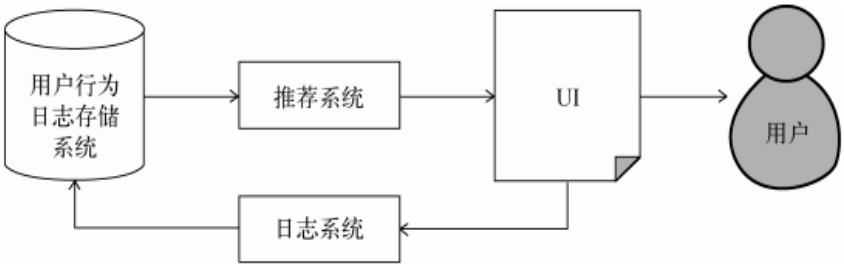
推荐系统实例

首先介绍推荐系统的外围架构，然后介绍推荐系统的架构，并对架构中每个模块的设计进行深入讨论。

外围架构

讨论推荐系统是如何和网站的其他系统接口的。一般来说，每个网站都会有一个 UI 系统，UI 系统负责给用户展示网页并和用户交互。网站会通过日志系统将用户在 UI 上的各种各样的行为记录到用户行为日志中。日志可能存储在内存缓存里，也可能存储在数据库中，也可能存储在文件系统中。而推荐系统通过分析用户的行为日志，给用户生成推荐列表，最终展示到网站的界面上。



推荐系统和其他系统之间的关系

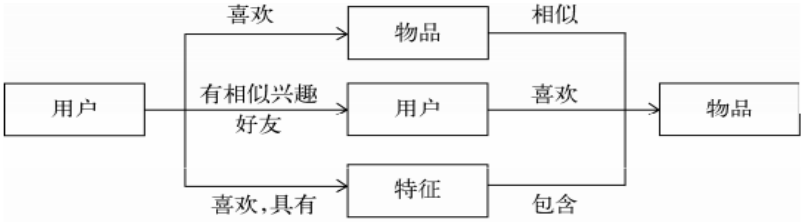
- 通过一定方式展示物品，主要包括物品的标题、缩略图和介绍等。
- 很多推荐界面都提供了推荐理由，理由可以增加用户对推荐结果的信任度。
- 推荐界面还需要提供一些按钮让用户对推荐结果进行反馈，这样才能让推荐算法不断改善用户的个性化推荐体验。

个性化推荐算法依赖于用户行为数据，而在任何一个网站中都存在着各种各样的用户行为数据。那么如何存取这些数据就是推荐系统需要解决的首要问题。

- 按照前面数据的规模和是否需要实时存取，不同的行为数据将被存储在不同的媒介中。一般来说，需要实时存取的数据存储在数据库和缓存中，而大规模的非实时地存取数据存储在分布式文件系统（如 HDFS）中。
- 数据能否实时存取在推荐系统中非常重要，因为推荐系统的实时性主要依赖于能否实时拿到用户的新行为。只有快速拿到大量用户的新行为，推荐系统才能够实时地适应用户当前的需求，给用户进行实时推荐。

推荐系统架构

推荐系统是联系用户和物品的媒介，而推荐系统联系用户和物品的方式主要有 3 种，如果认为用户喜欢的物品也是一种用户特征，或者和用户兴趣相似的其他用户也是一种用户特征，那么用户就和物品通过特征相联系。



3 种联系用户和物品的推荐系统

基于特征的推荐系统架构，当用户到来之后，推荐系统需要为用户生成特征，然后对每个特征找到和特征相关的物品，从而最终生成用户的推荐列表。因而，推荐系统的核心任务就被拆解成两部分，一个是如何为给定用户生成特征，另一个是如何根据特征找到物品。

推荐系统需要由多个推荐引擎组成，每个推荐引擎负责一类特征和一种任务，而推荐系统的任务只是将推荐引擎的结果按照一定权重或者优先级合并、排序然后返回推荐列表：

- 可以方便地增加/删除引擎，控制不同引擎对推荐结果的影响。对于绝大多数需求，只需要通过不同的引擎组合实现。
- 可以实现推荐引擎级别的用户反馈。每一个推荐引擎其实代表了一种推荐策略，而不同的用户可能喜欢不同的推荐策略。可以将每一种策略都设计成一个推荐引擎，然后通过分析用户对推荐结果的反馈了解用户比较喜欢哪些引擎推荐出来的结果，从而对不同的用户给出不同的引擎组合权重。

推荐引擎的架构

推荐引擎使用一种或几种用户特征，按照一种推荐策略生成一种类型物品的推荐列表。

- 该部分负责从数据库或者缓存中拿到用户行为数据，通过分析不同行为，生成当前用户的特征向量。不过如果是使用非行为特征，就不需要使用行为提取和分析模块了。该模块的输出是用户特征向量。
- 该部分负责将用户的特征向量通过特征-物品相关矩阵转化为初始推荐物品列表。
- 该部分负责对初始的推荐列表进行过滤、排名等处理，从而生成最终的推荐结果

生成用户特征向量

用户的特征包括两种，一种是用户的注册信息中可以提取出来的，对于使用这种特征的推荐引擎，如果内存够，可以将存储这些特征的信息直接缓存在内存中，在推荐时直接拿到用户的特征数据并生成特征向量。另一种特征主要是从用户的行为中计算出来的。

