知识库更新相关论文阅读总结报告

知识库更新，按照更新数据获取的方式可以分为：PUSH 和 PULL 两类方法

- **PUSH**

顾名思义，PUSH 是推送的意思，即知识库所采用的数据源主动向知识库推送需要被更新的数据，知识库在接受到更新数据流之后，对知识库中相关数据进行更新。

目前支持 PUSH 的数据源不多，维基百科便是其中之一。因此 DBpedia 可以利用维基百科提供的更新数据流对 DBpedia 中的数据进行实时、持续的更新，而不需要主动去寻找需要被更新的数据。Hellmann 等人的“DBpedia Live Extraction”中所提到的 DBpedia live extracion 系统，便是利用维基百科提供的数据更新流，实现对 DBpedia 的实时更新。该系统使得 DBpedia 和维基百科几乎同步，数据滞后时间不超过 2 分钟。

考虑 zhishi.me 的更新，zhishi.me 中的中文维基百科的数据便可以利用这种方式进行更新。另外百度百科和互动百科虽然没有维基百科的更新流接口，但是有用户的词条编辑动态，展示了哪些词条被更新，可以作为 PUSH 信息利用到知识库的更新之中。

- **PULL**

相对于 PUSH，PULL 类方法是知识库主动的去寻找需要被更新的数据，而不是数据源向知识库推送。由于支持 PUSH 的数据源很少，因此 PULL 类方法是使用最多、应用最广的。

PULL 类方法最理想的情形是把数据源整个数据取下来，和知识库中的数据相比较，然后对知识库进行增删改等更新操作。但是由于网络带宽、计算资源的限制，这种方式只适合于阶段性的进行，而不能高频率地进行这种操作。

如果想要提高知识库的更新频率，增加知识库的实时性同时减少每次更新所消耗的资源，PULL 类方法需要解决的最核心的问题是如何识别出哪些知识需要被更新。如果能有效识别出需要被更新的知识，再对这些知识进行更新，可以大大减少需要消耗的网络资源。具体细节将会在后面讨论。

前面提到，PULL 类方法的核心是识别出那些知识需要被更新，换句话讲，我们需要给知识库中的实体赋予不同的更新优先级，优先级更高的实体需要优先被更新。而优先级刻画方式的好坏，直接决定更新效果的好坏。总结来讲，常见的优先级刻画方式有如下几种：

1. 使用热词

肖仰华等人的“How to keep a knowledge base synchronized with its encyclopedia source”中便是使用了搜索引擎的热门搜索、门户网站的热门话题以及热门新闻标题中提取出来的实体作为热词。热词相对来说更有可能属性值发生改变或者是新词，因此优先更新热词是完全合理的。但是每天的热词数量是有限的，因此仅仅更新热词是不够的。这篇论文将和热词语义相关的词也加入待更新队列来提高更新的数量。

热词的来源有很多，但是这篇论文却忽略了一个更重要的来源，就是知识库的数据源本身。百度百科和互动百科首页均有热搜词条，而这些热搜词条应该拥有更高的更新优先级。

2. 使用实体的历史更新频率

- 直接使用实体的历史更新频率

  Umbrich 等人的“Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources”中研究发现，web 上的数据的改变频率大致符合“泊松分布”。Dividino 等人的“Strategies for efficiently keeping local linked open data caches up-to-date”这篇文章证明了基于数据动态性（data dynamics）或者历史改变频率(change frequency)进行更新的策略相比与其他策略，能够更有效地利用有限的网络带宽资源。数据的动态性是指较长一段时间内 change frequency 的总体状态，本质上还是 change frequency。因此可以利用百科页面的 snapshots 或者 change logs 去计算每个实体过去的 change frequency，根据这个值给予不同实体不同的优先级，即 change frequency 越大，这个实体需要被更新的优先级就越高。

- 使用实体的历史更新频率去训练预测模型

  由于每一次更新周期都重新计算所有实体的历史更新频率所花费的代价太大，并且这种更新方式的更新频率依赖于百科 snapshots 的更新频率，因此可以考虑使用固定的 snapshots 集合作为训练数据集和测试集，设计相应的特征作为输入，更新频率作为输出，训练出一个回归模型去预测每一个实体未来的更新频率。肖仰华的那篇论文中便是

