# 字串切割：STRTOK、STRTOK_R與STRSEP

Sep 6, 2015 • C, strtok, strtok_r, strsep

對於字串切割、C標準函式庫提供了這幾個函式：strtok，strtok_r，strsep；使用時、只需要包含表頭檔string.h即可。詳細說明，參考連結：strtok(3)，strsep(3)。

由於C語言本身的限制以及在執行效率/資源佔有方面的考量與妥協、這些函式顯得並不那麼好用，甚而讓人不禁懷疑「誒？這真的是標準庫提供的函式麼？」。這篇筆記用來記述它們的使用方法、以及一些注意點。

在分述它們的不同之處之前、先說說它們的共同之處吧：

- 其一、呼叫函式會修改原字串內容（將切割符改為`'\0'`），故待切割字串不可為無法修改的字串字面常數(string literal)。
- 其二、單次呼叫只能完成一次切割、而不是直接提供切割完畢的字串數組。
- 其三、關於切割符字串（更確切的說、切割符集合），其內任一字符（而不是整體）都是切割符，而且在每次呼叫函式時都可以依據需求採用相同/不同的字串。

## strtok

函式原型：char *strtok(char *str, const char *delim);

三個函式中，它的「資歷」最老、兼容性也是最好的。由於只修改字串內容、故待切割字串也可存放在字符數組中。但因其內置了用於指示下次掃描位置的指標，故不能用於多執行緒(multi-thread)情景中。在切割字串的過程中，參數str在呼叫一次之後必須設為空(NULL)。

在下面的範例中，將待分割字串存放在字符數組中：

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
    const char * const str = "MAC-:00-1C-42-A7-F1-D9";
    const char * const delim = "-:";
    char buf[30] = {0};
    char *substr = NULL;
    int count = 0;

    strcpy(buf, str);
    printf("original string: %s (@%p)\n", buf, buf);

    substr = strtok(buf, delim);
    do {
        printf("#%d sub string: %s (@%p)\n", count++, substr,
        substr = strtok(NULL, delim);
    } while (substr);

    printf("original string after 'strtok': ");
    for (count = 0; count < strlen(str); count++) {
        printf("%c", buf[count]?:'*');
    }
    printf(" (@%p)\n", buf);

    return 0;
}
```

輸出結果為：

```
original string: MAC-:00-1C-42-A7-F1-D9 (@0x7fff5fbff7e0)
#0 sub string: MAC (@0x7fff5fbff7e0)
#1 sub string: 00 (@0x7fff5fbff7e5)
#2 sub string: 1C (@0x7fff5fbff7e8)
#3 sub string: 42 (@0x7fff5fbff7eb)
#4 sub string: A7 (@0x7fff5fbff7ee)
#5 sub string: F1 (@0x7fff5fbff7f1)
#6 sub string: D9 (@0x7fff5fbff7f4)
original string after 'strtok': MAC*:00*1C*42*A7*F1*D9 (@0x7f
```

注：最後一列、被'\0'替換的字符已用'*'打印出來。下同、不再贅述！

由此可以看出，每次呼叫strtok、它都會從掃描位置開始掃描字串、將其後第一個出現的切割符改為'\0'、然後修改掃描位置、最後將原掃描位置返回。有趣的是、當切割符連續出現時、strtok並不會返回空字串（即""）、而是選擇跳過。

在上一範例中，第8列引入了固定長度的字符數組，在不確定待切割字串長度的情景中，容易引發緩衝區溢位(buffer overflow)，故不推薦，可參考下面的範例：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
    const char * const str = "MAC-:00-1C-42-A7-F1-D9";
    const char * const delim = "-:";
    char * const dupstr = strdup(str);
    char * substr = NULL;
    int count = 0;

    printf("original string: %s (@%p)\n", dupstr, dupstr);

    substr = strtok(dupstr, delim);
    do {
```

```
        printf("#%d sub string: %s (@%p)\n", count++, substr,
        substr = strtok(NULL, delim);
    } while (substr);

    printf("original string after 'strtok': ");
    for (count = 0; count < strlen(str); count++) {
        printf("%c", dupstr[count]?:'*');
    }
    printf(" (@%p)\n", dupstr);

    free(dupstr);

    return 0;
}
```

輸出結果為：

```
original string: MAC-:00-1C-42-A7-F1-D9 (@0x1002068d0)
#0 sub string: MAC (@0x1002068d0)
#1 sub string: 00 (@0x1002068d5)
#2 sub string: 1C (@0x1002068d8)
#3 sub string: 42 (@0x1002068db)
#4 sub string: A7 (@0x1002068de)
#5 sub string: F1 (@0x1002068e1)
#6 sub string: D9 (@0x1002068e4)
original string after 'strtok': MAC*:00*1C*42*A7*F1*D9 (@0x10
```

