敏感词系统，是专门用来扫描文本中得敏感词汇的服务工具。**开发语言是java，以单进程多线程模式的HTTP独立服务器模式运行在后台。最主要的特点是快速高效，服务器加载一次性把所有的敏感词加载到内存中，所有的查询业务都是在内存中完成**。

分析业务场景，如果一个文本在一万字左右，违禁词表十万级别，如果对这种级别的内容进行快速扫描，就需要很好地一个解决方案。先分析几种一般的解决方案：

     1）**最简单的方案是，直接用正则表达式，对文本内容进行扫描，这种方式的优点是简单，因为各种脚本语言，包括java等都有正则表达式的支持，因此实现简单方便。如果对文本长度不多，需要检测的单词不多的情况下，实现简单，时间花费还不算多够用。但是对大文本的扫描或者一定数量的被扫描的词，效率的问题就凸现出来了。正则表达式的效率是一个最大的瓶颈**。

     2）**如果使用PHP开发的话，用strpos,str\_replace等之类的函数直接查找或者替换文本内容，也是最简单的方式，但是问题仍然是效率问题，对大文本搜索查找时间过长。另外，java及其他的脚本语言也有类似的内置函数，这些函数对简单的文本替换和查找，没有什么问题，但是对大文本，一定数量的词的查找就会显现出查找效率的问题**。

       以上两种方法，其实过程都差不多，优点都是简单，一般是用在一些表单提交的时候，进行数据过滤的时候使用。但是对大文本多词搜索来说，效率太低。原因有两块，比如使用php来进行开发，首先每次运行的时候，要把文本和被查找词加载到内存中，运行完之后就会销毁，然后下次运行重新加载所有的需要被搜索的词到内存中。其次是，检索过程的是，针对每一个词循环进行多次扫描，比如被搜索的文本长度是n,敏感词是m个，那么至少被搜索次数是nm。

改进方案：首先，对高频的检索业务，独立出来，成为一个独立的系统。仍然以http服务器的方式进行运行，一次性启动加载所有的违禁词表加载到内存中，一次加载之后在整个服务运行期间常驻内存，可以循环的多次被利用。违禁词表，在内存中以高效的数据结构方式存在，使得每次查找词表的时候，都能以o(1)的时间复杂度进行快速查找。这样对一个长度为n的文本，检索的时间最短可以是o(1)\*n，也就是时间复杂度为n，整个查找过程是需要循环为n次，即只需要扫描文本一遍就刚好把所有的词都扫出来了。因为对文本的任何检索操作，最短的操作，也是要循环扫描整个文本所有词一遍。因此这已经是理想中最短的检索方式了。

提出以上设想方案之后。接下来的思路是，如何设计合适的数据结构和算法，来实现复杂度为n\*o(1)的检索方案。我们的思路是，首先是考虑到这个方案最核心的着眼点是，o(1)的实现，而这个决定这个o(1)的因素是在敏感词表的在内存存储的数据结构形式上。在所有的算法中，我们就可以快速联想到，hash树这个数据结构。利用java的hash算法就可以实现快速的o(1)的查找。

于是，我们先把敏感词词表的形式列出来，假定我们有一张敏感词表如下：

敏感词

小人

小苹果

日本

日本人

日本电影

这个是在存储在文本中的一般形式，我们可以把它想象成是一个矩阵结构，高度是所有的词的个数，而宽度是敏感词字数最多那个词的个数。对这个矩阵中，空白的词以空白字符来填满。这样我们可以接下来可以对这个矩阵进行压缩，把它变成一颗颗的树。这个过程是这样的，首先横向去看，字跟字之间是有线性关系，因此无法压缩，必须保持变形之后字跟字之间的先后关系不能被破坏。其次再看，竖向去看，每列上下之间的字是彼此无关联的，处理离散状态。因此可以进行压缩。于是经过竖向压缩之后,就变成如下结果：

敏感词

小人

    苹果

日本

        人

        电影

结果变成了三颗hash树， 同时为标记词的完整性，可以在每个结束单词或字的对应值存储一个\0的空白字符标注结束,这样，整个此表的hash树结构如下

敏=>感=>词\0

小=>人\0

        苹=>果=>\0

日=>本=>\0

               人\0

               电=>影=>\0

处于同一列表示在同一个set集合里面。如果用户PHP二维数组来表示的话，可以很形象的表示为：

        array(

            '敏' => array(

                '感' => array(

                    '词' => "\0"

                )

            ),

            '小' => array(

                '人' => "\0",

                '苹' => array(

                    '果' => "\0",

                    "\0"

                )

