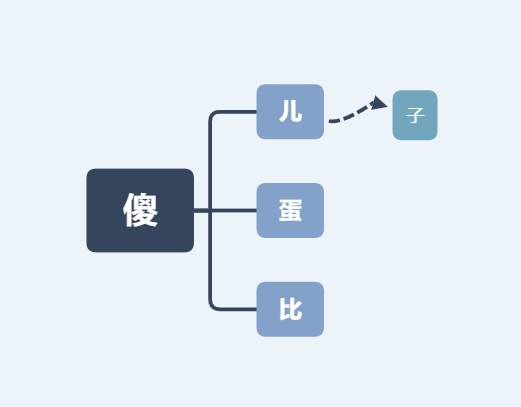
敏感词检测：

一种名为DFA（确定有穷自动机）的算法，基于状态转移来检索敏感词，只需要扫描一次待检测文本，效率高。

算法思路如下：

将敏感词集合构成如下的许多树形结构：



在这样的树结构的条件下，遍历待检测的文档，依次用每个字符去查找树的根节点，找到对应根节点的树后，再从子树中检测是否有对应的下一个字符，一直这样下去，如果能查找到树的叶子节点，说明敏感词匹配成功。

在具体的实现中，采用HashMap的数据结构来构成树，HashMap的特点是将键映射到具体的对象，且时间复杂度为O（n）。initMap函数负责将全部的敏感词生成一颗树，流程如下：

将全部敏感词以Set集合的形式作为参数传入，首先初始化一个HashMap类型的变量sensitiveWordsMap作为全部敏感词的根，对于每个敏感词，执行这样的操作：以第一个字符为键，查找sensitiveWordsMap，找到子树对应的Map则进入下一个节点继续以下一个字符为键查找子Map，如果没有则以此字符为键生成一个子Map加入到父Map中，以此类推，直到敏感词结束。initMap执行完毕后，类对象sensitiveWordsMap保存了敏感词对应的完整树结构。

**private void** initMap(Set<String> words) {  
  
 **sensitiveWordsMap** = **new** HashMap<>(words.size());  
 String currentStr;  
 Map<Object, Object> currentMap;  
 Map<Object, Object> subMap;  
  
 **for** (String word : words) {  
 currentStr = word;  
 currentMap = **sensitiveWordsMap**;  
  
 **for** (**int** i = 0; i < currentStr.length(); i++) {  
 **char** c = currentStr.charAt(i);  
 subMap = (Map<Object, Object>) currentMap.get(c);  
 **if** (subMap == **null**) {  
 subMap = **new** HashMap<>();  
 currentMap.put(c, subMap);  
 currentMap = subMap;  
 } **else** {  
 currentMap = subMap;  
 }  
 **if** (i == currentStr.length() - 1) {  
 currentMap.put(***END***, **"end"**);  
 }  
 }  
 }  
}

getSensitiveWordLength方法接受一个String参数，表示待查找的全文，一个int参数，表示待查找的位置，这个方法负责返回从位置i的字符开始敏感词的长度，如果不存在敏感词则返回0。

方法流程如下：定义变量currentMap用来表示当前的树或者子树对应的HashMap，初始指向类成员**sensitiveWordsMap**（此时它已经保存了敏感词的树结构），定义变量subMap表示当前currentMap的子树，length表示敏感词长度，flag表示是否查找成功。定义一个循环，先获取第i个位置的字符，在sensitiveWordsMap中以这个字符为键查找，如能找到子树对应的Map则将currentMap设为此子Map，length加一，进入下一次循环，找不到则退出循环。每次length+1时，要检查是否包含END这个Key(是我们在定义敏感词树时加入到叶节点下，用于判断是否为叶节点)，包含则将flag置为true说明匹配到了完整的敏感词。在方法返回之前会检查flag，如果不是true则返回0，说明未匹配到敏感词或只匹配上了一部分敏感词。

**private int** getSensitiveWordLength(String txt, **int** i) {  
  
 Map<Object, Object> currentMap = **new** HashMap<>();  
 Map<Object, Object> subMap = **new** HashMap<>();  
 currentMap = **sensitiveWordsMap**;  
  
 **int** length = 0;  
 **boolean** flag = **false**;  
 **char** currentChar;  
  
 **while** (**true**) {  
 currentChar = txt.charAt(i);  
 subMap = (Map<Object, Object>) currentMap.get(currentChar);  
  
 **if** (subMap == **null**) {  
 **break**;  
 } **else** {  
 length++;  
 **if** (subMap.containsKey(***END***)) {  
 flag = **true**;  
 }  
 currentMap = subMap;  
 }  
 i++;  
 }  
 **if** (!flag) {  
 length = 0;  
 }  
 **return** length;  
}

getSensitiveWords方法用来对全文进行完整的匹配，即循环传入的文档中的每一个字符，对这个字符调用getSensitiveWordLength，如果匹配到敏感词，则将敏感词的位置和长度打包成一个Set与敏感词本身这个String一起加入到HahsMap中，最后方法返回这个HashMap，供其他方法调用。

**public** Map<Set<Integer>, String> getSensitiveWords(String txt) {  
  
 Map location = **new** HashMap<Set<Integer>, String>(); *//Set里有两个元素，分别是初始位置和长度* **for** (**int** i = 0; i < txt.length(); i++) {  
 **int** sensitiveWordLength = getSensitiveWordLength(txt, i);  
 **if** (sensitiveWordLength > 0) {  
 Set key = **new** LinkedHashSet();  
 key.add(i);  
 key.add(sensitiveWordLength);  
  
 String value = txt.substring(i, i + sensitiveWordLength);  
  
 location.put(key, value);  
 i = i + sensitiveWordLength - 1;  
 }  
  
 }  
  
 **return** location;  
}