1. STL-string

1.1.string 源码解析

1.1.1 string 定义

C++标准库中关于 string 的定义在 stringfwd.h 头文件中，linux 操作系统下路径(以下所有代码路径都是 linux 操作系统下)为：/usr/include/c++/4.8.2/bits，具体定义如下截图：

```
/// A string of @c char
typedef basic_string<char>    string;
```

由如上截图中的内容可知，string 的定义其实是 basic_string 模板类对 char 的一个模板实例化。

basic_string 模板类定义在 basic_string 文件中，具体路径为(/usr/include/c++/4.8.2/bits)。关于 basic_string 模板类的具体实现放在了 basic_string.tcc 当中，c++里面模板的实现不能放在.cpp 文件中，必须写在头文件中，如果模板函数实现较复杂，就会导致头文件臃肿和杂乱，这里可以看到 stl 里面方法，就是把较复杂的实现放在.tcc 文件里面，然后当做头文件来包含，我们在写模板代码的时候也可以以此为参考。

1.1.2 basic_string 成员定义

首先，在 basic_string.h 顶头注释中我们看到关于几个主要的类成员定义，如下截图：

```
*
*  @code
*                                              [_Rep]
*                                              _M_length
*   [basic_string<char_type>]                  _M_capacity
*   _M_dataplus                                _M_refcount
*   _M_p ---------------->                     unnamed array of char_type
*  @endcode
*
```

这里其实是介绍了 basic_string 的内存布局，从起始地址出开始，_M_length 表示字符串的长度、_M_capacity 是最大容量、_M_refcount 是引用计数，_M_p 指向实际的数据。值得注意的是引用计数，说明该版本的 string 实现采用了 copy-on-write 的方式来减少无意义的内存拷贝。

由上图中的定义我们进行计算 std::string 的大小应该为 24 字节 (没有数据) 或者 32 字节（有数据）。其实不然，C++对象的大小是根据非静态成员的大小极其个数进行计算的，静态成员变量和成员函数是不计算在内的，因此根据 basic_string.h 中的定义可以看出，非静态成员变量只有一个，如下截图：

```
private:
  // Data Members (private):
  mutable _Alloc_hider  _M_dataplus;
```

关于_Alloc_hider 的定义(/usr/include/c++/4.8.2/bits/basic_string.h)如下图：

```
// Use empty-base optimization: http://www.cantrip.org/emptyopt.html
struct _Alloc_hider : _Alloc
{
    _Alloc_hider(_CharT* __dat, const _Alloc& __a)
        : _Alloc(__a), _M_p(__dat) { }

    _CharT* _M_p; // The actual data.
};
```

从上图我们可以看出，_Alloc_hider 是个结构体类型，其中存放的就是前面提到的_M_p 变量，即模板类实际指向的数据体。_Alloc_hider 的基类_Alloc 是一个分配器，无成员变量，大小为 0。则 sizeof(std::string)的大小为 8（64 位系统）/4（32 位系统）。

如果计算 std::string 所花费的总空间字节数应该为 sizeof(std::string) + sizeof(' ') + sizeof(_Rep) = 33 个字节。至于 sizeof(' ')和 sizeof(_Rep)后面会提到，即必须以 null 结尾多出来的一个字节大小和其他 3 个变量的所占字节大小，其中 sizeof(std::string)的大小为栈空间大小，这一点要区分开来。

由上我们已经解析了 basic_string 的四个主要成员中的一个，至于另外三个成员的定义如下(_M_length、_M_capacity、_M_refcount)：

```
private:
// _Rep: string representation
//   Invariants:
//   1. String really contains _M_length + 1 characters: due to 21.3.4
//      must be kept null-terminated.
//   2. _M_capacity >= _M_length
//      Allocated memory is always (_M_capacity + 1) * sizeof(_CharT).
//   3. _M_refcount has three states:
//      -1: leaked, one reference, no ref-copies allowed, non-const.
//       0: one reference, non-const.
//     n>0: n + 1 references, operations require a lock, const.
//   4. All fields==0 is an empty string, given the extra storage
//      beyond-the-end for a null terminator; thus, the shared
//      empty string representation needs no constructor.

struct _Rep_base
{
    size_type       _M_length;
    size_type       _M_capacity;
    _Atomic_word        _M_refcount;
};
```

从上图我们可以看出，这三个成员通过一个结构体来进行管理，而不是直接作为类成员进行管理，猪样就只需要定义一个_Rep 指针即可，最大限度的减小了 string 对象的大小，减小了对象拷贝的消耗。

从上图注释 1.String really contains _M_length + 1 characters: due to 21.3.4 must be kept null-terminated.中，我们可以看出，string 始终由_M_length+1 个字符，必须保证以 null 进行结尾。(2.3.4 注释后面继续解释)

1.1.3 basic_string 构造函数