            ),

            '日' => array(

                '本' => array(

                    '人' => "\0",

                    '电' => array(

                        '影' => "\0"

                    ),

                    "\0"

                )

            )

        );

假如，有一个文本是:”哪里有日本AV”,在检索整个文本的时候，依次是顺序是，先检查第一列的hash表，会发现，比如，”哪“很快就发现，不在集合里面就会直接跳过，这样快速的扫过直到出现”日“这个字的时候，发现在集合中出现了，然后再根据这个字索引到对应的集合中，看到它对应的集合是 “本,\0”这样继续扫描下一个词的时候，发现 “本”也在集合中，然后继续查找“本”对应集合”人，A，\0“,按照这样的顺序执行下去，最后发现，找到文本中的违禁词，“日本”，“日本AV”，但是这里有一个问题就是在于，最大匹配和最小匹配的问题，因为”日本”和”日本AV”都是违禁词，因此，如果是最小匹配的话，就是“日本”这个词会先被匹配出来，然后按照违禁词的级别被替换到或者直接发出警报该文本不合法，如果是最大匹配的话，即使匹配出“日本”也依然会继续查找，直到找到“日本AV”整个词，或者找不到适配的敏感词才会跳转回来，保留查找到的最小匹配的“日本”这个词。

从这个当中可以看得出，相对来说，最小匹配效率要比最大匹配效率要高一些，但是无论以哪种方式找到违禁词并覆盖它们的话，都可以。一般来说实际上，更多的是倾向于最大匹配，因为查找到最大匹配的敏感词之后，然后直接替换这个词，可以把最小匹配的敏感词也一起替换掉。这两种效果，分别是“哪里有\*\*电影”，“哪里有\*\*\*\*”。

平安好房敏感词系统的git地址是：[git@git.ipo](mailto:git@git.ipo).[com:hf-dev-attach/app-service-illegal.git](http://comhf-dev-attach/)

在命令行下输入

**$ git**[**clone git@git.ipo**](mailto:clone%C2%A0git@git.ipo)**.**[**com:hf-dev-attach/app-service-illegal.git**](http://comhf-dev-attach/)

**就可以获得该项目的源代码，目录如下**：

**test**：存放的是测试脚本。

**assets**: 存放服务的配置文件，以及敏感词表，是一个txt文本。

**src**: 存放敏感词的java源代码。

**lib**: 存放的是第三方依赖库。

**bin**: 是已经编译过的java的.class 文件。

**sdk**: 包含php和java的sdk文件，在客户端文件里，开发者可以用该sdk直接调用敏感词服务。

**bannedword.jar**: 是以runable方式打包之后的jar包文件，这个是程序的所有文件包括lib库文件。所以只要执行该jar包文件就可以了。

**bannedword.bat**:是DOS命令脚本，在windows下启动违禁词服务器。

**MANIFEST.MF**: 是java编译jar包的时候，所使用的配置文件

**bannedword**: 是服务器端启动shell文本，可以在linux环境下，/etc/paths文本中设置改命令脚本的地址，这样直接在命令行中就可以输入：

**$ bannedword -t|-d >> /user/logs/bannedword/sys.log 2>&1 #-t表示测试模式启动，-d表示守候进程模式**

**bannedword-cli**: 是客户端shell脚本，可以通过该脚本直接跟敏感词服务器系统进行交互，一般是给运维使用。脚本的功能，可以批量添加和删除违禁词，也可以查询违禁词系统。同时可以切换到不同的违禁词服务器系统进行操作。

**$ ./bannedword-cli**

=======================平安敏感词过滤服务器系统=======================

本系统功能主要是用于操作违禁词服务器，主要功能包括:

1)添加违禁词删除违禁词

2)查看目前违禁词列表

3)以及检索文本是否包含违禁词

4)批量添加违禁词

5)批量删除违禁词

敏感词服务器目前以单进程多线程的方式运行

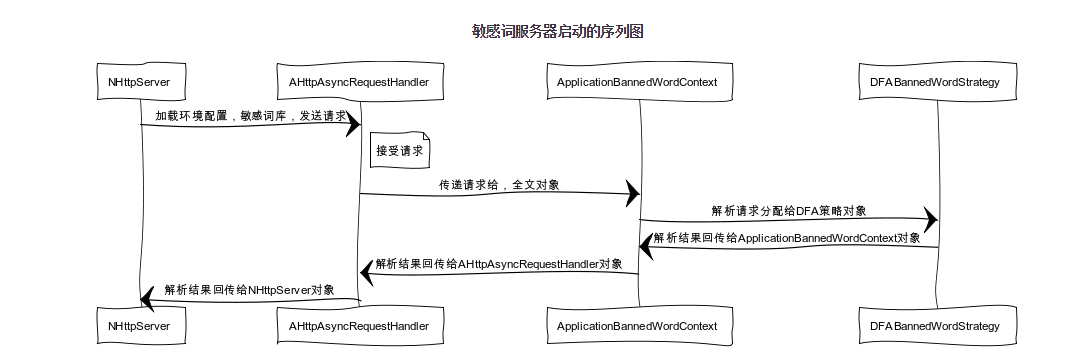
违禁词删除添加的操作都会实时持久保存。

每次重启都会加载上次操作的结果。

功能还会持续不断地改进，如果有好的建议也可以联系开发人员...