## strtok_r

函式原型：char *strtok_r(char *str, const char *delim, char **saveptr);

此為strtok的多執行緒安全的重入版本，新增參數saveptr，用於存放指示下次掃描位置的指標、其值不能為空；strtok_r在第一次呼叫時會忽略它的值（這意味著其初始值可為NULL）。與strtok一樣、待切割的字串亦可存放在字符數組中。

參考範例如下：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
    const char * const str = "MAC-:00-1C-42-A7-F1-D9";
    const char * const delim = "-:";
    char * const dupstr = strdup(str);
    char *saveptr = NULL;
    char *substr = NULL;
    int count = 0;

    printf("original string: %s (@%p)\n", dupstr, dupstr);
    substr = strtok_r(dupstr, delim, &saveptr);

    do {
        printf("#%d sub string: %s (@%p)\n", count++, substr,
        substr = strtok_r(NULL, delim, &saveptr);
    } while (substr);

    printf("original string after 'strtok_r': ");
    for (count = 0; count < strlen(str); count++) {
        printf("%c", dupstr[count]?:'*');
    }
    printf(" (@%p)\n", dupstr);

    free(dupstr);

    return 0;
}
```

輸出結果為：

```
original string: MAC-:00-1C-42-A7-F1-D9 (@0x1002068d0)
#0 sub string: MAC (@0x1002068d0)
#1 sub string: 00 (@0x1002068d5)
#2 sub string: 1C (@0x1002068d8)
#3 sub string: 42 (@0x1002068db)
#4 sub string: A7 (@0x1002068de)
```

```
#5 sub string: F1 (@0x1002068e1)
#6 sub string: D9 (@0x1002068e4)
original string after 'strtok_r': MAC*:00*1C*42*A7*F1*D9 (@0x
```

## strsep

函式原型：char *strsep(char **stringp, const char *delim);

最後說說strsep。與strtok_r同樣、它可以用在多執行緒情景中；但是、與前者不同的是、它不單修改字串的內容，而且還修改了字串的位址指標。

這就意味著：

- 使用字符數組存放待分割字串是不允許的。
- 存放待分割字串的位址的常數指標是不允許的。
- 待分割字串的位址是需要使用額外的指標進行備份的。

參考範例如下：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
    const char * const str = "MAC-:00-1C-42-A7-F1-D9";
    const char * const delim = "-:";
    char * const dupstr = strdup(str);
    char *sepstr = dupstr;
    char *substr = NULL;
    int count = 0;

    printf("original string: %s (@%p)\n", sepstr, sepstr);
    substr = strsep(&sepstr, delim);

    do
    {
        printf("#%d sub string: %s (@%p)\n", count++, substr,
```

```
        substr = strsep(&sepstr, delim);
    } while (substr);

    printf("original string after 'strsep': ");
    for (count = 0; count < strlen(str); count++) {
        printf("%c", dupstr[count]?:'*');
    }
    printf(" (@%p)\n", sepstr);

    free(dupstr);

    return 0;
}
```

輸出結果為：

```
original string: MAC-:00-1C-42-A7-F1-D9 (@0x1002068d0)
#0 sub string: MAC (@0x1002068d0)
#1 sub string:  (@0x1002068d4)
#2 sub string: 00 (@0x1002068d5)
#3 sub string: 1C (@0x1002068d8)
#4 sub string: 42 (@0x1002068db)
#5 sub string: A7 (@0x1002068de)
#6 sub string: F1 (@0x1002068e1)
#7 sub string: D9 (@0x1002068e4)
original string after 'strsep': MAC**00*1C*42*A7*F1*D9 (@0x0)
```

有一點須注意：與strtok/strtok_r不同、strsep會返回空字串。這在某些情景中相當有用。

## 小結

關於它們的概覽、可以參考以下表格：

| 函式 | 屬性 | 值 | 合規 |
| --- | --- | --- | --- |
| strtok | 執行緒安 | 非多執行緒安 | POSIX.1-2001, POSIX.1-2008, C89, |

| | | | |
|---|---|---|---|
| | 全 | 全 | C99, SVr4, 4.3BSD |
| strtok_r | 執行緒安全 | 多執行緒安全 | POSIX.1-2001, POSIX.1-2008 |
| strsep | 執行緒安全 | 多執行緒安全 | 4.4BSD |

那麼、何時使用它們？以下建議、僅供參考：

- 如果需要空字串、且不允許多個切割符相鄰、及不考慮可移植性，則可選用strsep。
- 如果允許多個切割符相鄰、且不需要空字串，則可選用strtok_r。
- 如果有萬分之一之可能、優先選用或自行實作strtok_r/strsep、而不是使用strtok；strtok可取的只有可移植性佳這一點，可以新作滿足多執行緒安全的函式呼叫之。