进程包括1个或多个线程，线程的尺度更小。
- 每个进程在执行过程中拥有独立的内存单元，而一个进程的多个线程在执行过程中共享内存。

python的多进程编程主要依靠multiprocess模块，可以利用multiprocess模块的Pool类创建多进程，Pool类可以提供指定数量的进程供用户调用，当有新的请求提交到Pool中时，如果进程池还没有满，就会创建一个新的进程来执行请求。如果池满，请求就会告知先等待，直到池中有进程结束，才会创建新的进程来执行这些请求。

python 3中的多进程编程主要依靠threading模块，可以使用Thread（）方法创建新线程，也可以通过继承Thread类重写run()方法来创建。一个进程所含的不同线程间共享内存，这就意味着任何一个变量都可以被任何一个线程修改，因此线程之间共享数据最大的危险在于多个线程同时改一个变量。

Ø 其中一个方法就是在修改前给其上一把锁lock，确保一次只有一个线程能修改它。threading.lock()方法可以轻易实现对一个共享变量的锁定，修改完后release供其它线程使用。

Ø 另一种实现不同线程间数据共享的方法就是使用消息队列queue。不像列表，queue是线程安全的

Python多进程和多线程哪个快?

由于GIL的存在，很多人认为Python多进程编程更快，针对多核CPU，理论上来说也是采用多进程更能有效利用资源。网上很多人已做过比较，我直接告诉你结论吧。

- 对CPU密集型代码(比如循环计算)　- 多进程效率更高
- 对IO密集型代码(比如文件操作，网络爬虫) - 多线程效率更高。

对于IO密集型操作，大部分消耗时间其实是等待时间，在等待时间中CPU是不需要工作的，那你在此期间提供双CPU资源也是利用不上的，相反对于CPU密集型代码，2个CPU干活肯定比一个CPU快很多。那么为什么多线程会对IO密集型代码有用呢？这时因为python碰到等待会释放GIL供新的线程使用，实现了线程间的切换。虽然多线程在系统资源上增加了一定程度的消耗，但是相对于网络资源响应以及 IO 传输产生的延迟和等待，仍然在效率上有了一定程度的提升。在请求某个数据时，程序并没有等待上一个请求完全解决，而是在结果返回之前又发起了新的请求。从而在一定程度上减弱了网络延迟对整个程序的阻塞效果。

is/setDaemon(bool): 将该子线程设置为父线程的守护线程（默认为非守护线程（False））。（需要在线程start之前设置）

关于“守护”的含义，我们可以这样理解，子线程为父线程的守护线程，意思是说子线程要守着父线程，一旦父线程执行完毕，也就不需要“守护”了，所以此时子线程就要结束。

Python全局解释器锁

再看一下实习代码和一些复杂的生产者消费者的例子

介绍一下中台这个概念

敏捷，解耦，可复用，[什么是中台业务架构？ - 知乎 (zhihu.com)]

有用到什么中台的平台或者框架或者架构吗

介绍自己是什么样的多线程服务，什么业务场景，

Helm和CICD

Gitlab cicd：

3. 微信机器人

异步回调

目标关联系统

大课题背景：目前视频侦查在保障公共安全领域是很重要的一部分，那整个这个课题其实就是想通过利用监控视频，物联信息等实现一个大范围的目标关联，得到目标的运动轨迹，然后后续通过对轨迹进行分析，来获取一些有价值的信息或者情报。然后我在做的这个融合视觉特征和时空约束的目标关联系统呢，就是应用视觉信息和时空信息，视觉信息是在利用人体信息的基础上，同时也考虑人脸的信息，因为人脸相比于人体这个整体来说更有区分性；然后时空约束就是通过两个摄像头之间实际的位置，距离，渡越需要的时间，在我们关联度过程中进行一定约束。

具体的关联过程是，应用了最佳优先检索流程。在输入首图及其时间位置方向信息后，根据摄像机拓扑结构获取时空约束，从而确定检索范围，树状创建新的搜索任务。每一次搜索任务根据视觉相似度匹配情况决定是否发散新的搜索任务。所有的搜索任务被存储在全局列表中，根据优先级在每一轮检索结束后进行重排序。

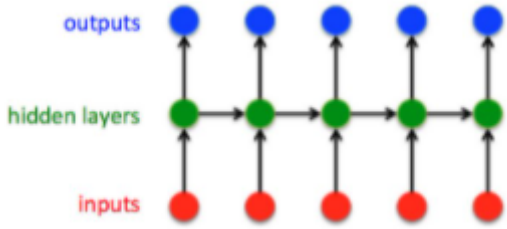
聊天系统

这是一个基于深度学习的开放性聊天系统，该系统可以满足闲聊对话和推荐电影对话两种模式。我在系统中主要负责首先通过seq2seq模型实现一个简单的对话系统，然后在此基础上加入了集束搜索，同时设计并且实现了基于规则的推荐意图检测模块，通过检测句子中是否出现关键词并且分析其位置关系，根据先验规则来判断是否需要接入推荐系统。