使用百度百科的多个 snapshots 作为训练集和测试集，设计了包括实体页面存在时长、实体页面总更新次数、页面访问量等八个特征，训练出一个回归模型，利用该回归模型给待更新实体不同的优先级，根据优先级去更新实体。

3. 使用实体所对应的三元组的生命周期

在知识库中，每一个实体的信息都是以 N 个三元组的形式存储，形如“<特朗普><身份><商人>”。当实体的某个属性值发生改变时，可以看作是存储旧的属性指的三元组的删除和存储新的属性值的三元组的插入。例如，当特朗普的身份变为总统时，“<特朗普><身份><商人>”这个三元组便会被删除，代表着该三元组的生命周期的结束，取而代之的是“<特朗普><身份><总统>”这个三元组。如果能够预测出每个三元组的生命周期，便可以赋予总的生命周期最短的实体更高的优先级，从而优先更新它们。

Nishioka 等人的“Keeping Linked Open Data Caches Up-to-date by Predicting the Life-time of RDF Triple”便是使用的该思想。该论文从训练数据集中训练得到一个回归模型用于预测每一个 RDF Triple 的生命周期，所有三元组的生命周期之和最短的实体将优先被更新。该回归模型的输入特征是一个“one-hot”特征向量。该特征向量的计算方法是将所有三元组的主谓宾（s、p、o）拆开，放入到同一个 set 中，形成一个大小为 N 的 set，而这个“one-hot”向量的维度便是 N，每一维度的值取决于对应的词是否是该三元组的主谓宾之一，如果是则取 1，否则取 0。为了降低特征向量的维度，可以将出现频率较低的词全部统一为“other objects”。回归模型输出便是该三元组的生命周期。

该方法从本质上讲还是预测实体被改变的可能性，但是是从三元组的角度而不是从实体的角度。通过多个三元组的预测情况综合去得出实体的预测情况，可以减少误差，提高更新效果。同时该方法所使用的特征向量是“One-hot”向量，相比于其他特征计算更简单，可以降低计算资源。

4. 其他更新策略

以上三种方法都可以看作是“change frequency based”更新策略，即根据实体过往的更新频率决定其未来被更新的优先级。除了这种更新策略之外，还可以采用如下一些更新策略：

- 用户驱动更新

  可以根据用户的反馈来选择要更新的实体，以 zhishi.me 为例，可以在网页端提供

错误或缺失属性标注功能，对于用户标注属性错误或者属性缺失的实体优先到百科进行更新检查。另外可以分析后台的 query log，提取出 query log 中用户查询不到的实体，优先到百科去检查是否有相关的百科页面，进行更新。

- Round Robin 更新策略

Round Robin 更新法就是按顺序每个更新周期去更新知识库中的部分数据，如此循环更新下去。这种策略的优点就是简单易行，并且可以保证每个实体在一定时间之后都会被更新。但是明显仅仅使用这种更新策略是不行的，需要和其他更新策略结合使用。

- Sampling-Based 更新策略

Sampling-based 策略是在 Junghoo Cho 等人的“Effective Change Detection Using Sampling”这篇论文中提到的。该论文提出，对于来源于大量不同站点的数据所构成的知识库，可以采用基于采样的更新策略。假设每一次更新周期允许进行更新检查的最大数量为 N，基于采样的更新策略首先从每一个站点中随机采样部分数据，然后通过采样数据中数据需要被更新的情况，去给不同的站点的数据赋予不同数量的更新资源，即更新检查的最大次数。

基于采样的策略并不局限于大量不同的站点这一条件，以 zhishi.me 为例，它的数据来源只有三个站点，再使用站点来区分数据明显不合适。但是我们可以根据实体的类别将数据分为多个不同的类(可以根据带宽和计算资源选择合适分类粒度)，然后对每个类中的数据进行采样分析，对于采样数据中需要被更新实体更多的类别，我们会赋予其更高的优先级。对于优先级更高的类别我们可以采用贪婪策略对该类别全部进行更新，也可以按照抽样数据中需要被更新的比例取分配带宽资源。

另外，sampling-based 更新策略和 Round Robin 更新策略也可以很好的结合使用。采样分析完成后，对于每一个类别的数据的选取可以采用 Round Robin 的策略。