======================================================================

**敏感词服务器启动的序列图**



源代码编写的时候，可以通过策略模式，继承IBannedWordStrategy对象的方式，扩展不同的过滤违禁词的算法。然后根据不同的算法来提升违禁词的查找效率。

核心代码分析：

public void addWord(String word, int level)

 {

 // 保存到字典中

 ApplicationBannedWordContext.dict.put(word, level);

 HashMap<String, Object> banWordSet = ApplicationBannedWordContext.banWordDictionarie;

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

 {

 String chr = String.valueOf(word.charAt(i));

 // System.out.println("1 add chr:" + chr);

 HashMap<String, Object> node = (HashMap<String, Object>) banWordSet

 .get(chr);

 if (node != null)

 {

 banWordSet = (HashMap<String, Object>) node;

 node = null;

 } else

 {

 HashMap<String, Object> stateHashTable = new HashMap<String, Object>();

 banWordSet.put(chr, stateHashTable);

 banWordSet = stateHashTable;

 }

 if (i < word.length() - 1)

 {

 banWordSet.put(KEY\_OVER, VALUE\_NOT\_END);

 } else

 {

 banWordSet.put(KEY\_OVER, VALUE\_END);

 }

 }

 banWordSet = null;

 }

通过迭代的方式，循环构造哈希树字典。

检索文本的核心代码：

private int findBanWordReturnLen(String txt, int begin, int flag)

 {

 HashMap<String, Object> banWordSet = ApplicationBannedWordContext.banWordDictionarie;

 int maxMatchRes = 0;

 int res = 0;

 int l = txt.length();

 String word = "";

 for (int i = begin; i < l; i++)

 {

 char chr = txt.charAt(i);

 word = String.valueOf(chr);

 // 过滤掉空白字符以及非英文中文数字字符

 // 查找该字符是否在哈希树中

 if (!banWordSet.containsKey(word))

 {

 txt = null;

 banWordSet = null;

 return maxMatchRes;

 }

 // 沿着哈希树继续查找下一个字符

 Object wordMap = banWordSet.get(word);

 // KEY\_OVER特殊字符会被忽略掉

 if (wordMap instanceof HashMap)

 {

 res++;

 banWordSet = (HashMap<String, Object>) wordMap;

 if (((String) banWordSet.get(KEY\_OVER)).equals(VALUE\_END))

 {

 if (flag == MATCH\_TYPE\_MIN)

 {

 wordMap = null;

 banWordSet = null;

 txt = null;

 return res;

 } else

 {

 maxMatchRes = res;

 }

 }

 }

 }

 txt = null;

 banWordSet = null;

 return maxMatchRes;

 }

通过迭代遍历hash树的方式，查找每个树的对应集合或者值是否含有违禁词。最外层是一个，遍历文本的循环，在这个循环中先从树的根部进行查找，如果发现根部符合，则进入内部循环，在这个循环中会，进入下一个节点分支继续查找，然后直到到最外部的叶节点，或者不符合的词，就退出整个内部循环，然后回到上一个循环继续查找。找到了对应的结果之后，就把违禁词出现的，位置和长度，都记录下来。

这样就可以，在文本中替换或者删除违禁词文本的操作了。

因为文本字符在java中是以定长的字符数组的方式存储，它是一个线性数组，所以遍历的的速度很快，可以知道位置和长度，可以直接一步定位到违禁词所在的位置，这样查询的速度是瞬间移动的。所以很快。

在这种算法中，最大的特点是，无论有词表中有多少的违禁词，查询的速度对指定的文本都是恒定的。也就是说，对同一个被扫描的文本来说，无论增加多少的违禁词，其查询的速度都不太会受到多少影响。实际上，一个文本上含有多个违禁词的概率也不是太多。有的文章一旦含有某些级别程度高的违禁词，再查出第一个违禁词的时候就可以直接拦截了。比如一些垃圾广告的文章，含有一些垃圾信息，基本可以直接拦截，根本不需要把所有的违禁词都扫出来！