1. 序列模型

seq2seq vs RNN:

1. 经典的RNN结构如下：



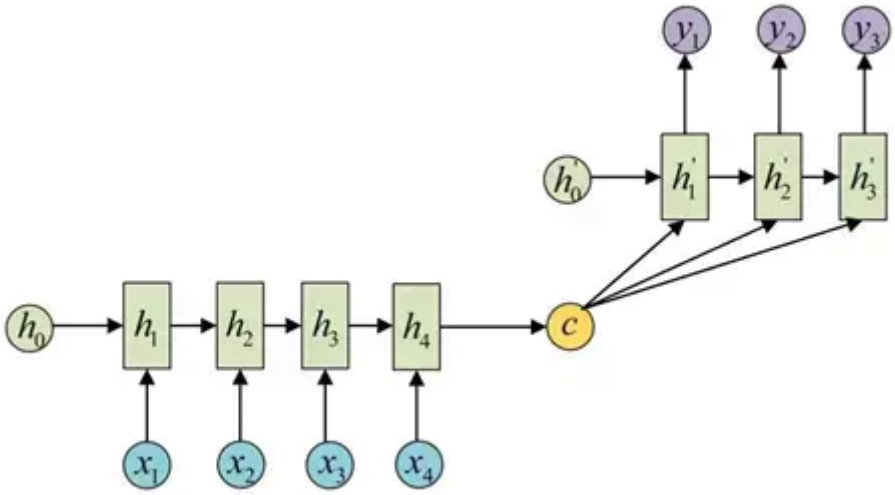
输入和输出序列必须有相同的时间长度

2. seq2seq模型：

seq2seq属于encoder-decoder结构的一种，这里看看常见的encoder-decoder结构，基本思想就是利用两个RNN，一个RNN作为encoder，另一个RNN作为decoder。**encoder负责将输入序列压缩成指定长度的向量**，这个向量就可以看成是这个序列的语义，这个过程称为编码，**获取语义向量最简单的方式就是直接将最后一个输入的隐状态作为语义向量C。**也可以对最后一个隐含状态做一个变换得到语义向量，还可以将输入序列的所有隐含状态做一个变换得到语义变量。

而**decoder则负责根据语义向量生成指定的序列**，这个过程也称为解码，如下图，最简单的方式是将encoder得到的语义变量作为初始状态输入到decoder的RNN中，得到输出序列。可以看到上一时刻的输出会作为当前时刻的输入，而且其中语义向量C只作为初始状态参与运算，后面的运算都与语义向量C无关。

decoder处理方式还有另外一种，就是语义向量C参与了序列所有时刻的运算，如下图，上一时刻的输出仍然作为当前时刻的输入，但语义向量C会参与所有时刻的运算。



RNN（循环神经网络）的优点：可以学到输入的上下文模型

RNN（循环神经网络）的缺点：随着时间长度的增大，无法从差得很远的时间步骤中获得上下文环境，只能记住短期存储序列

双向RNN有两种类型的连接，一种是向前的，这有助于我们从之前的表示中进行学习，另一种是向后的，这有助于我们从未来的表示中进行学习。

怎样表示序列中单独的单词 word embedding

1. 可以做一个词表，每个词有一个序号，但词表的容量是个问题。
2. one-hot表示：one-hot向量中只有一个是1，其余都是0

   在实际应用中，会导致词向量维度太大，而且编码过于稀疏，可能导致网络难以收敛
3. 使用one-hot编码进行嵌入过于稀疏，可以采用一种更优雅的方式

   首先生成一个embedding随机矩阵，然后使用one-hot编码乘以embedding矩阵，即获取embedding矩阵的某一行作为该单词的词嵌入。

   但这样无法表示词之间的关系
4. word2vec就是要通过训练得到embedding矩阵。具体的原理记不清楚了，只记得当中的一个细节：word2vec过每个词在语料中的出现次数建立huffman树，从huffman树根节点(root)开始，每次父节点向左子叶遍历编码为1，向右子叶遍历为0。使用huffman树的原因是可以降低训练时候的计算量。一般来说，训练用的语料库都非常大。而在语料库中，有一些词出现频率很高，还有一些词出现频率很低，而且这种频率差异是非常巨大的。那么采用huffman树后，出现频率很高的常用词路径短，计算量小，从而降低了整个word2vec模型在训练时的计算开销。

注意力机制：

背景：

encoder把所有输入序列都编码成语义向量context，contex可能存不下那么多信息，会造成精度的下降，并且decoder只利用了编码器最后一个隐藏层状态，信息利用率低

