# 检索

As Advanced search 高级搜索

Bs 基础索引 倒排索引

AS（Advanced Search）即高级搜索，2层架构的第一层，将WebPM.UI发送过来的检索请求发给BS，把BS的结果归并以及过滤后返回WebPM.UI。 AS是一个大模块，包含AC和BC两层模块。总体来说，AS接收US发送过来query，将query分发给BS计算，再merge BS的计算结果交给US，与US交互的是AC，与BS交互的是BC。

US（Universal Searcher） 通用搜索 是垂类检索资源整合的入口，主要功能包括两个部分：1、query阶段的query解析和分发，包括并行访问多个后端服务、合并及排序策略、缓存各个服务结果三个；2、数据返回阶段的结果整合，主要是将 aladdin 特殊需求结果，ecom 广告结果，ks 等搜索结果与大搜自然结果合并返回给前端。

BC（Basic searcher Controller），介于AS和BS之间的中间层。

模块定位：AC:Advanced Search Conroller，高级检索的控制模块（高级是相对于基础检索模块的BS而言的），高级体现在AC能在一定程度上理解用户的需求，返回内容除了具有文本相关性，还具有需求相关性

模块功能：AC将后端BC/BS返回的结果汇集起来，完成相关性排序并获取结果的摘要，将最相关的N条结果返回给US。

feedas load了所有笔记的一些特征和score之类的在内存 然后每天换库

冷启动是指在模型运行初期阶段，没有提前给模型赋予一定的初始条件，需要模型在整个运行期间实现各种指标数据从无到有的过程，比如模拟水位需要从0开始到趋近实际水位，模拟海流需要从0开始到趋近实际流速流向。就相当于这个初期就是让模型开始热起身来，先建立起该有的条件来，初期的数据是不准确的，是无法选用为研究结果的。如果模型模拟时间区段足够长，那么在模型运行中后期阶段，模型就已达到接近实际的各种条件了，此时模型才算是真正的开始模拟你的研究内容。

热启动是指在模型运行初期阶段，我们提前给模型赋予一定的初始条件，使模型从一开始就处于接近实际状况的状态，就相当于模型从一开始就已经热好身了，直接就可以上赛场开始干活了。热启动下的模型从模拟一开始就在真正的模拟你的研究内容了。

————————————————

版权声明：本文为CSDN博主「晏长街」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/m0_50132779/article/details/119236189>

<https://zhuanlan.zhihu.com/p/572998087>

除了倒排召回 还有ann召回

用户session召回

Ann（approximate nearest neighbor）是指一系列用于解决最近邻查找问题的近似算法。最近邻查找问题，即在给定的向量集合中查找出与目标向量距离最近的N个向量。

比较平凡的方法就是线性查找方法，也就是说每次检索都会遍历整个检索库，虽然准确率是100%，但是其效率是随着检索库的增加而线性增加的，当检索库的大小超过一定阈值之后，每次检索的时间开销将会变得巨大，不能忍受。在搜索集小的时候，还可以接受，但是实际中向量集合往往很大，线性查找方法的计算资源消耗较大且耗时较长，在工程上一般都不会采用这种方法。

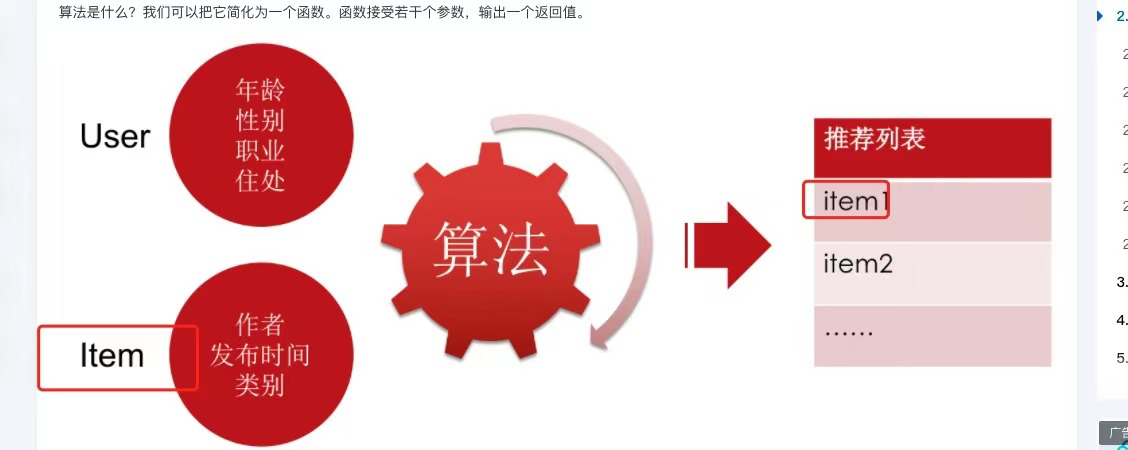
在 NN 的搜索上，提出了一些树的方法，比如说 KD-tree，以及进化的 ball-tree，像kd-tree等一些优化方法并没有提高高维空间里的搜索效率，效率甚至比线性扫描还要低，导致准确最近邻搜索难以直接用于实际问题。顾名思义，近似最近邻搜索找到的向量不要求一定是准确的最近邻，只要求距离准确的最近邻很近就可以了，可以看到近似近邻实际上是效果与性能的折衷

