

和算法相关，创建一个敏感词树（多叉），将每个敏感词的第一个字符（不重复）记入根节点的子节点，敏感词最后一个字符记上标记，在检查敏感词时，一个指针指向敏感词字符查到的位置，两个指针指向字符串（需要过滤敏感词的文本），一个指向敏感词头部（不是就往后移动一位），一个指向需要匹配的字符

具体算法：敏感词是bf，abc。 敏感词尾部是f和c

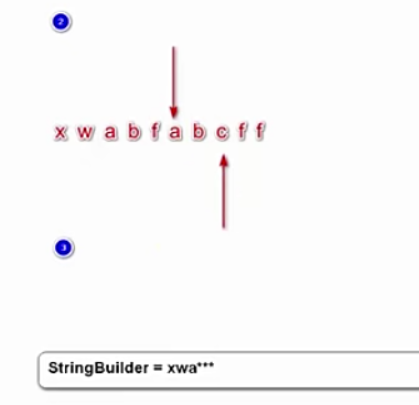
用两个指针ab走,ab从x开始,b开始走到w，发现没找到敏感词，则

ab都移到w，发现不是敏感词，stringbuilder append上

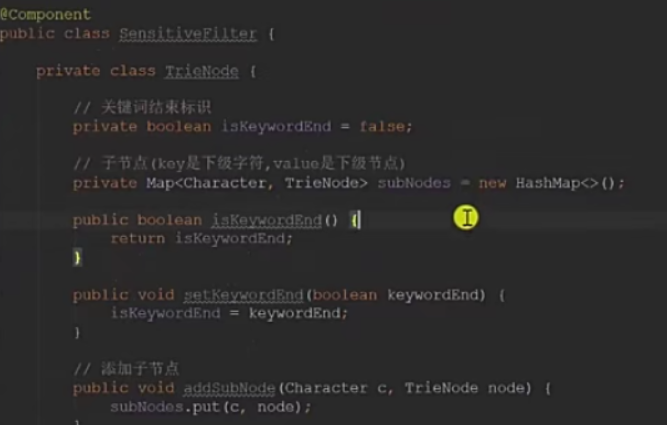
ab都到a，b走到b发现是(但不是敏感词尾部就继续走)，再走到f发现不是，stringbuilder

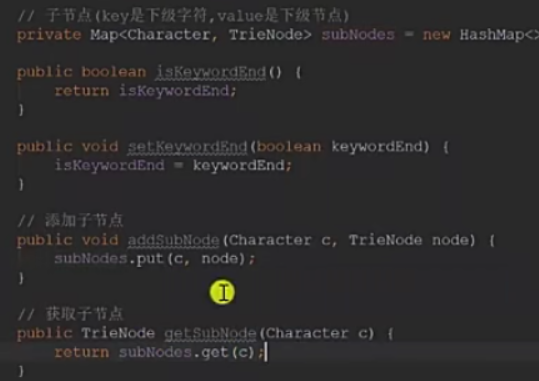
ab都到b，b走到f发现f是敏感词尾部，则说明是敏感词，在stringbuilder加上\*\*

当然算法里还会跳过符号，防止自&杀 这样跳过过滤

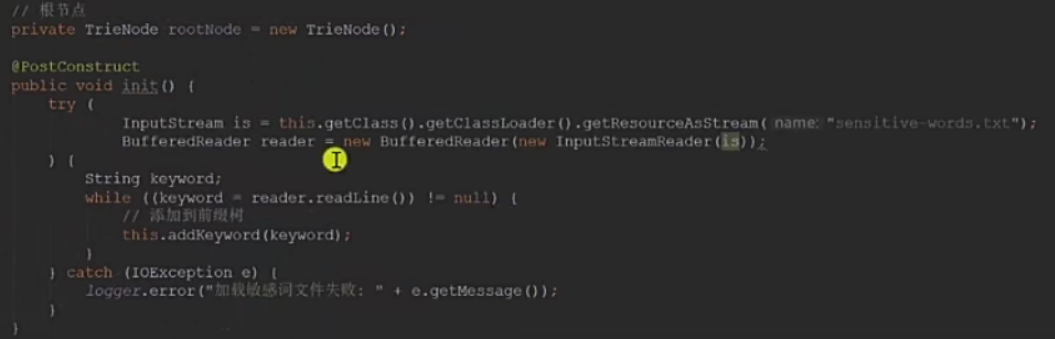


1.定义前缀树



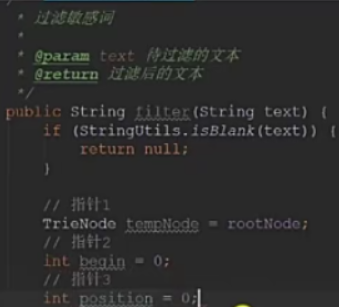


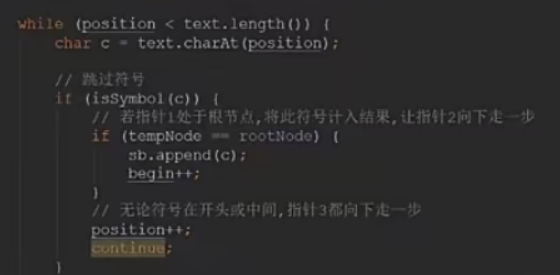
2.初始化





1. 过滤敏感词的方法









class Trie {

private Trie[] children;

private boolean isEnd;

public Trie() {

children = new Trie[26];

isEnd = false;

}

public void insert(String word) {

Trie node = this;

for (int i = 0; i < word.length(); i++) {

char ch = word.charAt(i);

int index = ch - 'a';

if (node.children[index] == null) {

node.children[index] = new Trie();

}

node = node.children[index];

}

node.isEnd = true;

}

public boolean search(String word) {

Trie node = searchPrefix(word);

return node != null && node.isEnd;

}

public boolean startsWith(String prefix) {

return searchPrefix(prefix) != null;

}

private Trie searchPrefix(String prefix) {

Trie node = this;

for (int i = 0; i < prefix.length(); i++) {

char ch = prefix.charAt(i);

int index = ch - 'a';

if (node.children[index] == null) {

return null;

}

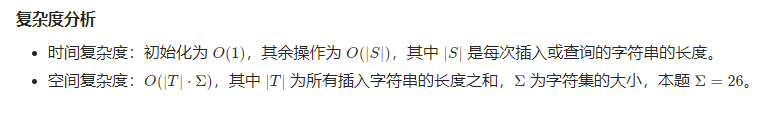
node = node.children[index];

}

return node;

}

}



根节点不放字符

空间换时间

100000个长度不超过10的单词。对于每一个单词，我们要判断他出没出现过，如果出现了，求第一次出现在第几个位置。

如果我们使用一般的方法，没查询一个单词都去遍历一遍，那么时间复杂度将为O(n^2),这对于100000这么大的数据是不能够接受的。假如我们要查找单词student。那我们通过前缀树只需要查找s开头的即可，然后接下来查询t开头的即可，对于大量的数据可以省去不小的时间







**优化空间复杂度**：

使用DoubleArrayTrie降低空间复杂度

简单的说，DoubleArrayTrie本质是一个确定有限状态自动机（DFA），使用两个线性数组（base数组和check数组）来表示Trie树，对Trie的结构进行了空间上的压缩，但同时不降低查询速度