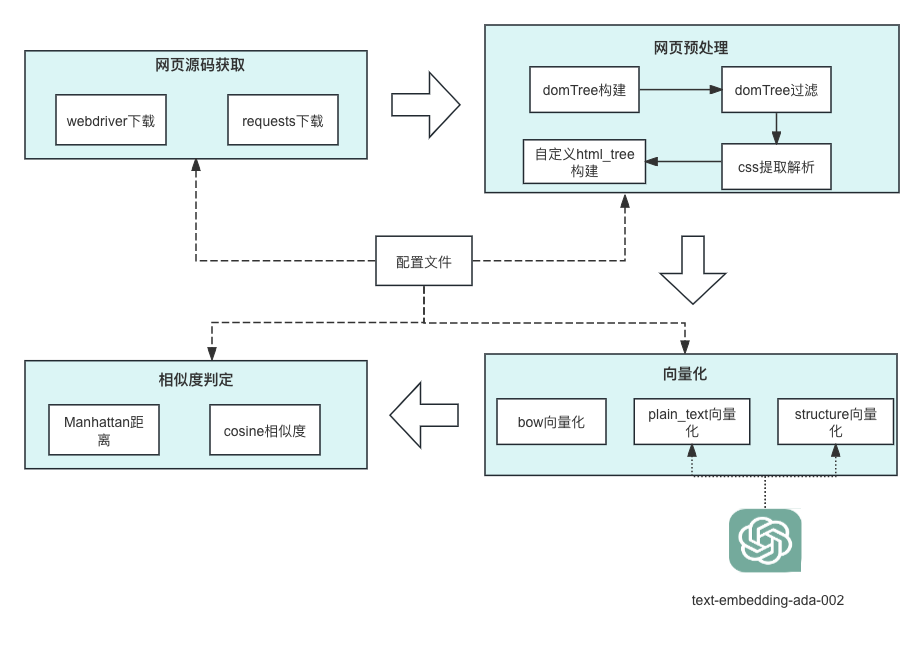
1. **整体流程图**

****

1. **各模块拆解设计与实现方案**

**2.1 网页源码获取**

实现两种根据url获取html,css源码的方法，**默认采用方法2**

方法1: 使用requests获取网页源码，这种方法需要手动配置headers，配置不合适，高频请求等情况下可能出现访问被限制，或者无法获取完整页面源码（页面存在动态加载元素）等情况，成功率较低，时延也较低

方法2：使用selenium获取网页源码，这种方法需要安装selenium库以及和机器chrome版本匹配的chromedriver；由于通过chromedriver发起请求，这种方法多数情况下可以 获取到网页完整源码，且对于需要登录的网站可以自动模拟账户密码输入过程，成功率高但是时延也相对较高

**2.2 网页预处理**

设计的网页预处理步骤如下：

1. 通过beautifulsoup解析html源码构建初始domtree
2. domTree预处理，去掉注释以及javascript,meta等对页面布局展示无影响的元素
3. 从style节点中提取css源码，以及根据配置标志（是否拉取远端css），尝试拉取远端css源码；使用cssutils工具将css源码解析为<selector,property\_dict>的KV对形式
4. 根据配置中是否将css融合到html结点的标志，对3中每个selector锁定对应的html元素结点，把property\_dict加入到元素的属性中，得到融合了css属性的dom tree
5. 对上述处理后的dom tree重新构建为自定义的树，结点类型为代码里定义的TreeNode，保存对向量化有用的属性特征，便于后续的向量化计算

注：第4步中可根据配置参数"include\_css\_in\_html"选择是否将css融合到html结点，这里如果选择融合可以只针对html向量化和相似度比较，且可以过滤掉对当前网页无影响的css属性，缺点是时延高，selector匹配dom tree的过程耗时大；如果不融合，需要单独对css采用下述的bag-of-words向量化，最终的相似度由html和css相似度融合得到

**2.3 向量化**

实现三种html和css向量化方法，**默认采用bag-of-words方案**

**bag-of-words方案：**参考百度2009年网页结构相似度比较专利中的思想，先序遍历得到结点列表，对每个结点，基于结点类型，属性类型和属性值构造伪word，以深度加权累加（深层结点对整体结构影响比上层小）作为每个伪word的权重； 相比原专利中的方法进行了几点优化：

1. 引入css信息
2. 对每个节点按不同属性类型拆分后构造word(原专利结点粒度构造word)，相比原方法有更好的泛化性
3. 对同父节点下的siblings，去除包含属性值且属性值完全相同的结点（通常是大板块下重复的子版块）
4. 引入可配置的参数warmup\_depth，支持从一定深度后再进行权重衰减