一个特征向量由特征以及特征的权重组成，在利用用户行为计算特征向量时需要考虑以下因素：

- **用户行为的种类** 在一个网站中，用户可以对物品产生很多不同种类的行为。这些行为都会对物品特征的权重产生影响，但不同行为的影响不同，大多时候很难确定什么行为更加重要，一般的标准就是用户付出代价越大的行为权重越高。
- **用户行为产生的时间** 一般来说，用户近期的行为比较重要，而用户很久之前的行为相对比较次要。因此如果用户最近购买过某一个物品，那么这个物品对应的特征将会具有比较高的权重。
- **用户行为的次数** 有时用户对一个物品会产生很多次行为。因此用户对同一个物品的同一种行为发生的次数也反映了用户对物品的兴趣，行为次数多的物品对应的特征权重越高。
- **物品的热门程度** 如果用户对一个很热门的物品产生了行为，往往不能代表用户的个性，反之如果用户对一个不热门的物品产生了行为，就说明了用户的个性需求。因此，推荐引擎在生成用户特征时会加重不热门物品对应的特征的权重。

特征—物品相关推荐

对于一个推荐引擎可以在配置文件中配置很多相关表以及它们的权重，而在线服务在启动时会将这些相关表按照配置的权重相加，然后将最终的相关表保存在内存中，而在给用户进行推荐时，用的已经是加权后的相关表了。特征—物品相关推荐模块还可以接受一个候选物品集合，候选物品集合的目的是保证推荐结果只包含候选物品集合中的物品。

如果需要在一个小的候选物品集合中给用户推荐物品，那么可以考虑上述方法。但如果是要在一个很大的候选物品集合中给用户推荐物品，那么可以考虑直接在初始推荐列表中过滤掉不在候选物品集合中物品的方法。特征—物品相关推荐模块除了给用户返回物品推荐列表，还需要给推荐列表中的每个推荐结果产生一个解释列表，表明这个物品是因为哪些特征推荐出来的。

```
def RecommendationCore(features, related_table):
    ret = dict()
    for fid, fweight in features.items()
        for item, sim in related_table[fid].items():
            ret[item].weight += sim * fweight
            ret[item].reason[fid] = sim * fweight
    return ret
```

过滤模块

在得到初步的推荐列表后，需要按照产品需求对结果进行过滤，过滤掉那些不符合要求的物品。

- **用户已经产生过行为物品**　因为推荐系统的目的是帮助用户发现物品，因此没必要给用户推荐他已经知道的物品，这样可以保证推荐结果的新颖性。
- **候选物品以外的物品**　候选物品集合一般有两个来源，一个是产品需求。另一个来源是用户自己的选择。
- **某些质量很差的物品**　为了提高用户的体验，推荐系统需要给用户推荐质量好的物品，那么对于一些绝大多数用户评论都很差的物品，推荐系统需要过滤掉。这种过滤一般以用户的历史评分为依据。

排名模块

新颖性排名

新颖性排名模块的目的是给用户尽量推荐他们不知道的、长尾中的物品。虽然前面的过滤模块已经过滤掉了用户曾经有过行为的物品，保证了一定程度的新颖性，但是用户在当前网站对某个物品没有行为并不代表用户不知道这个物品，

要准确了解用户是否已经知道某个物品是非常困难的，因此我们只能通过某种近似的方式知道，比如对推荐结果中热门的物品进行降权，仅仅在最后对热门物品进行降权是不够的，而应在推荐引擎的各个部分考虑新颖性问题。

$$p_{ui} = \frac{p_{ui}}{\log(1 + \alpha * popularity(i))}$$

如果使用 ItemCF 算法，根据前面的讨论可知计算出的相似度矩阵中，热门物品倾向于和热门物品相似。那么也就是说，如果用户对一个热门物品 j 产生了很多行为，有很大的r_{uj}，那么和这个热门物品相似的其他热门物品都会在用户的推荐列表中排在靠前的位置。因此，如果要提高推荐结果的新颖度，就需要对r_{uj}进行降权。

也可以引入内容相似度矩阵，因为内容相似度矩阵中和每个物品相似的物品都不是很热门，所以引入内容相似度矩阵也能够提高最终推荐结果的新颖度。

多样性

增加多样性可以让推荐结果覆盖尽可能多的用户兴趣。

- 将推荐结果按照某种物品的内容属性分成几类，然后在每个类中都选择该类中排名最高的物品组合成最终的推荐列表。

首先，选择什么样的内容属性进行分类对结果的影响很大。其次，就算选择了某种类别，但物品是否属于某个类别是编辑确定的，并不一定能够得到用户的公认。

- 控制不同推荐结果的推荐理由出现的次数。要提高推荐结果的多样性，就需要让推荐结果尽量来自不同的特征，具有不同的推荐理由，而不是所有的推荐结果都对应一个理由。

```
def ReasonDiversity(recommendations):
    reasons = set()
    for i in recommendations:
        if i.reason in reasons:
            i.weight /= 2
        reasons.add(i.reason)
    recommendations = sortByWeight(recommendations)
```

时间多样性

保证用户不要每天来推荐系统都看到同样的推荐结果。

- 首先要保证推荐系统的实时性，在用户有新行为时实时调整推荐结果以满足用户最近的需求。如果用户有实时行为发生，那么行为提取和分析模块就能实时拿到行为数据并转化为新的特征，然后经过特征-物品相关模块转换成和新特征最相关的物品，因而推荐列表中就立即反应了用户最新行为的影响。
- 在用户没有新的行为时，也要保证推荐结果每天都有变化：
  1）记录用户每次登陆推荐系统看到的推荐结果；
  2）将这些结果发回日志系统。这种数据不需要实时存储，只要能保证小于一天的延时就足够了。
  3）在用户登录时拿到用户昨天及之前看过的推荐结果列表，从当前推荐结果中将用户已经看到的推荐结果降权。

用户反馈

分析用户之前和推荐结果的交互日志，预测用户会对什么样的推荐结果比较感兴趣。