新闻推荐系统从海量新闻中推荐出你感兴趣的新闻，百度从海量的搜索结果中找到最优的结果，短视频推荐出你每天都停不下来的视频流，这些里面都包含有今天要介绍的ANN方法

Approximate Nearest Neighbor

ANN

## CF



协同过滤算法

协同过滤算法(Collaborative Filtering, CF)是很常用的一种算法，在很多电商网站上都有用到。CF算法包括基于用户的CF(User-based CF)和基于物品的CF(Item-based CF)。

基于用户的CF原理如下：

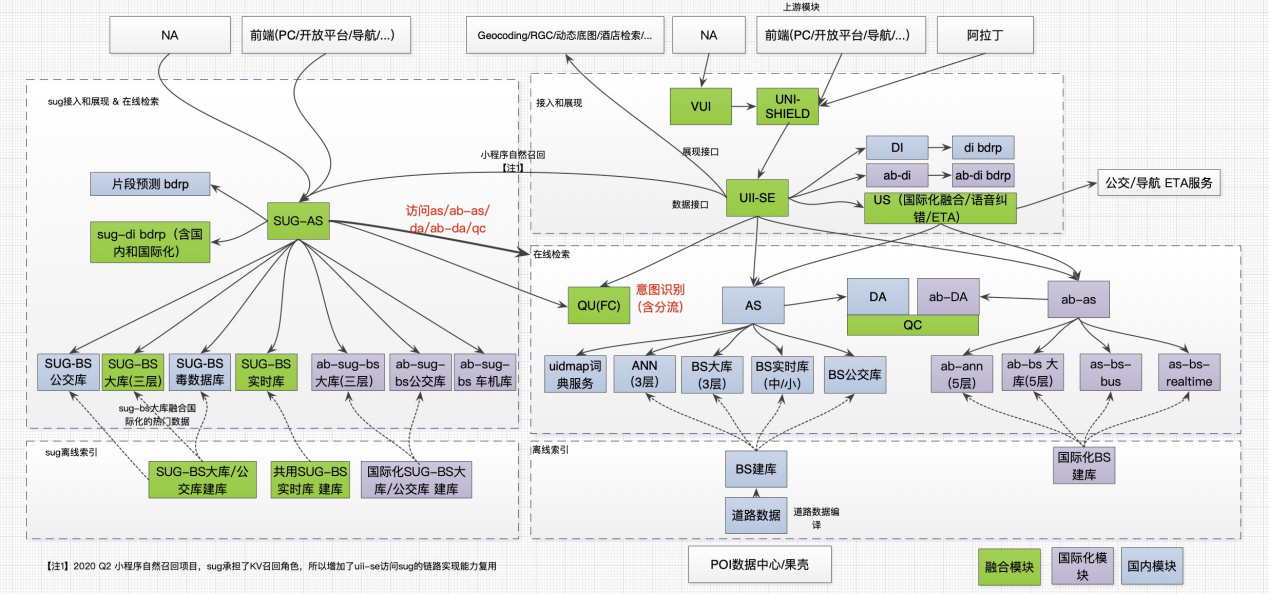
分析各个用户对item的评价（通过浏览记录、购买记录等）；

依据用户对item的评价计算得出所有用户之间的相似度；

选出与当前用户最相似的N个用户；

将这N个用户评价最高并且当前用户又没有浏览过的item推荐给当前用户

基于流行度的算法



与uii-se直接相关的模块各个模块功能如表格中所示，其中qu，da识别到正确的需求，as返回uids，di返回uid对应poi详细数据，带ab模块的是国际化数据，当判定为国际化数据时，会访问带ab的模块；下面我将从美食泛需求，简述下大框检索的过程