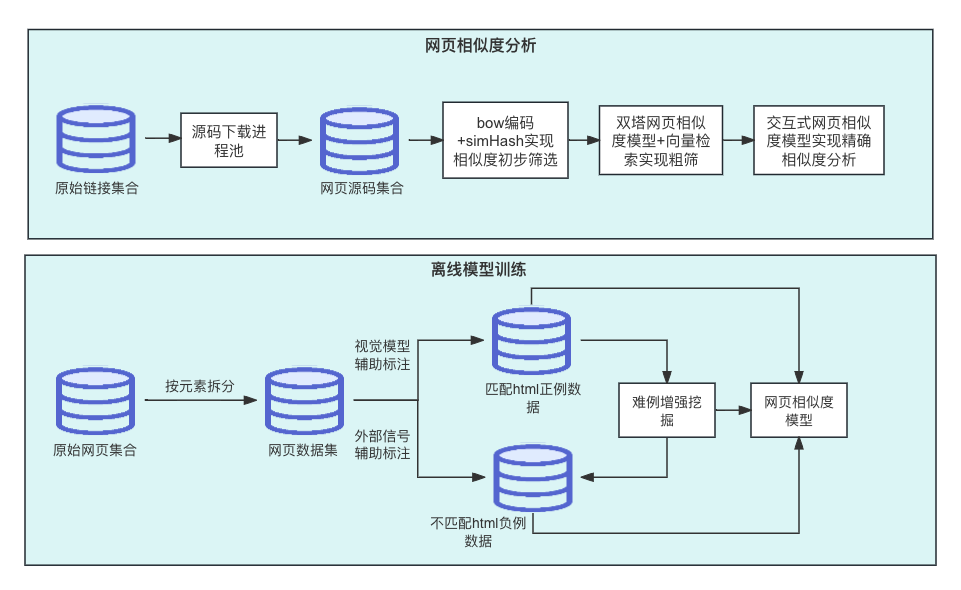
**序列化为文本后向量化：**对原始dom tree预处理后的树，通过先序遍历构造html样式的文本，对序列化后的文本通过openai text embedding接口得到嵌入；由于长度限制，这种方法忽略深度大于一定阈值max\_depth的结点，只保留上层子树；虽然这种方法考虑了结点间的序关系，但依赖于embedding模型本身对html代码的编码能力，未使用html-pairs数据微调的通用text-embedding模型通常效果不好，倾向于认为所有html源码相似度都很高（相比和普通文本），需要使用一些网页数据进行微调才能有更好的效果

**基于树结构的向量化：**使用上一种方法计算高度小于阈值的各子树embedding，然后子树embedding和父节点本身属性聚合得到父节点对应子树的embedding，自顶向上逐层聚合得到整个树的表征；这种方法解决了2中整棵树序列化长度过大的问题，但是实际对于较复杂的网页开销过高，且本身也依赖于text-embedding模型针对网页结构相似性任务的微调

**2.4 相似度判定**

相似度判定模块设计的较为直接，对于bag-of-words向量化后的特征，由于特征本身稀疏性高，基于曼哈顿距离计算特征向量间的相似度，相似度阈值默认设置为0.5；对于其它两种基于text-embedding的向量化方案，采用余弦计算向量相似度，相似度阈值默认设置为0.95，主要由于未经过html相似性任务微调的embedding模型对html的编码在欧式空间内都很接近

1. **大规模网页相似性分析方案**

****

对于大规模网页数据，相似度分析可通过下面几步漏斗式过滤实现：

1. 对于源码下载部分，通过进程池实现多实例并行下载大规模网页源码
2. 获得源码数据集合后，同样通过多进程等方式并行完成上述的dom tree预处理
3. 相似度分析时，首先基于上述bag-of-words方案结合sinHash，利用二值化向量进行高效初筛，筛选出相似度很高和很低的页面，这些数据不需要进入下一轮计算
4. 然后利用离线训练得到的双塔网页相似度判别模型，结合向量索引技术进行相似性粗筛，进一步筛选出相似度较高的网页
5. 最后利用基于交互的网页相似度计算方法，对上一步得到的候选进行两两判断，最终得到真正相似度很高的网页

同时离线侧利用视觉表征模型等方式构造远程监督信号，微调网页相似度模型用于相似度分析，提升相似度判别效果

**附：基于交互的网页相似度计算**

类似文本相似度任务，通常单塔模型由于从底层就进行了细粒度交互，效果比最上层通过两个embedding计算余弦相似度分数的双塔模型更好 对于domTree相似度判定也是类似，先把网页编码为向量，再计算向量间距离/相似度的方式， 由于没有页面子结构/细节之间的匹配，它适用于区分风格样式差别较大的网页，例如不同域的页面，对于需要页面内细节比较的情况并不适合，这种情况需要网页 间基于交互的相似度判定方式；网页html源码很长并且是典型的树型结构，适合通过分治的方法进行相似度计算，具体步骤为：

1. 待比较的两个网页通过上述方法构造domTree并预处理
2. 预定义基于属性类型和属性值的html结点间距离计算方法，以及基于text-encoder或编辑距离等字面距离的子树距离计算方法
3. 从根结点开始，自顶向下的比较两个domTree子树间相似度
   1. 如果两个domTree当前结点的深度达到阈值，利用两个结点代表的子元素的html源码（通过先序遍历获得）采用2中定义的方法直接计算当前子树距离（这一步是因为深度较大的通常是页面中的细粒度元素，对整体结构影响不大，不需要太精细的比较，可以当成父元素里的文本，直接通过文本距离比较；并且元素嵌套通常也不高，可以适配主流text-encoder最大长度）

3.2 否则需要分治计算；先利用2中定义的结点距离计算两个结点本身的距离，然后对每个子结点，递归的进入3计算子树距离，当前子树距离等于当前根结点间距离和所有子树距离之和