敏感词过滤

gaccob

2012 年 11 月 25 日

一个简单的敏感词过滤的设计:

- 最基本的思路是建立hash表来做查询.
- 敏感词的长度一般不会太长，一般也就2个汉字(4个字节)左右，可以再做一份敏感词长度的索引，这样子在处理时能提高效率.
- 编码问题：词库和源因为以往的历史原因，是gbk编码，而非utf.
- 目前是一个英文忽略大小写的设计.
- 代码在gbase中，关键部分的代码如下所示:

```
    // dirty word
    typedef struct dirty_t
    {
        char word[MAX_DIRTY_WORDS_LEN];
        int prev;
        int next;
    } dirty_t;

    // dirty context
    typedef struct dirty_ctx_t
    {
        int table_size;
        dirty_t table[MAX_DIRTY_WORDS_COUNT];
        int hash[MAX_DIRTY_WORDS_HASH_COUNT];
        int index[256][256];
    } dirty_ctx_t;

    // if > 0x80, means double bytes
    // else, single byte
    const uint8_t GB_SPEC_CHAR = (uint8_t)(0x80);

    // dirty words check
    int dirty_check(dirty_ctx_t* ctx, const char* src, int len)
    {
        static char lowcase[MAX_SOURCE_WORDS_LEN];
        int i, k, step, key;
        const uint8_t* from;

        if (!ctx || !src || len > MAX_SOURCE_WORDS_LEN) {
            return -3;
        }
        for (i = 0; i < len; i++) {
            lowcase[i] = tolower(src[i]);
        }

        for (i = 0, step = 0; i < len; i += step) {
            key = 0;
            from = (const uint8_t*)&lowcase[i];
            if (from[0] < GB_SPEC_CHAR) {
                key = ctx->index[0][from[0]];
                step = 1;
            } else if (i + 1 < len) {
                key = ctx->index[from[0]][from[1]];
                step = 2;
            } else {
                printf("source code error\n");
                return -2;
            }
            // no index, go ahead
            if (0 == key) {
                continue;
            }
            // found key
            for (k = 1; k < MAX_DIRTY_WORDS_LEN; k++) {
                // exceed len
                if (i + k > len) {
                    break;
                }
                // no dirty
                if (0 == CHECK_DIRTY_FLAG(key, k)) {
                    continue;
                }
                if (0 == dirty_hash_find(ctx, (const char*)from, k)) {
                    return -1;
                }
                // no need to loop all
                RESET_DIRTY_FLAG(key, k);
                if (0 == key) {
                    break;
                }
            }
        }
        return 0;
    }
```