1、首先在大框中输入美食，点击搜索，客户端会将请求发送至phpui，此时qt = s，这个qt是客户端带的，s标识大框检索，也就是说无论在大框中检索什么，此时的qt都是s；

2、phpui接受到qt = s的请求之后，会根据路由，s的路由第一个是反抓，正常经过反抓之后，s的第二个路由是search，此时qt仍然为s，此时只知道请求来自大框，但是并不清楚到底是poi检索，还是路线请求等，phpui会根据路由配置easy\_search直接访问uii-se；

3、uii-se接受到qt = s的请求，会访问qu对这个请求进行意图的识别，由于本例子为美食，会返回type = map\_poi到uii-se，uii-se将请求返回至phpui，

4、phpui接受到type = map\_poi之后，判断下次路由为poi，qt = poi，构造请求，访问vui，然后vui去访问uii-se；

5、此时uii-se收到的请求qt = poi，会对参数做些处理，然后访问as请求uids，当然也会访问sug\_client，和推荐，但是对检索成功失败不影响，此处就没做说明；

6、若as命中缓存，直接返回缓存，如果没有命中缓存，会请求bs召回poi点，as会更具poi点的一些参数进行排序，其中有tag，distance等等；

7、as排序好后，将rn（每页的poi点个数）\* pn(页数）~ rn\*pn + rn 的结果返回，一般rn为10，此时得到as的返回的uids，这10个poi就是本次检索的结果

8、uii-se再将这10个uid去请求di，di请求bdrp索引，检索di，返回到uii-se，uii-se得到10个poi详细数据，再进行一定的加工afterhandle，返回到vui；

9、vui对结果进行视觉展现调整，返回到phpui，phpui发送到客户端，完成检索；客户端上显示出这页结果；

功能上：uii-se负责接收前端请求并连接多个模块获取检索信息，as负责高级索引和cache及策略等，bs 维护索引，负责建立索引和检索 等 ， di 负责索引文件的数据 ，qu、da负责query分析

结构上：uii-se在前端中属于后端，在后端中属于前端。主要负责接收phpui等数据请求，根据请求调用后端服务并做整合返回给前端。

1、首先会根据actual\_server\_index确定下游的as还是ab-as，以及di还是ab-di；

2、获取一些必要的参数，并校验，比如word，pn这些，确定是不是正确的类型，确定来源，设置透传参数等；

3、解析cookie，然后对一些参数覆盖，比如src

4、判断是否为sug行政区划检索，如果是，替换bid直接请求di，新的bid无结果用旧的bid重试，最后返回比如用sug检索北京市；

5、如果不是sug行政区划，判断是否为特殊的query，这些特殊的query配置在这个js中，例如钓鱼岛，获取直接返回；

6、对商圈检索和多边形检索替换search\_bound；

7、访问qu，识别主意图（这个地方phpui请求的机制之前是不能将上一次的请求的结果作为下一个请求的数据)；

8、酒店导流，如果命中先对酒店访问（搜酒店能命中酒店导流），若酒店无结果，直接跳转拼音检索；

9、若不用酒店导流或者酒店无结果，依次判断属性检索，p2p检索，判断是否需要同时访问国内国外as，若不需要，判断是否需要sug直跳（manual\_query\_not\_direct\_jump），不用就访问as获取数据；

10、若命中酒店导流，会判断是否需要二次检索，这次检索好像是加筛选项的；

11、若需要对模糊和精确门址的处理，进行门址检索优化；

12、竞价排名、室内导航、城市列表等等；

13、获取来自as的di还是ab-di，二次判断；如果是sug直跳，更新cityid；

14、行政区划处理（和sug行政区划有别，我理解的是sug相当于已经检索一次，所以没有访问as）

15、填充as和di，到result中；

16、后面的操作相当于后置操作，其实也不需要放在这个controller中的，这地方有个小技巧，在bound\_rank也就是图像后面的访问服务的请求，都是在interceptor中的；

可以看到，poicontroller涉及下游很多，但是一般的泛需求检索，并不会都取访问所有下游，强依赖的是qu、as、di，其余下游不会导致无返回；

但是也可以看到其实很多模块都可以放到interceptor中，但是都一起放在controller的dohandle中，导致poicontroller比较臃肿

工业上都会有很多路召回，画像召回只是一路补充

一路相似召回

es做向量检索