注意力机制的基本思路：

给encoder中各时刻的隐藏层状态ht分配一个权重wt，在decoder的每一个时刻，首先通过各权重与对应的隐藏层状态相乘，得到一个向量c

$$c_1 = h_1 \cdot w_{11} + h_2 \cdot w_{12} + h_3 \cdot w_{13} \tag{16}$$

然后把向量c与前一个状态拼接在一起形成一个新的向量，输入到隐藏层，得到该时刻的输入，后面的时刻同理。

2. 集束搜索

背景：在用decoder生成输出序列时，每一步要选择一个最大概率的词，如果直接使用贪心策略的话，每一步选择当前最可能的词，会出现Amy is sixteen years old 变成 Amy is going to be sixteen years old. 因为在英语中，或者说训练的语料库中is going 更加常见。

方法：beam search中有一个叫做集束宽的值b，表示在每一次选择时，要挑选前top b个结果。在第一步的时候，选择前b个作为候选结果，对于每个候选结果，去考虑它第二个单词的概率。假设词库中共有1000个单词，这时一共会有1000*b中选择，在这些结果中，再选前b个，以此类推。

3. 推荐意图检测

最初的打算是通过训练一个分类器来实现，判断是否应该接入推荐系统。后面一来由于时间有限，二来没有合适的数据集，所以改成基于规则判断。为了能够确定出比较全面合理的规则，向周围的同学征集了一些想要被推荐电影时的一些各种各样不同的表达，然后对这些句子从语法和句式等角度进行分析。首先构造了一些特殊单词的近义词词库，例如电影，可以有movie film等，还有一些能表示推荐意图的单词的词库。除此之外，对句子进行分析时，需要确定句中提问的中心词，例如what is your favorite movie ? VS what is your favorite movie actor? ，然后也要通过句式来确定是哪种类型的提问。

最后，在遇到一些不确定的语句时，我们率先提问是否需要推荐电影，这样可以在一定程度上保障用户的使用体验感。当然这是一些比较工程的思维了。

论坛

数据库怎么设计的：例如一开始实现发帖回帖功能的时候，我们需要有一张用户的表，存储用户id，用户名，秘密等；帖子和评论也分别有一个表，存储帖子或评论的id和内容，但像每个用户可以发多个帖子，每个帖子也可以对用多条评论，对于这样一对多的关系，就需要用外键来实现，例如可以在发帖这一列新增一列，存储用户id，让它和用户表中的某条记录对应。

然后再例如我们发现，随着论坛功能的丰富，每个用户在论坛中存储的个人信息也越来越多，用户这张表变得很大，我们会通过把用户这个大表拆分成用户基本信息表，用户详细信息表两个一对一的表，很多时候只需要查询基本信息，这样也可以提高查询性能。

再例如，论坛在实现了大部分功能之后，我们发现，有可能很多查询会涉及到通过用户名来搜索，那这个时候可能就需要在表里，给用户名这一列增加一个索引来提高查询效率。

在用flask进行开发时，我们使用了sqlAlchemy这个工具来简化数据库操作，可以通过定义一个python的类来表示数据库中的一张表，可以通过对这个类进行各种操作来代替写sql语句。

Flask：Flask是一个基于Python实现的web开发的'微'框架，路由，视图函数，模板三大部分，Flask的基本模式为在程序里将一个视图函数分配给一个URL，每当用户访问这个URL时，系统就会执行给该URL分配好的视图函数，获取函数的返回值并将其显示到浏览器上

调用了”render_template()”方法来渲染模板。方法的第一个参数”hello.html”指向你想渲染的模板名称，第二个参数”name”是你要传到模板去的变量，变量可以传多个。模板在子目录templates下。

Request，

session和cookie的作用

HTTP**协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话。常用的会话跟踪技术是Cookie与Session。Cookie通过在客户端记录信息确定用户身份，Session通过在服务器端记录信息确定用户身份**。session 的出现，是为了解决 cookie 存储数据不安全的问题的。

装饰器，可以用来限制用户的权利

数据库

蓝本

前后端分离，restful服务

前后端不分离的概念是后端要控制前端的数据显示和模板渲染（django），它有一个缺点就是可复用性不强，也就是它的后端程序只适用于一种前端类型，比如返回的是网页模板，则它只能用于网页端，移动端要用只能重新渲染一个移动端的模板。

而前后端分离则解决了这一问题，它的可复用性极强，一个后端可对接多个类型的前端，因为它不使用模板，而是通过向前端传递json数据的方式，将页面渲染和显示数据交给前端去做。这样写出来的后端可以适用于任何类型的前端。

路由和视图函数：

客户端把请求发给 Web 服务器，Web 服务器再把请求发给 Flask 程序实例，Flask 程序实例需要知道每个 URL 请求要运行哪些代码，所以保存了一个 URL 到 Python 函数的映射关系。处理 URL 和函数之间关系的程序称为路由，这个函数称为视图函数。通常使用 app.route 修饰器来定义路由，app 指 Flask 程序实例对象，如果使用蓝本管理路由后，由蓝本实例对象来取代 app

注册的时候通过邮件确认，数据库关键信息加密

挑一个项目进行介绍，包括项目的背景，做了哪些工作之类的